Proceedings of IEEE Fourth International Conference on Space Mission Challenges for Information Technology (SMC-IT), Palo Alto, CA, August 2011, pp. 85-92.

Software Productivity of Field Experiments Using the Mobile Agents Open Architecture with Workflow Interoperability

William J. Clancey, Michael Lowry, Robert Nado, Maarten Sierhuis
Intelligent Systems Division
NASA Ames Research Center
Mountain View, CA 94035
William.J.Clancey@NASA.gov

***Abstract*—We analyzed a series of ten systematically developed surface exploration systems that integrated a variety of hardware and software components. Design, development, and testing data suggest that incremental buildup of an exploration system for long-duration capabilities is facilitated by an open architecture with appropriate-level APIs, specifically designed to facilitate integration of new components. This improves software productivity by reducing changes required for reconfiguring an existing system.**

Keywords-Open architecture; interoperability; service-oriented architecture; multiagent systems; exploration systems; workflow automation.

I. INTRODUCTION

Future human spaceflight missions to near Earth objects or Mars may be much longer and staged differently than expeditions in low Earth orbit. Unlike the International Space Station, systems may not be physically integrated, but still require interoperation [15]. In particular, exchanging data and commands among pressurized suits, vehicles/habitats, and robotic systems could improve safety and work productivity.

The Mobile Agents Project (2002-08) developed an open architecture for an end-to-end exploration system that facilitated crew self-reliance during simulated science EVAs [4][5]. This report systematically analyzes software productivity data from a series of field experiments involving ten system reconfigurations. The objective is to develop informative metrics for the design and value of software open architectures intended for efficient reconfiguration.

In particular, this study sought data to assess whether the incremental buildup of an exploration system for long-duration, remote operations is facilitated by an open architecture whose APIs are specifically designed to simplify integration of new components. This approach maximizes flexibility and minimizes costs by reducing changes to the existing system when reconfiguring components. The study shows the advantages of *composite APIs* that map conventional component APIs (which provide access to the functional methods and data objects of hardware and software) to a service-oriented language through which people and subsystems communicate.

The Mobile Agents Architecture (MAA) provides a common goal-oriented, model-based interface that automates communication of information and commands in a distributed, concurrent system of systems, consisting of a diversity of hardware and software interacting in a mobile, distributed environment [6]. MAA uses a service-oriented language for expressing *messages about tasks* (e.g., in terms of an EVA plan, astronauts, life support tools such as cameras, robotic assistants). This kind of composite API enables a diversity of hardware and software components (including COTS) to provide *task-oriented services* to the overall exploration system; in particular it enables highly flexible and efficient voice commanding.

The MAA open architecture, consisting of a workflow backbone of services ("agents" [17]) and composite APIs, enabled distributed, mobile agents to handle simultaneous goal-oriented requests on a non-reliable network. This architecture was developed from field experience with different existing hardware and software systems for both field science and routine habitat operations. The key architectural lessons concern how distributed mobile subsystems communicate, how task-level information and commands are represented, how services operating on the surface communicate with remote support teams, and how asynchronous services manage multiple, simultaneous requests.

In this paper we review the design of the MAA, the field experiments, and the data that was analyzed. We summarize the results of the analysis, interpreting charts and statistics in terms of the complexity and costs for reusing and modifying software. We describe what we learned about services workflow agents should provide and how they should be structured, and conclude with a summary of results and recommendations.

II. MOBILE AGENTS ARCHITECTURE

Workflow software agents in the Mobile Agents field experiments were implemented in the Brahms Language [1][16]. To employ a common messaging scheme each subsystem to be integrated is "agentified"—it is made to behave like a Brahms workflow agent by "wrapping" it with a Brahms Communications Agent (CA) using the Brahms JAVA API.

A subsystem is made into an agent by defining *SpeechActs* (structured messages, also "communicative acts") by which it will interact with other agents in the

system. For example a biosensor SpeechAct could correspond to the voice command, "What is the pulse?" The API of the subsystem is extended by the CA to translate its API calls to SpeechActs and vice versa. We refer to this as a "composite API" method. The subsystem's API and CA work together to enable communication with any workflow agent in the system: {Workflow Agents} ⇔ Communication Agent ⇔ Subsystem API ⇔ Subsystem.

Other agent-based languages could be used to implement the architecture described here. However, Brahms effectively provides direct support for an open architecture with interoperability via its SpeechAct and CA structures. Service-oriented architectures (SOA) promote a similar common messaging approach for interoperability, in which subsystems provide "services." The MAA illustrates how to provide an SOA by converting subsystems into agents. The diversity of MAA configurations demonstrates that agents are a good way to implement a software service; in particular, to make a component into a service, "agentify" it.

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

In itself, CORBA only enables exposing internal system classes in one object-oriented program to another program. This means that a program that uses exposed CORBA classes needs to understand the internal functions of the system with which it interacts. Every system has its own internal structure and functions, perhaps written in different programming languages. CORBA enables inter-process communication, by allowing indirect access to the local data storage of other processes.

In contrast, the agent approach does not expose a software process' internal methods/functions. It is based instead on a more abstract protocol that defines how agents communicate messages to each other and requires agents to respond to these messages by providing data or causing actions to occur. Using the agent approach, programmers do not need to provide CORBA object definitions to each developer of the integrated system. Instead, the team defines a communication message protocol—in MAA this is the syntax and language of SpeechActs—and then the developers may work independently.

For example, the voice commands for two robots "Boudreaux take a picture of me" and "Scout take a picture of me" differ in only one word in the language of the task. However, executing these requests involves different networks, APIs, command processing, and even different kinds of cameras, exemplifying how interoperability, decomposition, and abstraction relate. It is not necessary to pre-enumerate interactions—each agent handles cases that pertain to its own functionality, which decompose the request and pass on pieces to other agents, which do the same. Primitive actions are carried out (e.g., taking a picture) and requested data transformed and returned to the agents that requested it and who now enact the desired workflow (e.g., the photograph is passed to the agent creating a web site documenting the EVA). The dynamic decomposition and assembly each agent accomplishes avoids interfaces with n-factorial combinations—and correspondingly avoids subsystem developers requiring as many interactions with each other. Yet, in fact all of these combinations can be expressed in the task language and accomplished by the exploration system. Through the task-level abstraction and agent-based interoperability, the system's possible states and combinations of behaviors far exceed what designers have to explicitly plan and formalize.

In summary, in terms of software architecture, Brahms provides a language for creating agents and a communication infrastructure for agents to interact. Considerable flexibility in languages and methods is possible. For example, the API of the EVA Robotic Assistant (ERA [11]) exposes its C++ methods using CORBA objects, which the ERA CA, written in Java, translates to and from SpeechActs used by agents throughout the exploration system. A later generalization, called the *Collaborative Infrastructure* (CI) used in Exploration Technology and Development "autonomy for operations" projects for information exchange services and in OCAMS [9] for ISS file management, handles this translation by providing a toolkit of C++ and Java libraries that include SpeechActs for "agentifying" an external system [10].

Of special note are the communication agents that interact with subsystems having interfaces that people use directly. For example, ScienceOrganizer is a web-based display of EVA progress and data, organized according the objects and actions of the domain (e.g., plans, people, locations). Such subsystems often require particular communication methods for receiving data and/or commands. In particular, the ScienceOrganizer CA uses SOAP to interact with ScienceOrganizer; the Dialog CA uses OAA with the Dialog System; the Compendium CA uses SQL to communicate with Compendium's database (of plans and data), which Compendium displays in its own GUI. Each of these communication agents "agentifies" external systems, but their design and operation are focused on enabling people to communicate with the exploration system through different views in different modalities.

Thus we say at one level that the agents provide "services," emphasizing communications among workflow agents and subsystems. But more generally from the human-task perspective, the agents are providing people with direct control of robots and devices and immediate access to information, without concern for low-level commands and interfaces. That is, "agentifying" enables people to tell subsystems the goal they want accomplished or the information they seek in task-level terms (e.g., "Are the batteries charging?"). This design correspondingly reduces the training required for using the exploration system.

To recap, the MAA enables arbitrary object-oriented systems to become agents not just by exposing their methods, but by wrapping the software so communications

between subsystems occur using task-level messages (SpeechActs). The CA straddles the programming domain of the subsystem (its language, data, and functions) and the agent domain of the integrated workflow system (represented in terms of the objects and activities of the EVA system). Using a common SpeechAct language to integrate data and functionality follows the principles of "model-based" programming, which enables relating semantically different data across hardware and software systems [13]. Consequently, designing an agent to command a robot is handled in the same manner (at the same semantic level) as designing an agent to associate a photograph with a map location and sample.

III. Field Experiments and Data Analyzed

The field experiments for MAA were based on the methodology of *empirical requirements analysis*, in which prototype exploration systems were used for assisting crew members in simulated surface missions [3]. In particular, the project explored how existing components (robots, cameras, computers, biosensors, GPS devices, electric power systems, databases, email, heads-up display, etc.) could be made into an integrated exploration system that was easily reconfigured for different EVAs and settings.

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

- "Automating Capcom" Configurations for Desert-RATS and Mars Desert Research Station:
  - **DRATS02**, **MDRS03**, **MDRS04** (all using EVA Robotic Assistant)
  - **MDRS05**, **DRATS05** (Scout vehicle), **CDS05** [14] (K9 & Gromit robots)
  - **DRATS06** [8] (pressurized suits, JSC ExPOC, and GeoPhone Array, Fig. 1)
- "Power Agents": **MDRS06 [7]**
- Metabolic Advisor: **POGO07**
- iMAS Scientist's Field Assistant: **iMAS08**

Table I describes the systems, which are listed chronologically by field test; Table II summarizes configurations. "Platforms" are laptop computers running Brahms agents; "functions" are workflow capabilities (Table III); external systems are any devices or software with APIs communicating with agents (e.g., biosensors, cameras, email, power inverter, robots).

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

TABLE I. Description of Mobile Agents Field Experiment Configurations

Mobile Agents System Descriptions
DRATS02 Initial system build, single laptop platform, camera, biosensors, voice. Agent communications represented as Brahms Objects. Includes "proxy agents" for potentially queuing requests to agents on inaccessible platforms.
MDRS03 Addition of an EVA Robotic Assistant (ERA), 2nd astronaut laptop, and habitat computer (for HabCom crew member), i.e., four distributed platforms. Agent communication uses KaOS with CORBA as transport layer and directory service across network for multiple platforms.
MDRS04 Fully functional location and plan assistance. ERA follows astronauts in canyon, automated video tracking. Agent communication via KaOS/CORBA now uses FIPA SpeechAct envelope with Brahms "Communication Acts" as payloads; centralized directory service on mobile laptop (ATV) acceptable, but single point of failure.
MDRS05 2nd ERA relay controlled by human operator; temperature probe; dynamic reconfiguration of ERA roles during EVA. Personal agents for astronauts and robots decomposed to create service-oriented "assistants"; Plan Assistant enables agents to handle multiprocessing (simultaneous open requests) through task list.
DRATS05 Scout rover is configured to use ERA control system; heads-up display. Insufficient testing time provided in field to complete integration.
CDS05 Two weeks after DRATS05: Same software package as DRATS05 with bugs fixed. System incorporates Gromit and K9 robots as adapted ERA agents; one astronaut, no HUD; HabCom crew member interacts with EUROPA planner to control K9 via agent architecture.
PA06 "Power Agents"; no robots; all inside MDRS habitat; completely new functionality in monitoring electric power system, including generator, batteries, solar panels. Voice mail implemented during two-week shake down test; configuration retains all science data collection and EVA management capabilities. Introduces sound beeps to provide confirmation for certain routine commands.
iMAS06 Same software as PA06, but configured for one laptop operating in standalone (off-network) mode, for use by a field scientist.
DRATS06 Reconfiguration of DRATS/CDS05 to enable second commanding console off-site (JSC's Exploration Planning & Operations Center, ExPOC); automated control of geophone deployment from Houston. Voice commanding by crew in pressurized suits with special microphones. Demonstrated autonomous driving of Scout ("Go to waypoint" "Follow astronaut one").
POGO07 Based on iMAS06; integrates with Metabolic Algorithm (Excel VBA) connected to biosensors in pressurized suit during partial-gravity experiments. Language grammar rebuilt from scratch to be more compact, enabling much faster compilation.
iMAS08 Based on POGO07, with more automated science data logging; used in practical settings by geologists in Hi and NM and by divers with scuba gear (Belize).

TABLE II. MOBILE AGENTS SYSTEM CONFIGURATIONS

SYSTEM	FTE	# Platforms	# Functions	Robotic Systems (adapting ERA CA)					# External Systems
				ERA 1	ERA 2	SCOUT	K9	Gromit	
DRATS02	2	1	3	X					4
MDRS03	1.4	4	29	X					7
MDRS04	2.8	4	53	X					10
MDRS05	2.9	5	82	X	X				13
DRATS05	0.9	4	77			X			14
CDS05	0.9	4	80				X	X	11
PA06	1.1	5	45						10
iMAS06	.01	1	51						5
DRATS06	0.7	5	91			X			16
POGO07	0.3	1	63						7
iMAS08	0.05	1	50						5

TABLE III. CATEGORIZATION OF EVA WORKFLOW AUTOMATION CAPABILITIES

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

Capabilities are counted to group commands handled uniformly by a subsystem. For example, asking the Scout to power on/off its brakes, headlights, or motors counts as one capability. Here many different task-level commands map onto a single method in Scout's communication agent. However asking Scout to follow someone and asking it to halt are two capabilities because these task-level operations are handled by different CA methods, coordinating data and commands from different sources. One source of efficiency is that agents may retain methods relevant to different configurations. For example, the agents used in DRATS05 and CDS05 were identical except for adding CAs for the new robots and removing Scout and its CA; thus a voice command to Scout in CDS05 would be properly interpreted and result in the response that Scout is unavailable.

Overall, in the ten MAA system reconfigurations 134 workflow capabilities were developed for astronaut health monitoring, system health monitoring, location tracking, human-robot coordination, plan management, science data logging, voice command interface controls, and alert management. The Mobile Agents systems were developed to illustrate typical functionalities that may be useful to scientist-astronauts during EVAs in which real-time communication with mission support is not possible due to light-speed time delay [4]. Voice commanding capabilities were designed to assist creating documented EVA products while flexibly following an EVA plan, keeping on route and schedule, and remaining aware of logistic/safety constraints and limitations.

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

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

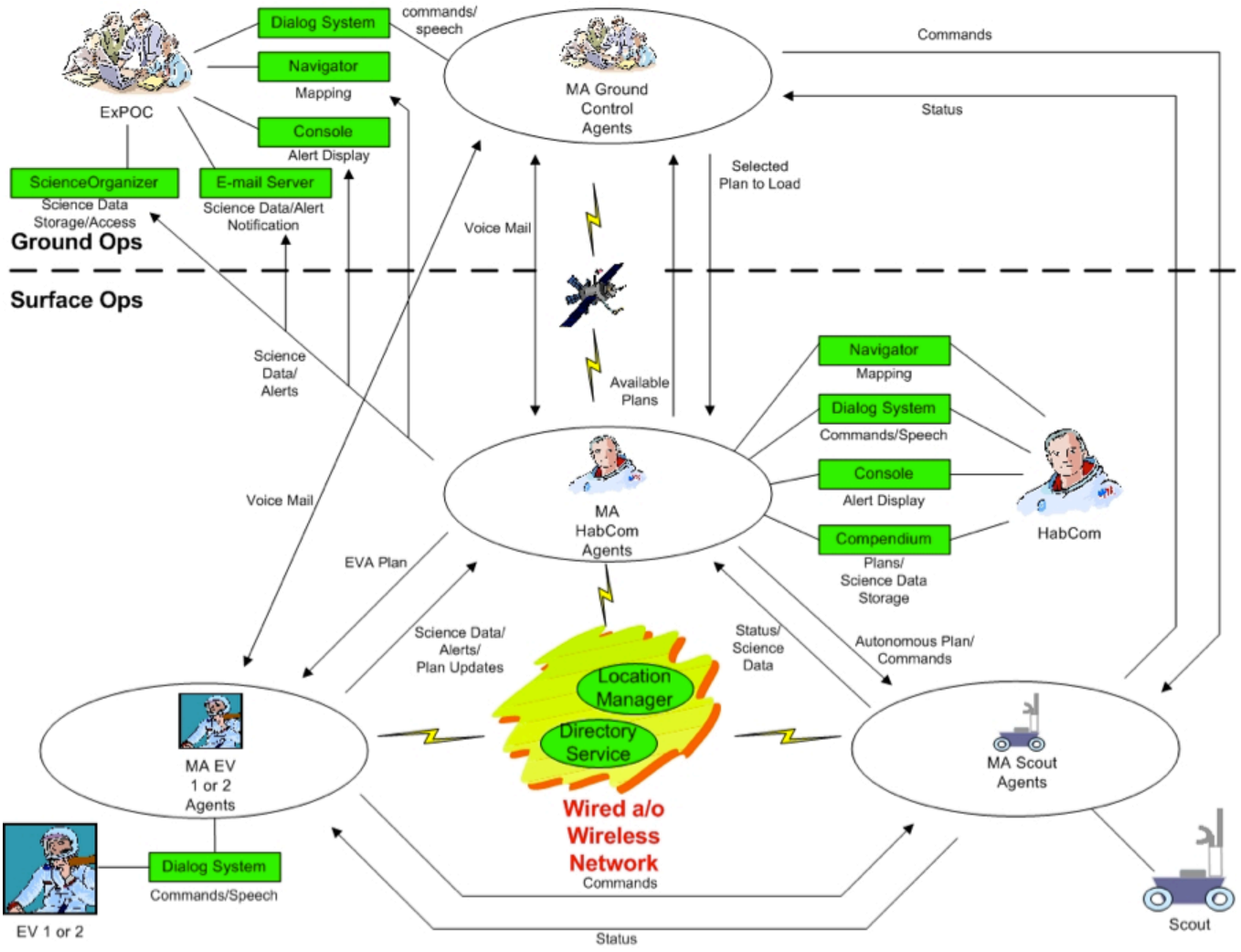

Figure 1. Desert-RATS 2006 EVA Exploration System Configuration (Meteor Crater, September 2006) [8]. Green rectangles are MAA subsystems; ovals are computer platforms containing a Brahms multiagent system: EVA astronauts (EV1, EV2), Scout, Habcom, and ExPOC. DRATS06 reconfigured MDRS05 to use Scout rover instead of ERAs. Agents control Scout's Geophone deployment system from the ExPOC agent platform in Houston. This configuration demonstrated how two kernel support systems, one local (HabCom) and the other remote (ExPOC), could be used to coordinate the flow of data, information, and goal directed commands coming from EVA astronauts, the surface habitat operator, and the remote ground support operator.

IV. Analysis Overview

For a closed architecture not designed for adding new capabilities, the amount of effort (in FTE cost and KSLOC) can increase rapidly when new capabilities are added. We hypothesized that an open architecture designed for flexibility would make reconfiguration and addition of components easier and less costly.

Our experience and analysis confirmed three primary hypotheses:

1. *An agent-oriented workflow system using composite APIs enables integrating new components (e.g., biomedical algorithms, robots, databases) without modifying them.*
2. *Incremental Cost/capability is linear or decreasing as new capabilities are added.*
3. *Incremental KSLOC/capability is linear or decreasing as new capabilities are added.*

Analysis showed that a *workflow service backbone* was established in the third year (MDRS05) and was reused for subsequent configurations, ranging from 85% carryover when entirely different robots and a planning system were incorporated (CDS05) to 100% carryover when shifting from an EVA system to a habitat power monitoring system (MDRS06). At the same time, in developing the workflow service backbone, nearly half of the APIs were unchanged. Architecture changes focused on reconfiguring workflow services to respond to interface requirements, rather than re-integration with subsystems. That is, the data show that enhancement of workflow capabilities is often possible without modifying existing systems and their APIs, given an open architecture in which integration occurs through task-level communications.

Furthermore, in the aggregate *80% of added KSLOC for system reconfigurations was for task-level API translators ("communication agents"; CAs) to integrate new components* (Fig. 2). But a relatively small amount was added for each configuration after the architecture was mature. On average Workflow KSLOC was increased by 14% and CA KSLOC by 13% for each reconfiguration.

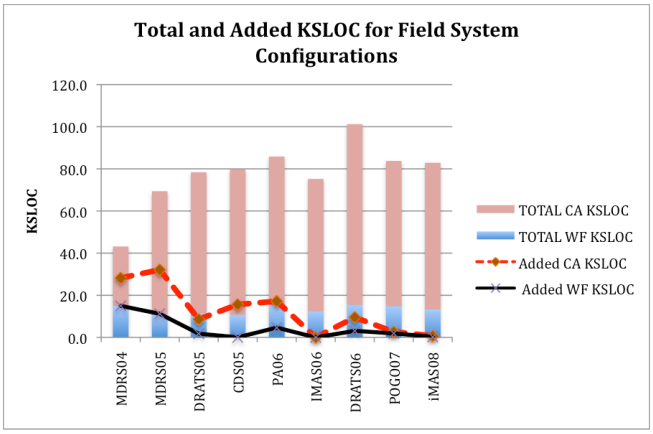

Figure 2. Total KSLOC (thousands of lines of code) for each system configuration (columns, broken into Workflow Backbone and Communication Agent parts) and new KSLOC (for new or modified agents; shown as lines). Code for Communication Agents dominates.

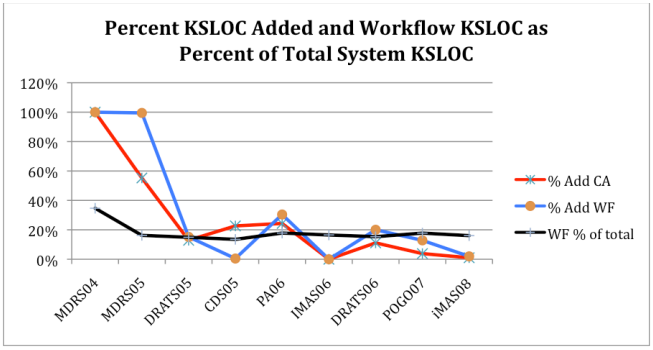

Figure 3. Percentage of KSLOC added to Communication Agents and Workflow Agents for each configuration; Percentage of KSLOC Workflow Agents relative to the system total KSLOC .

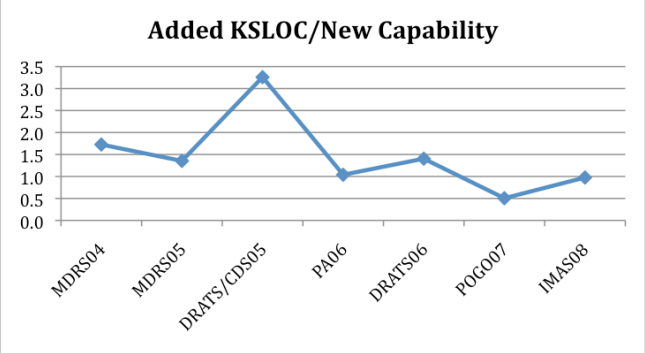

Figure 4. Average KSLOC added for each new capability.

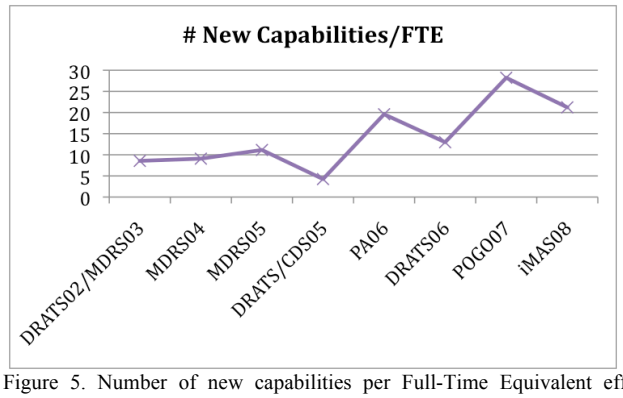

Figure 5. Number of new capabilities per Full-Time Equivalent effort (annualized over development period).

Depending on the component, API translators for complex systems such as rovers might require 20 KSLOC (1 FTE or more), while simple systems such as cameras might require 2 KSLOC (0.1 FTE).

The ratio of total workflow KSLOC to the total system KSLOC (Fig. 3) remained surprisingly constant at 16%, which demonstrates that *the amount of code required is linear with the number of capabilities, and additions require only incremental changes to affected workflow functions.*

The series of exploration system experiments demonstrated efficiency in designing, developing and testing new systems. The average (and median) DDT&E elapsed time per configuration was 172 days with five programmers on average (varying from nine to one). Programming effort varied from 3.4 FTE to less than 0.1 FTE, which underscores the minimal re-engineering required.

The three key productivity findings are: *code added for each new workflow capability trends downwards from 1.5 to less than 1 KSLOC* (Fig. 4); *new capabilities/FTE trends upwards from 10 to 20,* and *KSLOC/FTE trends upwards from 15 to 20*. These data show that size of the modifications and effort required are generally predictable and constant, with addition of task-level APIs for new automated hardware and software systems requiring more code and time. These data suggest that the overall exploration system architecture is stable and new capabilities neither interfere with existing capabilities nor require increasing complexity of interactions. Therefore, in using an open architecture with APIs providing task-level services, we can treat capabilities abstractly and make predictions at design time of the amount of code and effort required to build or modify an exploration system configuration. Specifically, the *analysis predicts that each workflow capability added by an experienced team will require 1.5 KSLOC or less and correspondingly 0.07 FTE or less*.

Viewing the field configurations in the aggregate, 134 capabilities were developed with 13 FTE in total DDT&E time of about 4 years. The overall average of 10 new capabilities per FTE closely fits the development efficiency of creating the mature architecture (2002-2005), when 64% of the total system capabilities were developed with 70% of the total FTE. Subsequent systems introduced relatively fewer new capabilities with increased productivity (Fig. 5). Effort for DRATS/CDS05 and DRATS06 reflects

significantly more complex commanding for four different robotic systems with a variety of peripheral subsystems.

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

V. Lessons Learned from Field Configurations

The development of the workflow agents involved a substantial learning process through trial and error in the field experiments, particularly in 2002-04. (It is humorous to recall how an old hand at DRATS02, unfamiliar with experimental prototyping, referred to the initial configuration as "not ready for prime time.") The methodology of empirical requirements analysis employed throughout involved using the prototype system for authentic scientific exploration (e.g., use by scientists in a new terrain that addressed their specialized expertise and interests). During these experiments we learned how people and systems interacted in practical work settings and realized the value of additional workflow automation. For instance, during iMAS08 it became obvious that explorers wanted to ask "Guide me to <location>"—receiving alerts thereafter to correct course—rather than repeatedly asking for the distance and bearing to the desired location. We recognized requirements and invented methods to make the wireless, distributed agent communications more robust and to handle loss of communications gracefully. We also made many improvements to the interaction between people and their personal agents and robotic assistants to make commanding more reliable and simpler, for example, substituting a beep confirmation for non-critical requests and not having an agent speak when you are speaking to someone else. We discovered some complications that will require future research and experimentation, such as how to avoid having an agent speak to you when you are listening to someone else.

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

- Use of a *distributed directory service* [10] to deal with unreliable wireless communication and to allow subsystems to be disabled and restored in a running system configuration.
- Use of a *workflow backbone*—consisting of individual agents for people and robots, complemented by shared functional workflow agents for navigation, planning, database management, communications (e.g., via email, GUI, voice loop, voicemail)—distinct from the component CAs that interface with component APIs provides great flexibility for reconfiguring components during an expedition for different EVA requirements (e.g., the transformation of MDRS05 to DRATS05 and CDS05; MDRS05 to PA06; PA06 to iMAS06, POGO07, and iMAS08).
- Use of a *web-based semantic database* to consolidate data from different sources for a common repository [1].
- Experimentation with a variety of *reconfigurable, mixed communication methods for controlling and getting data from arbitrary systems*: Voice (including shared loudspeaker), menu-based GUI, audible tones, heads-up display.
- Allowing *alternative data-exchange services* for CA ⇔ component API communications (e.g., SOAP, OAA).
- Providing *methods for creating and relating different types of data for different purposes* (e.g., photographs, GPS coordinates, biosensor telemetry, voice recordings, terrain maps).

VI. Conclusions and Recommendations

An *open software architecture* enables adding, upgrading and swapping components. From the perspective of human space exploration, the objective is to support upgrading and incorporating new elements (e.g., vehicles, robots, instruments), allowing for new forms of automation, migration and changing distribution of mission support functions, plus new and more complex simultaneous distributed operations [15]. This objective is often referred to in the context of "growth potential" and "incremental buildup," emphasizing technology upgrades. After analyzing a series of ten systematically developed surface systems that integrated a variety of hardware and software, we found evidence that *incremental buildup of an exploration system for long-duration capabilities is facilitated by an open architecture with appropriate-level APIs, specifically designed to facilitate integration of new components*, and this minimizes costs by reducing changes to the existing system.

Specifically, the study shows the advantages of composite APIs that map or translate conventional component APIs (which provide access to the functional methods and data objects of components) to the language of the task. Messages expressed as *SpeechActs*—in terms of an EVA plan; names and relationships of people, places, and robots; and features of work products—mediate communications between components and people through arbitrary media, including displays and voice commanding. Using this kind of secondary API (a "communication agent") as a wrapper effectively enables each component, including off-the-shelf hardware and software, to provide task-oriented services to the overall exploration system.

Acknowledgment

Funding for the Mobile Agents Project has been provided by NASA's Intelligent Systems, Moon and Mars Analogue Mission Activities (MMAMA), and Exploration Technology and Research Programs (ETDP). This study was supported by ETDP/Lunar Surface Systems Project. Many people in

three NASA centers, co-authors in the references cited here, collaborated to develop Mobile Agent systems over the past decade. Special thanks to Rick Alena, John Dowding, and the JSC Scout/ERA team (especially Rob Hirsh, Jeff Graham, and Kim Shillcutt Tyree), and to our geologist collaborators, Brent Garry and Abby Semple, who experimented with the system doing field science at MDRS and other analog sites.

REFERENCES

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

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

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

[4] W. J. Clancey, Automating Capcom: Pragmatic Operations and Technology Research for Human Exploration of Mars. In *Martian Expedition Planning,* vol. 107, C. Cockell Ed. AAS Science and Technology Series, 2004, pp. 411-430.

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

[6] W. J. Clancey, M. Sierhuis, R. Alena, D. Berrios, J. Dowding, J. S. Graham, K. S. Tyree, R. L. Hirsh, W. B. Garry, A. Semple, S. J. Buckingham Shum, N. Shadbolt, and S. Rupert, "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.

[7] W. J. Clancey, M. Sierhuis, R. Alena, J. Dowding, M. Scott, and R. van Hoof, "Power system agents: The Mobile Agents 2006 field test at MDRS," *Mars Society Annual Convention*, 2006. Available: http://homepage.mac.com/wjclancey/%7EWJClancey/ClanceyMarsSoc2006.pdf
Video report: http://www.youtube.com/watch?v=zez9JINQm44

[8] W. J. Clancey, M. Sierhuis, J. Dowding, D. Berrios, M. Scott, R. van Hoof, F. Delgado, S. Tourney, and J. Kosmo, "Mobile Agents integrate astronauts, rover, and mission support In Desert-RATS mission simulation," *Mars Society Annual Convention*. Los Angeles, 2007. Video report: http://www.youtube.com/watch?v=fTTrFDR9I1I

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

[10] W. J. Clancey, M. Sierhuis, R. Nado, and R. van Hoof, "Collaborative infrastructure conceptual overview," NASA Ames Research Center, Intelligent Systems Division, TM10-0001, unpublished, 2010.

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

[12] A. W. Johnson, D. J. Newman, J. M. Waldie, and J. A. Hoffman, "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.

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

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

[15] S. Rader, "Constellation's command, control, communications, and information architecture (C3I) overview," Software & Avionics Integration Office (SAVIO) PowerPoint presentation, 11 Dec 2008.

[16] M. Sierhuis, *Modeling and Simulating Work Practice.* Ph.D. thesis, Social Science and Informatics (SWI), University of Amsterdam, The Netherlands, 2001.

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