NASA's OCA Mirroring System
An application of multiagent systems in Mission Control

Maarten Sierhuis
Carnegie Mellon Silicon Valley
NASA Ames Research Center
Moffett Field, CA 94035
(+1) 650-604-4917
Man-Machine Interaction Group
Delft University of Technology
Maarten.Sierhuis@nasa.gov

William J. Clancey
NASA Ames Research Center
Moffett Field, CA 94035
(+1) 650-604-2526
IHMC
Pensacola, FL
William.J.Clancey@nasa.gov

Ron J.J. van Hoof
Chin H. Seah
Michael S. Scott
Robert A. Nado
Susan F. Blumenberg
Michael G. Shafto
NASA Ames Research Center
Moffett Field, CA 94035

Brian L. Anderson
Anthony C. Bruins
Chris B. Buckley
Thomas E. Diegelman
Timothy A. Hall
Deborah Hood
Fisher F. Reynolds
Jason R. Toschlog
Tyson Tucker
NASA Johnson Space Center
Houston, TX

ABSTRACT
Orbital Communications Adaptor (OCA) Flight Controllers, in NASA's International Space Station Mission Control Center, use different computer systems to uplink, downlink, mirror, archive, and deliver files to and from the International Space Station (ISS) in real time. The OCA Mirroring System (OCAMS) is a multiagent software system (MAS) that is operational in NASA's Mission Control Center. This paper presents OCAMS and its workings in an operational setting where flight controllers rely on the system 24x7. We also discuss the return on investment, based on a simulation baseline, six months of 24x7 operations at NASA Johnson Space Center in Houston, Texas, and a projection of future capabilities. The paper ends with a discussion of the value of MAS and future planned functionality and capabilities.

Categories and Subject Descriptors
I.2.5 [**Programming Languages and Software**]: Multiagent systems, tools and techniques, J.2 [**PHYSICAL SCIENCES AND ENGINEERING**]: Aerospace, Mission Control, I.6.2 [**Simulation Languages**]: Multiagent simulation, agent-based simulation, agent-directed simulation, I.6.3 [**Applications**]: Multiagent systems, tools and techniques, Agent-based simulation

General Terms
Design, Human Factors, Languages.

Keywords
Multiagent systems, MAS, agent-based simulation, agent-oriented languages, ISS, mission control, human centered design, human factors, applications

1. INTRODUCTION
The Mission Control Center (MCC) in Houston, Texas, of the National Aeronautics and Space Administration (NASA) is one of the most complex and best-known organizations for command and control of human space flight. This paper describes the development and deployment of the *first* multi-agent system (MAS) in the International Space Station (ISS) MCC.

NASA's ISS MCC is charged with managing, commanding and controlling every aspect of the ISS, from docking with the Space Shuttle and Soyuz spacecrafts to uplinking and downlinking all information to and from the ISS. The ISS MCC, as most MCCs, is divided into a "front room"—the room where the main flight controllers are located—and a "back room"—where the support flight controllers are located. Each flight controller in the front room has several support flight controllers in the back room. Together the people in the front- and back room are organized in flight control groups for the different subsystems for the ISS, overall named the Flight Control Team (FCT). The Orbital Communications Adapter (OCA) officer is a back room flight controller and a member of the Operations Planner (OPSPLAN) group. The ISS OCA Officer is responsible for manually uplinking and downlinking all files to and from the ISS. These files include schedules, procedures, commands, email, photographs, health data, newspapers, etc.

Over a six-month period, computer scientists and ethnographers in the Work Systems Design & Evaluation Group of the Intelligent Systems Division at NASA Ames studied and simulated OCA work practices in collaboration with the OCA team to identify possible process improvements. Using statistics generated from an agent-based simulation model, the team, over the next three-month period, designed and simulated an agent-based workflow system that automates the process of creating a ground-based replica of the ISS file system (the Mirror LAN). Simulation statistics *predicted* a reduction in mirroring time from 6.7% to 0.4% of the OCA Officer's shift—a more than 94% reduction. Using the Simulation-to-Implementation engineering method, Brahms *simulated agents* were then converted, in a one-month period, into a distributed Brahms MAS run-time tool called OCAMS [1]. Using the Brahms virtual machine (BVM), these agents now manage the workflow on multiple computers and servers using secure communications provided by the Brahms collaborative infrastructure (CI). The tool also automatically writes large parts of the OCA Handover Log.

In this paper, we present the OCA Mirroring System (OCAMS) as it was deployed on July 8, 2008, to support the OCA Officer 24x7 in the mirroring and logging activities of their uplinking and downlinking process. The paper is organized as follows; We start with a review of some relevant work in Section 2, followed by a technical description of the OCAMS system in section 3; Section

Cite as: NASA's OCA Mirroring System -- An Application Of Multiagent Systems in Mission Control, M. Sierhuis, W. J. Clancey et al., *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 85 – 92

4 discusses our software engineering approach for MAS we dubbed "from simulation to implementation;" In section 5 we discuss the return on investment of the OCAMS system, and we end in section 6 with a discussion of the value of multiagent architectures and future functionality of the OCAMS system.

2. RELEVANT WORK

The agent-oriented language (AOL) used to develop both the OCA Officer work practice simulations and the run-time OCAMS MAS is Brahms [2]. Brahms is a multi-agent modeling language for simulating human work practice that emerges from work processes in organizations [3]. The same Brahms language can be used to implement and execute distributed multi-agent systems, based on models of work practice that were first simulated. Brahms demonstrates how a multi-agent belief-desire-intention language, symbolic cognitive modeling, traditional business process modeling, activity- and situated cognition theories are brought together in a coherent approach for analysis and design of organizations and human-centered systems. Brahms was first developed in 1992 as a multiagent simulation language. Brahms is a full-fledged compiled AOL, as opposed to current operational agent languages that are developed as a set of Java libraries (e.g. Jade).

The OCAMS system can be compared with workflow management systems evolved from business process management. Traditionally, Petri-Nets are used to make workflow systems able to deal with concurrency [4, 5]. However, modeling complex human decision-making is often easier represented in a belief- and rule-based format developed in DAI and Expert Systems. More recently, BDI agents are being combined with workflow and service-oriented architectures [6]. As we have found with OCAMS as well, BDI agents provide a flexible way of configuring services in a distributed work system in which both people and agents work together to perform the work [7].

Agent-based modeling and simulation (ABMS) has become a wide-ranging field in modeling and simulation [8, 9]. Our approach is founded in ABMS of human work systems and practices [10], business anthropology [11], ethnography [12], and participatory design [13].

3. THE OCAMS MULTI AGENT SYSTEM

The OCA Officer is responsible for all ISS uplink and downlink activity on the ISS Operations LAN (OPS LAN). As part of the old OCA work process, the OCA Officer spent much of his or her shift archiving selected files and mirroring files to the ground-based Mirror LAN, which replicates the ISS onboard OPS LAN directory structure.

Many uplink/downlink activities have to be duplicated (mirrored) on the Mirror LAN, using a laborious manual process of logging, searching, transferring, and verifying. The information necessary to determine the actions required can be categorized and extracted from the uplink/downlink log maintained by the file transfer (KFX) application. Automating these mirroring activities gives the OCA Officer more time to work on other important tasks, as well as reducing file-mirroring errors. This is now the work of the OCAMS MAS.

Architecturally, the OCAMS system is divided into three separate distributed agent systems, each of which are running in a separate Brahms Hosting Environment (BHE) (see Figure 1). These BHEs can run on any desired computer and network configuration, making the architecture easily adaptable to the computer architecture and network safety concerns of the ISS MCC (see section 6.2).

The communications infrastructure between OCAMS components is established using the NASA Collaborative Infrastructure (CI). The CI is an infrastructure that provides an application programming interface and a set of services that allows components to interact with one another using structured messages (*transport service*), allows components to find one another (*directory service*), allows components to share data with one another using a publish/subscribe service (*data distribution service*), and allows components to be managed using a common interface (*management service*). The components that make use of the CI are called *agents* or *actors*. The CI, in addition, provides process management tools to manage the startup, monitoring, and shutdown of *actor hosting environments* and their actors.

3.1 OCAMS Agents

The OCAMS Release 1 MAS consists of nine individual software agents (see Figure 2). Three agents are rule-based BDI agents written in the Brahms language, while six are Java-based agents written using the Brahms Java application interface. These Java agents are referred to as a "communication agent," because their purpose is to communicate with external systems or files outside of the OCAMS system.

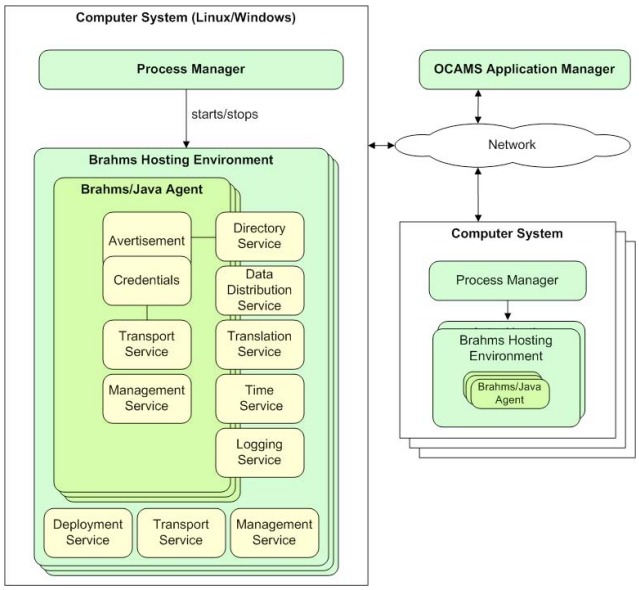

Figure 1. OCAMS Architecture

The agents are currently divided over three agent subsystems (BHEs), but could easily be loaded in a different number of BHEs. The *OCA BHE* consists of the central OCA Personal Agent that coordinates all work with the other agents in the system, and four human interface agents, one GUI agent and three Microsoft Office agents. The *Mirroring BHE* consists of the agents that interface with the KFX file uplink and downlink software currently used by the flight controllers and decides which file to mirror on the Mirror LAN. The *Monitoring BHE* is the workhorse of OCAMS. It has agents that monitor FTP and copy files to the Mirror LAN, and monitor for errors that might occur in the FTP and copying between file systems on the network, as well as problems with monitoring the specific file processing that needs

to occur on the Mirror LAN depending on the mirrored file type. Brahms agents (the solid circles in Figure 2) are used there where belief- and rule-based decision-making is needed. From our experience in developing BDI-based MAS, we developed some basic rules-of-thumb for deciding whether an agent should be a BDI agent (i.e. written in the Brahms language), or an imperative agent (i.e. written in the Java language). We will come back to this point in the Discussion section.

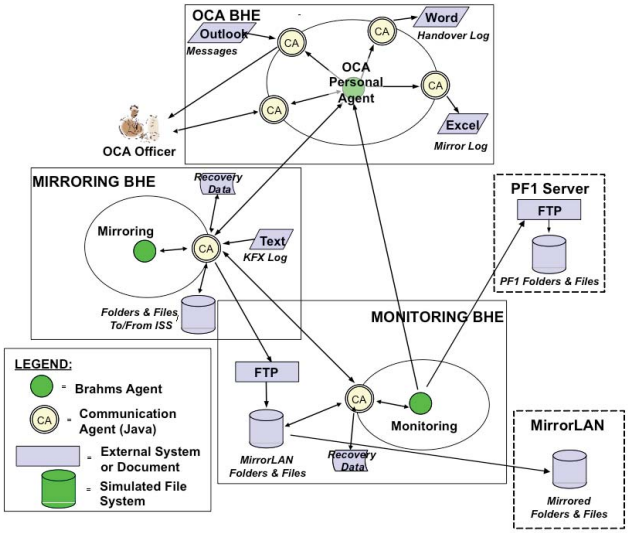

Figure 2. OCAMS Agent System Architecture

3.2 Agent Communication

3.2.1 Communicative acts

Communication between agents is performed using the Brahms/Java communication library [2]. This library allows local and distributed Brahms and Java agents to communicate with each other using an implementation of the FIPA agent communication protocol. The agent communication protocol is based on the FIPA ACL Message Structure Specification[1] and FIPA Communicative Act Library Specification[2]. An OCAMS communicative act (ComAct) is an object containing the envelope and payload attributes of a FIPA compliant ACL message. Both the envelope and payload attributes are map-types and contain the key-value pairs for each ACL message. The envelope attribute is used to contain address-related information for a ComAct, while the payload attribute is used to contain the content of a ComAct (see Table 1 for an example of ComActs to/from two agents in Figure 2).

Table 1. Example ComActs for OCAMS agents

Conv ID	*Sender*	*Receiver*	*FIPA Performative*	*Action*	*Content*
<1>	KFX Log Com Agent	Mirroring Agent	REQUEST	processNewBatch	Array of Java KFX File Objects
<1>	Mirroring Agent	KFX Log Com Agent	AGREE	processNewBatch	
<1>	Mirroring Agent	KFX Log Com Agent	REFUSE	processNewBatch	No Files & Explanation Message
<1>	Mirroring Agent	KFX Log Com Agent	FAILURE	processNewBatch	Explanation Message with array index of

3.2.2 Network communication and security protocols

OCAMS uses multiple network protocols to have agents communicate with one another and with systems external to OCAMS[3]. As needed for Homeland Security standards, the OCAMS agent and network communication protocols adhere to Federal Information Processing Standards Publications (FIPS) 140-2[4].

The CI transport service provides a pluggable transport allowing the transport to be expanded with different types of transport endpoints as needed by specific implementations, without affecting the components and their implementations (see Figure 3). OCAMS securely transmits all application-level data between the OCAMS Agent Systems (BHEs) over TCP/IP using the Secure Sockets Layer[5] (SSL). Other OCAMS subsystem components (Application Manager, Process Managers, Distributed Directory Service), that need to publish and subscribe to process status data, use Universal Data Packets[6] (UDP) over IP-multicasting. The application-level data communication includes *distributed agents* sending ComActs to one another that contain ACL messages (see Figure 3)[7].

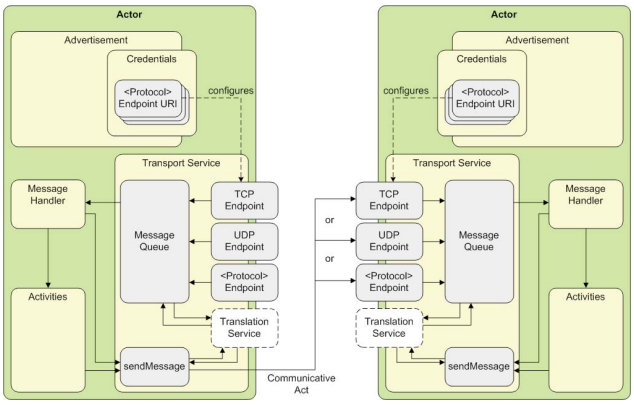

Figure 3. CI Transport Service

[1] http://www.fipa.org/specs/fipa00061/SC00061G.html

[2] http://www.fipa.org/specs/fipa00037/SC00037J.html.

[3] Because of security reasons, we cannot provide a description of the actual network configuration in NASA's MCC. All descriptions are generic.

[4] http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

[5] http://en.wikipedia.org/wiki/Transport_Layer_Security

[6] http://en.wikipedia.org/wiki/Universal_Data_Packet

[7] Brahms agents hosted within the same Brahms virtual machine use an optimized communication mechanism that bypasses the CI transport service.

4. FROM SIMULATION TO IMPLEMENTATION

This section describes the findings from the first three project phases: Current- and future simulation and system implementation. For a more extensive description of the "from simulation to implementation" methodology see [1].

4.1 Current OCA Simulation

The objective of the first phase (simulation of the current work system) was to develop a detailed agent-based OCA work practice simulation of the *mirroring activity* of the OCA Officer. This focus was chosen based on the work practice observations of the OCA Officers and observed issues involved with the current way of doing the mirroring of files from the ISS to the Mirror LAN machine (a Space Station Computer—SSC) located in the Flight Activities Multi Purpose Support Room (MPSR), i.e. the ISS backroom area for the OCA Officer.

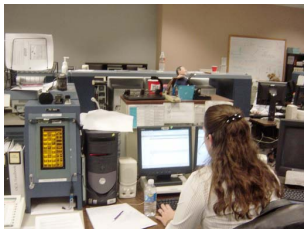

Figure 4. OCA Area in ISS MPSR

As part of the current OCA work process, the OCA Officer spends time after each file uplink/downlink activity mirroring the same files to the Mirror LAN. This cumbersome activity is both error prone, because of the manual "sneaker net" being used, and time consuming, because many uplink/downlink activities have to be duplicated on the Mirror LAN using a laborious manual process. OCA Officer observations, and modeling and simulation have given us insight into the intricate details of the work practice of the mirroring activity.

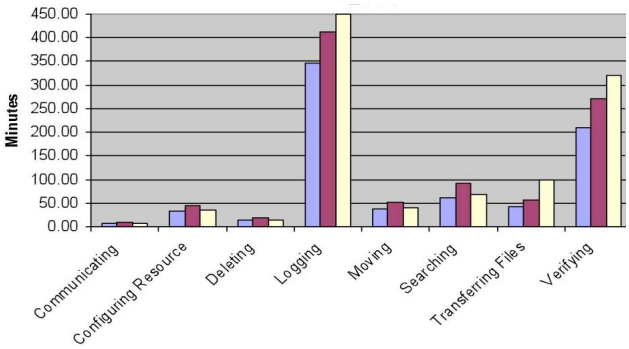

Figure 5. Simulation results for OCA Officer's Mirroring subactivities (Orbit 1-3)

Figure 5 shows the results of the Current OCA Mirroring Activity Brahms agent-based simulation for the three shifts (Orbits 1-3), based on actual ISS file mirroring data for November 2006. Striking in these results is the fact that the highest time-cost for the OCA Officer are the (handover) logging and mirroring verification (verifying) subactivities. Figure 6 shows a part of the simulated OCA Officer activities timeline for Orbit 3 (3rd shift). On the right top (1) you can see agent OCA Orbit 3 executing the Mirroring activity with its subactivities. Below that (2), the timeline also shows the actual file object being moved (due to agent OCA Orbit 3's copying subactivities) from folder "V:" to a folder on the Mirror LAN, via a USB "stick" (the colored bars above the black timeline shows location movements of the OCA Officer agent (1) and file object (2)).

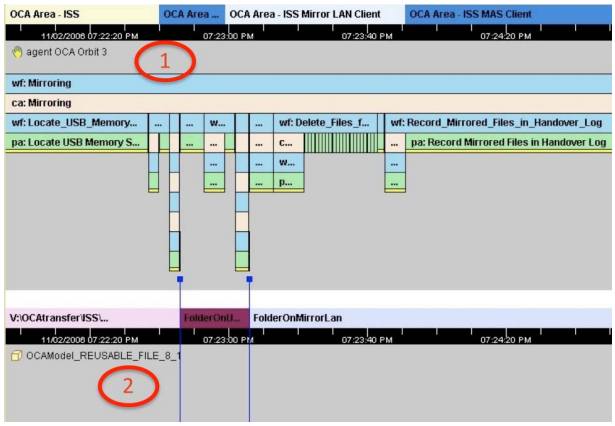

Figure 6. Brahms Agent-Viewer Timeline of an OCA Officer's Mirroring Activity during the 3rd shift (Orbit 3).

In the logging activity the OCA Officer writes the particulars of files uplinked/downlinked and mirrored in a Word document (not shown in Figure 6). This document provides the log of what happened during the shift, and is used for the next OCA shift to get a grasp of what happened in the previous shift. The verifying activity is where the OCA Officer verifies the processing of specific types of files (e.g. the ISS astronaut activity timeline updates files) by the Mirror LAN. As part of the copying processes, the Mirror LAN server executes and processes the "dropped" files. The OCA Officers have to verify that these batch processes execute correctly and the files are "absorbed" without errors. In case of errors, the OCA Officer needs to deliver the error files to the responsible flight controller in the MCC. Of course, all this needs to be logged in the handover log.

Our OCAMS design focused on automating the complete mirroring activity in such a way that the subactivities do not have to be done by the OCA Officer, but instead are done by the Mirroring-, OCA Personal- and Monitoring software agents from Figure 2, mimicking the work of the OCA Officer.

4.2 Future OCAMS Simulation

After the Current OCA simulation, an OCAMS requirements and design phase was started. The OCAMS MAS design from Figure 2 was implemented in a Brahms agent-based simulation of the future work system that includes both the OCA Officer and the OCAMS system.

This future simulation model included a simplified work practice model for the OCA Officer agent. The OCAMS agents perform all the mirroring and logging activities. The OCA Officer agent only has to select uplinked and downlinked files in the GUI, and review the OCAMS activity when complete. The GUI agent displays the mirroring status back to the OCA Officer who verifies the mirroring by OCAMS. All OCAMS activity is also reported in the handover log, which the OCA Officer uses to verify mirroring was completed correctly.

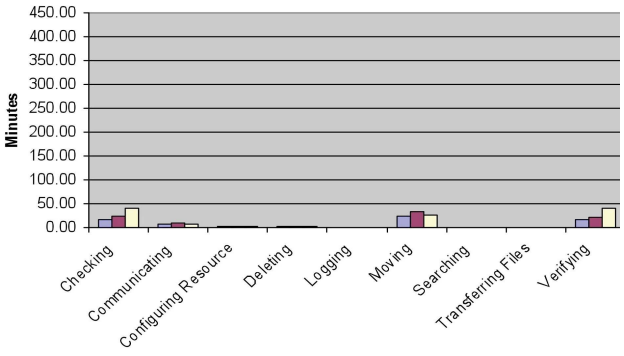

Figure 7. Simulation results for OCA Officer's Mirroring subactivities with OCAMS (Orbit 1-3)

Figure 7 shows the result of the Future OCA Mirroring agent-based simulation using OCAMS. The simulation is based on the same November 2006 uplink/downlink data as the Current simulation. Comparing Figure 5 with Figure 7, the amount of time spent by the OCA Officer on all mirroring subactivities, except Moving, have gone down dramatically (see Figure 8 for a comparison of how much time has been saved by OCAMS). There is a new activity that has been added, referred to as Checking. In this activity the OCA Officer is checking OCAMS for errors in mirroring. Figure 8 shows that the mirroring activity has dropped more than ten-fold[8].

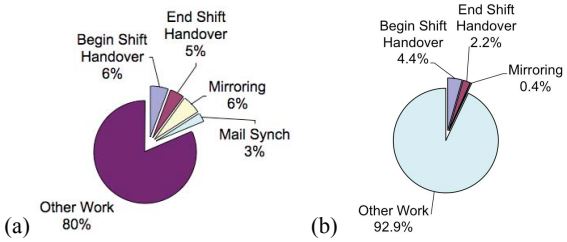

Figure 8. Percent OCA Officer Activities for (a) current simulation and (b) future simulation

4.3 OCAMS Implementation

The third phase was the implementation of the run-time MAS, based on the Brahms agents developed in the Future Simulation model. We accomplished this step in a one-month development effort with a team of three full-time developers. The development team consisted of one Brahms modeler for implementing the Brahms agents, and two Java developers implementing the Java agents.

The OCAMS Future Simulation in Brahms included all the major Brahms and Java agents that became part of the OCAMS MAS; The Mirroring, Monitoring, GUI CA, KFX Log Parser CA agents (see Figure 2). The work left to do during the transition from simulation to run-time implementation was to (a) make the OCAMS application robust in case of system failures (automatic restart of the BHE's and state recovery, etc), (b) add the MS Word CA (for generating handover logs) and MS Excel CA (for monthly generating statistics), and (c) make changes to the Brahms/CI environment allowing agents to communicate securely (being FIPS 140-2 compliant).

4.3.1 Robustness

OCAMS is robust in the sense that the OCAMS Application Manager (AM) detects a system failure of one of the three BHE's (see Figure 1). The BHE notifies the AM when an agent unexpectedly terminates, a deadlock is detected, or out of memory exceptions are raised. As part of the notification the BHE provides the AM with the detected error condition and basic state information such as the hostname, IP address of the host, current Java VM memory usage, and CPU load.

The Application Manager subscribes with the data distribution service for process status. Whenever the status of a process changes (i.e. a process shuts down) the AM receives a data object with the process identifier and status of a process. The AM uses this information to update its status display and, if a process shuts down unexpectedly generates an alert to the OCA Officer requesting the OCA officer to initiate recovery of the OCAMS system.

4.3.2 OCAMS Agent Process Flow

The KFX Log File CA agent parses all files uplinked and downlinked from the ISS and sends these parsed files to the Mirroring agent (see Figure 2). The Mirroring agent then decides if a file needs to be mirrored or not. The OCA Officer, through the OCAMS GUI (see Figure 9) selects which files need to be further processed by OCAMS as a session.

The OCA Personal Agent (see Figure 2) further processes files selected as part of a session. The OCA Officer has the choice to keep the mirror-or-not decision by the Mirroring agent, or overwrite this decision by selecting a file and changing the decision to be mirror, don't mirror, or mirror manually. This way the OCA Officer always has the last say in what OCAMS should do with the file. All files are processed through OCAMS so that they are logged in the handover log, which provides a permanent record of files uplinked, downlinked, and mirrored.

In case of errors during the mirroring of files (FTP-ing, copying, and monitoring), the Monitoring agent sends error messages to the OCA Personal Agent, which in turn sends these error messages to both the GUI CA and the HandoverLog CA. The OCA Officer can then handle the error appropriately; send error file to appropriate flight controllers, change status of file and reprocess, or manually mirror the file.

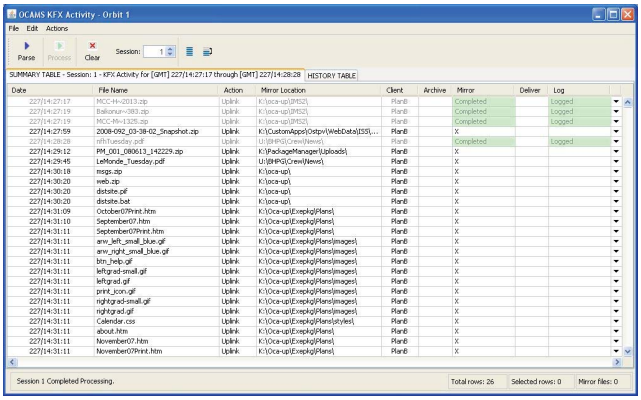

Figure 9. OCAMS GUI

[8] The Mail Synch activity is not in Figure 8(b), because the Future simulation excludes simulating the mail synch. The time spent on this is included in the Other Work category.

One of the requirements of the OCA team is that an OCA Officer can always return to manually performing the mirroring process. This way training of new OCA Officers can always be done on console, since new trainees (OJT's) need to know how to perform the job without OCAMS.

4.3.3 Getting OCAMS Operational

Deployment of OCAMS in the ISS MCC was not a trivial process. Security and safety of both astronauts and spacecraft is priority number one. To accomplish this, there are many levels of approvals needed for a new application to be accepted into operations. Implementing a MAS, almost by definition, means running a distributed application over one or more networks. OCAMS relies on interfacing with three separate communication networks inside the ISS MCC. These three networks had not been connected before, because of their different security levels. This was the main reason for a USB "sneaker net" to exist in the OCA Officer's work practice. For OCAMS to succeed in automating the mirroring, we had to get approval to interface to these three networks. It took six months of presenting the OCAMS architecture to several MCC engineering boards to get this approval. In section 6.2 we will briefly return to why we were able to convince so many that our solution would work; a big reason is the deployment flexibility MAS architectures provide.

5. Analysis of OCAMS Operations

As of this writing, OCAMS has been in operations for more than five months (June 26 – Dec 1, 2008). All 38 OCA certified flight controllers have been trained on using OCAMS. Around 10 OCA Officers are regularly on console and use OCAMS as part of their regular work. The other 28 are on console only when needed, which might be once or twice a month. All OCA Officers on console use OCAMS for mirroring their files.

5.1 OCAMS effectiveness

During the operational period, OCAMS has processed a total of 128,311 files. Of this total, OCAMS mirrored 19,711 files or ~15% (see Figure 10). Interesting to see is the differences, per month, in the number of files uplinked and downlinked from the ISS. These differences have to do with the crew that is onboard and the work the crew is doing—there have been two crews onboard the ISS since OCAMS became operational (Oct. 14, 2008 is when the Expedition 18 crew docked with the ISS).

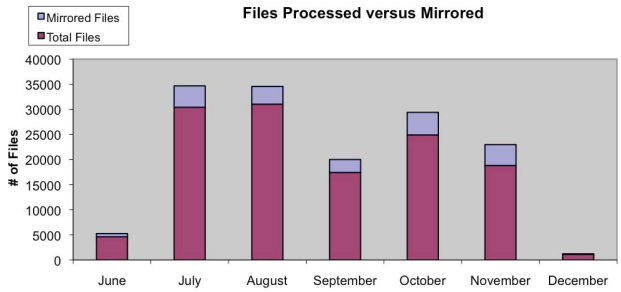

Figure 10. OCAMS files processed from June 26 - Dec 1, 2008

How well did OCAMS do in the first five months of operations? Subjectively—we asked the OCA Officers—the answer is that OCAMS is liked very much. OCAMS is automating the task that is perceived by the OCA Officers as very cumbersome and annoying. However, we want to have some objective measure of how well OCAMS performs. To do this OCAMS generates statistics about its file processing. In the period from June 16 – Dec 1, 2008, about 14% of the total files mirrored by OCAMS had an issue, causing the OCA Officer to mirror the file manually or deal with the failure in OCAMS (see Figure 11). Thus, we infer that the OCA Officer still had to be involved in mirroring ~14% more files than necessary, which makes OCAMS ~86% effective in the mirroring of files.

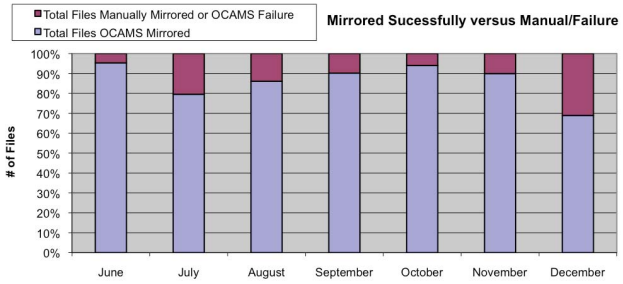

Figure 11. OCAMS mirrored files from June 26 - Dec 1, 2008

In Table 2 we compare OCAMS' *actual* effectiveness with the *simulated* effectiveness discussed in section 4 (going from ~6% shift time to ~0.4% shift time). When we assume that the simulated effectiveness is valid and that the amount of time spent—by the OCA Officers—on files with an issue is equal to the amount of time spent manually mirroring a file, then the actual OCAMS effectiveness is ~14% lower than the simulated effectiveness. In that case, we can say that *OCAMS currently saves ~79% of the OCA Officer's mirroring time*, instead of the ~93% predicted savings by the Future Simulation.

Table 2. OCAMS effectiveness

	Future OCA Mirroring Time	% Current OCA Mirroring Time (6% of shift time)	% Savings of Current OCA Mirroring Time
Simulated	0.40%	6.7%	93.3%
Actual	1.24%	20.6%	79.4%

5.2 Calculating return on investment

Design and development of OCAMS started in 2007 and will continue until only maintenance of OCAMS will remain. As development continues, the amount of OCA automation and thus the savings will increase.

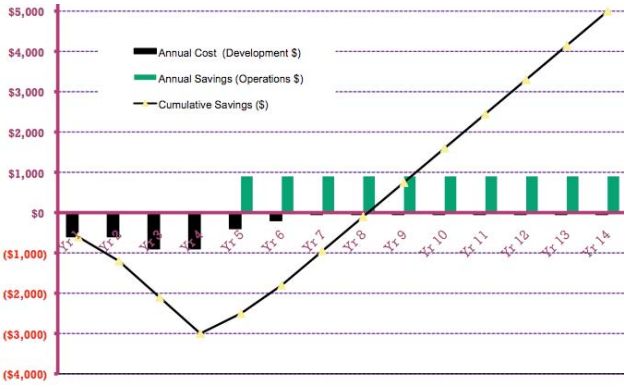

Figure 12. OCAMS ROI[1]

Figure 12 shows how we calculate the return of investment (ROI) of OCAMS[9], including both development cost and savings in the out years. As long as OCAMS does not automate the complete work of the OCA Officer, there is only development cost and no savings. Breakeven is calculated to happen somewhere in the future years, based on development cost and operations savings. Operations savings will end with the projected end of the ISS program.

6. DISCUSSION

In this section we discuss some of the benefits of our engineering approach and that of multi-agent architectures and BDI agents. We also touch upon the verification and validation (V&V) of our simulations, and end with planned future OCAMS functionality.

6.1 Work practice simulation

Our human-centered engineering approach is based on the following principles: 1) Development of software needs to be rooted in a thorough understanding of the work practice of the user and organization. This is done by performing in-situ work practice observations throughout the analysis and design process; 2) Work practice observations need to be modeled and simulated in a Current Simulation Model (CSM) and the designed software system in a Future Simulation Model (FSM), so that the result can be compared; 3) Design, modeling, and simulation of the new work system needs to follow a participatory design approach, involving the end-users and their management all throughout the design process. Brahms is a multiagent language and environment that supports this human-centered engineering approach.

As we have shown in the OCAMS project, the benefits of the OCA work practice simulations with Brahms were that a) the changes to the ISS MCC work system were analyzed upfront, allowing users and management to understand the impact of the new OCAMS system early on in the project; b) the use of the agent-based simulation environment enabled an easy transition from the FSM to the implementation of the OCAMS MAS; c) generation of statistics during the simulation were built into the actual system, allowing for return of investment analysis. Using this approach we have had 100% user acceptance from the moment OCAMS was put into operations, and continue to have strong user support.

The OCA Officers validated both the CSM and FSM by inspecting the output agent timelines from the simulation (e.g. see Figure 6). We verified the Current OCA model (CSM) by verifying that the OCA Officer agent correctly mirrored all files to the Mirror LAN server. To compare the current and future simulation runs, we used as input data the uplinked and downlinked files for the entire month of November 2006 (Nov. 1-30), with a total of 3843 ISS uplinked and downlinked files totaling 13 Gigabytes. The OCA team provided this data to us.

The Future OCA simulation model (FSM) was run from October 10 through October 12, 2007, on a Mac G5 multi-processor (4 CPUs) machine using 2GB of memory. The entire simulation ran in one BVM. It took about 31 hours to finish the simulation (BVM running time: 31:22:54.117).

[9] Actual time, cost and savings are changed in Figure 12.

6.2 Benefits of the MAS approach

The automation of the OCA work requires integrating multiple computer systems running on separate MCC networks. To integrate these systems, a distributed software- and hardware architecture is required. An agent architecture allows for a natural design of distributed system functionality as distributed services implemented by software agents. MAS architectures therefore naturally enable the design and implementation of a service-oriented architectures (SOA) [14]. In the OCAMS case, the Brahms environment enabled the easy transition from an agent-based simulation of people and systems to a system of multiple software agents, based on the functionality first simulated.

The ISS MCC is a complex environment with people, hardware systems and networks. Limitations in current OCA hardware, as well as the needed security requirements, made it necessary to change the OCAMS software and hardware architecture design multiple times over the course of two years. The flexibility of the MAS approach allowed the development team to adjust the OCAMS agent architecture to any new requirement that was put forward (e.g. on which machine and on what network different agents should be executed). Our customer was extremely pleased with the ease at which we were able to accommodate every change in system requirement.

A multi-agent architecture hides the details for distributing different software agents. Agents can run wherever and whenever they are needed, without impacting the integrity and overall functioning of the system. The agent developer does not have to deal with the intricate details of object and network communication protocols, such as Corba, SOAP or SSL. Agent communication is implemented at the designed agent communication language level. Low-level agent communication and distribution is taken care of by the agent framework used (Brahms/CI in our case). This enables an enormous flexibility in system architecture decisions.

Also, extension of functionality in a MAS is extremely easy. Adding new agents or extending old ones can provide new functionality without negative impact to the existing system functionality. This enables easy extension of OCAMS.

6.3 To BDI or not BDI

At the end of section 3.1 we mentioned that we developed rules-of-thumb for deciding when agents should be written in the Brahms language, and thus BDI-type agents, and when agents should be written in an imperative language, like Java. Here we briefly come back to this issue.

Although the BDI agent framework is well suited to develop "intelligent" agents, the implementation of the framework makes the agent computationally expensive. The main representational paradigm for BDI agents is declarative antecedent-consequence rules matching on the beliefs of the agents. Each BDI agent can be seen as a model-based system that represents the deliberation-action capability of a particular agent. We cannot justify using the BDI paradigm as if it were an imperative/procedural programming language, outside of programming convenience. We often cannot justify the use of the BDI paradigm to develop models for "non-deliberative" entities.

Our rule-of-thumb is that any agent that interacts with other systems should be a so-called Communication Agent written in an imperative/procedural language like Java. Such agents do not

perform any deliberation, but are merely proxy agents that "agentify" external systems. For example, the FTP-, MS Word and Excel and file copy and monitoring agents in OCAMS are all written as Communication Agents in Java, using the Brahms Java API.

6.4 Future functionality of OCAMS

The OCAMS system will be extended over the next three years. The objective is for the OCAMS system to enable the future concept of operations for the ISS OPSPLAN team. We have planned five separate releases that will increase the capability of OCAMS over time. Release 2 will include the automation of the archiving task. Release 3 will include the automatic notification of the uplinks and downlinks, and delivery of downlinked files to responsible flight controllers. In release 4 OCAMS will automate the actual uplinking and downlinking of files to the ISS. In the new concept of operation, flight controllers need to be able to run OCAMS from anywhere in MCC. This will require a web-based interface that is scheduled for Release 5.

The uplinking and downlinking of files to the ISS is a critical task for anyone who needs to send and receive file-based information from the ISS. We are planning to extend the reach of OCAMS to other flight controller positions.

7. CONCLUSIONS

This paper describes the development and operations of the OCA Mirroring Systems called OCAMS developed with NASA Ames' multiagent Brahms environment. OCAMS is a MAS deployed in NASA's Mission Control Center for the International Space Station. Release 1 of OCAMS automates the mirroring activity of the OCA Officer. In the mirroring activity files that are uplinked and downlinked from the Ops LAN onboard the ISS are mirrored to the ground-based Mirror LAN. OCAMS has been in operations since July 8, 2008 and is used 100% of the time by the OCA Officers in the ISS MCC. With using Brahms as an ABMS and MAS implementation environment in NASA's MCC, Brahms has transitioned from a research AOL to an operational AOL that is at NASA's Technology Readiness Level 9.

8. ACKNOWLEDGMENTS

We thank all the OCA Officers in DO4 at NASA Johnson Space Center for their continued support in the use and development of OCAMS.

We also thank our current funders at NASA Johnson Space Center, and all our past funders of the Brahms project.

9. REFERENCES

[1] Clancey, W.J., et al., Multi-Agent Simulation to Implementation: A Practical Engineering Methodology for Designing Space Flight Operations, in The Eighth Annual International Workshop "Engineering Societies in the Agents World" (ESAW 07) A. Artikis, et al., Editors. 2008, Springer: London.

[2] Sierhuis, M., W.J. Clancey, and R.J.J.v. Hoof, Brahms: An Agent-Oriented Language for Work Practice Simulation and Multi-Agent Systems Development in Multi-Agent Programming, 2nd Edition, M.D. Rafael H. Bordini, Jürgen Dix, Amal El Fallah-Seghrouchni, Editor. Submitted, Springer.

[3] Sierhuis, M., W.J. Clancey, and R.v. Hoof, Brahms: A multiagent modeling environment for simulating work processes and practices. International Journal of Simulation and Process Modelling, Inderscience Publishers, 2007. 3(3): p. 134-152.

[4] Aalst, W.v.d., Putting Petri Nets to Work in the Workflow Arena, in Petri Net Approaches for Modelling and Validation, J.M.C. W. van der Aalst, F. Kordon, G. Kotsis and D. Moldt, Editor. 2003, Lincom Europa: Munich. p. 125-143.

[5] Aalst, W.M.P.v.d. and A.H.M.t. Hofstede, YAWL: Yet Another Workflow Language. Information Systems, 2005. 30(4): p. 245--275.

[6] Burmeister, B., et al., BDI-Agents for Agile Goal-Oriented Business Processes, in Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008) - Industry and Applications Track, B. Berger, Nishiyama, Editor. 2008, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org): Estoril, Portugal. p. 37-44.

[7] Caire, G., D. Gotta, and M. Banzi, WADE: A software platform to develop mission critical applications exploiting agents and workflows, in Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008) - Industry and Applications Track, B. Berger, Nishiyama, Editor. 2008, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org): Estoril, Portugal. p. 29-36.

[8] Ören, T.I., Agent-directed Simulation - Challenges to meet Defense and Civilian Requirements, in The 2000 Winter Simulation Conference, J.A. Joines, Editor. 2000: Orlando, FL. p. 1757-1762.

[9] Yilmaz, L., T. Ören, and A. Nasser-Ghasem, Agents, Simulation, and Gaming. Simulation and Gaming Journal, 2006. 37(3): p. 339-349.

[10] Clancey, W.J., et al., Brahms: Simulating practice for work systems design. International Journal on Human-Computer Studies, 1998. 49: p. 831-865.

[11] Sachs, P., Transforming Work: Collaboration, Learning, and Design. Communications of the ACM, 1995. 38(9): p. 36-44.

[12] Blomberg, J., et al., Ethnographic Field Methods and Their Relation to Design, in Participatory Design: Principles and Practices, A.N. D. Schuller, Editor. 1993, Lawrence Erlbaum Associates: Hillsdale, NJ. p. 123-155.

[13] Greenbaum, J. and M. Kyng, eds. Design at Work: Cooperative design of computer systems. 1991, Lawrence Erlbaum: Hillsdale, NJ.

[14] Papazoglou, M.P. and v.d. W-J. Heuvel, Service oriented architectures: approaches, technologies and research issues. The VLDB Journal, 2007(16): p. 389–415.