Assessing Human-Agent Teams for Future Space Missions

Nanja J.J.M. Smets, Jurriaan van Diggelen, and Mark A. Neerincx,
TNO Human Factors

Jeffrey M. Bradshaw, *IHMC*

Catholijn M. Jonker, Lennard J.V. de Rijk, Pieter A.M. Senster, and Ot ten Thije,
Technical University of Delft

Maarten Sierhuis, *NASA Ames Research Center*

Running computer simulations of work practices early on lets researchers test human-agent teams in dangerous, complex environments by incrementally increasing fidelity, adding realistic features, and incorporating human participants.

A team sets out on a mission to Mars, coping with a dangerous and complex environment. Communication with Mission Control Center is difficult because one-way radio traffic can experience delays of up to 22 minutes. Human and machines, including sophisticated software and robotic agents, must collaborate closely and with relative independence from Mission Control Center for the mission to fulfill its goal.[1–3] Thus, the team must be able to cope with unexpected events on their own.

Testing requirements of such complex cognitive-support systems remains challenging. In general, we must account for three issues when testing human-robot team designs in these types of environments. First, although testing a prototype on Mars isn't an option, it's still important to test the prototype in a realistic environment to ensure the system is suitable for use in the eventual context. Second, testing a prototype for human-robot teams requires involving representative human participants, such as astronauts, who are hard to come by. Third, when testing an adaptive human-robot team in a complex environment with demanding tasks, controlling the experiment with laboratory precision is difficult.

We address these issues by choosing the right experiment in the test space with the appropriate level of fidelity and realism of the experiments. The test can start simply, with low fidelity and realism. (*Fidelity* refers to an adequate representation of relevant rules in a human-agent team that are addressed in the test and specifically the dependencies. *Realism* varies from one extreme—the real environment—to the other, a virtual environment.) We then increase the fidelity and realism in subsequent tests.

How do we design an iteration of experiments with the appropriate levels of fidelity and reality for each of the components? To help groups of humans and machines meet these demands, we are developing a mission execution crew assistant (MECA). For this

1541-1672/10/$26.00 © 2010 IEEE
Published by the IEEE Computer Society

purpose, we are deriving a requirements baseline for a distributed system of electronic partners (ePartners) to enhance astronauts' self-management in nominal (normal) and off-nominal (outside acceptable limits) actions in long-duration missions.[4] For the iterative development, testing, and refinement of the use cases, claims, and requirements baseline, we use the situated cognitive engineering (SCE) method.[5] Our proposed method involves choosing and combining different types of experiments, and the simulation tool helps fill in the blank areas in the test space.

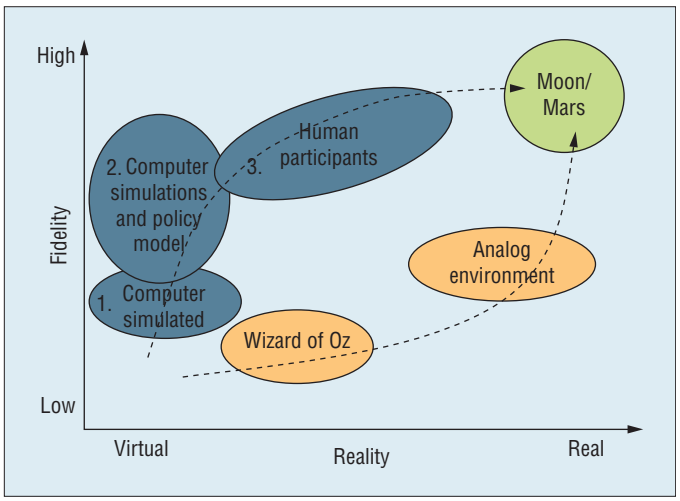

Figure 1. Test space. The gray ovals indicate the test environments of previous MECA project research, while the blue ovals depict the method we describe here, from a full computer simulation to one including human participants.

Test Method

It is possible to vary test experiments by altering the fidelity of the environment and actors and adjusting the test environment, from fully virtual to real world, to mixed reality (a combination of the two). Figure 1 depicts the test space, with reality on the *x*-axis and fidelity on the *y*-axis. In earlier research in the MECA project, we conducted a Wizard of Oz (WoZ) experiment[6] and an experiment in an analog environment on volcanic grounds (see the lower arrow in Figure 1). In the WoZ experiment, we used a virtual environment, virtual agents, and real participants. In the analog environment experiment, we used a real environment (but still not as realistic as the Moon or Mars), virtual agents, and real participants.

The upper arrow in Figure 1 depicts the approach we describe here. The first blue balloon depicts a full computer simulation, where the actors and environment are modeled in the Brahms work practice modeling tool.[7] Computer simulations let us run numerous tests and allow full control over the environment, agents, and actors, with low cost. A computer simulation of individual agent behavior, however, does not incorporate the group behavior that is also a product of the agents' organization. Hence, adding an organization model adds fidelity. We have accomplished this by extending Brahms with a KAoS policy model.[8,9] Eventually, we can enhance the realism by testing with one or more human participants (see Figure 1). To be able to control as many context factors as possible, it helps to simulate the environment and team members in a scenario. By simulating team members, we can induce certain off-nominal events, such as the fainting of a fellow astronaut.

Simulation Platform

Tools for testing human-agent teams in a mixed-reality environment with different levels of fidelity are currently lacking. We developed a simulation platform in which we model scenarios and actors by formulating work practices and policies. This corresponds to the idea that agent behavior is not only a product of an agent's individual mental attitudes, but also of the organization the agent belongs to and the world state.

To simulate the environment, individual agent behavior, and policies, we use a combination of Brahms and KAoS. Brahms let us model the agent's individual behavior, whereas KAoS provides the regulation from above. We had to integrate the Brahms and KAoS frameworks in order to use them together to simulate the human-agent teamwork. Brahms agents must be able to query KAoS for policies, while still respecting the agents' autonomy, which requires that the organizational structure should not interfere with the autonomy of the participating agents.[10] Agents must be able to disobey a policy. Because of the different communication languages and protocols in Brahms and KAoS, this requires some translation method.

Brahms

A *work system* is a natural setting for those who frequently work within it. A *workplace* is where the work system comes alive, where the daily work is continuously being performed on the basis of familiar past performance as well as unanticipated changes. In other words, work is like a symphony, well rehearsed, but always different. It is this "symphony" we are interested in composing (that is, designing changes) by using a modeling and simulation language that lets us model and predict the impact of a designed change on the current system.

Brahms is a multiagent modeling language for simulating human work practice that emerges from work

```
GROUPS are composed of
     AGENTS having
            BELIEFS and doing
            ACTIVITIES executed by
                  WORKFRAMES defined by
                        PRECONDITIONS, matching agents beliefs
                        PRIMITIVE ACTIVITIES
                        COMPOSITE ACTIVITIES, decomposing the activity
                        DETECTABLES, including INTERRUPTS, IMPASSES
                        CONSEQUENCES, creating new beliefs and/or facts
            DELIBERATION implemented with
                  THOUGHTFRAMES defined by
                  PRECONDITIONS, matching agents beliefs
```

Figure 2. A simple taxonomy of some Brahms language concepts. These concepts form the basic building blocks in any Brahms project.

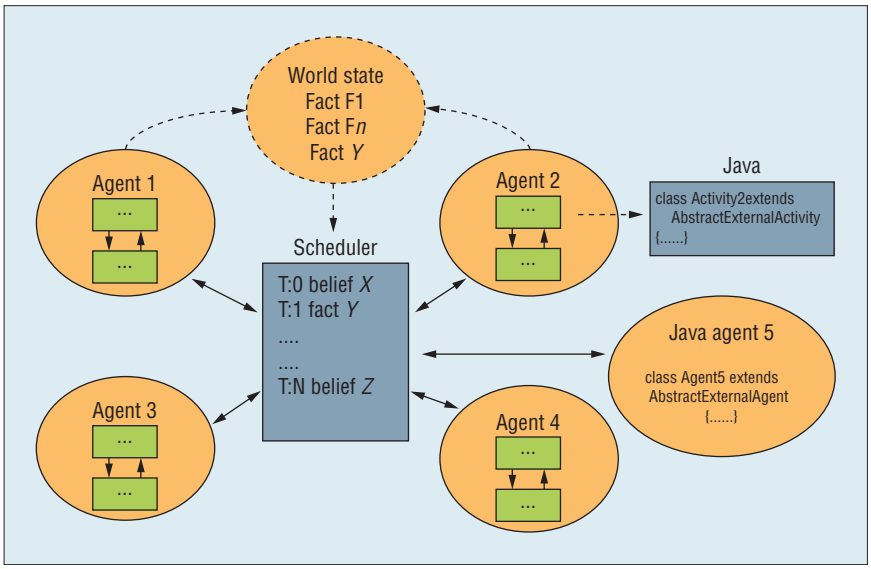

Figure 3. Brahms agent architecture. A Brahms simulation may run Brahms and Java agents.

processes in organizations.[1] The same Brahms language can serve to implement and execute distributed multiagent systems based on models of work practice that were first simulated. Brahms demonstrates how to integrate a multiagent belief-desire-intention (BDI) language,[11] symbolic cognitive modeling, traditional business process modeling, and situated cognition theories in a coherent approach for analysis and design of organizations and human-centered systems. Brahms is being developed and used by the Work Systems Design and Evaluation group in the NASA Ames' Intelligent Systems division.

The Brahms language supports various agent concepts such as mental attitudes, deliberation, adaptation, social abilities, and reactive- as well as cognitive-based behavior. The following Brahms language features are available to model agents:

- *Mental attributes* include attributes, relations, beliefs and facts, no explicit desires, and frame instantiations (intentions).
- *Deliberation* involves concluding new beliefs and using thought frames for reasoning.
- *Adaptation* includes changing beliefs, execution activity behavior, and reasoning based on context.
- *Social abilities* encompass groups and group inheritance, communication, and models of the environment (objects, geography, and location).
- *Reactive and cognitive-based behavior* involves modeling-activity behavior versus purely cognitive behavior, detectables, and workframe-activity subsumption.
- *Communication* includes communication activities and communicative acts.

Brahms is an agent-oriented language that lets us easily create agent groups that execute activities based on local beliefs. Figure 2 shows a simple taxonomy of some of the language concepts we discuss here.

Figure 3 shows the Brahms agent architecture. A Brahms virtual machine (BVM), written in Java, loads in compiled Brahms and Java agents. Brahms agents are written in the Brahms language, whereas Java agents are written in Java using the Brahms Java application interface (JAPI). In simulation mode, the BVM includes a scheduler that synchronizes

time, communicates beliefs, and detects facts in the world state.

To allow human-in-the-loop simulation, we added the capability to run in "wall-clock" time. This means that every simulated second takes a second in real time, which will allow people to participate in the simulation together with agents.

KAoS HART Framework

The KAoS Human-Agent-Robot Teamwork (HART) services framework has been adapted to provide the means for dynamic regulation on various agent, robotic, Web services, Grid services, and traditional distributed computing platforms.[8] It also provides the basic services for distributed computing, including message transport and directory services, as well as more advanced features such as domain and policy services.

All team members, human and agent, register with the directory service and provide a description of their capabilities. This lets them query the directory service to find other team members and match them based on capability. The domain and policy services manage the organizational structure among the agents, providing the specification of roles and allowing dynamic team formation and modification.

Two important requirements for the KAoS architecture are modularity and extensibility. These requirements are supported through a framework with well-defined interfaces that can be extended, if necessary, with the components required to support application-specific policies. Figure 4 shows the KAoS architecture's basic elements; its three layers of functionality correspond to three different policy representations:

- The *human-interface layer* uses a hypertext-like graphical interface for policy specification in the form of natural English sentences. The vocabulary is automatically provided from the relevant ontologies, consisting of reusable core concepts augmented by application-specific concepts.
- In the *policy-management layer*, the Web Ontology Language (OWL, www.w3.org/TR/owl-features) is used to encode and manage policy-related information. The Distributed Directory Service (DDS) encapsulates a set of OWL reasoning mechanisms.
- For the *policy-monitoring and enforcement layer*, KAoS automatically "compiles" OWL policies to an efficient format for monitoring and enforcement. This representation provides the grounding for abstract ontology terms, connecting them to the instances in the runtime environment and to other policy-related information.

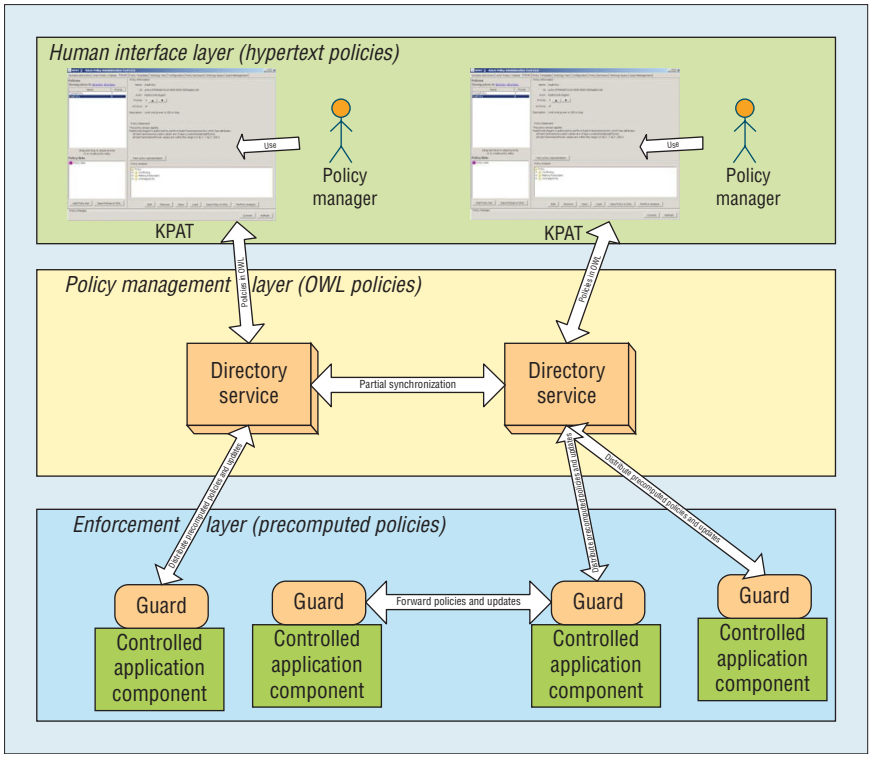

Figure 4. KAoS architecture. The human-interface, policy-management, and policy-monitoring and enforcement layers provide three different policy representations.

OWL semantics do not allow the expression of attribute constraints, but the KAoS role-value-map reasoner solves this problem.[8] During policy analysis, the OWL reasoner finds relations between action classes controlled by policies through subsumption reasoning. Description logic, however, does not recognize role-value map semantics. So when the subsumption reasoner finds a relation between actions and subsequently policies, it is still up to the manager to determine whether potential instances of role-value maps separate the actions and nullify the policy relation. As policies are distributed to guards, the reasoner classifies existing instances (such as the list of actors) so that relevant information of other kinds can be sent to the guards at the same time. As relevant policies are distributed to guards, they are compiled into an extremely efficient form that no longer requires an OWL reasoner. Policy decisions are

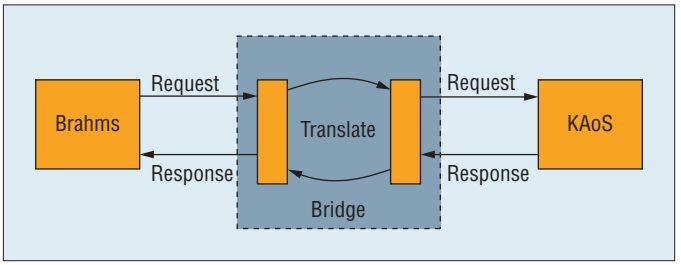

Figure 5. Schematic overview of the KAoS-Brahms bridge. A bidirectional translation is provided between KAoS concepts and Brahms concepts.

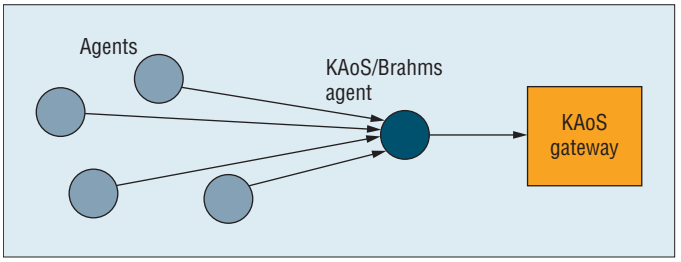

Figure 6. Agents query the KAoS policy library through one KAoS-Brahms agent. This agent forms the Brahms representative of KAoS.

rendered with near-table-look-up efficiency.

There are two main types of policies: authorizations and obligations. The set of permitted actions is determined by authorization policies that specify which actions an actor or set of actors are permitted (positive authorizations) or not allowed (negative authorizations) to perform in a given context. Obligation policies specify actions that an actor or set of actors is required to perform (positive obligations) or for which such a requirement is waived (negative obligations). From these primitive policy types, we build more complex structures that form the basis for team coordination.

Brahms-KAoS Integration

A Brahms agent must able to ask KAoS for a policy. We enable this communication using a KAoS-Brahms bridge, but KAoS and Brahms must still know the same concepts. Thus, we developed an ontology builder to map concepts in Brahms to the corresponding concepts in KAoS.

The bridge lets KAoS and Brahms communicate even though they use different languages. Figure 5 give a high-level overview of how the bridge works. Brahms sends a request, which the bridge translates. The bridge then sends the translated request to KAoS, which sends a response to the bridge. Lastly, the bridge translates the response and sends it to Brahms.

We can implement the bridge by either calling a Java activity within a Brahms agent or creating an external agent entirely implemented in Java (see Figure 6). We compared two alternatives to determine which of the two is the better solution.

To use Java activities in Brahms, the programmer creates a Java class with a special method. The BVM invokes this method when the Brahms agent invokes the activity. Java activities can be used anywhere that ordinary activities can be used.

When using an external Java agent, on the other hand, a Java class is created that implements a specific interface. This class is then registered as an external agent in the Brahms model. When the BVM runs the model, it instantiates the class and informs it of any events happening in the simulation. Other agents in the environment can communicate with the external agent using the standard communication activities in the Brahms language. The main advantages of using activities are that they do not require a complex communication protocol and that the activity semantics are similar to object-oriented method calls, which makes them intuitive to use. However, the system needs to work in an agent-oriented environment, which poses different requirements. Apart from that, activities cannot use state to speed up the process or help in debugging because new instances are created for every call. Finally, using activities results in numerous access points to KAoS. When an agent invokes an activity, that activity communicates with KAoS, which means there are at least as many access points as there are agents.

This problem does not occur when an external agent is used because in that case all communication runs through one agent specially designed as the bridge between Brahms and KAoS. This solution has several advantages. Because the bridge is an agent itself, it is a cleaner conceptual match with the surrounding system. Development is also easier because we can easily replace an agent with a stub while working on other parts of the system. Furthermore, an agent has a persistent state while the system runs, which lets it use resources more efficiently. However, this does come at the price of a more complex system because we need to design communication protocols.

We decided to use a KAoS-Brahms agent because of the ease of developing and debugging and the cleaner conceptual role of the agent.

KAoS-Brahms Agent

An important aspect of KAoS policies is the ability to authorize actions based on their properties. To unlock the full potential of policies, these properties must therefore be communicated to KAoS when a Brahms

agent issues a request to perform an action. A Brahms agent sends the information encapsulated in a message as defined by the Foundation for Intelligent Physical Agents (FIPA) Agent Communication Language.[12] In this case, a new Brahms class is created for each kind of action, containing the attributes that are relevant to the action's authorization.

When an agent wants to obtain authorization to perform an action, it constructs a new instance of this class, sets the values for the relevant attributes, and sends it to the KAoS gateway, encapsulated in a FIPA message. The KAoS gateway translates the action and its attributes and passes them on to KAoS. The result of the policy check is written back to the object, which the agent can then use to determine what it should do.

KAoS Gateway

Requests are formulated in the bridge's policy library. They are then sent to KAoS using the KAoS gateway, which serves as a translation gateway to KAoS. Translation is necessary because the Brahms agents produce Brahms objects, whereas KAoS does not. Also, KAoS's response to queries will have to be translated back into Brahms objects to be useful to the original requesting agents.

To realize the full potential of KAoS policies, this translation must be as complete as possible. This means that attributes of actions must be accounted for when translating from Brahms to KAoS. However, obligations and restrictions on attributes of obligated actions must also be expressible in Brahms objects when a response is to be sent back to an agent. This indicates the need for two subsystems in the KAoS gateway agent: one to handle translation and one to handle the communication of the translated objects, both to KAoS and Brahms.

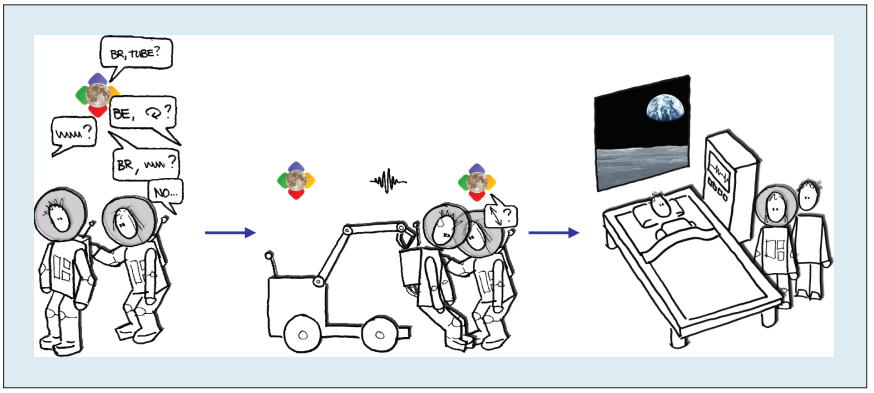

Figure 7. Short storyboard of the use-case scenario. When a problem occurs with Benny's spacesuit, he faints and must be brought to the habitat, where he receives medical attention.

Ontology Builder

When the KAoS policy services are used to enforce policies on a Brahms model, mapping is needed between concepts in Brahms and the corresponding concepts KAoS uses. We can manually construct both this mapping and the ontology used by KAoS, but it is a complex task to keep the model and ontology consistent. Policies in KAoS are defined using an OWL-based ontology.[13] Although the OWL standard does not enforce the way in which an ontology is serialized, using a Resource Description Framework (RDF) and XML syntax seems to be common practice. Fortunately, compiled Brahms models are also specified in XML files. Because of this, ontologies can be extracted from Brahms compiled code using a simple transformation from one XML file into the other.

The ontology builder constructs ontologies from a set of Brahms models for several reasons. First, Brahms models contain more than just the concepts relevant for policies; they also contain code that specifies when and how work frames and activities are performed. Generating Brahms models from an ontology would mean that code gets lost every time a new ontology is created. Although this means we must redefine policies, we can easily save and restore policies in KAoS if they are still applicable. Second, designing the system this way let us more easily extend previous Brahms projects with policies. Third, modeling agents in Brahms is considerably more explicit than modeling a formal set of concepts and their relationships, which lets the modeler think more about the work practice in the scenario that needs to be modeled than the actual formal representation.

For these three reasons, the simulation tool generates both the ontology and the mapping from the Brahms model. This enables automatic updating of the ontology whenever the model changes. To achieve this, the builder must load the model and extract an ontology defined in OWL.

Applying Brahms-KAoS for Testing

In a pilot experiment, we used a scenario-based design, the test space, and the simulation platform. This pilot shows that with the simulation platform, use cases, requirements, and claims can be tested systematically as formulated in the SCE method.

Figure 7 illustrates a storyboard for the use case from the pilot. This storyboard, which has been used before with different stakeholders and representative end users, proved very

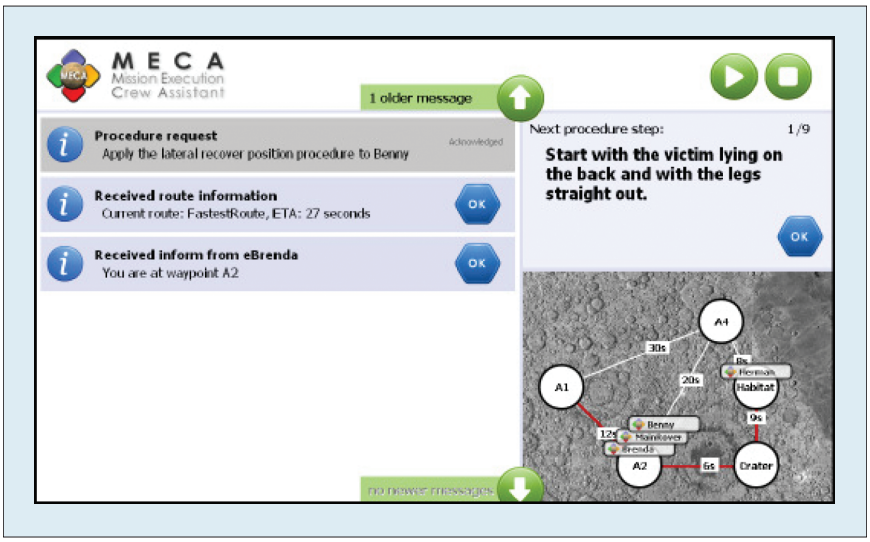

Figure 8. Human-in-the-loop interface. Messages are on the left, and procedure steps and a map with the actors are on the right.

successful for our purposes.[4] In our use-case scenario, Benny has a problem with his suit. Benny, Brenda, and their ePartner diagnose the problem and determine that the heater has broken. When Benny overheats and faints, he must be brought back to the habitat where he can receive medical attention. On the way to the habitat, Brenda performs first aid. Brenda's ePartner keeps her up to date on important events. During their ride to the habitat, the ePartner receives information about a crater that blocks their path. The route must be changed, so Brenda must accept or deny the alternative route.

The framework consists of the simulation platform extended with an interface for the human to interact with (see Figure 8). In addition to building the interface, we must change the simulation platform so that the simulation can run in real time. Brahms does not support this by default. We chose to model a separate agent that manages the time because this will help cleanly separate the code dealing with time from the Brahms model and it requires virtually no changes in the model.

At two moments in this scenario, the human or the machine must choose whether to obey a policy. For instance, Brenda can choose to take the diverted route or choose to take the original route (which in this case will cause a crash in the crater). This scenario shows that the actors can ignore policies, along with the resulting consequence. The simulation platform enhanced the test's fidelity.

In the future, we want to test this scenario with human participants and extend it with a virtual reality environment to add more realism. The use of virtual reality for training and testing has proven successful in other domains. For instance, we have used the virtual reality environment Unreal Tournament (from Epic Games) to test mobile decision support for first responders.[14] Because the platform is flexible, tests can be performed with one or multiple users, with a real or simulated rover, and so forth.

The scenario we discussed here is just one use case, but we have a library of use cases in the MECA project that can be deployed for testing. The proof of concept simulation is promising, and we intend to perform more tests in the future. We believe the method and simulation platform are particularly useful for larger groups and is a necessary test tool before setting foot on Mars.

Acknowledgments

MECA (www.crewassistant.com) is a development funded by the European Space Agency (contract number 21947/08/NL/ST).

References

1. J.M. Bradshaw, P. Feltovich, and M. Johnson, "Human-Agent Interaction," *Handbook of Human-Machine Interaction*, G.A. Boy, ed., Ashgate Publishing, to be published in 2010.
2. R. Doyle et al., "Progress on AI, Robotics, and Automation in Space: A Report from i-SAIRAS 08," *IEEE Intelligent Systems*, vol. 24, no. 1, 2009, pp. 78–83.
3. D. Schreckenghost et al., "Intelligent Control of Life Support for Space Missions," *IEEE Intelligent Systems*, vol. 17, no. 5, 2002, pp. 24–31.
4. M.A. Neerincx et al., "The Mission Execution Crew Assistant: Improving Human-Machine Team Resilience for Long Duration Missions," *Proc. 59th Int'l Astronautical Congress* (IAC 2008), 2008.
5. M.A. Neerincx and J. Lindenberg, "Situated Cognitive Engineering for Complex Task Environments," *Naturalistic Decision Making and Macrocognition*, J.M.C. Schraagen et al., eds., Ashgate Publishing, 2008, pp. 373–390.
6. N.J.J.M. Smets et al., "Game-Based versus Storyboard-Based Evaluations of Crew Support Prototypes for Long Duration Missions," *Acta Astronautica*, vol. 66, nos. 5–6, 2009, pp. 810–820.
7. M. Sierhuis et al., "Brahms: An Agent-Oriented Language for Work Practice Simulation and Multi-agent Systems Development," *Multi-agent Programming*, Springer, 2009.
8. A. Uszok et al., "New Developments in Ontology-Based Policy Management: Increasing the Practicality and Comprehensiveness of KAoS," *Proc. 2008 IEEE Workshop Policies for Distributed Systems and Networks* (*Policy 2008*), IEEE Press, 2008, pp. 145–152.
9. J. van Diggelen et al., "Policy-Based Design of Human-Machine Collaboration in Manned Space Missions," *Proc.*

3rd IEEE Int'l Conf. Space Mission Challenges for Information Technology (SMC-IT 09), IEEE CS Press, 2009, pp. 376–383.
10. V. Dignum, "A Model for Organizational Interaction: Based on Agents, Founded in Logic," doctoral dissertation, Institute of Computing Sciences, Utrecht Univ., 2004.
11. M.E. Bratman, *Faces of Intention: Selected Essays on Intention and Agency*, Cambridge Univ. Press, 1999.
12. *FIPA Communicative Act Library Specification*, Foundation for Intelligent Physical Agents, 2002.
13. T.R. Gruber, "A Translation Approach to Portable Ontology Specifications," *Knowledge Acquisition*, vol. 5, no. 2, 1993, pp. 199–220; http://dx.doi.org/10.1006/knac.1993.1008.
14. N.J.J.M. Smets et al., "Effects of Mobile Map Orientation and Tactile Feedback on Navigation Speed and Situation Awareness," *Proc. 10th Int'l Conf. Human Computer Interaction with Mobile Devices and Services* (MobileHCI 08), ACM Press, 2008, pp. 73–80.

THE AUTHORS

Nanja J.J.M. Smets is a research member of the Cognitive Systems Engineering group at TNO Human Factors. Her research interests include cognitive engineering, human-agent interaction, and astronaut-support systems. Smets has an MS in artificial intelligence and man-machine interaction from Vrije Universiteit Amsterdam. Contact her at nanja.smets@tno.nl.

Jurriaan van Diggelen is a research member of the Cognitive Systems Engineering group at TNO Human Factors. His research interests include cognitive engineering, human-agent teamwork, computational policies. Van Diggelen has a PhD in Artificial Intelligence from Utrecht University. Contact him at jurriaan.vandiggelen@tno.nl.

Jeffrey M. Bradshaw is a Senior Research Scientist at the Florida Institute for Human and Machine Cognition *(IHMC)*. His research interests include policy-based coordination of joint activity in humans and machines, Semantic Web technologies, adjustable autonomy. Bradshaw has a PhD in Cognitive Science, University of Washington. Contact him at jbradshaw@ihmc.us.

Mark A. Neerincx is a Senior Research scientist at TNO and Professor at Technical University of Delft. His research interests include cognitive engineering, electronic Partners, and cognitive taskload modeling for adaptive interfaces. Neerincx has a PhD in psychology from the University of Groningen. Contact him at mark.neerincx@tno.nl.

Catholijn M. Jonker is a full professor at Technical University of Delft. Her research interests include multi-agent systems, human-machine interaction, decision support systems. Jonker has a PhD in Artificial Intelligence from Utrecht University. Contact him at c.m.jonker@tudelft.nl.

Lennard J.V. de Rijk is a Master Student at Technical University of Delft. His research interests include Agent-based modelling. De Rijk has a bachelor degree in computer science from Technical University of Delft. Contact him at L.J.V.deRijk@student.tudelft.nl.

Pieter A.M. Senster is a Master Student at Technical University of Delft. His research interests include Agent-based modeling. Senster has a bachelor degree in computer science from Technical University of Delft. Contact him at pieter@pietersenster.nl.

Ot ten Thije is a Master Student at Technical University of Delft. His research interests include Agent-based modeling. Ten Thije has a bachelor degree in computer science from Technical University of Delft. Contact him at J.O.A.tenThije@student.TUDelft.nl.

Maarten Sierhuis is a Senior research scientist at the NASA Ames Research Center. His research interests include modeling and simulation of knowledge, organizations, human behavior and work practices. Sierhuis has a PhD in Social Science Informatics from University of Amsterdam. Contact him at maarten.sierhuis-1@nasa.gov.