TOWARDS THE VERIFICATION OF HUMAN-ROBOT TEAMS

Michael Fisher*, Edward Pearce, Mike Wooldridge
Department of Computer Science, University of Liverpool, UK

Maarten Sierhuis, Willem Visser
RIACS/NASA Ames Research Center, Moffett Field, CA, USA

Rafael H. Bordini
Department of Computer Science, University of Durham, UK

ABSTRACT

Human-Agent collaboration is increasingly important. Not only do high-profile activities such as NASA missions to Mars intend to employ such teams, but our everyday activities involving interaction with computational devices falls into this category. In many of these scenarios, we are expected to *trust* that the agents will do what we expect and that the agents and humans will work together as expected. But how can we be sure? In this paper, we bring together previous work on the verification of multi-agent systems with work on the modelling of human-agent teamwork. Specifically, we target human-*robot* teamwork. This paper provides an outline of the way we are using formal verification techniques in order to analyse such collaborative activities. A particular application is the analysis of human-robot teams intended for use in future space exploration.

INTRODUCTION

In our previous work, we have developed techniques for verifying (using model checking) multi-agent systems [1, 2]. The agents involved are rational/intelligent [25] and are represented in a high level language describing their beliefs, intentions, etc. While this has potential to be used in scenarios where rational agents are used to control critical applications, there is a need to consider the verification of situations with more human involvement.

Our work on agent verification has been developed in collaboration with NASA. Rather than deploying completely autonomous, human-free, space missions, NASA is now turning to joint human-agent (typically, human-robot) teams as a way to handle more advanced space missions

*Principal Contact: `M.Fisher@csc.liv.ac.uk` within the future. Yet, it is still important to ensure that the humans and agents will work together to achieve their goals.

Fortunately, attempts have been made to model human-agent activity, specifically via the Brahms framework [21, 20]. Brahms describes teamwork at a high level of abstraction yet, since its intended semantics is quite close to BDI models of agency [17], there is a possibility that agent verification techniques can be relevant here. In particular, if we can characterise (simple forms of) human behaviour as agent behaviour, then we ought to be able to utilise our previous work to verify human-agent interactions. This paper describes our approach in this area.

The structure of the paper is as follows. We first provide some background concerning agents and their use in space missions, then we outline our previous work on the verification of multi-agent systems. The following section then describes the need for human-agent (specifically human-robot) teams, and after that we address the problem of verifying the behaviour of such human-robot teams. Some concluding remarks are given at the end.

BACKGROUND

Rational Agents

An agent can be seen as an *autonomous* entity. Thus, the agent makes its own decisions about what to pursue. We are particularly concerned with *rational agents*, which can be seen as agents that make *explainable* decisions on how to act so as to achieve their goals in the best possible way. Thus, the key new aspects that such agents bring is the need to consider, when designing or analysing them, not just what they do but *why* they do it. Since agents are autonomous, understanding why an agent chooses a course of action is vital. In [26], a number of additional aspects of agents are described, in particular their *pro-active*, *reactive*, and *social* attributes. Although we do not see these as essential, most of the agents considered here do, indeed, incorporate such notions.

A number of logical theories of (rational) agency have been developed, such as the BDI [16, 17] and KARO [12, 14] frameworks (BDI stands for “Belief-Desire-Intention”, and KARO for “Knowledge, Abilities, Results, and Opportunities”). One reason for using logic in agent-based systems is that a formal semantics comes (more or less) for free. From this it follows that the semantics of logic-based agents is strongly dependent on their underlying logic. The foundations of such agent description frameworks are usually represented as (often complex) non-classical logics. In addition to their use in agent theories, where the basic representation of agency and rationality is explored, these logics often form the basis for agent-based formal methods and, as we will see later, agent-based formal verification techniques.

Towards Rational Agents in Space Missions

Software is fast becoming an enabling technology for space missions. For example, the International Space Station (ISS) contains millions of lines of code that amongst other things supports the inter-operability of US and Russian modules. The true future of software in space applications are however embodied in missions such as the Remote Agent from Deep-Space 1 [18] and the Demonstration of Autonomous Rendezvous Technology (DART) [8] where spacecraft were autonomously controlled by software systems. Deep space missions require such autonomous behaviour since communication delays make earth-controlled missions almost impossible. Autonomy is also a major cost driver for NASA since human controlled missions require large earth-based teams for support. See Figure 1 for an outline of why *rational agents* are increasingly used to provide a high-level abstraction/metaphor for building complex/autonomous space systems.

$$\begin{tikzcd}
\left.\begin{matrix}\mathit{Uncertain}\\ \mathit{environments}\end{matrix}\right\} \arrow[r] & \text{more ‘Intelligence’} \arrow[d] \\
\left.\begin{matrix}\mathit{Cooperation}\\ \mathit{Coordination}\end{matrix}\right\} \arrow[r] & \textbf{\textsc{Rational Agents}} \\
\left.\begin{matrix}\mathit{Communication}\\ \mathit{problems}\end{matrix}\right\} \arrow[r] & \text{more Autonomy} \arrow[u]
\end{tikzcd}$$

Figure 1: Motivations for using Rational Agents in Space Applications

Autonomous software is, however, hard to verify due to the uncertain/nondeterministic environments in which it executes. Yet, it is essential to attempt such verification since autonomous systems are amongst the most complex (and error prone) systems to develop. For example, the Remote Agent had a software deadlock during flight (although the mission completed successfully) and the DART mission failed (although the reasons are not clear yet) [11].

It is also very likely that autonomous systems will contain many rational agents that will need to interact and share information and resources: multiple space probes requiring collision avoidance, multiple surface rovers, autonomous docking between vehicles, etc. Additionally, NASA's new focus on doing human-robotic exploration of the Moon and Mars, brings human "agents" into the picture - this might simplify some of the autonomy requirements, but increase safety and certification concerns.

Thus, there is a clear path, not only towards rational agents, but also towards teams of such agents and humans. Such teams must be able to coordinate their activities, for example to avoid collisions and to cooperate to achieve some common goal. Although there has been relatively little work on the verification of BDI agents, there has been considerably more work on analysing teamwork, in particular where humans and agents interact to achieve common goals. A prominent approach is that of TEAMCORE developed by M.Tambe and colleagues [15], and an example of a significant application they have developed is the DEFACTO system [19], which coordinates the action of agents (e.g., representing fire engines) and humans for disaster response.

VERIFYING AGENT BEHAVIOUR

As seen above, the technology offered by autonomous software agents, particularly rational agents, is very appealing. Rational agents can:

- adapt to uncertain environments;

- solve problems independently;
- communicate and collaborate with other agents;
- learn new behaviour; etc.

However, can we be sure such agents will behave as expected? If rational agents are to have even partial control of critical missions, then we have to be able to trust them. But how can we check this? Our approach is to use *logical* verification techniques.

Logics for Precise Description — Logics provide an unambiguous formalism for describing the behaviour of systems. There is a wide variety of logics that can be developed for different scenarios, such as:

- dynamic communicating systems $\longrightarrow$ temporal logics;
- systems managing information $\longrightarrow$ logics of knowledge;
- autonomic/intelligent systems $\longrightarrow$ logics of goals, intentions, aims;
- situated systems $\longrightarrow$ logics of belief, contextual logics;
- systems in uncertain environments $\longrightarrow$ probabilistic logics.

Importantly, combinations of such logics are needed, so that we can specify high-level properties such as:

> "a rational agent has a problem that it $\boldsymbol{A}$ims to solve, but $\boldsymbol{B}$elieves that it needs help (i.e., the agent cannot solve the problem itself), so in the $\boldsymbol{N}$ext moment in time, its $\boldsymbol{G}$oal will be to get help."

in a compact and precise (unambiguous) formula like this:

$$(\boldsymbol{A}\,\texttt{solve-problem} \wedge \boldsymbol{B}\,\texttt{need-help}) \rightarrow \boldsymbol{N}\,(\boldsymbol{G}\,\texttt{get-help}).$$

When writing specifications of the properties required of agent-based systems in particular, it helps to use the same kinds of abstraction that we use to build the system itself (such as those mentioned above, particularly beliefs and goals).

Verifying Logical Descriptions — Analysing agent systems in order to make sure they will run as expected is a complex task. However, if we can provide both an abstract description of the agent system in question, and a logical description of the requirements/properties to be checked, then we can carry out logical verification, using a variety of techniques.

One such technique is Model Checking [7], and one of the reasons for this technique being particularly popular is that it can, potentially, make the whole verification process completely automated. This led to the development of very sophisticated model checkers such as SPIN [13] and NASA's Java Pathfinder [24, 10].

Verifying Multiple Agents — This form of verification (by model checking), which checks all possible behaviours of an agent situated in an environment, extends naturally to multi-agent scenarios, as seen in Figure 2.

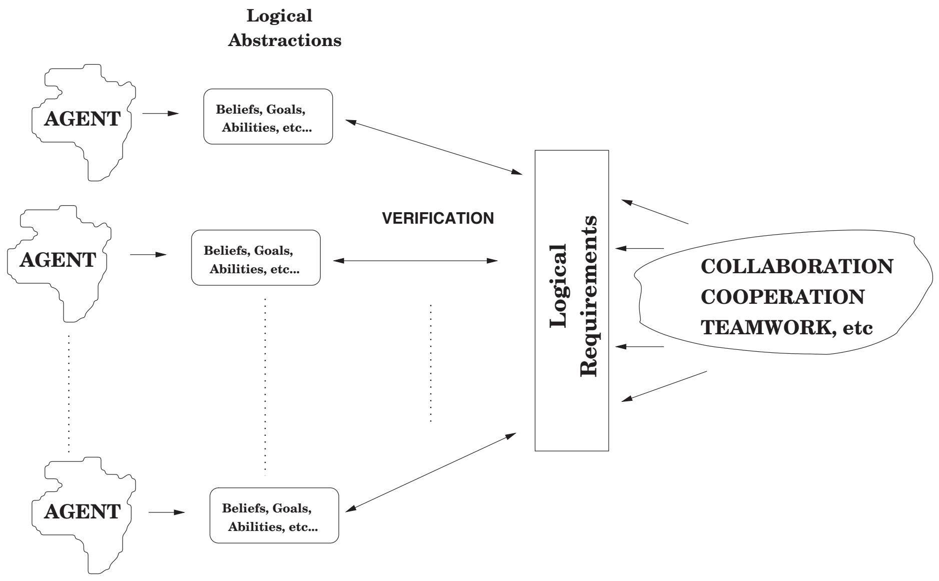

Figure 2: Verifying Systems of Multiple Rational Agents

As seen in the figure, we can combine the abstract representation of the various agents in the system and verify properties that refer not only to the properties that we require of each individual rational agent, but also to the properties that we require of the whole agent team. In particular, we can refer to its collaborative operation. Our particular approach to multi-agent verification is an instance of this general scheme.

Our Approach to Multi-Agent Verification — Our approach to verification of agent-based systems was first introduced in [1] (see also [2] for an overview of the approach). The idea is to do verification of agent programs directly, rather than a high-level design of the system. In particular, we have produced tools that can translate an agent-oriented programming language into the input language of the SPIN model checker as well as Java, thus allowing us to use NASA's Java Pathfinder as target model checker. We also defined a (relatively simple) logical language that allows us to write — using the types of abstractions typical of rational agents (such as beliefs, goals, etc.) — specifications of the properties the system is expected to satisfy. With this, the model checker can automatically verify whether the system satisfy the required properties or not.

Examples of properties we can verify are as follows. Imagine a scenario in which an astronaut is interacting with a robot on Mars. The robot is given some tasks for the day by the ground team, but the astronaut can interrupt the robot at any time and give alternative tasks. The original plan was for the robot to take panorama pictures at locations `l1` and `l2`, but the astronaut tells the robot to take a panorama at location `l3` instead. We may want to make sure that the robot will interrupt its original task as soon as the new course of action is suggested by the astronaut, that the requested panorama will be taken exactly at `l3` (i.e., the right location), that the original tasks are resumed once the astronaut instructs the robot to do that, that panoramas of locations `l1`, `l2`, and `l3` will all be eventually available, and so forth.

HUMAN-AGENT TEAMS

Human-agent teams refer to a complex human-agent work system in which people interact not only with each other, but also with software and hardware systems coordinated by rational software agents. This creates extra constraints, because even though software agents interact according to well-defined communication protocols, people do not. In this section we first discuss the reason why human-agent teams are relevant for space exploration. We then briefly describe the Brahms environment, developed at NASA Ames Research Center. Brahms was specifically developed for modelling human-machine interaction and work practice. Finally, we describe a typical human-agent teamwork scenario.

Why Human-Robot Teams in Space?

Although completely autonomous space missions (i.e., missions consisting solely of autonomous agents) are possible, they present a number of problems. One is that autonomous programs are difficult to control and analyse. An equally important one is that, for critical missions, human decision making is still essential at some level. Thus, there is a move towards combining the advantages of humans and agents into *human-agent teams*, specifically for planetary exploration. Crucially, human-agent (specifically, human-robot) teams are expected to carry out collaborative activity.

However, while such teams are appealing, in principle, several problems remain. The most significant are that such teams are difficult to program and analyse,

- individually, e.g. agents understanding what humans will do, and
- globally, e.g. programming and analysing teamwork and joint goals.

In spite of this, the future of space exploration (in particular planetary exploration) is likely to involve such human-agent teams. For example, NASA is planning to use human-robot teams in Mars exploration. Closer to home, we can see that the future of ubiquitous/pervasive computing essentially involves cooperation and collaboration amongst human-agent teams.

Brahms: Modelling Human-Agent Teams

Brahms is a multi-agent rule-based language developed at NYNEX (New York and New England Baby Bell Telephone Company, now Verizon), at the Institute for Research on Learning (IRL), and since 1998 at NASA Ames Research Center. The Brahms environment consists of a language definition, compiler, an integrated development environment (the Composer) and a Brahms Virtual Machine (the BVM) running on top of the Java virtual machine to load and execute Brahms models. Brahms was originally developed as a multi-agent language for modelling and simulating human work practice behaviour in organisations [6]. While Brahms can run in simulation mode, and is still used as a simulation environment [22], we have extended the BVM by allowing agents to run as real-time software agents without a simulation clock and event scheduler to synchronise the agents. This makes Brahms both a simulation and a software agent development environment. With Brahms you can test a multi-agent system by running the system as a simulation. When the system is debugged (using the Brahms AgentViewer), you can "flip the switch" and run the same system as a real-time distributed agent system. We refer to this as *from simulation to implementation*, a software engineering method that uses simulation as a system design and integration test environment.

Brahms agents are BDI-like agents (recall the Belief-Desire-Intention rational agent architecture mentioned earlier). However, Brahms does not use a goal-directed approach, but rather an approach we refer to as activity-based [5, 23]. Brahms agents are both deliberative and reactive. Each Brahms agent has a separate subsumption-based inference engine [3]. Brahms agents execute multiple activities at different levels at the same time. At each belief-event change (creation or changing of beliefs), situated-action rules (i.e., workframes) and production rules (called "thoughtframes") are evaluated at every active activity-level.

Brahms is a modelling language designed to model human activity. Agents, therefore, were developed to represent people. Brahms agents can belong to one or more group, inheriting attributes, initial beliefs, and activities, workframes and thoughtframes from multiple groups (multiple inheritance). This allows the abstraction of agent behaviour into one or more groups. Because Brahms was developed to represent people's activities in real-world context, Brahms also allows the representation of artifacts, data and concepts in the form of classes and objects. Both agents and objects can be located in a model of the world (the geography model) giving agents the ability to detect objects and other agents in the world and have beliefs about objects. Agents can move from one location in the world to another by executing a move activity, simulating the movement of people.

The Brahms modelling approach is based on a method that divides any system to be modelled into a number of more or less interdependent sub-models: the Agent, Object, Geography, Knowledge, Activity and Communication models. The Brahms model development environment — the Composer — supports this model-based approach, and allows the modeller to create groups and agents using a graphical user interface.

Group Hierarchy

The agent model consists of a group hierarchy representing the social, organisational, or functional groups of which agents are members. In the mission operations domain we can represent the mission operation workers according to their functional roles, such as the science team. Members of the science team are responsible for the science deliverables of the mission. They are often world-class scientists in specific domains, such as specialised science instruments that are carried onboard the robot. The science team members are divided into science theme groups that represent the functional roles during the mission, such as the "instrument synergy team", the "science operations team", and the "data analysis and interpretation team". Excerpt 1 shows the definition of some of the groups in Brahms source code (the excerpt shows partial source code: '`...`' means that source code is left out).

Excerpt 1. Partial Agent Model

```
group MyBasegroup memberof BaseGroup {
        attributes:
                public symbol groupMembership;
}

group VictoriaTeam memberof BaseGroup {...}

group ScienceTeam memberof VictoriaTeam, MyBaseGroup {
        location: Building244;

        attributes:
                ...
        initial_beliefs:
        // everyone knows where the rover is at the start of the sim
                (VictoriaRover.location = ShadowEdgeInCraterSN1);

        activities:
                ...
        workframes:
                ...
        thoughtframes:
}

group ScienceOperationsTeam memberof ScienceTeam {...}

agent Agent1 memberof ScienceOperationsTeam {
        initial_beliefs:
                (current.groupMembership = ScienceOperationsTeam);

        intial_facts:
                (current.groupMembership = ScienceOperationsTeam);
}
```

We will go step-by-step through the source code of Excerpt 1 explaining how groups and agents are defined. Note that this excerpt describes the definition of four groups and one agent. Bold font is used to denote keywords of the Brahms language. Every Brahms language element definition is actually placed in a separate source file, but is here shown as if it were part of one source code file for illustration purposes.

The first two groups are `MyBaseGroup` and `VictoriaTeam`. `MyBaseGroup` is a group defined by the modeller and is used to define common features for all groups. It is a non-domain specific "root" of the group hierarchy, used by the modeller to define common group properties. `MyBaseGroup` and `VictoriaTeam` are both members of the group `BaseGroup`, which is the root of all groups and is part of a base library that comes with the Brahms language, with certain predefined standard attributes. Here the `MyBaseGroup` group defines a common attribute for all groups, i.e., the `groupMembership` attribute. The `groupMembership` attribute is used in the model to allow agents to know to what group they belong. The third group that is defined is `ScienceTeam`. The group `ScienceTeam` is a member of two parent groups, `VictoriaTeam` and `MyBase`. This example shows that Brahms supports multiple inheritance for groups and agents. Group inheritance means that the subgroups and/or agents inherit all the elements defined in the parent group. The Brahms compiler will recognise naming conflicts in multiple inheritance and will report these at compile time. At this moment Brahms does not support "late-binding" and thus there are no possible inheritance conflicts at run-time. Next, the group `ScienceOperationsTeam` is defined as a member of the `ScienceTeam` group. Finally, we see the definition of an actual agent. The keyword **`agent`** declares agents, and in this example `Agent1` is an agent that is a member of the `ScienceOperationsTeam` group. Thus, the definition of groups and agents in Excerpt 1 explicitly defines the group hierarchy in Figure 3.

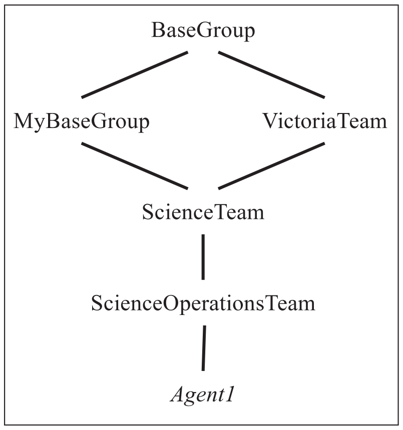

Figure 3: Group Hierarchy from Excerpt 1

Workframes

In Excerpt 2, two workframes are shown: a “high-level” workframe called `wf_SearchForWaterIce` (at the end of the excerpt), and a workframe part of the `FindingWaterIce` activity called `wf_WaitingForData`. Workframes allow for the execution of activities and the representation an agent’s activity execution constraints. Since activities take time, a workframe has a duration based on the time that the activity takes. Workframes “fire” according to a pattern-matching process in which workframe preconditions are tested and workframe variables are bound. The body of a workframe (i.e., the *do*-part) can have conclude statements and activity calls. Conclude statements in workframes are meant to represent the belief-state of the agent in relation to the activity that is going to be executed (i.e., before the activity call) or has finished executing (i.e., after the activity call), and are not meant to represent reasoning of the agent (for this we use the thoughtframes).

One way of thinking about the role of workframes is to view them as constraints on when an agent can perform an activity. Workframe (WFR) `wf_SearchForWaterIce` constrains when the agent can perform the FindingWaterIce activity. The constraints are represented as the preconditions of the workframe. The preconditions encode what beliefs the agent needs to have in its belief-set to enable it to perform the activity or activities (there can be more than one activity call in the workframe body). In plain English `wf_SearchForWaterIce` says: “When I believe that the `VictoriaRover` is currently in the activity `SearchForWaterIce` and I believe that the `VictoriaRover` is currently located in a crater, first bind the name of the crater to the variable `rover-loc`, then execute the workframe body with priority zero” (Brahms allows for parallel execution of workframes, but uses a “time-sharing” approach using priorities). Note also that `wf_SearchForWaterIce` has the `repeat:false` statement at the top. This means that this workframe will only fire once for a particular set of beliefs that match all its preconditions. The result is that the agent will only execute `wf_SearchForWaterIce` once for any crater the `VictoriaRover` visits.

When the agent’s inference engine has determined that the preconditions of `wf_SearchForWaterIce` are satisfied (due to finding matching beliefs in the agents belief-set) and it is the WFR with the highest priority, the agent will start executing the first statement in

Excerpt 2. Partial Activity Model for the ScienceOperationTeam Group

```
composite_activity FindingWaterIce (Crater crater, int pri) {
        priority: pri;

        activities:
                primitive_activity WaitingForData( ) {
                        priority: 0;
                        max_duration: 3600;
                } //end activity
                ...
        workframes:
                workframe wf_WaitingForData {
                        repeat: true;
                        priority: 0;
                        detectables:
                                detectable ReceiveHydrogenData {
                                 detect((VictoriaRover.nextSubActivity = DoDrilling))
                                 then abort;
                                } //end detectable
                                ...
                        when (knownval(current.nextSubActivity = WaitForData))
                        do {
                                WaitingForData( );
                        } //end do
                } //end workframe
                ...
        thoughtframes:
                ...
} //end composite_activity

workframe wf_SearchForWaterIce {
        repeat: false;
        variables:
                foreach(Crater) rover-loc

        when (knownval(VictoriaRover.currentActivity = SearchForWaterIce) and
              knownval(VictoriaRover.location = rover-loc))
        do {
              conclude((current.currentActivity = SearchForWaterIce));
              FindingWaterIce(rover-loc, 0);
        } //end do
} //end workframe
```

the body of the WFR, which in Excerpt 2 is the conclude statement that creates the belief for the agent that says that its current activity is `SearchForWaterIce`. This represents that the agent knows that it is currently in the activity of searching for water ice. Next, the engine calls the activity `FindingWaterIce`. Matching of beliefs preconditions, binding variables and firing the workframe, executing the conclude statement and calling the activity `SearchForWaterIce`, is all done in the same simulation time-event. Thus, although these processes take actual CPU time, they do not take any simulation time for the agent.

An important aspect of making Brahms into a software agent development language is the ability to seamlessly integrate Brahms agents within Java. Brahms has a JAPI defined to write agent activities in Java, so that they can be called from workframes. Brahms agents can also be completely written in Java, which enables the wrapping of existing external systems as a Brahms agent, enabling Brahms agents to communicate with external systems. For a more detailed description of the Brahms language we refer the reader to [21] and [4].

Sample Human-Agent Teamwork Scenario

There follows an outline of a scenario where human-agent (specifically, human-robot) teams can be used. This essentially concerns ensuring robust network connectivity for planetary exploration through the use of multi-agent (and human-agent) teamwork.

Two surface astronauts are going on an extra-vehicular activity (EVA) to explore a region, defined by the crew in an EVA plan. One or more network relays provide connectivity to each astronaut. A network relay can be one of two robots, or a robotically deployed relay device. The robots can be autonomous relays, as well as astronaut assistants, carrying tools and sample bags, taking pictures and panoramas. Each astronaut can "team up" with a robot. When a robot is teamed up with an astronaut, the robot's personal agent automatically performs the "following" and "watching" activities, and also automatically begins to monitor network connectivity to the astronaut (the astronaut's personal agent automatically begins to monitor connectivity to the robot). The personal agents also monitor network connectivity back to the habitat. The personal agent notifies the astronaut when network connectivity fails. Using a defined model of teamwork the robot and astronaut personal agents will provide simple recovery steps to try to reestablish communication.

VERIFYING HUMAN AGENT TEAMS

Semantics of Brahms Descriptions

The Brahms language is organised around the following representational constructs:

- Groups of groups containing
  - Agents who are located and have
    - Beliefs that lead them to engage in
      - Activities specified by
      - Workframes
- Workframes in turn consist of
  - Preconditions of beliefs that lead to
    - Actions, consisting of
      - Communication Actions
      - Movement actions
      - Primitive Actions
      - Other composite activities
    - Consequences of new beliefs and facts
    - Thoughtframes that consist of

Preconditions and
Consequences

Using Verification Techniques

Because of the success of the previous work in verifying AgentSpeak agents and multi-agent systems (written using AgentSpeak(F), a restricted version of AgentSpeak), and because Brahms, like AgentSpeak, extends the Belief-Desire-Intention notions of rational agency, it would seem reasonable to follow one or both of these approaches:

1. converting Brahms models into AgentSpeak models or
2. duplicating, with Brahms, as far as possible, the techniques used to make AgentSpeak verifiable.

The advantage of (1) is that it builds on existing work, and because it would involve translating one (essentially) BDI-based language into another, the task appears simpler. It would, however, require finding areas of common linguistic (syntactic/ontological) ground, which in turn could involve the simplification or re-formatting of the Brahms language, a step that might itself require significant formalisation and proof or verification.

A much larger task would be to convert Brahms models directly into a model checker's input language (e.g., Promela for Spin, Java for JPF). The complexity here centres on the fact that these input languages do not have BDI notions as part of their foundation and, for Spin, on the limitations of the Promela language. As this has been successfully done with programs written in AgentSpeak, such work might be used as an inspiration for doing similar translations with Brahms descriptions. Brahms, however, is more complex syntactically than AgentSpeak and, again, the translation process might require altering Brahms descriptions and would require a formal translation process (to ensure semantic equivalence between two quite different languages).

A pictorial view, following the previous description of multi-agent verification, of such human-agent teams is given in Figure 4.

Note. There is perhaps a subtle difference in the rationale behind using Brahms and using other agent languages. Brahms is fundamentally an application for modelling work systems, i.e., environments which include agents, objects, locations and other concepts and in which agents will typically collaborate. An agent language tends to focus more on the development and behaviour of the agents themselves, which may include competitiveness and may lack a framework of shared achievement.

In translating, the fundamental aim is to replicate the semantics of the source in the target language, and formal approaches often exist to make this as unambiguous as possible. In such a situation, the logic of the Brahms model would need to be described and this description would act as a pre-programming formal specification for the target language. Ideally, formal specifications will exist that were used in building the Brahms scenario in the first place and these may be a starting point for describing the underlying logic. As Brahms is modelling task-oriented behaviour

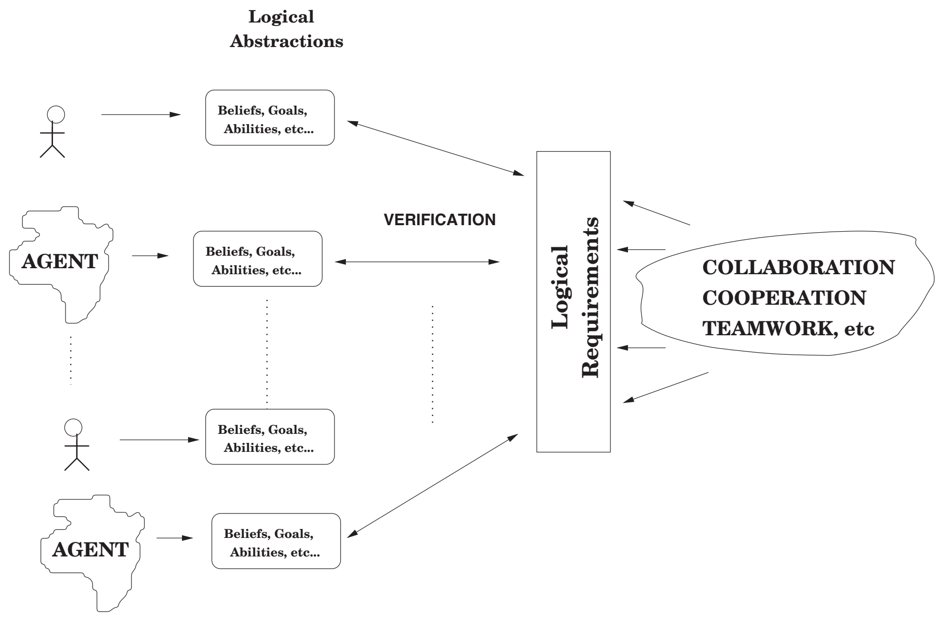
Logical Abstractions
Beliefs, Goals, Abilities, etc...
AGENT
Beliefs, Goals, Abilities, etc...
VERIFICATION
Beliefs, Goals, Abilities, etc...
AGENT
Beliefs, Goals, Abilities, etc...
Logical Requirements
COLLABORATION
COOPERATION
TEAMWORK, etc

Figure 4: Verifying Teams of Humans and Agents

in a dynamic, collaborative environment, many of the range of logics listed in the "Background" section may be applicable. In particular, BDI and temporal logics can be used to describe the agents' mental states and associated activities. In Brahms, plans are represented as workframes, for example:

```
workframe wf_moveToRestaurant {
    repeat: true;
    variables:
        forone(Diner) dn;
        forone(Building) bd;

        when(knownval(current.howHungry > 20.00) and
        knownval(current.chosenDiner = dn) and
        not(current.location = dn.location) and
        knownval(dn.location = bd ))
        do {
        moveToLocation(bd);
        conclude((current.readyToLeaveRestaurant = false),
                 bc:100, fc:0);
        }
}
```

The `when` descriptor effectively describes the belief context and `do` section shows the actions taken. It also includes a `conclude` which describes the state at the end of the activity. It can be seen that a temporal logic formula can be produced to describe this workframe in the broad sense. For greater conformity to the AgentSpeak model, how desires and intentions are differentiated in Brahms and how plan and option generation and selection are performed, will need to be considered.

A further option is to examine the interpretation cycle of an AgentSpeak agent (e.g., given in the documentation for *Jason*, an interpreter for an extended version of AgentSpeak [9]), which is closely representative of a practical reasoning system, and again to compare that structure with a Brahms agent and to assess how the latter would need to be altered to give it some sort of equivalence to the AgentSpeak agent cycle. The cycle describes the process of perceiving the environment, updating beliefs, retrieving plans according to perceived events, selecting plans as intended means, and taking action. These are all actions that are present in the Brahms language semantics, but obviously in a different form and perhaps with different characteristics.

CONCLUDING REMARKS

Our approach to the formal verification of human-robot (and, in general, human-agent) teams consists of two parts:

1. the formalisation of team activity, using the Brahms framework, and
2. the use of our agent verification tools to analyse the team formalisations.

In particular, our aim is to analyse whether our approach to verification of BDI (Belief-Desire-Intention) agents, for example through the verification of AgentSpeak programs, can capture (simplified) Brahms descriptions. One direction is to consider the syntax of both AgentSpeak and Brahms in order to assess whether, or how, they can be matched. For example, consider the following excerpt from the grammars of both languages:

AgentSpeak　　　　**Brahms**

```
ag ::= bs ps        agent   ::=  agent
                                 agent-name {GRP.group-membership}
                                 {
                                  {GRP. attributes}
                                  {GRP. relations}
                                  {GRP. initial-beliefs}
                                  {GRP. initial-facts}
                                  {GRP. activities}
                                  {GRP. workframes}
                                  {GRP. thoughtframes}
                                 }
```

Comparing the languages semantically reveals conceptual differences in the way that models are constructed and may provide a basis for determining how, should it be necessary, Brahms descriptions can be simplified or re-formatted (inside or outside of Brahms) for the purposes of verification specifically.

In this paper, we described our approach to verifying Belief-Desire-Intention agent-based (space) applications using sophisticated model checkers, specifically by means of our model-checking techniques for the high-level AgentSpeak language. If successful, this provides a mechanism for verifying (at least part of) Brahms team activities.

There remain, of course, problems. Specifically, there is likely to be a complexity boundary separating viable agent verification from inviable ones. There also remain the problems of comprehensively (and accurately) describing team activity and representing the environment appropriately. Finally, there must be an appropriate logic language available for specifying the properties to be checked, which again can be inspired by our existing work on model checking for AgentSpeak.

Our aim is to develop this approach further, and to use realistic scenarios taken from NASA examples, in which it is essential to make sure that human-agent teams can always recover from error situations/problems. Also of interest are scenarios involving human intervention/override, such as advisory agents and semi-autonomous spacecraft docking.

REFERENCES

[1] R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking AgentSpeak. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2003), Melbourne, Australia, 14–18 July*, 2003.

[2] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Model checking rational agents. *IEEE Intelligent Systems*, 19(5):46–52, September/October 2004.

[3] Brooks, R., A. Intelligence without representation. *Artificial Intelligence*, 47:139-159. 1991

[4] Brahms Website: `http://www.agentisolutions.com`, Ron van Hoof, 2000.

[5] W. J. Clancey. Simulating Activities: Relating Motives, Deliberation, and Attentive Coordination. *Cognitive Systems Research* 3(3):471-499. 2002.

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

[7] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, December 1999.

[8] DART. `http://www.nasa.gov/mission_pages/dart/main/`.

[9] *Jason*. `http://jason.sf.net`.

[10] Java PathFinder. `http://javapathfinder.sf.net`.

[11] K. Havelund, M. Lowry, S. Park, C. Pecheur, J. Penix, W. Visser, and J.L. White. Formal Analysis of the Remote Agent Before and After Flight. In *5th Langley Formal Methods Workshop*, June 2000.

[12] W. van der Hoek, B. van Linder, and J-J. Meyer. A Logic of Capabilities. Rapport IR-330, Vrije Universiteit, Amsterdam, July 1993.

[13] G. J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, November 2003.

[14] B. van Linder. *Modal Logics for Rational Agents*. PhD thesis, Universiteit Utrecht, 1996.

[15] D. V. Pynadath and M.Tambe. Automated teamwork among heterogeneous software agents and humans. *Journal of Autonomous Agents and Multi-Agent Systems*, 7:71–100, 2003.

[16] A. S. Rao and M. P. Georgeff. Modeling Rational Agents within a BDI-Architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proc. of the Second International Conference (KR'91)*, pages 473–484. Kaufmann, San Mateo, CA, 1991.

[17] A. S. Rao and M. Georgeff. BDI Agents: from theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, San Francisco, CA, June 1995.

[18] Remote Agent. `http://citeseer.ist.psu.edu/bernard98design.html`.

[19] N. Schurr, J. Marecki, P. Scerri, J.P. Lewis, and M. Tambe. The defacto system: Coordinating human-agent teams for the future of disaster response. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, chapter 8. Springer-Verlag, 2005.

[20] M. Sierhuis. Multiagent Modeling and Simulation in Human-Robot Mission Operations. (See `http://ic.arc.nasa.gov/ic/publications`).

[21] M. Sierhuis. *Modeling and Simulating Work Practice. BRAHMS: a multiagent modeling and simulation language for work system analysis and design.* PhD thesis, Social Science and Informatics (SWI), University of Amsterdam, SIKS Dissertation Series No. 2001-10, Amsterdam, The Netherlands, 2001.

[22] M. Sierhuis and W. J. Clancey. Modeling and Simulating Work Practice: A human-centered method for work systems design. *IEEE Intelligent Systems* 17(5) (Special Issue on Human-Centered Computing, 2002).

[23] L. A. Suchman. *Plans and Situated Action: The Problem of Human Machine Communication*. Cambridge, MA, Cambridge University Press, 1987.

[24] W. Visser, K. Havelund, G. Brat, and S. Park. Model checking programs. In *International Conference on Automated Software Engineering (ASE)*, September 2000.

[25] M. Wooldridge. *An Introduction to Multiagent Systems.* John Wiley & Sons, 2002.

[26] M. Wooldridge and N. R. Jennings. Agent theories, architectures, and languages: A survey. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents*. Springer-Verlag, 1995.