Verifying Brahms Human-Robot Teamwork Models

Richard Stocker Louise Dennis Clare Dixon Michael Fisher

Department of Computer Science, University of Liverpool, U.K.,
contact: R.S.Stocker@liverpool.ac.uk

Abstract. Collaboration between robots and humans is an increasingly important aspect of industrial and scientific settings. In addition, significant effort is being put into the development of robot helpers for more general use in the workplace, at home, and in health-care environments. However, before such robots can be fully utilised, a comprehensive analysis of their safety is necessary. Formal verification techniques are regularly used to exhaustively assess system behaviour. Our aim is to apply such techniques to Brahms, a human-agent-robot modelling language. We show how to translate from Brahms scenarios, using a formal semantics for Brahms, into the input language of a model checker. We illustrate the approach by defining, translating, and verifying a domestic robot helper example.

1 Introduction

As autonomous devices are increasingly being developed for, and deployed in, both domestic and industrial scenarios, there is an increasing requirement for humans to at least interact with, and often work cooperatively with, such devices. While the autonomous devices in use at present are just simple sensors or embedded hardware, a much wider range of systems are being developed. These consist not only of devices performing solo tasks, such as the automated vacuum cleaners we see already, but are likely to include robots working cooperatively with humans. For example, there will be robot 'helpers' to assist the elderly and incapacitated in their homes [1, 2], there will be manufacturing robots which will help humans to make complex artifacts [3], and there will be robots tasked with ensuring that humans working in dangerous areas remain safe. All these are being developed, many will be with us in the next ten years, and all involve varying degrees of cooperation and teamwork.

The above examples highlight robots deployed in both domestic and safety-critical industrial situations where human safety can be compromised. Thus, it is vital to carry out as much analysis as is possible not only to maximize the safety of the humans involved, but to ascertain whether the humans and robots together 'can', 'should', or 'will' achieve the goals required of the team activity.

There are several *challenges* facing such analysis. One is to to accurately describe human, and indeed robot, behaviour. Even when we have described such behaviours, how can we exhaustively assess the possible interactions between the humans and robots? While some work has been carried out on the safety analysis of low-level human-robot interactions [4], a detailed analysis of the *high-level* behaviours within such systems has not yet been achieved.

In this paper, we tackle the general problem of matching a set of requirements (which could concern safety, capabilities, or interactions) against scenarios involving humans, robots, and agents. Within this, we use important work on high-level modelling of human-agent-robot teamwork that has already been carried out using the Brahms framework [5]. Thus, we assume that the key interactions and behaviours of any human-agent-robot scenario have been captured within a Brahms model. We also assume that a set of informal requirements have been constructed. The work described in this paper essentially describes the *solid* arrows within Fig. 1.

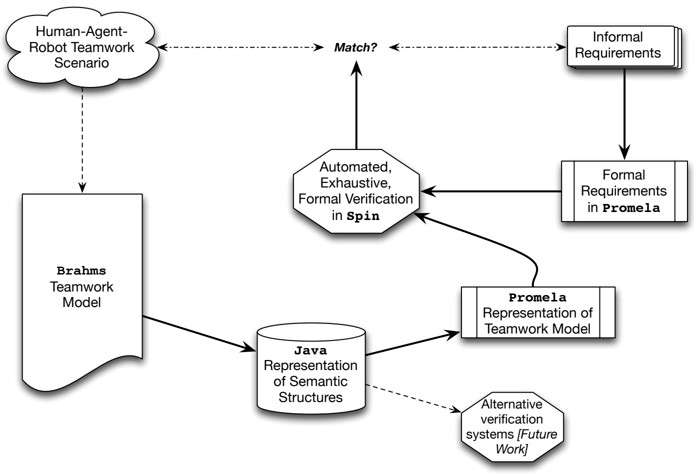
Human-Agent-Robot Teamwork Scenario
Match?
Informal Requirements
Automated, Exhaustive, Formal Verification in Spin
Formal Requirements in Promela
Brahms Teamwork Model
Promela Representation of Teamwork Model
Java Representation of Semantic Structures
Alternative verification systems [Future Work]

Fig. 1. Overview of the processes described in this paper.

Thus, given a Brahms model of a human-agent-robot scenario, we use the formal semantics described in [6] to generate a `Java` representation of the semantic structures relevant to this scenario. We then translate these structures into `Promela` process descriptions, which represent partial instantiations of the semantics, suitable for input to the `Spin` model checker [7]. In parallel, we translate the informal requirements (where possible) to `Promela` code representing these properties. Finally, we apply the `Spin` model checker to the `Promela` descriptions and feed the results back to the high-level scenarios for evaluation. This, then, provides a mechanism for formally verifying properties of human-agent-robot teamwork scenarios.

Our paper describes this framework and exhibits its use on a specific domestic scenario, where a helper robot and a house agent work together with a person to to monitor, remind and assist a person with their daily activities in a home environment.

2 Background

2.1 Brahms

Brahms is a multi-agent modelling, simulation and development environment devised by Sierhuis [5] and subsequently developed at NASA Ames Research Center. Brahms is designed to model both human and robotic activity using *rational agents*. Rational agents can be seen as autonomous entities, able to make their own choices and carry out actions in a *rational* and *explainable* way [8]. As Brahms was developed in order to represent people's activities in real-world contexts, it 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 giving agents the ability to detect both objects and other agents, have beliefs about the objects and move between locations. For a more detailed description of the Brahms language we refer the reader to [5] and [9]. The *key aspects* of Brahms are:

- *activities*: actions (with durations) an agent can perform;
- *beliefs*: each agent's own personal perceptions of itself, the environment and other agents;
- *facts*: the actual state of the agents and the environment (which agents/objects observe using *detectables*);
- *detectables*: allowing facts to be brought into the an agent's belief base and determining how the agent will react in response;
- *workframes*: sequences of events required to complete a task, together with any belief updates resulting from the task completion;
- *thoughtframes*: reasoning/thought processes, e.g. "I believe it is raining therefore I believe I need an umbrella";
- *time*: Brahms models incorporate a time-line of events and belief changes.

Brahms has been judged ideal for describing human-agent-robot teamwork, for example astronaut-robot teamwork on Mars [10].

A Brahms simulation contains a set of agents (representing robots, humans or actual agents) and a scheduling system which manages a clock recording global time within the simulation. Since agent actions have durations, the scheduler will examine each agent to see how much longer any action the agent is performing will take and then advance the clock to the next significant point in time, typically when the agent action finishes. When two agents finish actions at the same time (and at the start of the simulation) the scheduler also manages the order in which the agents execute their reasoning processes in order to determine their next action. In such cases the agents have a pre-determined order of priority within the scheduler.

2.2 Brahms Formal Semantics

In [6], we provided a formal *operational semantics* for Brahms. This provides the theoretical basis for our verification. A Brahms model is represented as a 5-tuple:

$$\langle Ags, ag_i, B_\xi, F, T_\xi \rangle$$

Where Ags is the set of all agents, ag_i the agent currently under consideration, B_ξ the belief base of the system (used to synchronise the agents, e.g. agent i's next event finishes in 1000 seconds), F the set of facts in the environment (e.g. temperature is $20°C$) and T_ξ is the current time of the system.

The agents (Ags, and ag_i) have a 9-tuple representation:

$$\langle ag_i, \mathcal{T}, W, stage, B, F, T, \mathit{TF}, \mathit{WF}\rangle$$

Here ag_i is the identification of the agent; $\mathcal{T}$ the current thoughtframe; W the current workframe; $stage$ the current stage of the agent's reasoning cycle; B the agent's beliefs; F the set of facts about the world; T the agent's internal time; TF the agent's thoughtframes; and WF is the agent's set of workframes. In addition, $stage$ controls which rules in the operational semantics are currently applicable to the agent or if the agent is in a finish (fin) or idle ($idle$) stage. The (operational) semantics is then represented as a set of transition rules of the form

$$\langle \mathit{StartingTuple}\rangle \xrightarrow[ConditionsRequiredForActions]{ActionsPerformed} \langle \mathit{ResultingTuple}\rangle$$

Here, '$\mathit{ConditionsRequiredForActions}$' refers to conditions which must hold before the rule can be applied, while '$\mathit{ActionsPerformed}$' represents changes to the agent, object or system state which, for presentational reasons, can not be easily represented in the resulting tuple. Finally, it is assumed that all agents and objects can see and access everything in the overall system's tuple, e.g. T_ξ.

The semantics is split into two groups of rules: the first group concerns the global system and represents the functioning of the scheduler; the other acts upon individual agents. Rules for the scheduler act as a global arbiters instructing agents when to start, suspend, or terminate. Rules for the individual agents choose actions and update beliefs, etc. An agent first processes *thoughtframes*, then *detectables* (both of which may update the beliefs), and then *workframes* which may initiate actions, referred to as *activities*.

For example, there are rules informing an agent on how to select a thoughtframe based on whether its beliefs match the thoughtframe guard conditions and whether the thoughtframe's priority is sufficiently high. The rules governing activities communicate with the system to inform it of the activity's duration. When no agent can apply any more operational rules, control returns to the scheduler which examines all the agents' activities to determine which will conclude first and at what time it will finish. The scheduler then moves the global clock forward accordingly, and hands control to the rules governing the behaviour of the individual agents once more.

2.3 Formal Verification, `Promela` and `SPIN`

Formal verification represents a family of techniques aimed at assessing whether a system always/ever satisfies its specification. We consider a fully automated, algorithmic technique known as *model checking* [11]. A model checker takes a description of the system together with some requirement expressed in a formal logic. The model checker exhaustively checks the formal requirement against *all* paths through the system. If a path is found in which the property does not hold then a trace of that path is provided.

In this paper we use the `Spin` model checker [7]. `Promela` (Process/Protocol Meta Language) is the input language for `Spin`. `Promela` was designed to be a simple multi-process language, allowing the models generated to be small. Processes are a key part of `Promela`. They are asynchronous and are declared by the key word `proctype`. `Promela` provides three basic control flow constructs: case selection; repetition; and unconditional jump.

`Spin` itself is an on-the-fly reachability analysis system [7]. It accepts specifications in the form of *linear temporal logic* properties, which are translated into *Büchi automata* — finite automata over infinite input sequences. `Spin` then examines all possible runs through the `Promela` program, running the *Büchi automaton* in parallel in order to assess whether the temporal requirements are satisfied.

3 Case Study: "Domestic Home Care"

We will describe our translation and verification procedure through the development of one specific example; further details are available in the extended technical report [12]. We first describe the, necessarily very simple, scenario and outline its Brahms implementation. Though we can only provide an English overview here, we provide a sample Brahms workframe in Fig. 2; the full Brahms implementation is provided in [12].

3.1 Overview of Scenario

In this scenario there is a person, a helper robot, a human care worker and a house agent. The helper robot is mobile and can move about the house assisting the person with various tasks. The house agent has the role of detecting information, informing the person and issuing reminders where necessary. The care worker is called when the robot/agent are unable to assist. Such domestic health-care scenarios typically involve assisting the elderly or infirm; see for example [1, 2].

The helper robot: fetches drinks, cooks food, and delivers meals to the person; collects dirty dishes and puts them in the dishwasher; fetches medicines; records whether the person has taken these; informs the person of what to do in case of an emergency, e.g. a fire; and communicates with the house agent. The house agent: informs the helper robot of the person's location; issues reminders to the person (e.g. to flush the toilet); and monitors the person's location. The care worker is called for when the person fails to take their medication. We assume the care worker is always successful in administering the medication. The person is modelled very simply, watching TV, requesting food, eating and going to the toilet at regular intervals.

3.2 Brahms Representation

This scenario is modelled using five agents and one object: *Robot*, *House*, *Care-Worker*, *Environment* and *Bob* (our elderly person) are the agents and *Clock* is the object. The *Clock* is used for termination of the simulation (i.e. after 20 hours) and provides the notion of time used by the simulation e.g. governing when the human's hunger increases. The *Environment* is a simple agent that decides if, and when, a fire alarm will occur.

Bob's role is to watch television and perform simple everyday tasks such as eating and going to the toilet. Thoughtframes are used to update beliefs about how hungry he is and how much he needs the toilet. When his hunger reaches a certain threshold a workframe activates and *Bob* requests food. A similar workframe will trigger a visit to the toilet. These workframes have a higher priority than the workframe for watching television, so when they become active the 'television' workframe *suspends*. The workframe for going to the toilet activates other workframes to flush the toilet and wash his hands once finished. Two versions of these workframes exist: representing whether or not he remembers to perform the task, each have the same guard conditions and priority so only one will execute at random. *Bob* also has workframes for taking his medication and thoughtframes that govern whether or not he chooses to do so.

The helper *Robot* remains idle until it receives a command or it detects *Bob* requires attention. When *Bob* requests food, the *Robot* prepares and delivers it. There is a detectable in the *Robot's* "wait for instructions" workframe which detects when *Bob* has finished eating; this triggers a belief update which in turn triggers a workframe to clear the plates. The *Robot* also has workframes to deliver medicine to *Bob*; activated at pre-allocated times. The *Robot* places the drugs on *Bob's* tray and then monitors them hourly to check if they have been taken. The workframe governing this is shown in Fig. 2. A detectable `takenMedicationC` aborts the workframe if the drugs have been taken and then updates the *Robot's* beliefs. If the drugs have not been taken the workframe reminds *Bob* to take his medication. The *Robot* counts the number of times it reminds *Bob*, and after 2 reminders it notifies the *House*. The *Robot* also instructs *Bob*to evacuate the house in the case of fire and answers the door to the *Care Worker*.

```
workframe wf_checkMedicationC {
   repeat: true;
   priority: 3;
   detectables:
      detectable takenMedicationC{
         when(whenever)
         detect((Bob.hasMedicationC = false),
          dc:100)
         then abort; }
   when(knownval(current.perceivedtime > 14)and
      knownval(Bob.hasMedicationC = true) and
      knownval(current.checkMedicationC = true))
   do {
      checkMedication();
      remindMedicationC();
      conclude((current.checkMedicationC = false));
      conclude((current.missedMedicationC =
       current.missedMedicationC + 1)); }}
```

Fig. 2. The *Robot's* workframe to remind *Bob*about medication

The *House* is 'intelligent'. It is responsible for monitoring *Bob*, giving him instructions based on his location, and detecting any fire. The *House's* default workframe monitors *Bob*, and has detectables which update the *House's* beliefs about *Bob's* location. When *Bob's* location is at the toilet a new workframe is fired, containing an 'abort' detectable which quits the activity when *Bob* leaves the toilet and activates a new workframe which detects *Bob's* location and uses this to decide whether or not *Bob* has left without flushing the toilet. *Bob*is then reminded if necessary. The default monitoring workframe also has a detectable for a fire, this aborts the current activity and activates a workframe which sounds an alarm and notifies the *Robot* and *Bob*. Finally, while the *House* is notified that *Bob* has failed to take his medicine, it informs the *Care Worker*.

The *Care Worker* performs outside activities which are abstracted into a single "busy" activity. When the *Care Worker* is called he/she will only make their way to the house once they have finished their current activity. When the *Care Worker* arrives they ring the door bell. Once they have been let in by the *Robot* they administer the medication and inform the *Rpbpt* that the patient has now taken the medication. The *Care Worker* then leaves and continues with their outside activities.

Note that each of the agent's behaviours are here deliberately chosen to to be simple. We can, of course, add much more complex behaviour though our aim here is just to use this scenario to exhibit the overall approach.

4 From Brahms to `Promela`

We automatically build a `Promela` version of the Brahms scenario. In practice we translate Brahms into `Java` data structures corresponding to the semantic configurations (i.e., the various tuples mentioned in section 2.2) [6]. This is to allow us to (later) target several different model-checkers from the same intermediate representation. Here, however, we just discuss the final `Promela` code and do not detail the intermediate `Java` representation. `Promela`'s restrictive data types and control structures make it difficult to model the operational semantics for Brahms directly. Agents, workframes, thoughtframes and the tuples representing the system model all have to be represented via arrays. This makes it complex to write generic code that will apply to *any* model. As such we choose to generate a *partial instantiation* of the operational rules tailored for a particular model of interest. This partial instantiation is generated automatically from the `Java` representation.

4.1 From the Scheduler to a Promela Process

Representing the Scheduler in `Promela`. Given a specific model, we generate partial instantiations of the scheduler rules which act, not on a list of unknown agents, Ags, but upon the specific agents we know to exist in the model. The only variables used by the scheduler are an integer to represent its current time, '`cntEnvionment`'; an enumeration, '`turn`', which can be either an object/agent's name or the Environment; and a Boolean, '`EnvironmentActive`', which decides when the system is to terminate.

The `Promela` translation imitates the Brahms system scheduler by representing it as a `proctype`. The global clock is represented by an integer. Agents are also represented using `proctypes` and the scheduler determines their order of execution through '`turn`'. Once an agent has executed, '`turn`' is re-assigned to the scheduler.

Matching the Scheduler's Rules. The `Promela` code captures all the scheduler rules in a loop containing a conditional expression with one condition representing the guard for each rule. If the condition evaluates to true then code representing the rule's semantics is executed. We give an example of one of the scheduler rules, Sch_run, and discuss its instantiation as `Promela` code.

RULE: Sch_run

$$\langle Ags, ag_i, B_\xi, F, T_\xi \rangle \xrightarrow[\forall ag \in Ags | stage_{ag} \in \{fin, idle\}, (T_\xi \neq -1)]{stage_{ag_i} = Set\_Act} \langle Ags, ag_i, B_\xi, F, T_\xi \rangle$$

Sch_run becomes active if all the agents are either finished (in the *fin* stage) or idle (the *idle* stage) and the simulation hasn't finished ($T_\xi \neq -1$). In `Promela`:

– a set of Boolean variables represent when agents are *idle* (e.g., '`RobotActive`') is set to *false* if the *Robot* is *idle*); '
– a set of integers representing the time remaining for each agent's current activity are used to judge whether an agent is in the *fin* stage. (e.g., if '`Robot_timeRemaining`' is zero then the *Robot* is in the *fin* stage); and
– `Promela` will terminate if the simulation has concluded so it isn't necessary to check explicitly for $T_\xi = -1$.

The condition representing the rule's guard checks all these variables ('`RobotActive`', '`Robot_timeRemaining`' etc.) explicitly. In the generic rule, Sch_run sets the *stage* of ag_i to Set_Act. In `Promela` the value of the agent's enumeration '`turn`' represents the agent's stage and this is set accordingly.

4.2 From Agent Semantics to Promela Processes

Representing the Agent's Data Structures in `Promela`. The components of the 9-tuple that represent an agent are primarily represented by arrays. These arrays are referred to by name in the partial instantiations of the operational rules.

For instance, $\mathcal{T}$, the agent's current thoughtframe is represented as a one-dimensional array and treated as a stack. The array is labelled '`tf_stack`' followed by the agent's name e.g. '`tf_stackRobot`'. The current workframe is represented in a similar fashion. The first six indices (three in the case of the current thoughtframe) of the array (elements 0-5) are used to store the workframe header data. Below the header information are a stack of *deeds* which may represent belief updates or activities. Sets of thoughtframes and workframes are stored in the same format but in two-dimensional arrays where the first index represents the thoughtframe or workframe and the second represents the elements of the thoughtframe or workframe. These are named '`tf`' or '`wf`' followed by the name of the agent. e.g. an agent *Robot* may have a set of workframes as follows:

Index	Workframe at index 0	index 1
0	Workframe ID number = 0	ID = 1
1	Boolean guard condition = 1 (workframe is active)	Guard = 0
2	Priority of the workframe = 4	Priority = 10
3	Repeat = 3 (always repeat)	Repeat = 0 (never repeat)
4	Boolean to flag a communication or move activity = 0	Comm/Move = 0
5	Boolean to flag if workframe is in impasse = 0	impasse = 1
6 . .	Last deed on stack . .	Last deed on stack . .
i	Top deed on stack	Top deed on stack

We do not represent the current stage of the agent's reasoning cycle explicitly, but do so implicitly by the order in which rules are represented in the `Promela` code.

Beliefs and facts in Brahms are tied to the `attributes` and `relations` of an agent; where attributes are defined properties of agents and relations are connections between agents. So agent `Robot` could believe agent `Bob's` attribute `AskedForFood` is true or that `Bob` has the relation of `isPatientOf` with the `Carer`. To model this in `Promela` every agent is assigned a belief about every `attibute` and `relation`, even if it does not own that attribute. This belief is represented as a Boolean array. The name of the belief is the name of the agent followed by the name of the attribute, e.g. `RobotAskedForFood` represents the `Robot's` beliefs about the attribute `AskedForFood`. The index of the array is the ID number of the agent whom the belief concerns, e.g.

0 = Robot's ID	*Robot* believes the Robot askedForFood = false
1 = Clock's ID	*Robot* believes the Clock askedForFood = false
2 = Bob's ID	*Robot* believes that *Bob* AskedForFood = true
3 = House's ID	Robot believes the House AskedForFood = false

Matching the Agent's Semantic Rules in `Promela`. When the scheduler's '`turn`' enumeration is an agent name then control passes to the agent rules. Like the scheduler rules these are represented by a loop that checks the rule pre-conditions in turn. To explain how the `Promela` translation matches Brahms we show how one of the operational semantic rules is represented in `Promela`.

RULE: *Wf_Select*

$$\langle ag_i, \emptyset, \emptyset, \, Wf\_{*}, B_i, F, T_i, \, TF_i, \, WF_i \rangle$$
$$\xrightarrow[\exists W \in WF_i | B_i \models W^g]{\beta = Max_{pri}(W \in WF_i | B \models W^g)}$$
$$\langle ag_i, \emptyset, \beta, \, Wf\_(true/false/once), B_i, F, T_i, \, TF_i, \, WF_i \rangle$$

Wf_Select determines which workframe is to be selected. For the rule to be activated there needs to exist a workframe in the set of workframes whose guard conditions evaluates to true with respect to the belief base ($\exists W \in WF_i | B_i \models W^g$). At the start of each cycle the agent first identifies which workframes have guard conditions that evaluate to true: those which are active have a 1 entered at index 1 in the 2-dimensional array of workframes above, those that are not have a 0. The rule also states that the "current workframe" entry in the tuple must be empty, which is represented in `Promela` by a pointer to the current workframe's top element. If this is -1 then there is no current workframe. If the current workframe is empty and some workframe is active then Wf_Select will be selected.

Wf_Select performs a selection process to find the active workframe with the highest priority ($\beta = Max_{pri}(W \in WF_i | B \models W^g)$). The `Promela` translation loops through the array of workframes, checks the guard condition and the priority of each workframe; index 1 and 2 in the workframe array shown earlier. It builds a temporary array of workframes that share the maximum priority among all the active workframes. Finally the `Promela` code arbitrarily selects one workframe from this temporary array. For a further comparison with the semantics rules we refer the reader to the technical report [12].

4.3 Correctness Issues

As can be seen, we have not implemented the Brahms semantics directly in `Promela`. At present, analysis of this implementation consists of an informal comparison of the `Promela` arrays against the complex data structures of the semantics and an informal analysis of the operational rules against the partial instantiations produced for the specific example of the "Home Care" system. Parts of this analysis have been reproduced here and the full version can be found in [12]. In future work we intend to produce a more general, though still informal, discussion of the translation mechanisms themselves. It should be noted that there is also no proof that the operational semantics accurately capture Brahms. So both systems can be viewed separately as mechanisms for exploring models of human-agent teamwork even if they are not provably equivalent.

5 "Home Care" Verification

We next consider the actual verification of human-agent-robot teamwork; again we focus on the "home care" scenario.

5.1 Requirements

We develop a range of logical requirements for the scenario; recall that in *temporal logic*, $\Diamond\phi$ means that "ϕ will be true at *some* moment in the future", while $\Box\phi$ means that "ϕ will be true at *all* future moments". We describe some of the properties verified and classify these just by the core aspect they represent, i.e. properties labelled F*n* relate to the fire alarm; labelled by T*n* relate to the toilet; H*n* relate to hunger and M*n* relate to medicine. For space reasons, we only provide the temporal formulae in the case of the fire alarm. The axioms used in the properties are all based on the beliefs of the agents or facts in the system. We expect all of these properties to hold apart from M1.

F1: If a fire actually occurs then, eventually, *House* will generate a fire alarm. Logical requirement is: $\Box(a \Rightarrow \Diamond\, b)$ where
  a = there is a fire
  b = *House* believes there is a fire alarm

F2: If fire alarm is sounding, and *Bob* leaves *House* then fire alarm finishes. Logical requirement is: $\Box((a \wedge b) \Rightarrow \Diamond\neg a)$ where
  a = *House* believes there is a fire alarm
  b = *Bob* believes he has evacuated the house

F3: If fire alarm is sounding and *Bob* has not left *House*, then *Robot* reminds *Bob*. Logical requirement is: $\Box((a \wedge b) \Rightarrow \Diamond\neg c)$ where
  a = *House* believes there is a fire alarm
  b = *Bob* believes he has evacuated the house
  c = *Robot* believes it has alerted *Bob* of the fire 0 times

T1: Eventually *Bob* will go to toilet.

T2: If *Bob* goes to the toilet he can forget to flush it and, if so, he will be reminded by the House. So, if *Bob* goes the toilet then eventually he will flush the toilet.

H1: If *Bob* requests food then eventually *Robot* will deliver the food within an hour.

H2: Once *Bob* has finished eating, *Robot* will then retrieve the dishes and place in the dishwasher.

M1: Either *Bob* always takes his medication or the *Robot* never reminds him to do so. (This should be false since *Bob* may not take his medication even if reminded).

M2: If *Bob* has medication, but not taken it, then *Robot* will eventually remind *Bob* to take it.

M3: *Bob* takes medicine or *House* is informed that *Bob* has not taken it.

M4: If *Care Worker* is informed that *Bob* has not taken his medication then the *Care Worker* is with *Bob* within 2 hours and *Bob* takes his medication.

5.2 Verification Results

The properties F1, F2, F3, T1, T2, H1, H2, M2, M3 were all verified using Spin (i.e. the property holds on all paths from every initial state) in times ranging from T1 of 29.9 seconds to H1 of 848 seconds. As expected Spin shows that the property M1 is false and the time taken to find a trace in the model was 421 seconds. The property M4 was run multiple times to observe how changing the duration of the *Care Worker's* other duties affected the outcome. Spin was able to verify M4 so long as the *Care Worker's* other duties took less than 2 hours.

6 Conclusions

In this paper we have presented an overview of our work in verifying human-agent teamwork using the Spin model checker and the Brahms teamwork modelling system. Brahms enables the description of human-agent teamwork scenarios where the defining factors are the actions taken, their timing, duration and results. It has proven useful in the analysis of such scenarios via simulation. By adding verification to Brahms we extend its usefulness by allowing *all* possible simulations (with fixed time granularities) to be explored, thus ensuring that undesirable outcomes *can not* arise within the model.

A simple case study was presented, demonstrating the kind of human-agent teamwork scenarios we intend to verify. This case study included sample verified properties. The case study demonstrates most of the core capabilities of Brahms: multi-tasking by suspending actions in favour of higher priority ones; detecting changes in the environment; aborting actions; choosing between actions of equal priority; and communicating to coordinate actions.

The properties we verified were necessarily simple. However, it should be clear that as long as the properties can be represented in a straight-forward temporal language, then model-checking can be carried out. When humans are involved, we abstract their behaviour within the Brahms model and describe their requirements in logical terms. Whether human participants live up to these requirements is, of course, up to others to assess.

6.1 Related and Future Work

There are relatively few tools available for the analysis of human-agent teamwork. Brahms is one of the few that is used in the analysis of real systems. At present Brahms is, essentially, a testing tool and is used to examine multiple simulations of a model in a search of undesirable outcomes (e.g. Extra Vehicular Activities in space [13, 10]).

As far as we are aware there is no tool for the formal analysis via model checking of such scenarios. However BDI-style agent programming languages are a natural tool for creating such models, with their emphasis on modelling the reasoning of autonomous agents in terms of their beliefs and goals. A number of systems have been developed for model-checking programs in agent languages [14–16] though none of these have yet been applied to human-agent-robot teamwork.

In future we aim to improve the efficiency of the verification and to analyse more complex scenarios with multiple agents, cooperating and coordinating efforts in a much larger team. Scalability of the verification will be tested on these new scenarios. Scenarios under consideration include search and rescue; factory work; and hospitals. We also aim to investigate the verification of Brahms models in other model checkers, particularly ones with input languages which let us capture the operational semantics in a more intuitive fashion. This would provide a better guarantee of equivalence to Brahms simulations and it would also provide a point of comparison for evaluating the efficiency of the model checkers. The Java Pathfinder system [17] is an obvious candidate for this, either by implementing the Brahms semantics directly in Java or by using the AIL tool-kit for modelling agent languages and AJPF, its associated JPF based model checker [16].

Acknowledgements This work was funded by EPSRC through EP/F033567 and EP/F037201.

References

1. Montemerlo, M., Pineau, J., Roy, N., Thrun, S., Verma, V.: Experiences with a mobile robotic guide for the elderly. In: Eighteenth national conference on Artificial intelligence, Menlo Park, CA, USA, American Association for Artificial Intelligence (2002) 587–592

2. Pineau, J., Montemerlo, M., Pollack, M., Roy, N., Thrun, S.: Towards robotic assistants in nursing homes: Challenges and results. Robotics and Autonomous Systems **42** (2003) 271–281
3. Lenz, C., Nair, S., Rickert, M., Knoll, A., Rosel, W., Gast, J., Bannat, A.: Joint-action for Humans and Industrial Robots for Assembly Tasks. In: Proc. 17th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN). (2008) 130–135
4. CHRIS: Cooperative Human Robot Interaction Systems: `http://www.chrisfp7.eu` (2011)
5. Sierhuis, M.: 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)
6. Stocker, R., Sierhuis, M., Dennis, L., Dixon, C., Fisher, M.: A Formal Semantics for Brahms. In: Proc. 12th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA). Volume 6814 of Lecture Notes in Computer Science., Springer (2011) 259–274
7. Holzmann, G.: The Spin Model Checker: Primer and Reference Manual. Addison-Wesley (2003)
8. Wooldridge, M.: An Introduction to Multiagent Systems. John Wiley & Sons (2002)
9. Sierhuis, M.: Multiagent Modeling and Simulation in Human-Robot Mission Operations. (See `http://ic.arc.nasa.gov/ic/publications`) (2006)
10. Clancey, W., Sierhuis, M., Kaskiris, C., van Hoof, R.: Advantages of Brahms for Specifying and Implementing a Multiagent Human-Robotic Exploration System. In: Proceedings of the Sixteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS), AAAI Press (2003) 7–11
11. Clarke, E., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (2000)
12. Stocker, R., Dennis, L., Dixon, C., Fisher, M.: Spin Verification of Brahms Human-Robot Teamwork Models. (See `http://www.csc.liv.ac.uk/∼rss/Publications.html`) (2012)
13. Hirsh, R., Tyree, K., Johnson, N., Johnson, N.: Intelligence for human-assistant planetary surface robots. In: In IntelZigence for Space Robotics, TSI Press (2006) 261–279
14. Jongmans, S.S., Hindriks, K., van Riemsdijk, M.: Model checking agent programs by using the program interpreter. In: Computational Logic in Multi-Agent Systems. Volume 6245 of Lecture Notes in Computer Science. (Springer) 219–237
15. Bordini, R.H., Fisher, M., Pardavila, C., Wooldridge, M.: Model checking AgentSpeak. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS). (2003)
16. Dennis, L.A., Fisher, M., Webster, M., Bordini, R.: Model Checking Agent Programming Languages. Automated Software Engineering **19** (2012) 5–63
17. Visser, W., Havelund, K., Brat, G.P., Park, S., Lerda, F.: Model Checking Programs. Automated Software Engineering **10** (2003) 203–232