Webster, M, Dixon, C, Fisher, M, Salem, M, Saunders, J, Koay, KL, Dautenhahn, K and Saez-Pons, J

Toward Reliable Autonomous Robotic Assistants Through Formal Verification: A Case Study

Toward Reliable Autonomous Robotic Assistants Through Formal Verification: A Case Study

Matt Webster, Clare Dixon, Michael Fisher, Maha Salem, Joe Saunders, Kheng Lee Koay, Kerstin Dautenhahn, *Senior Member, IEEE*, and Joan Saez-Pons

***Abstract*—It is essential for robots working in close proximity to people to be both safe and trustworthy. We present a case study on formal verification for a high-level planner/scheduler for the Care-O-bot, an autonomous personal robotic assistant. We describe how a model of the Care-O-bot and its environment was developed using Brahms, a multiagent workflow language. Formal verification was then carried out by automatically translating this model to the input language of an existing model checker. Four sample properties based on system requirements were verified. We then refined the environment model three times to increase its accuracy and the persuasiveness of the formal verification results. The first refinement uses a user activity log based on real-life experiments, but is deterministic. The second refinement uses the activities from the user activity log nondeterministically. The third refinement uses "conjoined activities" based on an observation that many user activities can overlap. The four samples properties were verified for each refinement of the environment model. Finally, we discuss the approach of environment model refinement with respect to this case study.**

***Index Terms*—Autonomous systems, formal verification, human–robot teams, model checking, robotics.**

I. Introduction

ROBOTIC assistants are likely to be used for a variety of applications including personal healthcare, exploration within remote environments, and manufacturing. These robots will operate in close proximity to their human operators and, therefore, must be safe and trustworthy in their operations. One of the aims of the EPSRC-funded *Trustworthy Robotic Assistants* (TRA) project[1] is to develop tools and techniques for the verification and validation of robotic assistants. The TRA project uses three different methodologies for this: formal verification, simulation-based testing, and end-user validation. In this paper, we present a case study on the application of formal verification to the Care-O-bot: an autonomous robotic assistant deployed at the University of Hertfordshire's Robot House (see Fig. 1).

Manuscript received May 1, 2014; revised August 12, 2014 and January 31, 2015; accepted March 19, 2015. Date of publication May 13, 0215; date of current version March 11, 2016. This work was supported by EPSRC grants EP/K006193 and EP/K006509. This paper was recommended by Associate Editor N. Rungta.

M. Webster, C. Dixon, and M. Fisher are with the Department of Computer Science, University of Liverpool, Liverpool, L69 3BX, U.K. (e-mail: matt@liverpool.ac.uk; cldixon@liv.ac.uk; MFisher@liverpool.ac.uk).

M. Salem, J. Saunders, K. L. Koay, K. Dautenhahn, and J. Saez-Pons are with the School of Computer Science, University of Hertfordshire, Hertfordshire AL10 9AB, U.K. (e-mail: m.salem@herts.ac.uk; j.1.saunders@herts.ac.uk; k.l.koay@herts.ac.uk; K.Dautenhahn@herts.ac.uk; j.saez-pons@herts.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/THMS.2015.2425139

[1]http://www.robosafe.org/

Fig. 1. Care-O-bot robotic assistant operating in the University of Hertfordshire's Robot House.

Verification is the process of assessing whether a system meets its requirements; formal verification is the application of formal (i.e., mathematical) methods to the verification of systems. The formal verification approach used in this paper is based on model checking [1], in which a model of a program or process is constructed. This model is typically nondeterministic so that each "run" (or simulation) of the model can be different from the last. A program called a model checker exhaustively analyzes all possible executions of the model in order to establish that some property, usually derived from system requirements, holds. Therefore, it is possible, for example, to use a model checker to formally verify that in every execution of a given program, the program will always reach a desirable situation. In other words, we can formally verify that a given requirement holds.

The model checker used for this case study, SPIN [2], has been publicly available since 1991 and has been used for the formal verification of a wide variety of systems, including flood control barriers, telecommunications switches, and several space missions [3]. SPIN, which stands for Simple PROMELA Interpreter, verifies programs and processes written in PROMELA, the Process Meta-Language. Rather than writing in PROMELA directly, we utilize an intelligent agent modeling language and simulation environment called Brahms [4] to develop models of the Robot House and Care-O-bot. Brahms can be used to develop detailed models of systems with multiple interacting agents and has been used to model human–robot teams [5] and complex workflows [4] for space exploration. We translate the Brahms models automatically into PROMELA using the *BrahmsToPromela* translator software [6] based on a formal semantics for Brahms [7]. The autonomous control systems used in the Robot House were written at a similar level of abstraction to constructs in the Brahms modeling language.

Fig. 2. Plan view of the ground floor of the University of Hertfordshire Robot House. Numbered boxes show the locations of sensors.

Therefore, encoding these autonomous systems using Brahms reduced modeling errors due to changes in abstraction level.

Brahms is a rich language for specifying the behavior of multiagent systems and includes features such as inheritance, geography, message passing, and probabilities. While some of the features were necessary in modeling the Robot House (e.g., message passing), others were useful but not strictly necessary (e.g., inheritance). It would have been possible to use a lower level language to model the Robot House environment directly, e.g., PROMELA, and therefore avoid the additional step needed to translate from Brahms to PROMELA. However, as Brahms workframes closely resembled the IF–THEN rules used in the autonomous control systems within the Robot House, modeling these rules using Brahms could be achieved with limited effort. In addition, the *BrahmsToPromela* translator had already been developed, and therefore, the quickest way to obtain a PROMELA model from the autonomous control system code was to first encode the IF–THEN rules as Brahms workframes and then translate into PROMELA automatically using *BrahmsToPromela*.

In the remainder of this section, we examine the Care-O-bot and the Robot House in more detail and compare our approach to related work. In Section II, we describe the way in which a model of the Care-O-bot's high-level planner/scheduler for autonomous decision-making system was developed using Brahms. Then, in Section III, we show how that model was formally verified using the SPIN model checker, and in Section IV, we improve the environment models used by the model checker. Finally, in Section V, we provide a concluding discussion.

A. Robot House and Care-O-bot

The University of Hertfordshire's Robot House is a suburban three-bedroom house near Hatfield, U.K. (see Figs. 1 and 2). In addition to house furnishings and décor, the Robot House is equipped with more than 50 sensors that provide real-time episodic information on the state of the house and the individuals occupying it. These sensors range across electrical (e.g., refrigerator door open/closed sensor), furniture (e.g., cupboard drawers open), services (e.g., detect when toilet flush is being used), and pressure (e.g., chair sensors to detect when someone is seated) devices [8], [9].

The Robot House hosts a number of different robots that are used to conduct human–robot interaction experiments (see, e.g., [10]). One of these robots is the commercially available Care-O-bot robot manufactured by Fraunhofer IPA [11]. It has been specifically developed as a mobile robotic assistant to support people in domestic environments and is based on the concept of a "robot butler" [12]. The Care-O-bot robot is equipped with a 7-degree-of-freedom manipulator arm extended with a three-finger gripper and also comprises an articulated torso, stereo sensors serving as "eyes," LED lights, and a tray. Accordingly, the robot's sensors include its current location, the state of the arm, torso, eyes, and tray. By means of a text-to-speech synthesizing module, the robot is also capable of expressing given text as audio output.

The robot's software is based on the robot operating system (ROS) and a number of ROS packages (e.g., drivers, navigation, and simulation software) are available online[2]. For example, to navigate to any designated location within the house, the robot uses the ROS navigation package[2] in combination with its laser range-finders to perform self-localization, map updating, path planning, and obstacle avoidance in real time while navigating along the planned route.

High-level commands are sent to the robot via the ROS *script server* mechanism, which are then interpreted into low-level commands by the robot's software. For example, these high-level commands can take the form "raise tray," "move to location x," "grab object on tray," "say hello," etc.

The Care-O-bot's high-level decision making is determined by rules (stored in a MySQL database) of the form:

```
Guard
==
RobotAction*
```

Here, `Guard` is a sequence of propositional statements that are either true or false, linked by Boolean AND (`&`) and OR (`|`) operations, while `RobotAction*` is a sequence of actions which the Care-O-bot will perform only if the `Guard` is true. In practice, the `Guard` is implemented as a set of SQL queries and the `RobotActions` are implemented through the ROS-based `cob_script_server` package, which provides a simple interface to operate Care-O-bot. For example, take the following rule which lowers the Care-O-bot's tray:

```
SELECT * FROM Sensors WHERE sensorId=500
  AND value = 1     &
SELECT * FROM Sensors WHERE sensorId=504
  AND value = 1
==
light,0,yellow
tray,0,down,,wait
light,0,white,,wait
cond,0,500,0
cond,0,501,1
```

[2]http://wiki.ros.org/care-o-bot, http://wiki.ros.org/navigation

The SQL table called `Sensors` stores the values of all sensors in the Robot House. One row at most is stored in the Sensor table for each `sensorId/value` pair so that the SELECT queries above will return exactly one value if the proposition they represent is true, and false otherwise. (The word "sensors" is used loosely and includes variables, which describe the Robot House's knowledge about the state of the house, e.g., the "trayRaised" variable.)

In the rule above, the guard checks whether variable 500 (trayRaised) and 504 (trayEmpty) are set to 1 (true) or not by performing the SQL SELECT queries. If the guard is true, the Care-O-bot will change the light color to yellow (indicating that the robot is in motion), set the tray to the lowered position, wait for completion, set the light to white (indicating that the robot has stopped moving), and wait for completion. Finally, the variables 500 (trayRaised) and 501 (trayLowered) are set to "false" and "true," respectively. Note that twice the robot `waits` for short periods of time (around one second). The `wait` command is used to prevent the robot's actions from executing at the same time, e.g., to prevent the light being set to white (indicating to the user that the robot has stopped moving) before the robot actually stops moving.

The Care-O-bot's rule database is composed of multiple rules for determining a variety of autonomous behaviors, including checking and answering the front doorbell and reminding a person to take medication. The full Robot House rule database includes a set of 31 default rules and can be obtained from the EU ACCOMPANY project's Git repository.[3]

B. Related Work

Stocker *et al.* [6] describe the development of the *BrahmsToPromela* software tool, which is utilized in this study to automatically translate a model of the Care-O-bot written in Brahms into a PROMELA specification which can then be formally verified using the SPIN model checker. The authors examine an assisted living scenario similar to the Robot House system tackled in this case study. However, the work here expands on the work of Stocker *et al.* by modeling a real-life robotic system and scenario where the rules are directly derived from actual code used in practice. Additionally, this work has used user activity logs from real-world experiments within the Robot House in order to increase the verisimilitude of the person agent within the model.

Saunders *et al.* [8] and Duque *et al.* [9] described the University of Hertfordshire Care-O-bot and Robot House systems, which are used as a basis for this work. More information on the development of these systems can be found on the ACCOMPANY project website.[4]

Formal verification has been used before for robotic systems. For example, Mohammed *et al.* [13] used hybrid automata and hybrid statecharts for formal modeling and verification of multirobot systems, Cowley and Taylor [14] used dependent-type theory and linear logic for the formal verification of assembly robots, and Kouskoulas *et al.* [15] formally verified control algorithms for surgical robots. However, very little work has been conducted to formally verify the safety and trustworthiness of robotic home companions. This is where our work aims to complement existing research in the area of formal verification of autonomous and robotic systems.

[3]https://github.com/uh-adapsys/accompany

[4]http://accompanyproject.eu

II. Modeling the Care-O-bot Using Brahms

The 31 default rules are similar in structure to common constructs within the Brahms multiagent workflow programming language. The first step in modeling was to convert the full set of Care-O-bot rules into a more convenient IF–THEN rule representation. For example, the above rule was rewritten as

```
IF tray_is_raised AND tray_is_empty
THEN set_light(yellow)
    move_tray(lowered_position)
    wait()
    set_light(white)
    wait()
    set(tray_is_raised,false)
    set(tray_is_lowered,true)
```

Once translated into this format, these rules could then be straightforwardly translated into Brahms. A key concept in Brahms is the "workframe," which specifies a sequence of things to be done when a given condition holds. The Robot House rules were translated into Brahms workframes within the Care-O-bot agent, with the `IF a THEN b` rules translated into `if a then do {b}` construct in Brahms. For example, the rule above was translated into a Brahms workframe called f_lowerTray:

```
workframe wf_lowerTray {
repeat: true;
priority: 10;
when(knownval(current.trayIsRaised
 = true) and
  knownval(current.trayIsEmpty = true))
do{
  conclude((current.lightColour =
     current.colourYellow));
  lowerTray();
  wait();
  conclude((current.lightColour =
     current.colourWhite));
  wait();
  conclude((current.trayIsRaised
  = false));
  conclude((current.trayIsLowered
  = true));
 }
```

This workframe is set to "repeat," meaning that it can be used more than once by the agent. Multiple workframes can be eligible for execution by the Brahms interpreter at the same time; therefore, the `priority` sets the importance of the workframe relative to other workframes (with larger numbers being more important). `If a then do {b}` construct states that when some conditions are true, in this case `trayIsRaised` and `trayIsEmpty`, the agent should do the actions that follow. In the action list, the `conclude()` construct is used to determine

when beliefs should be updated within the Brahms agent. The `lowerTray()` and `wait()` statements are programmer-defined primitive actions whose function is to denote that something has happened and has a certain duration, e.g.:

```
primitive_activity wait()
{ max_duration: 1}
```

In general, the Robot House rules were translated into Brahms workframes on a one-to-one basis. However, in some cases, it was necessary to use more than one Brahms workframe for a rule. This generally happened when a rule contained interactions with the user via the GUI. For instance, when the person sits down and watches television (detected via sensors in the sofa seats and the television power outlet), the Care-O-bot approaches the person and asks whether he or she would like to watch the television together. The person has three options, which are presented by the Robot House via a GUI on a tablet computer: to tell the Care-O-bot to watch television, to return to its charging station, or to continue with its current task. This behavior is modeled using a Brahms workframe within the Care-O-bot agent, in which these options are communicated to the person using the `announceQuery-ToUser_ThreeOptions()` activity:

```
workframe wf_watchTV {
repeat: true;
priority: 10;
when(knownval(robotHouse.sofaSeat
Occupied =   true) and
 knownval(robotHouse.televisionWattage
 > 10)  and  ...)
do{
 conclude((current.queryUserOption1 =
    current.activityWatchTV));
 conclude((current.queryUserOption2 =
    current.activityReturnHome));
 conclude((current.queryUserOption3 =
    current.activityContinue));
 conclude((current.userQueried = true));
 conclude((current.queryUser_
 ThreeOptions = true));
 announceQuery ToUser_ThreeOptions();
 conclude((current.askedToWatchTV
 = true));
}
```

The "person" agent (simulating simple human behavior) then selects a response and sends it back to the Care-O-bot model. In this example, the person agent was set to always choose to watch TV with the Care-O-bot *when asked* by the Care-O-bot. The following Care-O-bot workframe is executed when the person decides to watch TV:

```
workframe wf_optionSelectedWatchTV {
repeat: true;
priority: 10;
when(knownval(Person.userRespondedToQuery
= true)  and
 knownval(Person.queryResponse =
    current.activityWatchTV))
do{
 conclude((Person.userRespondedToQuery
 = false));
 conclude((Person.guiSetWatchTV
 = true));
 conclude((current.userQueried
 = false));
}
```

Here, the guard uses the Care-O-bot's belief about the person agent's belief about whether the person has responded to the query. (While the belief is referred to as `Person.userRespondedToQuery`, it concerns the Care-O-bot agent's belief about the person agent's belief as this workframe is part of the Care-O-bot agent. For more information on belief handling in Brahms, see [16].). If this belief is set to "true," and the person has responded to the query, then the workframe will execute. Note that later in the workframe the Care-O-bot agent sets this belief to false so that the next time the person agent responds to a query; this workframe will be able to execute correctly.

A. Modeling a Scenario

After translating the Robot House rules into Brahms, it was necessary to set up a model of the Robot House environment, or *scenario*. The scenario determines the range of possibilities within the Robot House environment consisting primarily of the Care-O-bot, the person being assisted by Care-O-bot, and the Robot House. For example, the scenario consists of a model of the person and the Robot House, where each is defined as an agent within Brahms. Another agent, the "Campanile Clock," measures the passage of time in the model and keeps the other agents updated with the current time.

Information within the Brahms model is stored as agent beliefs, which can be public or private. Agents can reason about their own beliefs as well as the beliefs of other agents. For example, the Campanile Clock agent maintains the current time within the model as a public belief and periodically announces the time to the other agents. Therefore, the Campanile Clock's `time` variable can be used by all agents to refer to the current time within the model. (This is useful when defining properties for formal verification.) Other information in the model is held by the relevant agent. For example, the Robot House agent maintains sensor data for Robot House, the Care-O-bot agent maintains information about the location and state of the Care-O-bot, and the Person agent maintains the location and state of the person occupying the Robot House.

The layout of the Robot House is encoded using a *geography* within Brahms, providing an hierarchical description of the different places that the agent can occupy. For example:

```
areadef House extends BaseAreaDef { }
areadef Room extends House { }
areadef areaOfInterest extends Room { }
area LivingRoom
instanceof Room partof House { }
area LivingRoomTV instanceof areaOf
Interest
  partof LivingRoom { }
```

This Brahms geography states that there is a type of area called `House`. `House` is an extension of a built-in Brahms class called `BaseAreaDef`. Another area type, `Room`, extends the `House` class, and `areaOfInterest` extends `Room`. The `LivingRoom` is an instance of the `Room` class, and is part of the `House`, and the `LivingRoomTV` is an instance of `areaOfInterest` and is part of the `LivingRoom`. Note that the `House` area definition is different from the Robot House agent described above; the former describes the layout of the robot house, while the latter encapsulates the data from the Robot House's sensors together with functionality for communicating sensor state with other agents.

Our scenario lasts from noon to 9 P.M. At any given point in the day, the person may choose to sit and watch TV, move into the living room area, or move into the kitchen (e.g., to prepare food to eat), or may choose to send the Care-O-bot into the kitchen or the living room. At 5 P.M., the person needs to take medication.

The person can act nondeterministically, that is, behave in a manner which is unpredictable within the model of the scenario. This nondeterminism is implemented in Brahms as a set of five workframes within the person agent, all of which will fire at a given point. Each workframe has a priority. The highest priority workframe is executed, and a belief is modified within the agent (using the "conclude" keyword in Brahms). This belief is modified with a level of certainty (known as the belief-certainty), which states that the belief will be updated with a given probability. If the belief is updated, this information is communicated to the Care-O-bot agent or the Robot House agent (depending on the workframe), which causes these agents to know that the person has done something, e.g., sent the Robot to the kitchen via the GUI, or that the person has moved into the living room. If the belief is not updated, then the next workframe fires. It is possible for none of the five workframes to update a belief within the person agent, and this special case models the ability of the person to do nothing.

Based on this simple scenario, we can establish a number of high-level requirements for the Care-O-bot. For example, the Care-O-bot should remind the person to take medication at 5 P.M. Another requirement is that the Care-O-bot should go into the living room if it is told to go into the living room by the person. In the next section, we formalize these kinds of requirements using temporal logic and verify them using the SPIN model checker.

III. Formal Verification of Brahms Models Using *BrahmsToPromela* and SPIN

Brahms refers to a multiagent workflow specification language, as well as a software package consisting of an agent simulation toolkit and an integrated development environment. The Brahms software does not come with formal verification tools built-in; for formal verification, we used the *BrahmsToPromela* translator [6]. *BrahmsToPromela* allows models written using a subset of Brahms to be automatically translated into PROMELA, the process metalanguage used by the SPIN model checker. Once translated, SPIN can be used for the automatic formal verification of the Brahms model with respect to particular requirement. In our case, we formalize these requirements using linear temporal logic, which allows the formalization of concepts relating to time, e.g., "now and at all points in the future" (via the $\Box$ operator), "now or at some point in the future" ($\Diamond$) and "in the next state" ($\bigcirc$) [17]. This enables formalisation of safety requirements (something bad never happens, $\Box\neg$bad), liveness properties (e.g., something good eventually happens, $\Diamond$good) and fairness properties (e.g., if one thing occurs infinitely often so does another, e.g., $\Box\Diamond$send $\Rightarrow \Box\Diamond$receive).

Using *BrahmsToPromela* extends SPIN's property specification language with a *belief* operator, "B." This is parameterized by the agent that holds the belief—therefore, $\mathsf{B}_{\text{Care-O-bot}}x$ means that the Care-O-bot believes x is true. Beliefs in Brahms are translated into Boolean variable arrays in PROMELA. For example, $\mathsf{B}_{\text{Care-O-bot}}x$ is modeled by "`bool CareOBot_`x`[`n`]`," where n is the number of agents and objects within the Brahms simulation. Beliefs are stored as arrays as Brahms allows agents to (dis)believe other agents' beliefs; therefore, it is necessary to use an array to store whether or not each agent believes a particular agent's belief. Using this framework, we model $\mathsf{B}_{\text{Care-O-bot}}(\mathsf{B}_{\text{Person}}x)$ (i.e., whether the Care-O-bot believes that the Person believes x) as the Boolean array index "`Person_`x`[cob]`" where `cob` is a constant that refers to the Care-O-bot. Similarly, $\mathsf{B}_{\text{Person}}x$ is modeled as "`Person_`x`[person]`."

All of the properties in this paper use beliefs rather than facts about the environment. Beliefs may or may not hold, i.e., beliefs can be incorrect. However, in this system, we assume complete sensor accuracy so that if x is true, then $\mathsf{B}_a x$ is true for all agents a.

To explore possibilities, the following sample requirements were translated and their formalized properties verified using SPIN for the Brahms model.

1) It is always the case that if the Care-O-bot believes that the person has told it to move into the kitchen, then it will eventually move into the kitchen
$$\Box\left[\begin{array}{l}\mathsf{B}_{\mathrm{Care-O-bot}}(\mathsf{B}_{\mathrm{Person}}\mathrm{guiGoToKitchen})\\ \Longrightarrow\ \Diamond\mathsf{B}_{\mathrm{Care-O-bot}}(\mathsf{location}=\mathrm{Kitchen})\end{array}\right].$$
2) It is always the case that if the Care-O-bot believes that the person has told it to move to the sofa in the living room, then it will eventually move there
$$\Box\left[\begin{array}{c}\mathsf{B}_{\mathrm{Care-O-bot}}(\mathsf{B}_{\mathrm{Person}}\mathrm{guiGoToSofa})\\ \Longrightarrow\ \Diamond\mathsf{B}_{\mathrm{Care-O-bot}}(\mathsf{location}=\mathrm{LivingRoomSofa})\end{array}\right].$$
3) It is always the case that if the Robot House believes that the sofa seat has been occupied for at least 1 h,[5] and if the Robot House believes that the television power consumption is higher than 10 W, and if the Care-O-bot believes that it has not yet asked the person if he or she wants to watch television, then eventually, the Care-O-bot will move to the living room sofa and ask the person if he

[5]This ensures that the Care-O-bot will only ask the person if he or she wants to watch television once every hour at most.

TABLE I
FORMAL VERIFICATION OF FOUR PROPERTIES FOR THE ORIGINAL MODEL

Prop.	States	Depth	Memory (MB)	Time (s)
1	302 160	74 819	399	9.9
2	302 160	74 819	399	9.7
3	576 317	81 341	410	18.0
4	302 160	74 819	399	9.7

or she wants to watch the television

$$\Box\left[\begin{array}{c}\mathsf{B}_{\mathrm{RobotHouse}}\mathrm{sofaOccupied1Hour}\wedge\mathsf{B}_{\mathrm{RobotHouse}}\mathrm{tvPower}>10\\ \wedge\mathsf{B}_{\text{Care-O-bot}}\neg\mathrm{askedToWatchTV}\\ \Longrightarrow\Diamond\left(\begin{array}{c}\mathsf{B}_{\text{Care-O-bot}}\mathrm{location}=\mathrm{LivingRoomSofa}\wedge\\ \mathsf{B}_{\text{Care-O-bot}}\mathrm{askedToWatchTV}\end{array}\right)\end{array}\right].$$

4) It is always the case that if the time is 5 P.M., then the Care-O-bot will believe that the medicine is due

$$\Box\left[\begin{array}{c}\mathsf{B}_{\text{Campanile\_Clock}}\mathrm{time}=5\mathrm{pm}\Rightarrow\\ \Diamond\mathsf{B}_{\text{Care-O-bot}}\mathrm{medicineDue}\end{array}\right].$$

The first two requirements are derived from a higher level requirement that, in general, the Care-O-bot should follow instructions given to it by the person. The third property is important for maintaining the social activity of the person within the Robot House, who is temporarily under the care of the Care-O-bot. The fourth property is derived from a higher level requirement that the Care-O-bot should remind the person to take medication at the correct time.

Table I summarizes the verification results.[6] These results were obtained using an eight-core Intel Core i7-3720QM CPU (2.60 GHz) laptop with 16 GB of memory running Ubuntu Linux 12.04 LTS. In each case, the same PROMELA model was used; any difference in the number of states or time taken is due to the complexity of the property being verified and the resulting automaton used by the model checker. For requirements 1, 2, and 4, the resources used were almost identical, and this is to be expected as these properties were similar in structure and produced similar automata. Property 3 produced a slightly more complex automaton requiring more resources to verify.

IV. IMPROVING THE ENVIRONMENT MODEL

In the previous section, we were able to formally verify the high-level decision making system within the Care-O-bot. This provides a degree of assurance that the Care-O-bot will satisfy the requirements/properties against which it has been verified. However, this degree of assurance is dependent on the quality of the model of the Care-O-bot's environment. The environment model determines the behavior of all agents beyond the Care-O-bot, including any people within the Robot House, the Robot House, and the sensors within the Robot House. In the case of the Care-O-bot, which has a deterministic control system, the environment model is the sole source of nondeterminism within the model and generates the state space, which is then model checked.

However, the environment model used in the Brahms model is simple. It describes a scenario that lasts from 12 P.M. to 9 P.M. In each hour, the person agent can choose from one of six possible options: sit down and watch TV, move into the living room area, move into the kitchen, send the Care-O-bot into the kitchen, send the Care-O-bot to the living room, or do nothing. At 5 P.M., the person will need to take medication and is reminded to do so by the Care-O-bot. Which one of the six possible options is chosen by the person is nondeterministic, and during simulation of the model the choice made is random.

In a real-world Robot House scenario, there are many more things that could happen than in our environment model. The person may go to bed, for example, or choose to leave the house. The doorbell could ring, or another person could enter the house.

In order to generate a more interesting environment model, we used the activity recognition system in [9] to collect data from an actual person living in the Robot House for a period of four days. The person's behavior was monitored by sensors throughout the Robot House (see Fig. 2). There are a total of 59 sensors in the Robot House, which allow the house to be monitored in a variety of ways. A real-time energy monitoring system can detect the activation and deactivation of electrical devices like refrigerators and kettles. A different sensor network is able to monitor the movement of people through the Robot House using reed sensors, temperature sensors, and pressure mats. Via these sensors, it is possible to detect when someone sits down on a chair, or opens the bedroom door, for example [9].

One person occupied the house for approximately four days and 11 h. Combinations of sensor outputs were used to determine the current activity of the person within the Robot House during this time. For instance, if the "bed contact" sensor is activated, then it was assumed that the `In_Bed` activity was underway. In other words, the person must be in bed. Activities can also be related to other activities. For example, if the activity `In_Bed` is active, and the activity `Lamp_on_Bedroom` has been deactivated for at least 5 s, then the activity `Sleeping_Bedroom` is inferred. If the person is in bed and has just turned the light off, then the person must be sleeping in the bed.

The user activity log generated automatically in the Robot House contained 569 different entries, each being a tuple containing the activity name, whether the activity was beginning (i.e., being activated) or finishing (i.e., being deactivated) and the date/time. For example, the following state that the activity `In_Bed` began at 11:03 P.M. on May 19, 2013 and finished at 5:25 A.M. on May 20, 2013:

```
In_Bed   1   2013-05-19 23:03:50
In_Bed   0   2013-05-20 05:25:03
```

The entries covered a range of activities, including `In_Bathroom`, `Toiletting`, `Preparing_Hot_Drink`, `Actively_Watching_TV`, and `Water_Sink_ON`.

[6]The computational resources used are less than those reported in our earlier paper [18] due to the use of hash compression in the SPIN model checker. For consistency, hash compression was used throughout this paper.

Three different approaches to modeling the person were taken in light of the user activity log. They are as follows.

A. Deterministic Environment Model

In the deterministic environment model, the entries in the user activity log were converted directly into Brahms workframes without abstraction so that if an event occurred at a particular time within the Robot House, then the same event occurs within the Brahms model at exactly the same time. For example, take the following entries in the activity log:

```
  Preparing_Hot_Drink  1  2013-05-
20 09:07:06
  Preparing_Food       1  2013-05-
20 09:07:06
```

Both activities start at the same time, 9:07:06 A.M. These were translated into the following Brahms workframe within the person agent:

```
  workframe wf_36196 {
         repeat: true;
         priority: 100;
         when(knownval(Campanile_Clock.
  time = 36196 )      and known-
val(current.wf36196 = false))
         do {
           conclude((current.Preparing
           HotDrink = true));
           conclude((current.PreparingFood
           = true));
           conclude((current.wf36196
           = true));
         wait();
     }
    }
```

This workframe was set to repeat (although repetition would not happen due to the way the workframe is defined) with priority 100, the same as all other workframes generated from the activity log. The `when` statement says that when the campanile clock time is set to 36 196,[7] to set the person agent's beliefs about preparing a hot drink and preparing food to true, and to set the belief `wf36193` to true. A precondition of this `when` statement is that `wf36196` is false; setting it to `true` prevents this workframe from executing again in the future.

A custom Java application automated the process to translate a comma-delimited input file containing the complete activity log into a set of Brahms workframes of the form above. Brahms agent belief declarations, such as `PreparingFood`, were also generated by the Java application. Since all beliefs were describing the activity of the person within the Robot House, these belief declarations were included in the Person agent's specification in Brahms.

The resulting Brahms file was 316 kB in size and contained over 500 workframes in the person agent—including one for each user activity log entry. This is larger than the earlier Brahms model (which was $\sim$ 107 kB), and Brahms was able to execute the model in 10 s on the same laptop used for model checking. However, translating this Brahms model into SPIN resulted in a model too large to be model checked. There were too many `mtype` elements (symbolic names for constant values) within the PROMELA model. (SPIN allows a maximum of 255 distinct names within an `mtype` declaration.) The `mtype` elements are used by the *BrahmsToPromela* translator to keep track of agent states within the PROMELA model. One strategy was to see whether it was possible to verify a fragment of the user activity log instead.

[7]The clock time of 36 196 was based on 9:07:06 A.M. being 36 196 s after the first entry in the activity log.

TABLE II
FORMAL VERIFICATION OF FOUR PROPERTIES FOR DETERMINISTIC ENVIRONMENT MODEL

Prop.	States	Depth	Memory (MB)	Time (s)
1	114 501	229 001	1463	8.4
2	45 272	90 543	674	3.3
3	109 329	218 655	1394	8.1
4	51 009	102 017	742	3.6

A fragment corresponding to the first 21 h of user activity was the largest fragment that could formally be verified. The Robot House model incorporating the deterministic environment model was formally verified with respect to the four properties given in Section III. Table II summarizes the time and memory usage.

The model checker was able to formally verify these properties much more quickly (a minimum of 3.3 s compared to 9.7 s), and with fewer states (a minimum of 45 272 compared with 302 160). This is to be expected, as the model is deterministic, resulting in a much smaller state space.

From a model-checking perspective, the deterministic environment model can be seen as a single run (or simulation) of a nondeterministic model. However, this model is still interesting from a formal verification perspective as it allows validation of the robot's autonomous control system as modeled within Brahms and translated using *BrahmsToPromela*. For example, we can observe the behavior of the real-world Care-O-bot over the 21-h period contained in the user activity log, and check whether properties 1–4 actually held in the real world. If they did not, this would indicate that there was a problem with the Brahms model and that redesign would be necessary. Furthermore, use of a deterministic model within a model checker allows us to verify a formally defined property and, therefore, provide a more rigorous result than simply executing a simulation of the Brahms model and checking the output trace to see whether the requirement corresponding to the formal property has held.

B. Extended Nondeterministic Environment Model

One of the strongest aspects of model checking is to automatically explore the state space of a nondeterministic model. Within such a model, there are points at which a number of different things could happen. For example, a person may sit down,

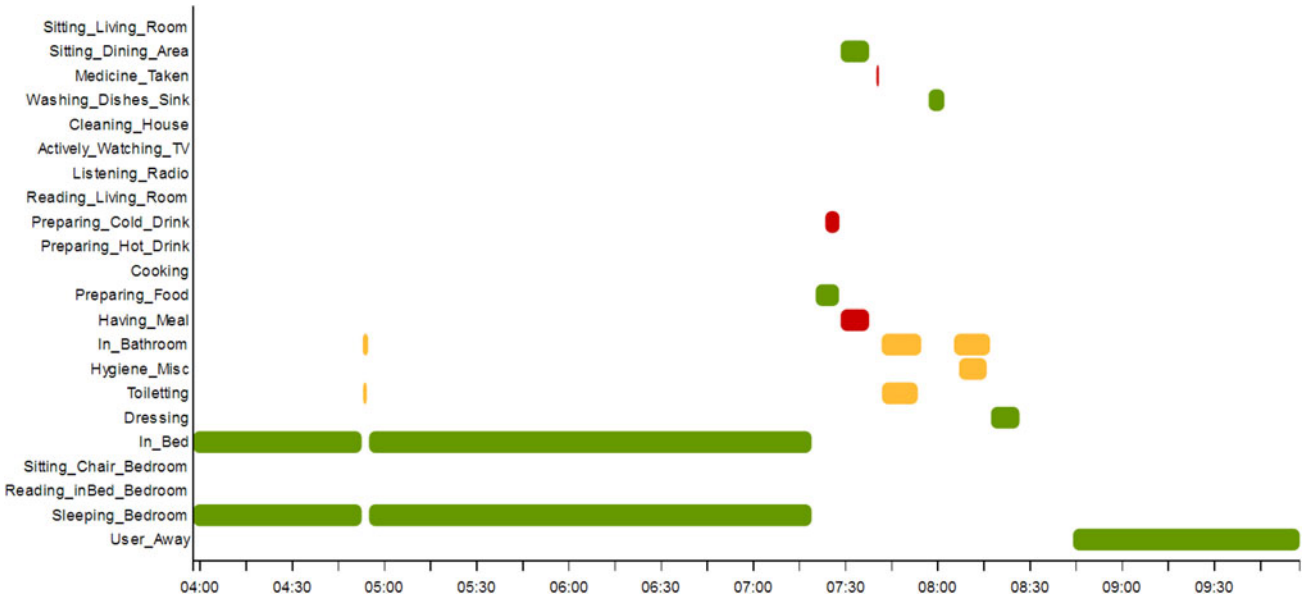


Fig. 3. Part of the UH Robot House user activity log for a 6-h period beginning at approximately 4 A.M. on Thursday May 23, 2013.

TABLE III
FORMAL VERIFICATION OF FOUR PROPERTIES FOR EXTENDED NONDETERMINISTIC ENVIRONMENT MODEL

Prop.	States	Depth	Memory (MB)	Time (s)
1	1 122 836	113 519	1223	107
2	1 122 836	113 519	1223	104
3	1 300 161	113 519	1118	111
4	1 122 836	113.519	1223	104

watch TV, or prepare a meal. These nondeterministic choices produce a branching state space, which typically grows exponentially with the number of such choices. One job of a model checker, therefore, is to check all the possible paths through this state space in order to ensure that a particular property holds at all points during the execution of the model. For example, in Section III, we showed that the Care-O-bot will always remind the person in the house to take their medication at 5 P.M. By using a model checker, we know that this is always true in every possible run through the model, and in every possible state, with respect to the environment model used.

In order to improve the environment model in Section II-A, another nondeterministic model was developed. This new model operates in a similar manner: There are a number of possible actions that the person can take, and the choice of which action to take is nondeterministic. However, this environment model is a significant improvement over the environment model in Section II-A as it contains a higher number of possible actions.

In the environment model from Section II-A, there were only six things the person could choose to do in a given time step. Inspection of the user activity logs revealed that there were many more than six things that the person did and these activities were used as the basis for the new environment model.

A fragment of the user activity log can be seen in Fig. 3. Each of the activities shown on the left-hand side of the figure were modeled using Brahms. There were a total of 26 different user activities from which the person agent could choose (22 activities shown in Fig. 3, and four user activities from the environment model in Section II that were not duplicated in the user activity log).

The multiagent version of the Robot House scenario, including the extended nondeterministic environment model described above, was model checked with respect to the four properties given in Section III. No errors were found. The time and memory requirements are summarized in Table III.

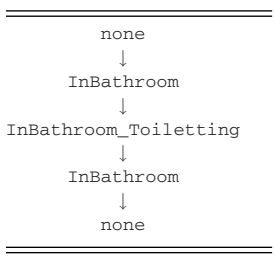
Increased nondeterminism in the environment model has resulted in the use of more computational resources during model checking. For example, the number of states required has roughly tripled compared to the results in Table I. This is to be expected as there are more activities available to the person agent.

C. Nondeterministic Conjoined Activity Environment Model

Examination of the user activity log showed that a number of activities overlap and were not mutually exclusive. (A number of overlapping activities can also be seen in Fig. 3.) Consider the following:

```
In_Bathroom    1    2013-05-20    05:25:46
Toiletting     1    2013-05-20    05:26:01
Toiletting     0    2013-05-20    05:26:56
In_Bathroom    0    2013-05-20     05:27:05
```

Here, the `In_Bathroom` activity begins, and a few seconds later, the `Toiletting` activity begins and then ends. Shortly, the `In_Bathroom` activity ends. Clearly, these two activities have overlapped. We could describe what has happened with a simple state diagram:



Here, none denotes a state in which no activities are happening. In the next state, In_Bathroom is active, and in the next state, InBathroom_Toiletting is happening. Next In_Bathroom occurs again (as Toiletting has now finished), and finally, we return to the none state.

The nondeterministic model in Section IV-B is, therefore, less accurate than we thought, as the user can select from 26 mutually exclusive activities, whereas in reality, these activities are not mutually exclusive and may overlap. In order to improve the accuracy of the environment model, we therefore needed to account for overlapping activities. This was done by "conjoining" the activities in the way we did with the In_Bathroom and Toiletting activities above: These overlapping activities can be converted into the mutually exclusive states none, In_Bathroom, and InBathroom_Toiletting. In other words, by conjoining the names of the currently occurring activities, we can describe them as mutually exclusive states of the form we had in the example in Section II-A.

For the conversion, a Java application was implemented. The application reads in an activity log from a comma-delimited input file. This is then converted into a data structure D, which maps timestamps to sets of activities, which are active at that time. For example, for the activity log excerpt above, D would be

$$\begin{aligned}\{\ &2013-05-2005:25:46 \mapsto && \{\texttt{InBathroom}\},\\ &2013-05-2005:26:01 \mapsto && \{\texttt{InBathroom}, \texttt{Toiletting}\},\\ &2013-05-2005:26:56 \mapsto && \{\texttt{InBathroom}\},\\ &2013-05-2005:27:05 \mapsto && \emptyset\}\end{aligned}$$

Each set in the range of D is converted into a string denoting a conjoined activity so that $\{\texttt{InBathroom}, \texttt{Toiletting}\}$ becomes InBathroom_Toiletting. The conversion is performed deterministically so that each set maps to a single conjoined activity string, with no possibility of having, say, InBathroom_Toiletting distinct from Toiletting_InBathroom. This set of strings is then used to output Brahms workframes describing those conjoined activities. For example, the SittingDiningArea, HavingMeal and MedicineTaken activities all overlap and so were converted to the conjoined activity SittingDiningArea_HavingMeal_MedicineTaken, in turn translated into the following Brahms workframes:

```
workframe wf_SittingDiningArea_
HavingMeal_Medicine-Taken {
        repeat: true;
        priority: 25;
        when(knownval(Campanile_Clock.time
        < 5) and
            knownval(current.doneSitting
            DiningArea_-
             HavingMeal_MedicineTaken
             = false) and
               knownval(current.considered
              Sitting-DiningArea_
              HavingMeal_MedicineTaken
                       = false))
      do {
         conclude((current.considered
             SittingDiningArea-
           _HavingMeal_MedicineTaken
            = true));
           conclude((current.goalDo
           SittingDiningArea_-
           HavingMeal_MedicineTaken
           = true),bc:20);
           wait();
       }
workframe wf_doSittingDining
        Area_HavingMeal_-
MedicineTaken {
                repeat: true;
                priority: 25;
      when(knownval(current.goalDoSitting
         DiningArea_-
        HavingMeal_MedicineTaken
          = true) and
            knownval(current.doneSitting
             DiningArea_-
             HavingMeal_MedicineTaken
              = false))
           do {
           conclude((current.doneSitting
             DiningArea_-
           HavingMeal_MedicineTaken
            = true));
           conclude((current.Sitting
           DiningArea = true));
           conclude((current.HavingMeal
            = true));
         conclude((current.MedicineTaken
          = true));
          wait();
       }
```

The first workframe says that if the time is before 5 P.M., and the conjoined activity SittingDiningArea_HavingMeal_MedicineTaken has not been done, and if this conjoined activity has not yet been "considered," then consider it by making a belief update with certainty 20% (bc:20). Therefore, 20% of the time this update is successful, which will cause the next workframe to execute and update the beliefs of the person agent to reflect that the activities SittingDiningArea, HavingMeal and MedicineTaken are now taking place. 80% of the time this

TABLE IV
FORMAL VERIFICATION OF FOUR PROPERTIES FOR NONDETERMINISTIC CONJOINED ACTIVITY ENVIRONMENT MODEL

Prop.	States	Depth	Memory (MB)	Time (s)
1	4 470 619	337 437	5354	872
2	4 248 201	373 391	5938	871
3	4 587 998	384 157	6190	859
4	4 430 294	372 549	5926	848

belief update is not successful, which does not trigger the second workframe. In this case, another workframe corresponding to a distinct conjoined activity may execute. Ultimately, it is possible that none of the conjoined activities happens, in which case the person agent is doing nothing for that time step.

Using the ActivityLog Java application, a total of 133 conjoined activities were found. Each of which was converted into Brahms workframes as described above.

The multiagent system of the Robot House scenario, including the nondeterministic conjoined activity environment model, was model checked with respect to the four properties given in Section III. No errors were found. The time and memory requirements are presented in Table IV.

This environment model contained a larger number of activities for the person to be engaged in than in the nondeterministic environment model used previously (there are now 133 different conjoined activities compared with 26 different activities previously). It can be seen that the increased nondeterminism in the environment model has increased the computational resources required by the model checker. For example, the number of states used is around four times as many as for the extended nondeterministic environment model and around 14 times as many as for the environment model used in Section III.

V. DISCUSSION

In this case study, we formally verified an autonomous decision making planner/scheduler system for the Robot-House-assisted living environment and the Care-O-bot robotic assistant. This was done by converting the Robot House planner/scheduler rules into Brahms: a workflow language for defining the behavior of multiple agents. These rules matched closely the Brahms style. Brahms was also used to model the Robot House environment, including the Care-O-bot, the Robot House, and a resident. The Brahms model was then translated into the PROMELA language using the *BrahmsToPromela* tool [6]. Once in PROMELA, the Brahms workflows (and, consequently, the Care-O-bot's decision making system) were formally verified. Four properties were formally verified, demonstrating that this process can be used for verification of autonomous decision making systems for robotic systems.

The simplistic nondeterministic environment model used in Section III was enhanced in three different ways in order to increase its fidelity. First, it was enhanced to a deterministic model that covered a larger set of user activities. Then, it was modified to include a total of 26 different activities for the person agent to choose from (20 activities more than the initial nondeterministic environment model). Finally, it was modified to allow for the overlapping of the person agent's activities, resulting in 133 different conjoined activities. In all cases, the model checker showed that there were no errors found, therefore providing assurance that the Care-O-bot's high level decision-making system meets requirements.

The person agent in the environment model is the chief source of nondeterminism, as is the case with the resident in the real-life Robot House. Extending the environment model in three different ways (to give a total of four different environment models) means that the unpredictability of the person agent has been modeled in four different ways. Each of the four models provide a degree of assurance that the Care-O-bot is safe for use by a single person in the Robot House, and together, they provide an even greater degree of assurance. Obviously, a model is an abstraction of the real world and will never be able to fully represent all aspects of the actual situation. However, by modeling human behavior in as many ways as possible, we gain a greater confidence in the system being formally verified and a greater level of trust in the autonomous robotic assistant.

The aim of this case study is to provide a detailed example of how formal verification can be applied to an existing personal robotic system. We show how formal verification can be used to provide assurance that the robotic system will behave correctly with respect to a small subset of the robot's requirements. While the size of this subset is small, it is sufficient as a proof of concept.

The formal verification techniques used were effectively "off-the-shelf" software components, consisting of Brahms, a multiagent simulation framework, *BrahmsToPromela*, an automatic translator from Brahms to PROMELA, and SPIN, a well-known model checker for the PROMELA language. The use of modular components expedites the modeling and formal verification process and minimizes errors that may be introduced by developing new formal verification methods.

REFERENCES

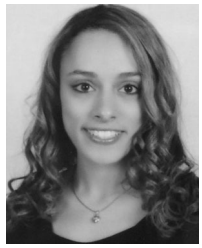
[1] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.
[2] G. J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*. Reading, MA, USA: Addison-Wesley, 2003.
[3] G. Holzmann. (2013). Inspiring applications of Spin. [Online]. Available: http://spinroot.com/spin/success.html
[4] M. Sierhuis and W. J. Clancey, "Modeling and simulating work practice: A method for work systems design," *IEEE Intell. Syst.*, vol. 17, no. 5, pp. 32–41, Sep./Oct. 2002.
[5] W. J. Clancey, M. Sierhuis, C. Kaskiris, and R. van Hoof, "Brahms mobile agents: Architecture and field tests," in *Proc. Human–Robot Interaction: Papers AAAI Fall Symp.*, (2002). [Online]. Available: http://www.aaai.org/Press/Reports/Symposia/Fall/fs-02-03.php
[6] R. Stocker, L. A. Dennis, C. Dixon, and M. Fisher, "Verification of Brahms human–robot teamwork models," in *Proc. 13th Eur. Conf. Logics Artificial Intell.*, 2012, pp. 385–397.
[7] R. Stocker, M. Sierhuis, L. A. Dennis, C. Dixon, and M. Fisher,, "A formal semantics for Brahms," in *Proc. 12th Int. Workshop Comput. Logic Multi-Agent Syst.*, 2011, pp. 259–274.
[8] J. Saunders, N. Burke, K. L. Koay, and K. Dautenhahn, "A user friendly robot architecture for re-ablement and co-learning in a sensorised home," in *Proc. 12th Eur. AAATE Conf.*, 2013, pp. 49–58.
[9] I. Duque, K. Dautenhahn, K. L. Koay, I. Willcock, and B. Christianson, "Knowledge-driven user activity recognition for a smart house—Development and validation of a generic and low-cost, resource-efficient

system," in *Proc. 6th Int. Conf. Adv. Comput.-Human Interactions*, Nice, France, 2013, pp. 141–146.
[10] D. Syrdal, K. Dautenhahn, K. L. Koay, M. Walters, and W. Ho, "Sharing spaces, sharing lives—The impact of robot mobility on user perception of a home companion robot," in *Proc. 5th Int. Conf. Soc. Robot.*, 2013, pp 321–330.
[11] U. Reiser, C. Connette, J. Fischer, J. Kubacki, A. Bubeck, F. Weisshardt, T. Jacobs, C. Parlitz, M. Hägele, and A. Verl, "Care-O-bot 3: Creating a product vision for service robot applications by integrating design and technology," in *Proc. IEEE/RSJ Int. Conf. Intell. Rob. Syst.*, 2009, pp. 1992–1998.
[12] U. Reiser, T. Jacobs, G. Arbeiter, C. Parlitz, and K. Dautenhahn, "Care-O-bot 3—Vision of a robot butler," in *Your Virtual Butler—The Making-of* (ser. LNAI), vol. 7407, R. Trappl, Ed. New York, NY, USA: Springer, 2013, pp. 97–116.
[13] A. Mohammed, U. Furbach, and F. Stolzenburg,, "Multi-robot systems: Modeling, specification, and model checking," in *Robot Soccer*. Rijeka, Croatia: InTech, 2010, pp. 241–265.
[14] A. Cowley and C. J. Taylor, "Towards language-based verification of robot behaviors," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 4776–4782.
[15] Y. Kouskoulas, D. W. Renshaw, A. Platzer, and P. Kazanzides,, "Certifying the safe design of a virtual fixture control algorithm for a surgical robot," in *Proc. Hybrid Syst.: Comput. Control*, 2013, pp. 263–272.
[16] A. Acquisti, W. J. Clancey, R. van Hoof, M. Scott, and M. Sierhuis, *Brahms Tutorial TM01-0002, Version 1.2*. Edison, NJ, USA: Agent iSolutions, Mar. 2011.
[17] M. Fisher, *An Introduction to Practical Formal Methods Using Temporal Logic*. New York, NY, USA: Wiley, 2011.
[18] M. Webster, C. Dixon, M. Fisher, M. Salem, J. Saunders, K. L. Koay, and K. Dautenhahn, "Formal verification of an autonomous personal robotic assistant," in *Proc. Formal Verification Modeling Human-Mach. Syst.: Papers AAAI Spring Symp.*, 2014, pp. 74–79.

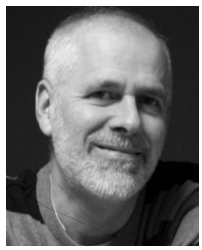
Matt Webster received the Ph.D. degree in computer science from the University of Liverpool in 2009. He is currently working as a Postdoctoral Research Associate with the Department of Computer Science, University of Liverpool, Liverpool, U.K. He is also working on the Trustworthy Robotic Assistants project funded by the Engineering and Physical Sciences Research Council. He is a Member of the Centre for Autonomous Systems Technology and the Logic and Computation Research Group with the University of Liverpool. His research interests include autonomous systems, formal verification, robotics, and unmanned aircraft.

Clare Dixon received the Ph.D. degree in Computer Science from the University of Manchester in 1995. She is a Senior Lecturer with the Department of Computer Science, University of Liverpool, Liverpool, U.K. Her research interests include formal specification and verification and temporal and modal theorem-proving techniques. She is a Member of the Logic and Computation Research Group, University of Liverpool, and she is a Member of the Centre for Autonomous Systems Technology, the Institute for Risk and Uncertainty, and is the Deputy Director of the Liverpool Verification Laboratory.

Michael Fisher received the Ph.D. degree in Computer Science from the University of Manchester in 1987. He is a Professor of computer science and Director of the Centre for Autonomous Systems Technology, University of Liverpool, Liverpool U.K. His research interests include logical methods in computer science and artificial intelligence, particularly temporal reasoning, programming languages, practical formal verification, and the development and analysis of autonomous and agent-based systems.



Maha Salem received the Ph.D. degree in Cognition and Robotics from the University of Bielefeld in 2012. She is a Postdoctoral Research Fellow with the Adaptive Systems Research Group, School of Computer Science, University of Hertfordshire, Hertfordshire, U.K. Her research interests include social human–robot interaction (HRI) with particular focus on safety aspects for the design of trustworthy robotic assistants. She is further interested in cross-cultural HRI as well as in multimodal interaction and nonverbal expressiveness based on the design and use of communicative robot gesture.



Joe Saunders is a Senior Research Fellow with the Adaptive Systems Group, University of Hertfordshire, Hertfordshire, U.K. His research interests include applying theoretical studies to practical real-world environments. He has recently carried out trials on end-user robot teaching in a care environment as part of the European Union ACCOMPANY project.

Kheng Lee Koay received the B.Sc. degree in robotics and automated systems and the Ph.D. degree from the University of Plymouth,, Plymouth, U.K., in 1997 and 2003, respectively.

He is currently a Senior Research Fellow with the Adaptive Systems Research Group, School of Computer Science, Faculty of Engineering and Information Sciences, University of Hertfordshire, Hertfordshire, U.K. His research interests include mobile robotics, human–robot interaction, social robotics, home companion, and agent migration.

Kerstin Dautenhahn (SM'13) received the Ph.D. degree in Biological Cybernetics from the University of Bielefeld in 1993. She is a Professor of artificial intelligence with the School of Computer Science, University of Hertfordshire, Hertfordshire, U.K., where she is a Coordinator of the Adaptive Systems Research Group. Her main research interests include human–robot interaction, social robotics, socially intelligent agents, and artificial life.

Joan Saez-Pons received the Ph.D. dgree in Automation and Robotics at Sheffield Hallam University in 2011. He is a Postdoctoral Research Fellow with the Adaptive Systems Research Group, University of Hertfordshire, Hertfordshire, U.K. His research interests are within the broad area of robotics, with particular interest in autonomous mobile robots, coordination and cooperation of multirobot systems, human–robot interaction, artificial intelligence, social robotics, and computer vision.