Towards Verification of Domestic Robot Assistants-Part 2

Clare Dixon

Department of Computer Science
University of Liverpool
cldixon@liverpool.ac.uk
www.csc.liv.ac.uk/∼clare
www.robosafe.org

Collaborators

Farshid Amirabdollahian[2,*]
Kerstin Dautenhahn[2,*]
Louise Dennis[1]
Kerstin Eder[3,*]
Michael Fisher[1,*]
Paul Gainer[1]
Dejanira Araiza Illan[3,*]
Kheng Lee Koay[2,*]
Anthony Pipe[3,*]
Maha Salem[2,*]
Joe Saunders[2,*]
Maarten Sierhuis[5]
Richard Stocker[1,4]
Matt Webster[1,*]
David Western[3,*]

1 University of Liverpool
2 University of Hertfordshire
3 Bristol Robotics Lab
4 Nasa Ames Research Centre
5 Nissan Research Centre
* Trustworthy Robotic Assistants Project

Talk Structure-Part 2

- Introduction
- Tools and Techniques
- Brahms
- Formal Semantics of Brahms
- Brahms to Promela
- Properties
- Discussion
- Conclusions

Introduction

- In the previous slides we showed how we modelled the Care-O-bot behaviours and carried out model checking.
- First we provide a quick detour into temporal theorem proving that might be useful in verifying user defined behaviours.
- Then we discuss an approach to verification via a tool called Brahms.

Verification of Added Behaviours I

- We are currently working with UoH to validate newly added behaviours.
- UoH have an interface (Teach-me) that allow the input of new personalised behaviours (with priority zero).
- These are constructed by selecting and combining values from existing primitives such as sensors, robot actions and timings.
- They would like to flag issues as conflict within the actions for example trying to move to two different places or say two things simultaneously.
- For example
  “If it is 2pm remind me to watch my favourite TV programme.”
  “If it is 2pm remind me to take my medicine.”

Verification of Added Behaviours II

- We are currently discussing what sort of *conflicts* should be flagged.
- Given that only one behaviour can execute at once this is more of a question of behaviours never being executed eg never reminding about taking the medicine.
- Thought is needed about how these issues should be reported back to the user.
- The Teach-me system allows potentially complex timing constraints which may be problematic for verification.
- One solution might be to use a model checking approach.
- Alternatively we could use a temporal theorem prover.

There Is More than Model Checking

- Although we have focused here on model checking there are temporal theorem proving tools.
- In particular at Liverpool we have developed resolution-based provers for PTL (trp++).
- Tableau calculi and their implementations also exist (which try to construct a model for the formula).
- Both tableau and resolution calculi are refutation based, i.e. to show a formula valid (i.e. it holds in all models) it is negated and the calculus applied.
- That is to show a specification of a system S implies a property P, i.e. $S \Rightarrow P$ is valid we negate and show $S \wedge \neg P$ is unsatisfiable (doesn't hold in any model).

The Resolution Procedure PTL

1. Translation to normal form - complex subformulae renamed using new propositions, and temporal operators reduced to a core set. Clauses hold at all reachable states.
2. Step resolution - similar to classical resolution.
3. Temporal resolution - identification of sets of formulae which together imply a $\Box$-formula for resolution with a $\Diamond$-formula.
4. The derivation of false means the set of clauses is unsatisfiable. If no new clauses can be derived the set of clauses is satisfiable.

Normal Form (SNF)

Formulae in normal form are of the form

$$\Box \bigwedge_{i} T_i.$$

where each T_i is known as a *clause* and must be one of the following.

$$\textbf{start} \Rightarrow \bigvee_{b=1}^{r} l_b \qquad \text{(an } \textit{initial} \text{ clause)}$$

$$\bigwedge_{a=1}^{g} k_a \Rightarrow \bigcirc \bigvee_{b=1}^{r} l_b \qquad \text{(a } \textit{step} \text{ clause)}$$

$$\bigwedge_{a=1}^{g} k_a \Rightarrow \Diamond l \qquad \text{(a } \textit{sometime} \text{ clause)}$$

Resolution Rules

- Initial resolution

$$[\text{IR}]\quad \frac{\begin{array}{lcl}\textbf{start} & \Rightarrow & (A \vee \neg p) \\ \textbf{start} & \Rightarrow & (B \vee p)\end{array}}{\begin{array}{lcl}\textbf{start} & \Rightarrow & (A \vee B)\end{array}}$$

- Step resolution.

$$[\text{SR}]\quad \frac{\begin{array}{rcl}X & \Rightarrow & \bigcirc(A \vee p) \\ Y & \Rightarrow & \bigcirc(B \vee \neg p)\end{array}}{\begin{array}{lcl}X \wedge Y & \Rightarrow & \bigcirc(A \vee B)\end{array}}$$

- Temporal resolution

$$\frac{\begin{array}{lcl}A & \Rightarrow & \bigcirc \Box p \\ C & \Rightarrow & \Diamond \neg p\end{array}}{\begin{array}{lcl}C & \Rightarrow & (\neg A)\, \mathcal{W}\, \neg p\end{array}}$$

We must find a set of step clauses that together imply $\bigcirc \Box p$ to apply this rule.

Other Rules

- Rewriting of clauses that give false in the next moment in time. (RW)

$$\{A \Rightarrow \bigcirc \textbf{false}\} \longrightarrow \left\{ \begin{array}{lll} \textbf{start} & \Rightarrow & \neg A \\ \textbf{true} & \Rightarrow & \bigcirc \neg A \end{array} \right\}$$

- Subsumption/simplification
- Termination

$$\begin{array}{lll} \textbf{start} & \Rightarrow & \textbf{false} \\ \textbf{true} & \Rightarrow & \bigcirc \textbf{false} \end{array}$$

Example: The Specification of the Moving Robot

We have the following clauses (S)

$$\begin{aligned} \textbf{start} &\Rightarrow \neg kitchen \\ send &\Rightarrow \bigcirc kitchen \\ kitchen \wedge \neg send &\Rightarrow \bigcirc kitchen \\ \neg kitchen \wedge \neg send &\Rightarrow \bigcirc \neg kitchen \end{aligned}$$

Assume we want to try prove the property (P)

$$\Diamond(send \wedge \bigcirc kitchen)$$

holds (i.e. $S \Rightarrow P$).
This should not be valid as we may never satisfy send.
We negate P and obtain

$$\Box(send \Rightarrow \bigcirc \neg kitchen)$$

Example: Sample Proof I

1.	$\textbf{start} \Rightarrow \neg kitchen$	[*Given*]
2.	$send \Rightarrow \bigcirc kitchen$	[*Given*]
3.	$kitchen \wedge \neg send \Rightarrow \bigcirc kitchen$	[*Given*]
4.	$\neg kitchen \wedge \neg send \Rightarrow \bigcirc \neg kitchen$	[*Given*]
5.	$send \Rightarrow \bigcirc \neg kitchen$	[*Given*]
6.	$send \Rightarrow \bigcirc \textbf{false}$	$[SR, 2, 5]$
7.	$\textbf{start} \Rightarrow \neg send$	$[RW, 6]$
8.	$\textbf{true} \Rightarrow \bigcirc \neg send$	$[RW, 6]$

Although we could apply other resolution steps we never obtain a contradiction.

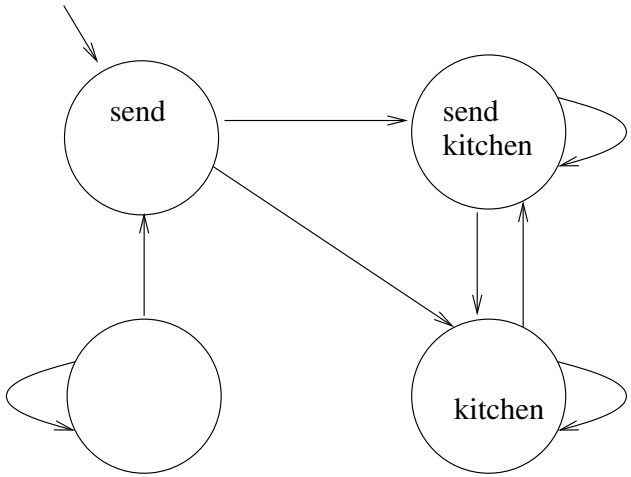
So the negated formula is satisfiable and the original is not valid.

Example: Sample Proof II

However if we add that *send* holds initially we can derive a contradiction.

1*a*.	**start** $\Rightarrow$ *send*	[*Given*]
	...	
7.	**start** $\Rightarrow \neg$*send*	[*RW*, 6]
8.	**true** $\Rightarrow \bigcirc \neg$*send*	[*RW*, 6]
9.	**start** $\Rightarrow$ **false**	[*IR*, 1*a*, 7]

This shows (when *send* holds initially) $S \Rightarrow \Diamond(send \wedge \bigcirc kitchen)$ is valid.



Verification of the Care-O-bot via Brahms

- We next discuss an approach to verification via a tool called Brahms.
- Our previous approach to verifying the Care-O-bot via direct translation of behaviours to NuSMV isn't very general for example it doesn't help given a different robot using other ways of controlling the robot.
- Additionally, whilst we have considered the decision making of the robot the person has not been modelled (or only a very simple model has been considered).
- With robotic assistants we may need a better representation of the person so we can reason about interactions between the robot and the person.

Brahms

- To address these issues we use Brahms, a language explicitly developed to model human-robot-agent teamwork that is potentially applicable to a wider range of human robot scenarios.
- Brahms is a simulation/modelling language (rather than a programming language) in which complex human-robot-agent teamwork scenarios can be described and is based on the concept of rational agents.
- The system has been extensively and successfully used within NASA for the modelling of astronaut-robot planetary exploration teams.
- We use Brahms to capture the key interactions and behaviours of any human-robot-agent scenario.
- We describe a tool (developed by Stoker) that can be used to translate Brahms models into input to a model checker.

Brahms in Action

Examples of uses of Brahms are:-

- to model the NYNEX telephone exchange;
- simulation of the moon Apollo Lunar Surface Exploration;
- to model and simulate the Mars Exploration Rover mission operation work processes;
- simulation of crew members on board the International Space Station (ISS);
- OCAMS: Orbital Communications Adapter Mirroring System is a multi-agent software system that helps a flight controller at NASA to manage interactions with the file system on board the ISS. Brahms was used to model the human behaviour of the officers' work practice.

Brahms in Action: Victoria Crater

- Nasa's Victoria mission is a semi-autonomous robotic mission to study rocks and soil in Mars's Victoria Crater.
- However, in the crater, power is a problem because it is too dark for solar power and the robot would need to move to a sunny spot to re-charge its batteries.
- Brahms has been used to model this including moving around the crater, signaling to base, energy, heating, drilling, picking up samples, movement paths etc.

Brahms: Overview

- Brahms is designed to model both human and robotic activity using *rational agents* (autonomous entities, able to make their own choices and carry out actions in a *rational* and *explainable* way).
- It allows the representation of artifacts, data, and concepts in the form of classes and objects.
- Both agents, representing autonomous entities, and objects can be located in a model of the world giving agents the ability to detect both objects and other agents, to have beliefs about the objects/agents, and to move between locations.
- When Brahms executes, it produces a simulation of the humans, robots and agents interacting within some model.
- Among other things it tracks the time taken over tasks.

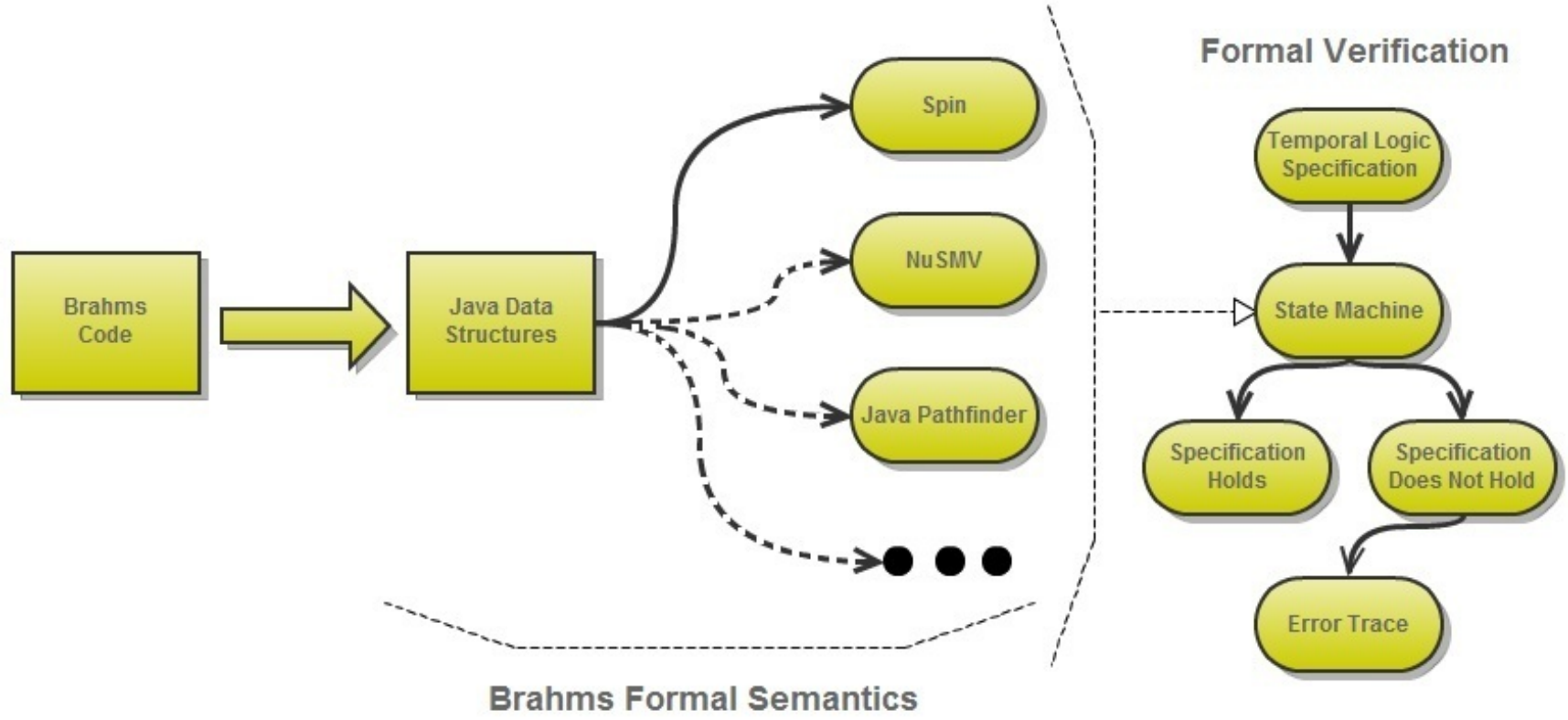
Advantages of Brahms for Modelling the Scenario

- Workframes (the Brahms mechanism for representing representing work processes) have a guard-action structure that can be used to represent behaviours.
- Workframes have priorities to allow selection between them (similar to the behaviours).
- Workframes can be interrupted if the guards to higher priority workframes become satisfied (similar to the interruption of behavious).
- Activities (the Brahms mechanism to represent robot actions) have durations that can be used to model the timing aspects.
- The geography model from Brahms can be used to represent the locations.

Issues with Using Brahms

- Whilst being a useful tool for modelling human-robot-agent teamwork there was no formal semantics, Brahms is proprietary software and, as such, we had no access to the source code.
- To apply model checking we first had to develop a formal semantics for Brahms.
- Using this, we then had to develop a tool that translates Brahms models into an intermediate representation that can be translated into input to a number of model checkers.
- Our tool currently translates this intermediate representation into `Promela`, the input language of the model checker `Spin`.

Overview of the Approach I



Overview of the Approach II

- A Brahms model is interpreted using the formal semantics to generate a `Java` representation of the semantic structures relevant to this scenario.
- These `Java` data structures can then be used to generate `Promela` processes for each human, robot, and agent in the scenario which are suitable for input into the `Spin` model checker.
- We also translate our requirements into `Promela` properties, and then are able to apply the `Spin` model checker to verify that the required properties hold.
- This tool thus provides us with a mechanism for formally verifying properties of human-robot-agent teamwork scenarios modelled using Brahms.

Brahms: Components

- Agents and Objects - modelling autonomous entities and objects: agents react to their internal beliefs; objects to react to external factors.
- Groups and Classes - allow hierarchical structuring of agents and inheritance.
- Attributes, Relations, Beliefs and Facts - attributes are characteristics of agents/objects, relations show the relationships to other agents/objects. Beliefs relate to attributes and relations. Facts represent the real value.
- Geography - representation of the location of the agents represented hierarchically.
- Workframes and Thoughtframes - represent work or thought processes needed to complete a task.

Brahms Simulations

- 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 in 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 with the shortest duration action finishes.
- By doing this the scheduler maintains synchronicity throughout a simulation, ensuring that the order in which the agents execute does not affect the outcome of the simulation.

From the Robot House Scenario to Brahms

- We identify five agents in the scenario
  - Person
  - Robot
  - RobotHouse
  - TheEnvironment
  - Campanile_Clock (this keeps track of time)
- We represent the robot behaviours as a set of `IF a THEN b` rules (as previously).
- These are then translated into Brahms workframes using the construct `when a do { b }`.
- Here `a` is termed the *guard* and the workframe will only be eligible for execution if the guard is satisfied.

Brahms: Example Agent

```
agent Robot{
    location: chair;
    attributes:
        .......
        public int timeSinceMedANotification;
        public boolean trayIsRaised;
        public boolean trayIsLowered;
        public boolean trayIsEmpty;
        public int lightColour;
        ........
    initial_beliefs:
        (current.colourWhite = 0);
        (current.colourYellow = 1);
         ......
        (Person.location = chair);
        (robotHouse.doorbellRang = false);
        ..........
    ......
    }
```

Brahms: Sample Activities

Activities are part of the robot agent. The main type of activities are

- primitive activities: to model basic actions with a duration;
- move activities: to change the agents' location changing the beliefs other agents have about its location in its old and new location;
- communication activities: for passing messages between agents.

```
activities:
    move moveToLivingRoom() {
        location: LivingRoom;
}
    primitive_activity sayAndWait(){
        max_duration: 1;
}
```

Brahms: Workframes

- Workframes and thoughtframes govern how agents, objects and the world vary over time.
- Workframes represent the work processes involved in completing a task.
- Thoughtframes represent the reasoning process carried out on the current beliefs.
- They contains a sequence of activities/actions (workframes only) and belief/fact updates (termed *concludes*).
- Workframes can detect (using *detectables*) changes in the environment, update an agent's beliefs accordingly and then decide whether or not to continue executing.
- Workframes/thoughtframes have priorities showing their relative importance that allow the scheduler to select the one(s) with the highest priority.

Modelling Behaviours

- In general, Robot House behaviours 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 interaction with the user via the GUI.
- For example, in the `S1-alertFridgeDoor` behaviour the robot asks the person via a user interface whether it should go to the kitchen or wait where it is.
- These options are communicated to the person using the `announceQueryToUser()` activity and the result of these choices is modelled using a Brahms workframe both within the robot agent.

Brahms: Example Workframe I

```
workframe wf_alertFridgeDoor {
                    // cob rule 5, before user response
  repeat: true;
  priority: 10;

  when(knownval(robotHouse.fridgeFreezerIsOn = true) and
    knownval(robotHouse.fridgeFreezerIsOnTime > 30) and
    knownval(robotHouse.goalFridgeUserAlerted = false) and
    knownval(current.userQueried = false))
  do{
    conclude((current.lightColour = current.colourYellow));
    moveToLivingRoom();
    waitUntilInLivingRoom();
    conclude((current.lightColour = current.colourWhite));
    waitForLightColourChange();
    sayAndWait();   // "The fridge door is open!"
```

Brahms: Example Workframe II

```
        conclude((current.goalGoToCharger = false));
        conclude((current.goalGoToTable = false));
        conclude((current.goalGoToSofa = false));
        conclude((current.queryUserOption1 =
                          current.activityGoToKitchen));
        conclude((current.queryUserOption2 =
                          current.activityWaitHere));
        conclude((current.queryUser = true));
        announceQueryToUser();
        conclude((current.userQueried = true));
        conclude((current.goalFridgeUserAlerted = true));
}
```

Brahms: Example Workframe III

This represents the workframe once the person has selected the option *go to kitchen*.

```
workframe wf_alertFridgeDoorGoToKitchen {
  // cob rule 5, after user response, option "go to kitchen"
  repeat: true;
  priority: 10;

  when(knownval(Person.userRespondedToQuery = true) and
  knownval(Person.queryResponse = current.activityGoToKitchen)
  do{
    moveToKitchen();
    conclude((Person.userRespondedToQuery = false));
    conclude((current.userQueried = false));
    }
}
```

The Scenario Modelled

In the modeling in the previous translation to NuSMV no person was explicitly modeled and the the environment sensors such as sofa sensors, fridge door sensor, the doorbell being pressed were allowed to change non-deterministically.

Here we model a scenario from 12.00 to 18.00.

At any point in the day the person can choose to

- sit down and watch TV;
- move into the living room area;
- move into the kitchen (eg to prepare food);
- send the Care-O-bot into the kitchen;
- send the Care-O-bot to the living room;
- no nothing.

At 5pm the person will need to take their medication.

Modelling the Scenario in Brahms

- These choices are modelled using workframes within the person agent.
- Each workframe has a priority. The highest priority workframe is executed, and a belief is modified within the agent (using `conclude`).
- 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 or the Robot House agent (depending on the workframe) which causes these agents to know that the person has done something, eg, 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.

Overview of the Semantics

Before we could perform any formal verification we first had to develop an operational semantics for Brahms.

The semantics is split into two groups of rules:

- the first concerns the global system and represents the functioning of the scheduler;
- the second acts upon individual agents.

Rules for the scheduler act as global arbiters, instructing agents when to start, suspend, or terminate.

Rules for the individual agents choose activities and update beliefs, etc.

Brahms Formal Semantics

A Brahms semantic 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 is the agent currently under consideration;
- B_ξ is the belief base of the system (used to synchronise the agents, e.g. agent i's next event finishes in 1000 seconds);
- F is the set of facts in the environment (e.g. the `doorbell` is set to false i.e. hasn't rung); and
- T_ξ is the current time of the system.

Operational Semantics Rules

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

Here,

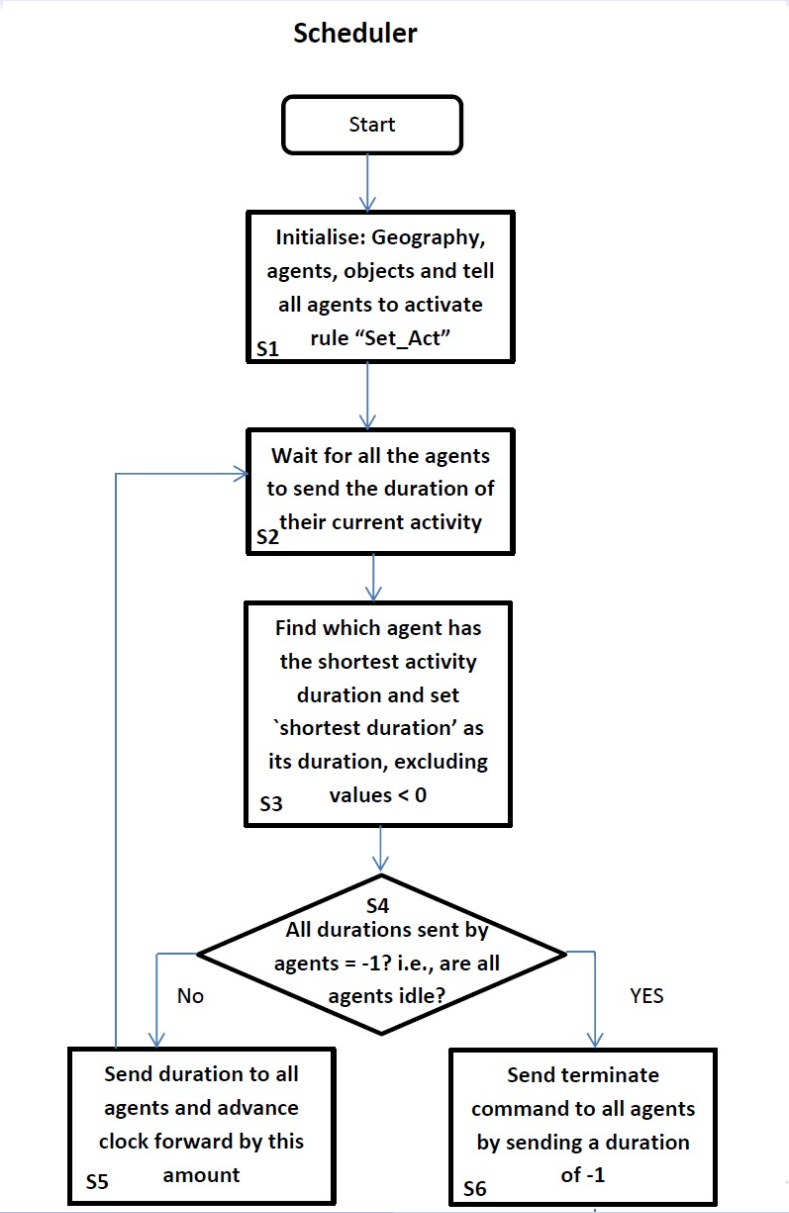
- '*ConditionsRequiredForActions*' refers to conditions which must hold before this rule can be applied,
- '*ActionsPerformed*' represents changes to the agent, object or system state which, for presentational reasons, can not be easily represented in the resulting tuple.

It is assumed that all agents and objects can see, and access, everything in the overall system's tuple, e.g. T_ξ.

Overview of the Semantics

- An agent first processes *thoughtframes*, then *detectables* (both may update beliefs), and then *workframes* which may initiate activities.
- There are rules that represent how an agent selects a thoughtframe based on the thoughtframe guard conditions and priority.
- 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 (simulation) clock forward accordingly, and hands control to the rules governing the behaviour of the individual agents again.

Overview of the Scheduler Semantics



Semantic Rules: Timing

The timing in Brahms works by the use of a global system clock coupled with agents having their own internal clocks.

The system scheduler (ξ) asks each agent (Ag_i) how long each of their activities are, finds the time of the shortest activity and then tells each agent to move their clock forward by this time.

$$\frac{Ag_0 \xrightarrow{LocalClock+t} X, X \xrightarrow{Choice} Ag_0' \quad \cdots \quad Ag_n \xrightarrow{LocalClock+t} X, X \xrightarrow{Choice} Ag_n'}{\xi \xrightarrow{LocalClock+t} \xi'}$$

Scheduler Semantics Rules

There are three scheduler semantics rules

- Sch_run: deals with starting the agents running (box S1).
- Sch_rcvd: receives activity from agents and advances the global clock (box S3).
- Sch_Term: when all agents are in an idle state the global clock is set to -1 to signal termination (box S6).

Semantic Rules: Scheduler Semantics

RULE: Sch_rcvd

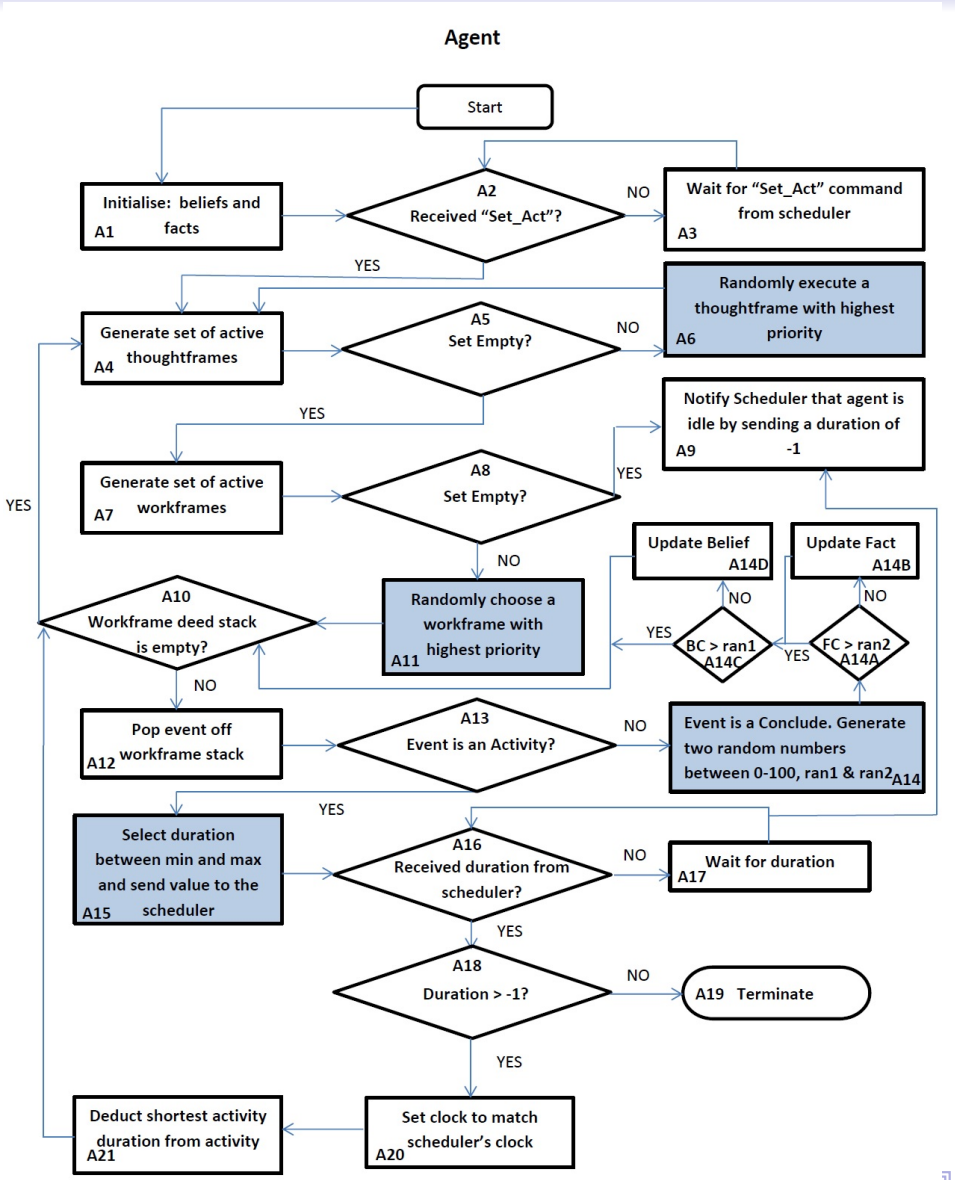
$$\langle Ags, ag_i, B_\xi, F, T_\xi \rangle$$

$$\frac{T_{\xi'} = T_\xi[T_\xi / T_\xi + MinTime(\forall ag_i | T_i \in B_\xi)]}{\forall ag_i \in Ags | stage \in \{Pop\_PA*, Pop\_MA*, Pop\_CA*)\} \vee idle, (T_\xi \neq -1)}$$

$$\langle Ags, ag_i, B_\xi, F, T_{\xi'} \rangle$$

- The rule receives the activity durations from all agents.
- If all agents are in a waiting or idle state then the Scheduler will check the end time for all agents' activities, calculate the smallest value and set its time to this.
- To activate all the agents need to be at the stage where the rules *Pop_PA*$*$, *Pop_MA*$*$ or *Pop_CA*$*$ relating to primitive activities, move activities or communication activities applicable (where * represents a wild card).

Overview of a Brahms Agent's Semantics



Representation of the Agent the Scheduler Semantics

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

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

- ag_i is the identification of the agent;
- $\mathcal{T}$, the current thoughtframe;
- $\mathcal{W}$, 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 set of thoughtframes; and
- *WF*, the agent's set of workframes.

Here *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 Agent Semantics

The Brahms system operates on a simple cycle of handling:

$$\textit{Thoughtframes} \rightarrow \textit{Detectables} \rightarrow \textit{Workframes}$$

The agents semantics rules are split into rules relating to

- the start of each cycle: Set_* rules
- thoughtframes: Tf_* rules
- workframes: Wf_* rules
- detectables: Det_* rules
- variables (provide quantification in Brahms): Var_* rules
- popstack (relating to timing of activities): Pop_* rules

Semantic Rules: Agent's Semantics

RULE: Pop_PASend

$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$

$$\xrightarrow[T_\xi = T_i \wedge \beta = \langle \beta_d, Prim\_Act^t ; \beta_{ins} \rangle]{B_\xi = B_\xi \cup (T_i = T_i + t)}$$

$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$

Note $\langle \beta_d, Prim\_Act^t ; \beta_{ins} \rangle$ represents a workframe β with header information β_d, $Prim\_Act^t$ is a primitive activity of duration t and β_{ins} is the stack of instructions the workframe is to perform.

The agents use this rule send the duration of their next event to the scheduler.

From Brahms Models to Input to a Model Checker

- We first translate into a intermediate representation using Java Data Structures
- This facilitates the translation into the input languages of a variety of model checkers for example allowing us to check probabilistic and epistemic properties as well as purely temporal ones.
- Initially we chose the `Spin` model checker as this is a widely used, effective and stable system and its input language, `Promela`, is a higher level language than many model checking input languages, making it easier to represent the Brahms agents/objects.
- `Spin` also has the ability to run `Promela` code as a simulation, making comparison with the Brahms simulation possible.

Intermediate Java Representation

- The `Java` classes developed capture the structural aspects of the Brahms models required by the operational semantics of Brahms.
- This is a syntactic transformation of the Brahms model and its underlying elements into Java data structures.
- The `Java` class `MultiAgentSystem` is used to represent the 5-tuple from the operational semantics for Brahms models.
- The `agent` class represents the 9-tuple from the operational semantics for agents.

Representation of Workframes

- Although `Promela` is an appropriate input language for model checking it has more restrictive data types and control structures than a typical high level programming language.
- Hence a one to one correspondence between the `Java` data structures and associated rules in the operational semantics and `Promela` was not possible.
- Arrays are the main data structure available in `Promela` and so an array-based representation was used for most of the `Java` data structures.
- The `Promela` translation has a separate process (termed *proctype* in `Promela`)
- For instance, the agent's current workframe is represented as a one-dimensional array and treated as a stack.

Representation of Workframes

Index	Description
0	Workframe ID number
1	Boolean guard condition, e.g.,1 = workframe is active
2	Priority of the workframe
3	Repeat, e.g.,0 = delete, 3 = always
4	Boolean to flag a communication or move activity
5	Boolean to flag the workframe is in impasse
6	Last deed on stack
...	...
...	...
i	Top deed on stack

- Sets of workframes for an agent are two dimensional arrays.
- One dimension captures the details of a workframe and the other dimension shows the workframes for that agent.

Representation of Thoughtframes, Beliefs and Facts

- Thoughtframes are represented in a similar way to workframes.
- Relationships between agents or objects are also modelled using two dimensional arrays.
- Beliefs and facts in Brahms are tied to the attributes and relations of an agent, eg, the robot believes the person's location is on the sofa.
- To model this in Promela every agent is assigned a belief about every attribute, even if it does not own that attribute.
- This is modelled in Promela using a one dimensional array for each attribute.
- Facts are modelled in a similar way.

Correctness Issues

We aim to verify Brahms programs, however we are not using the *actual* Brahms interpreter.

We need to show that a program that has been declared correct by our system would actually behave correctly if executed in the existing Brahms simulation engine.

There are several aspects to this:

- the correctness of our Brahms *semantics*;
- the correctness of our *translation* from the Brahms semantics into Java data structures; and
- the correctness of the *translation* into `Promela`.

Correctness of the Brahms Semantics

- As Brahms is proprietary software we had no access to the source code.
- We developed the semantics in collaboration with the Brahms designer.
- The semantics were discussed/confirmed with NASA engineers who have used used Brahms in a number of projects.
- These semantics were later used by NASA in their own work.

Correctness of our Translation into Java Data Structures

- The translation takes the components of Brahms and translates them into corresponding Java data structures.
- The Java classes model the key aspects of the Brahms language capturing elements of the Brahms semantics eg the key aspects were the `MultiAgentSystem` is used to represent the 5-tuple from the semantics and `agent` class represents the 9-tuple for agents.
- Code inspection was used to provide an informal correctness justification.
- The intermediate representation was also used by NASA in their own work.

Correctness of the Translation into `Promela`

- A direct comparison between the `Spin` simulation and the Brahms simulation was carried out.
- Model checking of properties that are or are not expected to hold was carried out and any discrepancies investigated.

Properties I

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.

$$\Box B_{Care-O-bot}(B_{Person} guiGoToKitchen) \Rightarrow \Diamond B_{Care-O-bot}(location = Kitchen)$$

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 into there.

$$\Box(B_{Care-O-bot}(B_{Person} guiGoToSofa) \Rightarrow \Diamond B_{Care-O-bot}(location = LivingRoomSofa))$$

Properties II

It is always the case that if the Robot House believes that the sofa seat is occupied, and if the Robot House believes that the television wattage is higher than 10 watts, then eventually the Care-O-bot will move to the living room sofa and ask the person if they want to watch the television with the Care-O-bot.

$$\begin{aligned}&\Box((B_{RobotHouse}\, sofaSeatOccupied \wedge \\ &B_{RobotHouse}\, televisionWattage > 10) \\ \Rightarrow &\Diamond(B_{Care-O-bot}\, location = LivingRoomSofa \wedge \\ &B_{Care-O-bot}\, askedToWatchTV))\end{aligned}$$

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

$$\begin{aligned}&\Box(B_{Campanile_C lock}\, time = 5pm \\ \Rightarrow &\Diamond B_{Care-O-bot}\, medicineDue)\end{aligned}$$

Results

Property	States	Depth	Memory(MB)	Time(s)
1	652,573	46,617	10,132	20.7
2	652,573	46,617	10,132	20.7
3.	746,479	53,009	11,596	30.7
4.	652,573	46,617	10,132	20.3

The formal verification was carried out on an eight-core Intel® Core™ i7-3720QM CPU (2.60GHz) laptop with 16 GB of memory running Ubuntu Linux 12.04 LTS

Property 3 produced a slightly more complex automaton and therefore required more resources to verify.

Performance I

- As usual with model checking the number of states produced is a limiting factor to the size of the models that can be checked.
- The tool is a prototype and could be made more efficient.
- The performance of the translator was considered by analysing different aspects of the system.
- The main issues that increase the number of states are adding agents, adding workframes and thoughtframes, workframes with no activities, communication and the number of activities and concludes (updating beliefs and facts) for an agent.
- Issues with workframes with no activities need more investigation but was thought to relate to checks whether to suspend the current workframe being carried out unnecessarily.

Performance II

- Similarly the communication issues relating to the use of *collectall* (syntax relating to quantification meaning work on all objects/agents that satisfy a guard condition at once, rather than *forone* select one or *foreach* work on all one after another) needs more investigation.
- Regarding the increases relating to workframes and thoughtframes it was identified that better use could be made of *deterministic wrappers* to wrap multiple states into one.
- The addition of *deterministic wrappers* to group several lines of code achieved a state space reduction of 25-33%.
- It was thought that a code re-organisation would help with further state space gains in this way.

Use of the Translator in Other Work

- Researchers at NASA/Middlesex have developed an alternative tool for verifying Brahms models based on our work.
- It uses our formal semantics for Brahms and the translation into the intermediate Java data structures.
- The latter is converted into an executable form and then uses Java Pathfinder to produce a state model (termed the *MAS connector*).
- The state model can then be converted into standard model checkers such as `Spin`, `NuSMV`, or Prism.

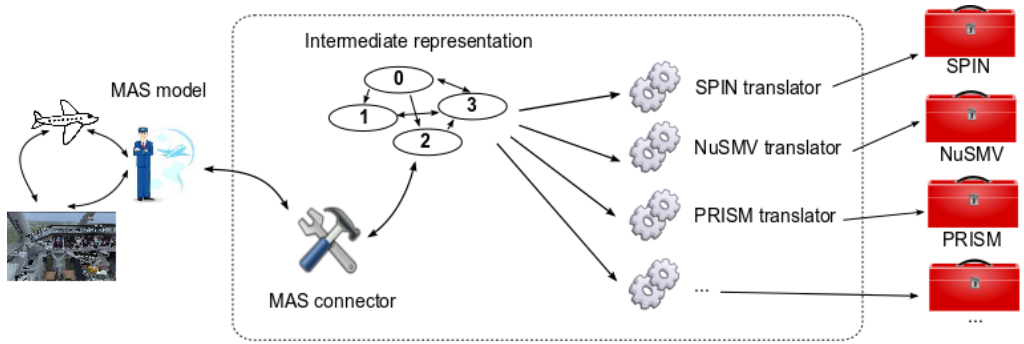


Image from Hunter/Raimondi/Rungta/Stocker.

Discussion: Modelling in Brahms

- The approach via Brahms potentially enables re-use of the translator for other robotic systems.
- However they must first be modelled in Brahms.
- As it was a first attempt the Brahms model we developed here doesn't capture the behaviour priority or interruptions as described previously.
- However it could be improved to capture this using the priorities of workframes.
- Interruptions can be modelled using the Brahms priorities and guards and by aborting the current workframe.
- We can model the timings using Brahm's `max_duration` syntax within activities.
- This is part of our current work.

Discussion: Efficiency and Correctness

- To follow this route we first had to develop a semantics for Brahm's semantics which was non-trivial.
- As mentioned previously, the correctness of the translation has not been shown formally (and cannot be done without access to the source code).
- But similarly the correctness of the hand-crafted translation to NuSMV, or automatic translator via the `CRutoN` was not formally proved.
- Whilst the state explosion problem will always be an issue it is thought that some re-organisation of the code to make better use of deterministic wrappers and further investigation into the previously mentioned issues would improve the tool's performance.

Back to the Robot House: UoH

One experiment that was carried out considered trust in the robot in two scenarios where the robot appeared faulty or not.

The scenario related to a situation where the householder had been called away.

In the *faulty* scenario the robot moved in an erratic manner, didn't respond correctly when asked to play a certain type of music etc.

In both scenarios the person was asked to carry out a task with the robot (setting the table).

As part of this they were asked to throw some personal letters to the householder away, pour juice into a plant and login to the householder's laptop, asking whether they had ever read someone else's emails.

Experiments in the Robot House: UoH

The experiment was carried out with the Sunflower robot in the robot house with 40 participants.

Participants took part in an interaction with the robot, completed a questionnaire afterwards and took part in an interview.

The findings suggest that although errors in a domestic robot's behavior are likely to affect participant's perception of its reliability and trustworthiness, this doesn't seem to influence their decisions to comply with instructions (or not).

Their willingness to comply with the robot's instructions seem to depend on the nature if the task, in particular, whether its effects are irrevocable.

Concluding Remarks

- Personal assistant robots are in development.
- We discussed our experiences with applying formal verification to a robot assistant in the robot house at UoH.
- We developed a by hand translation and an automatic translator from the sets of behaviours into input to a model checker as well as a translator from an simulation modelling language Brahms.
- Such verification results provide a route towards proving safety requirements to convince users of trustworthiness.
- These results should be used along with techniques such as simulation based testing and experiments with real people to give more confidence in reliability, safety and trustworthiness of robotic assistants.

Papers

Salem, M., Lakatos, G., Amirabdollahian, F. and Dautenhahn, K.
Would You Trust a (Faulty) Robot? Effects of Error, Task Type and Personality on Human-Robot Cooperation and Trust.
Proceedings of the 10th ACM/IEEE International Conference on Human-Robot Interaction (HRI) 2015.

Stocker, R., Sierhuis, M., Dennis, L., Dixon C., and Fisher M.,
A Formal Semantics for Brahams.
Proceedings of the 12th International Workshop on Computational Logic in Multi-Agent Systems. LNCS, Springer. 2011

Stocker, R., Dennis, L., Fisher M. and Dixon C.
Verification of Brahams Robot-Human Teamwork Models
Proceedings of the 13th European Conference on Logics in Artificial Intelligence. LNAI, Springer. 2012

Webster, M., Dixon C., Fisher M., Salem M., Saunders J. Koay K.L., and Dautenhahn, K.
Formal Verification of an Autonomous Personal Robotic Assistant,
Proceedings of Workshop on Formal Verification in Human Machine Systems (FVHMS), AAAI, 2014