Verifying BDI Agents in Dynamic Environments

Blair Archibald, Muffy Calder, Michele Sevegnani, Mengwei Xu
School of Computing Science, University of Glasgow, Glasgow, UK
{blair.archibald, muffy.calder, michele.sevegnani, mengwei.xu}@glasgow.ac.uk

***Abstract*—The Belief-Desire-Intention (BDI) architecture is a popular framework for rational agents, yet most verification approaches are limited to analysing the behaviours of an agent in a subset of all possible environments. However, in practice, BDI agents operate in dynamic environments where the exact occurrence of external changes is difficult to predict. For safety/security we need to assess whether the agent behaves as required in all circumstances. To address this, we define environments, accounting for both sensor information about physical changes and new tasks to be completed, as a non-deterministic finite-state automata. We give an environment-enabled extension to the Conceptual Agent Notation (Can) language including an executable semantics via an encoding to Milner's bigraphs and the BigraphER tool. We illustrate the framework through a simple Unmanned Aerial Vehicle (UAV) example that is verified using mainstream tools including PRISM model checker. Results show our approach can automatically identify agent design flaws to aid agent programmers in design, debugging, and analysis.**

***Index Terms*—BDI Agents, Formal Methods, Environments**

I. Introduction

Belief-Desire-Intention (BDI) is one of the most popular agent development frameworks and forms the basis of many agent-oriented programming languages including AgentSpeak [1], Can [2], and 3APL [3]. (B)eliefs represent what the agent knows, (D)esires what the agent wants to bring about, and (I)ntentions the desires the agent is currently acting upon. BDI features a collection of mature software and platforms including JACK [4] and Jason [5].

BDI agents rarely stand alone but instead exist in an environment—a source of information that the agent has been designed to understand—and operate by means of a reasoning cycle (see in Fig. 1) with three consecutive steps: (1) *perceive*, (2) *deliberate* and (3) *act*. The first step is to perceive the environment to update the agent's beliefs, the second to deliberate to determine, for example, what plan to select under current beliefs (specified by language semantics), and the third to act on the external environment by executing actions of the selected plans. This approach facilitates practical agents that effectively interact with their environments. However, designing agents that ensure correct behaviours in *all* possible environments is very difficult. As agent systems (*e.g.* robotic systems [6]) are increasingly employed in many real-life (*e.g.* domestic and industrial) settings, we must tackle this problem as it is crucial to analyse agent behaviours under all circumstances.

To illustrate the issues, we consider scenarios from Unmanned Aerial Vehicles (UAVs). In searching operations, UAVs patrol a designated area to identify objects of interest, *e.g.* missing persons, returning to base when detection is successful. During the mission, UAVs are expected to interact with the physical world. For example, UAVs should activate parking mode if the weather becomes harsh. The changes of these physical attributes (*e.g.* the weather and objects available to be detected) can happen at any time during UAV's mission and all combinations of these changes may produce complex behaviours. Therefore, we need mechanisms (*e.g.* verification techniques), involving an automated exhaustive analysis, in order to assess whether agents will always behave correctly under any environmental conditions.

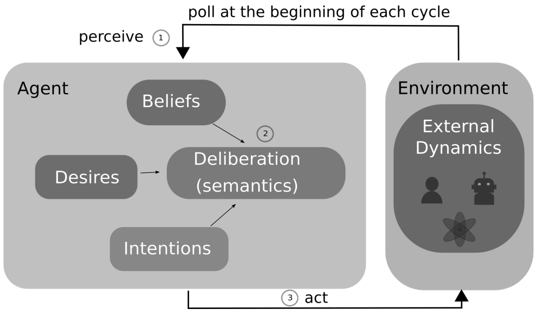

Figure 1: Agent interactions with an environment consisting of external physical changes due to *e.g.* natural phenomena. Numbers give the order of operation. All information is fed into the belief base.

Traditionally the analysis of agent behaviours is carried out by computational simulations. For example, the BDI platform Jason supports simulated environments that provide certain services to the agent (the ability to perceive and take action). Although computational simulations are quick to run, they are generally non-exhaustive in terms of agent behaviours in a given environment. As such, *actual* agent behaviours in deployment may not be the seen in simulation and rare cases may not manifest despite many simulation runs. To analyse all possible runs of agent behaviours in one environment, formal verification can be applied to build a mathematical model of the agent system [7]. However, if the agents are to be used in safety-critical areas, or where agent mistakes might involve financial penalties, this approach guarantees very little about agent behaviours in *actual* environments (which are difficult to accurately predict at design stage). To provide stronger guarantees, we instead assess *all* possible agent behaviours in *all* possible environments. A graphical comparison between these approaches is in Fig. 2.

We provide a verification framework based on Bi-

DOI reference number: 10.18293/SEKE2022-149

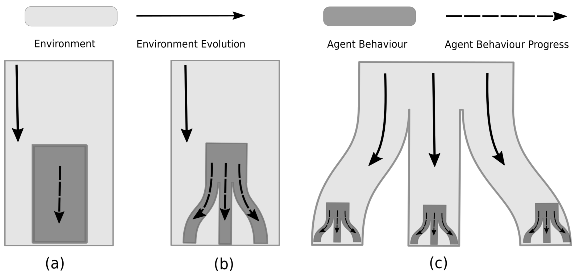

Figure 2: Approaches to analyse agent behaviours in environments. (a) Simulation: one run of agent behaviour in one environment; (b) existing verification approaches: all possible agent behaviours in one environment; (c) Proposed approach: all possible agent behaviours in all possible environments.

graphs [8]—a graph-based universal modelling formalism—that models both BDI agents and environments. We build on previous work [9] on a bigraph encoding of CAN semantics [2] (that includes classical BDI features and advanced features such as declarative goals), that provides an *executable* semantics for BDI agents operating in a (dynamic) environment. Verification is achieved through mainstream software tools including BigraphER and PRISM [10], and demonstrated by verifying several properties of UAVs.

As the external environment *with respect to an agent* may change while the agent is reasoning, the key of our approach is to provide a formalisation of external dynamics that summarises the main environment changes (due to *e.g.* natural phenomena) over each of the reasoning cycle. In other words, how the external environment has been updated in reality over each cycle is subject to the nature of the environment and what matters is that the agent can access the final effects of what has changed over this cycle. By focusing on reasoning cycles, we remain agnostic to the actual time required for each cycle, which can be difficult to anticipate at design stage due to the delay or variation in real process deployed on the hardware. This approach offers a feasible way to specify the dynamics of external environments while enabling practical verification by avoiding both state explosion and difficult time synchronization problem between agent reasoning and environment simulation when using real-time.

We make the following research contributions:

- We formalise the dynamics of external environments due to *e.g.* natural phenomena and other agents as a finite non-deterministic finite-state automata.
- We provide a verification framework for environment-enabled CAN agents via bigraphs that guarantees expected agent behaviours under all possible environments.
- We showcase our formal verification approach and illustrate how it can improve the design of the BDI agents.

The paper is organised as follows. In Section II we review BDI agents and Bigraphs. In Section III we define the framework for agents in dynamic environments. In Section V we demonstrate our approach using a UAV example. We discuss related work in Section VI and conclude in Section VII.

II. BACKGROUND

A. BDI Agents

The CAN language formalises classical BDI agents consisting of a belief base $\mathcal{B}$ and a plan library Π. The belief base $\mathcal{B}$ is a set of formulas encoding the current beliefs from a language $\mathcal{L}$ and has belief operators for entailment (*i.e.* $\mathcal{B} \models \varphi$), and belief atom addition (resp. deletion) $\mathcal{B} \cup \{b\}$ (resp. $\mathcal{B} \setminus \{b\}$)[1]. A plan library Π is a collection of plans of the form $e : \varphi \leftarrow P$ with e the triggering event, φ the context condition, and P the plan-body. Events can be either external or internal (*i.e.* sub-goals that the agent tries to accomplish). We also use E to denote the set of events (i.e. the triggering event) in the plan library. The language used in the plan-body is defined by $P = nil \mid act \mid e \mid P_1; P_2 \mid P_1 \triangleright P_2 \mid P_1 \parallel P_2 \mid e : (\!|\varphi_1 : P_1, \cdots, \varphi_n : P_n|\!) \mid goal(\varphi_s, P, \varphi_f)$ with nil an empty program, act a primitive action, and e an internal event. In addition, we use $P_1; P_2$ for sequence, $P_1 \triangleright P_2$ to first try P_1 and use P_2 in case of failure, and $P_1 \parallel P_2$ for interleaved concurrency. A set of relevant plans (those that respond to the same event) is denoted by $e : (\!|\psi_1 : P_1, \cdots, \psi_n : P_n|\!)$. A goal program $goal(\varphi_s, P, \varphi_f)$ states that the declarative goal φ_s should be achieved by repeatedly executing P, failing if φ_f holds and exiting successfully if φ_s holds (see [11] for full details). The action act is in the form of $act : \varphi \leftarrow \phi^-; \phi^+$ where φ is a precondition, and ϕ^- and ϕ^+ are the sets of belief atoms to be deleted and added after the action is executed.

CAN semantics is specified by two types of transitions. The first, denoted $\rightarrow$, specifies *intention-level* evolution on configurations $\langle \mathcal{B}, P \rangle$ where $\mathcal{B}$ is the belief base, and P the plan-body currently being executed. The second type, denoted $\Rightarrow$, specifies *agent-level* evolution over $\langle E^e, \mathcal{B}, \Gamma \rangle$, detailing how to execute a complete agent where E^e is the set of pending external events to address (a.k.a. desires), $\mathcal{B}$ the belief base, and Γ a set of partially executed plan-bodies (intentions).

Fig. 3 gives rules for evolving any single intention. For example, the rule act handles the execution of an action, when the pre-condition ψ is met, resulting in a belief state update. Rule $event$ replaces an event with the set of relevant plans, while rule $select$ chooses an applicable plan from a set of relevant plans while retaining un-selected plans as backups. With these backup plans, the rules for failure recovery $\triangleright_;$, $\triangleright_\top$, and $\triangleright_\bot$ enable new plans to be selected if the current plan fails (*e.g.* due to environment changes). Rules $;$ and $;_\top$ allow executing plan-bodies in sequence, while rules $\|_1$, $\|_2$, and $\|_\top$ specify how to execute (interleaved) concurrent programs. Rules G_s and G_f deal with declarative goals when either the success condition φ_s or the failure condition φ_f become true. Rule G_{init} initialises persistence by setting the program in the declarative goal to be $P \triangleright P$, *i.e.* if P fails try P again, and rule $G_;$ takes care of performing a single step on an already initialised program. Finally, the derivation rule $G_\triangleright$ re-starts the original program if the current program has finished or got blocked (when neither φ_s nor φ_f becomes true).

[1] Any logic is allowed providing entailment is supported. A propositional logic is used in our example.

$$\frac{act : \psi \leftarrow \langle \phi^-, \phi^+ \rangle \quad \mathcal{B} \vDash \psi}{\langle \mathcal{B}, act \rangle \to \langle (\mathcal{B} \setminus \phi^- \cup \phi^+), nil \rangle}\ act \qquad \frac{\Delta = \{\varphi : P \mid (e' = \varphi \leftarrow P) \in \Pi \wedge e' = e\}}{\langle \mathcal{B}, e \rangle \to \langle \mathcal{B}, e : (\mid \Delta \mid) \rangle}\ event \qquad \frac{\varphi : P \in \Delta \quad \mathcal{B} \models \varphi}{\langle \mathcal{B}, e : (\mid \Delta \mid) \rangle \to \langle \mathcal{B}, P \rhd e : (\mid \Delta \setminus \{\varphi : P\} \mid) \rangle}\ select$$

$$\frac{\langle \mathcal{B}, P_1 \rangle \to \langle \mathcal{B}', P_1' \rangle}{\langle \mathcal{B}, P_1 \rhd P_2 \rangle \to \langle \mathcal{B}', P_1' \rhd P_2) \rangle}\rhd_{;} \quad \frac{}{\langle \mathcal{B}, (nil \rhd P_2) \rangle \to \langle \mathcal{B}', nil \rangle}\rhd_{\top} \quad \frac{P_1 \neq nil\ \langle \mathcal{B}, P_1 \rangle \nrightarrow \ \langle \mathcal{B}, P_2 \rangle \to \langle \mathcal{B}', P_2' \rangle}{\langle \mathcal{B}, P_1 \rhd P_2 \rangle \to \langle \mathcal{B}', P_2' \rangle} \quad \rhd_{\bot} \quad \frac{\langle \mathcal{B}, P \rangle \to \langle \mathcal{B}', P' \rangle}{\langle \mathcal{B}, (nil; P) \rangle \to \langle \mathcal{B}', P' \rangle};_{\top}$$

$$\frac{\langle \mathcal{B}, P_1 \rangle \to \langle \mathcal{B}', P_1' \rangle}{\langle \mathcal{B}, (P_1; P_2) \rangle \to \langle \mathcal{B}', (P_1'; P_2) \rangle}; \quad \frac{\langle \mathcal{B}, P_1 \rangle \to \langle \mathcal{B}', P_1' \rangle}{\langle \mathcal{B}, (P_1 \| P_2) \rangle \to \langle \mathcal{B}', (P_1' \| P_2) \rangle}\|_1 \quad \frac{\langle \mathcal{B}, P_2 \rangle \to \langle \mathcal{B}', P_2' \rangle}{\langle \mathcal{B}, (P_1 \| P_2) \rangle \to \langle \mathcal{B}', (P_1 \| P_2') \rangle}\|_2 \quad \frac{}{\langle \mathcal{B}, (nil \| nil) \rangle \to \langle \mathcal{B}, nil \rangle}\|_{\top}$$

$$\frac{\mathcal{B} \models \varphi_s}{\langle \mathcal{B}, goal(\varphi_s, P, \varphi_f) \rangle \to \langle \mathcal{B}, nil \rangle}G_s \quad \frac{\mathcal{B} \models \varphi_f}{\langle \mathcal{B}, goal(\varphi_s, P, \varphi_f) \rangle \to \langle \mathcal{B}, ?false \rangle}G_f \quad \frac{P \neq P_1 \rhd P_2 \quad \mathcal{B} \nvDash \varphi_s \quad \mathcal{B} \nvDash \varphi_f}{\langle \mathcal{B}, goal(\varphi_s, P, \varphi_f) \rangle \to \langle \mathcal{B}, goal(\varphi_s, P \rhd P, \varphi_f) \rangle}G_{init}$$

$$\frac{\mathcal{B} \nvDash \varphi_s \quad \mathcal{B} \nvDash \varphi_f \quad \langle \mathcal{B}, P_1 \rangle \to \langle \mathcal{B}', P_1' \rangle}{\langle \mathcal{B}, goal(\varphi_s, P_1 \rhd P_2, \varphi_f) \rangle \to \langle \mathcal{B}', goal(\varphi_s, P_1' \rhd P_2, \varphi_f) \rangle}G_{;} \quad \frac{\mathcal{B} \nvDash \varphi_s \quad \mathcal{B} \nvDash \varphi_f \quad \langle \mathcal{B}, P_1 \rangle \nrightarrow}{\langle \mathcal{B}, goal(\varphi_s, P_1 \rhd P_2, \varphi_f) \rangle \to \langle \mathcal{B}, goal(\varphi_s, P_2 \rhd P_2, \varphi_f) \rangle}G_{\rhd}$$

Figure 3: Intention-level CAN semantics.

$$\frac{e \in E^e}{\langle E^e, \mathcal{B}, \Gamma \rangle \Rightarrow \langle E^e \setminus \{e\}, \mathcal{B}, \Gamma \cup \{e\} \rangle}A_{event} \quad \frac{P \in \Gamma \quad \langle \mathcal{B}, P \rangle \to \langle \mathcal{B}', P' \rangle}{\langle E^e, \mathcal{B}, \Gamma \rangle \Rightarrow \langle E^e, \mathcal{B}', (\Gamma \setminus \{P\}) \cup \{P'\} \rangle}A_{step} \quad \frac{P \in \Gamma \quad \langle \mathcal{B}, P \rangle \nrightarrow}{\langle E^e, \mathcal{B}, \Gamma \rangle \Rightarrow \langle E^e, \mathcal{B}, \Gamma \setminus \{P\} \rangle}A_{update}$$

Figure 4: Agent-level CAN semantics.

The agent-level semantics are given in Fig. 4. The rule A_{event} handles external events, that originate from the environment, by adopting them as intentions. Rule A_{step} selects an intention from the intention base, and evolves a single step w.r.t. the intention-level transition, while A_{update} discards any unprogressable intentions (either already succeeded, or failed).

B. Bigraphs

Bigraphs are a graph-based universal modelling formalism, introduced by Milner [8] for describing systems with both spatial relationships and non-local linking. They have been used for modelling ubiquitous systems including [12], [13], and as a unifying theory of process calculi [14]. Bigraph dynamics are described using a rewriting system specified via reaction rule $l \longrightarrow r$ that replace a bigraph matching l (in some larger bigraphs) with a bigraph matching r. Given an initial bigraph and set of reaction rules we derive a (non-deterministic) transition system capturing the behaviour of the system through repeated rule application.

We have used bigraphs to encode the existing CAN language semantics to symbolically analyse BDI agent behaviour [9]. The encoding defines an equivalent bigraph for any CAN agent, and defines reaction rules that faithfully model the operational semantics. To execute bigraphical reactive systems, we use BigraphER [15], an open-source language and toolkit for bigraphs. BigraphER allows exporting transitions systems, *e.g.* DTMCs, for analysis in PRISM. To aid writing logical formulas over transition systems, states are labelled using bigraph patterns that assigns a state a *predicate* label if it contains (a match of) a given bigraph pattern.

III. FRAMEWORK

In this section, we first formalise our notion of an environment and then extend the existing CAN semantics to enable the agent to perceive and alter the environment.

A. Environments

Most existing BDI frameworks either omit the environment (*e.g.* CAN) or provide it with an informal treatment (*e.g.* AgentSpeak). In order to analyse an agent behaviour in an environment we first formalise environments. Following [16], we consider environments accounting for both sensor information (a set of beliefs literals) and external events (a set of tasks to complete). Recall the formula in a belief base of a BDI agent is from the language $\mathcal{L}$ and the set of events the agent can respond to is E. The alphabet for an environment is then $\mathcal{E} = \mathcal{L} \cup E$. An environment state, representing the environment at a point in time, is defined as follows:

Definition 1. *An environment state is $\Theta \in Q$ such that $Q = 2^{\mathcal{E}}$ and for any formula b from language $\mathcal{L}$, if $b \in \Theta$, then $\neg b \notin \Theta$.*

Environment states are sets of belief formulae and newly requested events which are held in the environment at some point. We assume an underlying mechanism that converts from real-world environments to symbolic literals, *e.g.* through pattern detection from sensor input. The condition ensures that no environment states has information indicating that both b and $\neg b$ are true, *i.e.* the law of the excluded middle.

Example 1. *A UAV that perceives an environment state $\Theta = \{\neg \mathsf{harsh\_weather}\}$ believes (based on sensor information) that the weather is not harsh.*

To support cases when an agent changes the environment but the effects of acting are undone, *e.g.* by humans, before the agent starts perceiving in the next cycle, we separate environment changes imposed by the agent from environment changes *e.g.* due to external factors. Environment dynamics contributed by the agent will be formalised through the extensions of CAN semantics in Section III-B. Meanwhile, we give the following external dynamics *with respect to an agent* for an environment over each agent reasoning cycle.

Definition 2. *The external dynamics for an agent in an environment is a tuple $\langle Q, \Theta_0, \epsilon \rangle$ where $Q = 2^{\mathcal{E}}$ is a set of environment states, and $\Theta_0 \in Q$ an initial (or start) state, and $\delta : Q \to 2^Q$ a finite nondeterministic function transitioning a state $\Theta \in Q$ to a finite set of successor states $\{\Theta_1, \ldots, \Theta_n\}$.*

Essentially, external dynamics of an environment is formalised as a non-deterministic finite-state automata and it, by definition, is *relative* to the agent under consideration, as any other agent is a part of the external environment.[2] Although we assume the automata to be finite-state, this does not preclude infinite behaviour, *e.g.* looping between existing states indefinitely.

Example 2. *Following Example 1, we assume the current environment state* $\Theta = \{\neg\mathsf{harsh\_weather}\}$. *Then a possible finite set of successor states can be* $\Theta_1 = \{\neg\mathsf{harsh\_weather}\}$, $\Theta_2 = \{\neg\mathsf{harsh\_weather}, \mathsf{e\_deliver}\}$, $\Theta_3 = \{\mathsf{harsh\_weather}\}$ $\Theta_4 = \{\mathsf{harsh\_weather}, \mathsf{e\_deliver}\}$ *where* e_deliver *is a new (external) event. These four states describes all combinations of physical attribute* harsh_weather *and the event* e_deliver[3].

In the following section, we extend CAN semantics to interact with the external changes in an environment.

B. Perceiving and Acting

Currently CAN does not support explicit environments. To allow environments, we augment both intention level configuration $\langle \mathcal{B}, P\rangle$ (resp. agent-level configuration) in CAN with an environment state Θ, namely $[\langle \mathcal{B}, P\rangle, \Theta]$ (resp. $[\langle E^e, \mathcal{B}, \Gamma\rangle, \Theta]$) where $\mathcal{B}$ is the belief base, P the current intention, E^e the external event set, and Γ the set of intentions.

When perceiving, the belief base $\mathcal{B}$ of an agent is updated to reflect[4] the current environment state Θ where $\mathcal{B} = \Theta$. While sensed information may remain in the environment, the perceived new events should be deleted to avoid them being adopted twice, *i.e.* these are explicitly removed from the environment. The following rule $A_{perceive}$ is given.

$$\frac{}{[\langle E^e, \mathcal{B}, \Gamma\rangle, \Theta] \Rightarrow [\langle E^e, \Theta, \Gamma\rangle, \Theta \setminus \{e \mid e \in \Theta\}} \; A_{perceive}$$

After perceiving, the agent assimilates perceived new events to the external event set at some later reasoning cycle and subsequently removed from the belief base (rule $A_{assimilate}$).

$$\frac{e \in \mathcal{B}}{[\langle E^e, \mathcal{B}, \Gamma\rangle, \Theta] \Rightarrow [\langle E^e \cup \{e\}, \mathcal{B} \setminus \{e\}, \Gamma\rangle, \Theta]} \; A_{assimilate}$$

Alongside external dynamics, the agent can alter the environment through acting. Although there is already an intention-level rule in CAN for acting, it only changes the belief base (rule act in Fig. 3) which assumes that the agent immediately believes the action has had an effect on the environment. However as, for example, the actions on an agent might be undone another agent, an agent should wait for the effects to take place in the environment and update its beliefs only *after* sensing from the environment in the next reasoning cycle (through rule $A_{perceive}$). As such, we replace the old act rule with the following rule act^{new} to apply effects of actions always in the *environment*, not the belief base.

$$\frac{act : \psi \leftarrow \langle \phi^-, \phi^+\rangle \quad \mathcal{B} \vDash \psi}{[\langle \mathcal{B}, act\rangle, \Theta] \rightarrow [\langle \mathcal{B}, nil\rangle, (\Theta \setminus \phi^-) \cup \phi^+]} \; act^{new}$$

Finally, the update of an environment due to the external dynamics is applied in the beginning of each reasoning cycle according to Definition 2 and is detailed in Section IV

IV. AGENT REASONING CYCLE

We encode the agent cycle (Fig. 1) including external dynamics of an environment, into bigraphs to obtain an executable semantics (available online[5]) that we use for verification. The cycle is a three step process described as follows:

Step 1 (environment update): Apply the non-deterministic function δ in Definition 2 to update the current environment state Θ to a finite set of states $\delta(\Theta) = \{\Theta_1, \ldots, \Theta_n\}$.

Step 2 (perceive): Using rule $A_{perceive}$, an agent perceives everything in the current (newly updated) environment.

Step 3 (progress): Progress the agent using rule $A_{assimilate}$ or rules in Fig. 4. When A^{new}_{step} is applied, it selects an intention non-deterministically from the intention base, and evolves a single step w.r.t. intention-level transition (seen in Fig. 3).

The reasoning cycle is repeated until all external events and intentions are fully addressed and no new tasks are requested. As we seek to examine the agent behaviours, we stop analysis when the agent stops operating.

To analyse an agent in all possible environments, model checking is applied to the transition system generated from our executable semantics. As we use bigraphs, the transition system has bigraphs as states and rewrite rules as transitions. To reason over the transition system, we label states with *bigraph patterns* [13], and specify dynamic properties using linear or branching time temporal logics such as Computation Tree logic (CTL) [17]. As we generate a transition system, the property specification language is constrained by the model checker. Here we use the non-probabilistic and non-reward logics provided by PRISM[6].

V. UAVS EXAMPLE

To illustrate our framework, we consider a small example taken from UAV surveillance mission systems [9].

A. Design

A UAV patrols a pre-defined area to identify objects of interest and can park itself upon harsh weather to avoid damage. The agent design and environment specifications are given in Fig. 5. The first plan (line 2) addresses event e_patrol_init and is always applicable (true context). The play-body consists of a declarative goal, goal(detection, e_patrol_task, false), that continually pursues e_patrol_task until an object of interest is detected (*i.e.* detection). No failure condition is specified. If this goal completes, the action return is executed to return to base, the effect is that result in that returned holds (unshown here). The event e_patrol_task is handled by plan on line 3 which contains a further declarative goal goal(harsh_weather ∨ battery_low, e_patrol, false) instructing

[2] This definition is a special case when the agent is fixed and can be extended in future to allow multi-agents in a shared environment.

[3] Unlike belief formulae built from a language, the negation of an event in an environment is denoted as the absence of such an event.

[4] Partial observability is not currently supported.

[5] https://bitbucket.org/uog-bigraph/bdi_env_model_seke22/src/master/

[6] As this is supported natively by BigraphER.

```
1  Plan library
2  e_patrol_init : true ← goal(detection, e_patrol_task, false); return
3  e_patrol_task : true ← goal(harsh_weather, e_patrol, false); e_pause
4  e_patrol : true ← patrol
5  e_pause : harsh_weather ∧ ¬parked ← activate_parking; wait
6  e_pause : harsh_weather ∧ parked ← wait
7  initial environment state
8  Θ₀ = {¬a, ¬b, ¬c, ¬d, e_patrol_init}
9  environment transition function
```

$$\delta(\Theta) = \begin{cases} \{\Theta, (\Theta \setminus \{\neg a\}) \cup \{a\}, (\Theta \setminus \{\neg b\}) \cup \{b\}, (\Theta \setminus \{\neg a, \neg b\}) \cup \{a, b\}\} & \\ & \text{if } \neg a \land \neg b \in \Theta \quad (1) \\ \{\Theta, (\Theta \setminus \{\neg a\}) \cup \{a\}\}, & \text{if } \neg a \land b \in \Theta \quad (2) \\ \{\Theta, (\Theta \setminus \{\neg b\}) \cup \{b\}\}, & \text{if } a \land \neg b \in \Theta \quad (3) \\ \{\Theta\}, & \text{if } a \land b \in \Theta \quad (4) \\ \{(\Theta \setminus \{b, c\}) \cup \{\neg b, \neg c\}\}, & \text{if } b \land c \in \Theta \quad (5) \end{cases}$$

where a = detection, b = harsh_weather, c = waited (the effect of action wait) and d = returned (the effect of action return).

Figure 5: Patrolling Task Design for BDI Agents.

```
1  Plan library
2  e_patrol_init : true ← goal(¬harsh_weather ∧ detection, e_patrol_task, false);
   goal(¬harsh_weather ∧ returned, e_return_task, false)
3  e_patrol_task : true ← goal(harsh_weather, e_patrol, false); e_pause
4  e_return_task : true ← goal(harsh_weather, e_return, false); e_pause
5  e_patrol : true ← patrol
6  e_return : true ← return
7  e_pause : harsh_weather ∧ ¬parked ← activate_parking; wait
8  e_pause : harsh_weather ∧ parked ← wait
```

Figure 6: Discovered Corrections of Fig. 5.

an agent to patrol continuously unless the weather is harsh when it should pause (*i.e.* event e_pause). Plans in lines 5 to 6 handle the e_pause by waiting until weather becomes better, while the plan in line 4 performs the patrolling. For succinctness, descriptions of actions (*i.e.* their precondition and effects) such as return and wait are not shown but can be found in our online model.

To specify environments, we assume favourable conditions (as often in practice) in the initial state (line 8). The environment transition function (line 9) describes the set of successor states given the current states: the object becoming available for detection or harsh weather appearing can happen at any time so long as they have not already occurred (cases 1–3); no update is available if detection and harsh weather are present (case 4); and, for practicality, the harsh weather will always get better after some time (case 5).

B. Analysis

We check that a UAV never returns to base under harsh weather (a safety property). To formalise this property, we represent state formulae by bigraph patterns: $\varphi_1 \overset{\text{def}}{=} \mathsf{B}(\text{“harsh\_weather”})$ and $\varphi_2 \overset{\text{def}}{=} \mathsf{B}(\text{“returned”})$. Using CTL, we have a property $\neg\mathbf{E}[\mathbf{F}(\varphi_1 \land \neg\varphi_2 \land (\mathbf{XX}\varphi_2))]$ that check there does not exist any agent behaviours in any environment that when the weather is harsh and UAV has not returned to base, the UAV is believed to be returned in two states[7] afterwards, *i.e.* we check that it does not travel through the bad weather.

However, this safety property does not hold for design in Fig. 5. To aid in addressing this problem, the violated states can be automatically located (by PRISM model checker). For debugging, the graphical output of each state in the transition system provided by BigraphER provides a diagrammatic representation of each state enabling us to locate subtle design issues. In this case, there are two failing situations. (1) when the weather becomes harsh, the agent can execute plans on line 5 or 6 to activate parking to avoid any potential damage. However, before the completion of parking (*e.g.* the execution of the action wait), the truth of detection which happens to hold can makes goal(detection, e_patrol_task, false) succeed, leading the agent to proceed returning prematurely without fully handling the negative environment. (2) occurs when the weather becomes harsh shortly after the UAV completes a detection, causing the UAV to return under harsh weather. To fix agent design flaws, we need to add ¬harsh_weather to the success condition of goal(detection, e_patrol_task, false) (blue in Fig. 6) for first case. For second case, we also require a declarative goal structure for the return task (red in Fig. 6). The property now holds for new agent design in Fig. 6.

We find that both designs in Fig. 5 and Fig. 6 can lead to a loop in the agent behaviours. Such a loop includes two situations: (1) persistent patrolling when no object is available to be detected and (2) whenever the agent is about to patrol or return, the weather becomes harsh so that the agent has to wait for the weather to be normal again. We can denote the bigraph pattern for the completion of the given intention as $\varphi_3 \overset{\text{def}}{=} \mathsf{Intent.1}$ if and only if it is the only intention in the base (which is our case), that checks that along all paths eventually the intention is completed either with success or with failure.

We also check that whenever a new event is requested from the environment, it will be responded by the agent eventually. We denote the bigraph pattern for the presence of new event request in an environment $\varphi_4 \overset{\text{def}}{=} \mathsf{Environment.(e\_patrol\_init} \mid \mathsf{id)}$ and the successful assimilation of such event in external event set (a.k.a. desires) as $\varphi_5 \overset{\text{def}}{=} \mathsf{Desires.(e\_patrol\_init} \mid \mathsf{id)}$ where the symbol id (called a site in bigraphs) stands for the part of model that is abstracted away. We can have the property $\mathbf{A}[\varphi_4 \implies \mathbf{F}\varphi_5]$ which checks that along all paths, if a new event is requested (*i.e.* φ_4 holds), this implies that eventually it will be added in the desires. To check that the agent actually committed to progressing this desire (leaving empty desire set in this case), we have the property $\mathbf{A}[\varphi_5 \implies \mathbf{F}\varphi_6]$ where $\varphi_6 \overset{\text{def}}{=} \mathsf{Desires.1}$. Both designs satisfy both of these properties.

A summary of these property checking is given in Table I. It also details the transition system that was used in the evaluation of each property: the number of states and

[7] When an action is executed by an agent (one $\mathbf{X}$), it requires another step to perceive the effects of action (another $\mathbf{X}$). Hence, two $\mathbf{X}$s are needed.

	Design in Fig. 5	Design in Fig. 6
Saftey Property	False	True
Completion Property	False	False
Response Property	True	True
Commitment Property	True	True
States	167	282
Transitions	242	373
Build time (s)	54.05	128.89
Rule applications	1306	2152

Table I: Properties checked: where safety property is $\neg\mathbf{E}[\mathbf{F}(\varphi_1 \wedge \neg\varphi_2 \wedge (\mathbf{XX}\varphi_2))]$, completion property $\mathbf{A}[\mathbf{F}\varphi_3]$, response property $\mathbf{A}[\varphi_4 \implies \mathbf{F}\varphi_5]$, and commitment property $\mathbf{A}[\varphi_5 \implies \mathbf{F}\varphi_6]$.

transitions, build time (which are in the order of minutes), and rule applications. The rule applications are the number of applications of reaction rules, including instantaneous reaction rules—an advanced feature of BigraphER—that allows agents to progress an intention without showing all sub-steps. For example, it includes environment revision, where we see only final output of a step of executing an action by an agent.

VI. Related Work

When modelling environments for BDI agents, most existing work, similarly to this work, separates the specification of environments from the agent designs. For example, the JaCaMo platform [18] includes an artifact-based framework to allow programming and executing virtual environments. While agent computational simulations are essential, they can, by their nature, only analyse one possible run of agent behaviours in one environment.

Verifying BDI agent behaviours through model checking and theorem proving has also been explored. For example, the work [19] (resp. [20]) applys the Java PathFinder model-checker (resps. Isabelle/HOL proof assistant) to verify BDI programs. However, none of them has addressed how the environment should be transitioned from state to state and they only examine agent behaviours in a subset of environment. On the contrary, we support the analysis of all possible agent behaviours in all possible environments.

VII. Conclusions

A computational modelling and verification framework for BDI-agents can aid design-time specification by allowing us to reason about the behaviour of rational agents operating in dynamic environments, *e.g.* those that feature to changes of external world situations (harsh weather etc.).

We have provided a formalisation of an environment, including the sensor information, incoming external events which the agent needs to respond to, and the external dynamics (relative to an agent) in such an environment. The computational modelling of a dynamic environment is enabled by an extension to the Can language (that formalises the behaviour of a classical BDI agent). The extended semantics are executable (via bigraphs) to allow both the development of agent designs and the specification of dynamic environments, exposing any potentially anomalous agent behaviour in any environment.

Through an UAV example, we have shown it is possible to reason about agent behaviours in all possible environments. In particular, we found that our approach can aid automatically identifying subtle agent design flaws rendered under some dynamic situations. The future work is to investigate uncertain environments to support numerical analysis, *e.g.* the probability of completing an intention in adversarial conditions.

Acknowledgements

This work is supported by the EPSRC, under PETRAS SRF grant MAGIC (EP/S035362/1) and S4: Science of Sensor Systems Software (EP/N007565/1).

References

[1] A. S. Rao, "AgentSpeak (L): BDI agents speak out in a logical computable language," in *European Workshop on Modelling Autonomous Agents in a Multi-Agent World*. Springer, 1996, pp. 42–55.

[2] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah, "Declarative and procedural goals in intelligent agent systems," in *Conference on Principles of Knowledge Representation and Reasoning*, 2002.

[3] K. V. Hindriks, F. S. D. Boer, W. V. d. Hoek, and J.-J. C. Meyer, "Agent programming in 3APL," *Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 4, pp. 357–401, 1999.

[4] M. Winikoff, "JACK intelligent agents: an industrial strength platform," in *Multi-Agent Programming*, 2005, pp. 175–193.

[5] R. H. Bordini *et al.*, "Programming multi-agent systems in AgentSpeak using Jason," vol. 8. John Wiley & Sons, 2007.

[6] L. Royakkers and R. van Est, "A literature review on new robotics: automation from love to war," *International journal of social robotics*, vol. 7, no. 5, pp. 549–570, 2015.

[7] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge, "Verifying multi-agent programs by model checking," *Autonomous Agents and Multiagent Systems*, vol. 12, no. 2, pp. 239–256, 2006.

[8] R. Milner, *The space and motion of communicating agents*. Cambridge University Press, 2009.

[9] B. Archibald, M. Calder, M. Sevegnani, and M. Xu, "Modelling and verifying BDI agents with bigraphs," *Science of Computer Programming*, vol. 215, p. 102760, 2022.

[10] M. Kwiatkowska *et al.*, "PRISM 4.0: Verification of probabilistic real-time systems," in *International Conference on Computer Aided Verification*, ser. LNCS, vol. 6806. Springer, 2011, pp. 585–591.

[11] S. Sardina and L. Padgham, "Goals in the context of BDI plan failure and planning," in *International Conference on Autonomous Agents and Multiagent Systems*, 2007, pp. 16–23.

[12] M. Sevegnani, M. Kabác, M. Calder, and J. A. McCann, "Modelling and verification of large-scale sensor network infrastructures," in *Conference on Engineering of Complex Computer Systems,*, 2018, pp. 71–81.

[13] S. Benford, M. Calder, T. Rodden, and M. Sevegnani, "On lions, impala, and bigraphs: modelling interactions in physical/virtual spaces," *ACM Transactions on Computer-Human Interaction*, vol. 23, pp. 1–56, 2016.

[14] M. Bundgaard and V. Sassone, "Typed polyadic pi-calculus in bigraphs," in *International Conference on Principles and Practice of Declarative Programming*, 2006, pp. 1–12.

[15] M. Sevegnani and M. Calder, "BigraphER: Rewriting and analysis engine for bigraphs," in *the 28th International Conference on Computer Aided Verification*, 2016, pp. 494–501.

[16] S. Sardina and L. Padgham, "A BDI agent programming language with failure handling, declarative goals, and planning," in *Autonomous Agents and Multi-Agent Systems*, vol. 23, no. 1. Springer, 2011, pp. 18–70.

[17] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," in *Workshop on Logic of Programs*, 1981, pp. 52–71.

[18] O. Boissier, R. H. Bordini, J. Hubner, and A. Ricci, *Multi-agent oriented programming: programming multi-agent systems using JaCaMo*. MIT Press, 2020.

[19] L. A. Dennis, M. Fisher, M. P. Webster, and R. H. Bordini, "Model checking agent programming languages," *Automated software engineering*, vol. 19, no. 1, pp. 5–63, 2012.

[20] A. B. Jensen, "Machine-checked verification of cognitive agents," in *Proceedings of the 14th International Conference on Agents and Artificial Intelligence*, 2022, pp. 245–256.