Probabilistic BDI Agents: Actions, Plans, and Intentions

Blair Archibald, Muffy Calder, Michele Sevegnani, and Mengwei Xu

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 qualitative properties, for example whether an intention completes. BDI-based systems, however, operate in uncertain environments with dynamic behaviours: we may need quantitative analysis to establish properties such as the probability of eventually completing an intention. We define a probabilistic extension to the Conceptual Agent Notation (CAN) for BDI agents that supports probabilistic action outcomes, and probabilistic plan and intention selection. The semantics is executable via an encoding in Milner's bigraphs and the BigraphER tool. Quantitative analysis is conducted using PRISM. While the new semantics can be applied to any CAN program, we demonstrate the extension by comparing with standard plan and intention selection strategies (*e.g.* ordered or fixed schedules) and evaluating probabilistic action executions in a smart manufacturing scenario. The results show we can improve significantly the probability of intention completion, with appropriate probabilistic distribution. We also show the impact of probabilistic action outcomes can be marginal, even when the failure probabilities are large, due to the agent making smarter intention selection choices.

Keywords: BDI Agents; Quantitative Analysis; Bigraphs

1 Introduction

A well-studied and popular architecture for developing rational agents is the Belief-Desire-Intention (BDI) paradigm. BDI paradigm builds upon a sound theoretical foundation to model an agent where (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 agents have inspired many agent-oriented programming languages including AgentSpeak [1], Can [2], and CanPlan [3], 3APL [4], and 2APL [5] along with a collection of mature software toolkits and platforms including JACK [6], Jason [7], and Jadex [8]. BDI agents have been recognised for their efficiency and scalability in areas such as business [9], healthcare [10], and engineering [11].

In BDI languages, desires and intentions are often represented using a plan library. Each plan describes a course of actions which an agent can perform to

address an event given some beliefs hold, while the set of intentions are the plans currently being executed. Typically BDI languages: (1) assume that action outcomes (*i.e.* the effects on *external* environment) are deterministic, (2) remain agnostic *internally* to the choice of an applicable plan to bring about its desires, (3) remain agnostic *internally* to the order that intentions are progressed. These assumptions facilitate the verification of agent behaviour through a non-deterministic underlying transition system (*e.g.* [12, 13]), where plan and intention selection denotes branching choices and actions have a single outcome. Unfortunately, this often does not adequately represent behaviour in uncertain environments such as cyber-physical robotics systems (*e.g.* surveyed in [14]) with uncertain sensors, and actuators.

For example, the outcome of an action may be probabilistic due to sensor noise and imprecise actuation, and plans and intentions are not created equally and are likely to have different characteristics such as preference and urgency. As a result, there is a growing need for formal techniques that can handle quantitative properties of agent-based systems under uncertainty.

We employ the following robot packaging task for smart manufacturing as an example, giving detailed quantitative analysis in Section 4. The robot insulates products with suitable wrapping bags, to prevent temperature rise and consequent spoilage, and then transfers the wrapped products to a storage location. There are two types of wrapping bags: premium and standard. The standard wrapping is preferred as the cheaper option, however it may not be effective if the product temperature is already too high, and/or the packaging occasionally breaks, which results in damaged product (*i.e.* a negative action outcome). Before wrapping the products, the robot also has to decide which product to handle first (as there may be multiple products waiting), meaning handling a product before it spoils requires a notion of urgency. While it is important to prioritise the more urgent products, it is also sensible to progress less urgent ones from time-to-time, before they also become urgent. So we need to model and *quantify* agent behaviour when there is a range of choices, inherent uncertainty, and characteristics of preference and urgency. For example, we may wish to know the probability the robot can complete packaging under different schedules, negative outcomes, and decisions.

In the BDI community, probabilistic action outcomes are usually implicit—requiring the agent to *sense* failures and *revise* the beliefs (*i.e.* to enable new plans)—and are often disregarded when modelling. Although most agent language semantics specify non-deterministic *plan selection*, *e.g.* in [2], it is typical in practice for plans to be ordered—either statically [7] or at run-time [15]—to enforce deterministic branching. While desirable to exploit the highest ordered plan, it may be worthwhile exploring other non-highest order plans every now and then to avoid being stuck in a local maximum. Similarly, *intention selection* is also not implemented in a fully non-deterministic fashion either, but in a fixed schedule, *e.g.* round robin (executing a step of each intention in turn).

We argue that the highest ordering (*i.e.* local maximum) and fixed schedules (*e.g.* round robin) are not always the best approach to plan/intention selec-

tion and suggest agents should support probabilistic plan/intention selection along with the need to evaluate the undesired outcomes of actions. Therefore, we present a formal approach to specify, model, and quantitatively analyse BDI agents with probabilistic action outcomes and plan/intention selections drawn from a probability distribution. Quantitative verification, *e.g.* asking the probability some intention completes, aids the design of agents by enabling plan and intention selection functions to be explored, and mitigates the risk of operating in the uncertain environments by providing quantitative assurance.

We have extended the operational semantics of CAN language in [2] to a probabilistic setting. CAN is chosen as it features a high-level agent programming language that captures the essence of BDI concepts without describing implementation details such as data structures. As a superset of AgentSpeak [1], CAN includes advanced BDI agent behaviours such as reasoning with *declarative goals*, *concurrency*, and *failure recovery*, which are necessary for our smart manufacturing example modelled in Section 4. Importantly, although we focus on CAN, the language features are similar to those of other mainstream BDI languages and the same modelling techniques would apply to other BDI programming languages. We build on our previous work on an executable semantics of CAN [16], based on Milner's Bigraphs [17] and provide the resulting probabilistic *executable* semantics.

We use probabilistic bigraphs [18] that allow a (relative) weight to be assigned to bigraph reaction rules and we extend the rules in CAN specifying plan selection, intention selection, and probabilistic action outcomes (specified by the user). For automated verification, we export a Discrete Time Markov Chain (DTMC) from the bigraph model for analysis in probabilistic model checkers, *e.g.* PRISM [19]. We believe this is the first rigorous quantitative analysis through formal modelling applied to plan selection, intention selection, and action execution within mainstream BDI agents.

We make the following research contributions:

- a probabilistic extension of the semantics of CAN language;
- an executable semantics of CAN based on probabilistic bigraphs;
- an evaluation, in a smart manufacturing case, of probabilistic plan and intention selection under probabilistic action outcomes, against standard counterparts, *e.g.* ordered plan selection and round robin intention selection.

The paper is organised as follows. In Section 2 we provide a brief overview of BDI agents and Bigraphs. In Section 3 we propose the probabilistic extension of CAN semantics. In Section 4 we evaluate our approach on a smart manufacturing example. We discuss related work in Section 5 and conclude in Section 6.

2 Background

2.1 BDI Agents

A BDI agent has an explicit representation of beliefs, desires, and intentions. The beliefs correspond to what the agent believes about the environment, while

the desires are a set of *external* events that the agent can respond to. To respond to those events, the agent selects a plan (given its beliefs) from the pre-defined plan library and commits to the selected plan by turning it into a new intention.

The CAN language formalises a classical BDI agent 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 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. The triggering event e specifies why the plan is triggered, while the context condition φ determines *when* the plan-body P is able to handle the event. Events can be either be external (*i.e.* from the environment in which the agent is operating) or internal (*i.e.* sub-goals that the agent itself tries to accomplish). The language used in the plan-body is defined by the following grammar:

$$P = nil \mid +b \mid -b \mid act \mid ?\varphi \mid e \mid P_1; P_2 \mid P_1 \triangleright P_2 \mid P_1 \parallel P_2 \mid \\ e : (|\varphi_1 : P_1, \cdots, \varphi_n : P_n|) \mid goal(\varphi_s, P, \varphi_f)$$

where nil is an empty program, $+b$ and $-b$ belief addition and deletion, act a primitive action, $?\varphi$ a test for φ in the belief base, and e is a sub-event (i.e. internal event). Actions act take the form $act = \varphi \leftarrow \langle \phi^+, \phi^- \rangle$, where φ is the pre-condition, and ϕ^+ and ϕ^- are the addition and deletion sets (resp.) of belief atoms, i.e. a belief base $\mathcal{B}$ is revised to be $(\mathcal{B} \setminus \phi^-) \cup \phi^+$ when the action executes. To execute a sub-event, a plan (corresponding to that event) is selected and the plan-body added in place of the event. In this way we allow plans to be nested (similar to sub-routine calls in other languages). In addition, there are composite programs $P_1; P_2$ for sequence, $P_1 \triangleright P_2$ that executes P_2 in the case that P_1 fails, and $P_1 \parallel P_2$ for interleaved concurrency. A set of relevant plans (those that respond to the same event) is denoted by $e : (|\psi_1 : P_1, \cdots, \psi_n : P_n|)$. Finally, a declarative goal program $goal(\varphi_s, P, \varphi_f)$ expresses that the declarative goal φ_s should be achieved through program P, failing if φ_f becomes true, and retrying as long as neither φ_s nor φ_f is true (see in [3] for details).

The operational semantics for CAN are defined over configurations $\mathcal{C}$ and transitions $\mathcal{C} \rightarrow \mathcal{C}'$. A transition $\mathcal{C} \rightarrow \mathcal{C}'$ denotes a single execution step between configuration $\mathcal{C}$ and $\mathcal{C}'$. We write $\mathcal{C} \rightarrow$ (resp. $\mathcal{C} \nrightarrow$) to state that there is (resp. is not) a $\mathcal{C}'$ such that $\mathcal{C} \rightarrow \mathcal{C}'$. A derivation rule consists of a (possibly empty) set of premises p_i $(i = 1, \ldots, n)$ on $\mathcal{C}$, and a conclusion, denoted by

$$\frac{p_1 \quad p_2 \quad \cdots \quad p_n}{\mathcal{C} \rightarrow \mathcal{C}'}\ l$$

where l is a rule name. We write $\mathcal{C} \xrightarrow{l} \mathcal{C}'$ to denote $\mathcal{C}$ evolves to $\mathcal{C}'$ through the application of derivation rule l.

A basic configuration $\langle \mathcal{B}, P \rangle$, where P is the plan-body program being executed (*i.e.* the current intention), is used in rules that define the execution of

[1] Any logic is allowed providing entailment is supported. A propositional logic with natural number comparisons is used in our examples.

a single intention. The agent configuration is defined as $\langle E^e, \mathcal{B}, \Gamma \rangle$ where E^e stands for the a set of pending external events and Γ the current set of intentions (partially executed plan-body programs). The semantics of CAN language is specified by two types of transitions. The first transition type, denoted as $\rightarrow$, specifies *intention-level* evolution in terms of basic configuration $\langle \mathcal{B}, P \rangle$ and the second type, denoted as $\Rightarrow$, specifies *agent-level* evolution over the agent configuration $\langle E^e, \mathcal{B}, \Gamma \rangle$. For example, in the *intention-level* evolution, the transition for belief addition and a belief test can be given as follows:

$$\frac{}{\langle \mathcal{B}, +b \rangle \rightarrow \langle \mathcal{B} \cup \{b\}, nil \rangle} \; +b \qquad \frac{\mathcal{B} \models \varphi}{\langle \mathcal{B}, ?\varphi \rangle \rightarrow \langle \mathcal{B}, nil \rangle} \; ?$$

We refer the reader to [2, 20] for a full overview of the semantics of CAN.

2.2 Bigraphs

Bigraphs are a graph-based universal modelling formalism, introduced by Milner [17], and extended to probabilistic systems [18]. As a graph-based rewriting formalism, over rules called reaction rules, bigraphs not only provide an intuitive diagrammatic representation, which is ideal for visualising the execution process of the systems, but also offer compositional reasoning via explicit abstractions (sites/regions/names), customised rewriting rules, and multiple ways to relate entities (placement and linking). They have been used both for modelling ubiquitous systems [21, 22, 23] and as a unifying theory of existing process calculi [24, 25] and their semantics.

The evolution of bigraphs is described through over a rewriting system specified via reaction rule $l \longrightarrow\!\!\blacktriangleright r$ that replace a bigraph matching l with a bigraph matching r in some larger bigraph[2]. Given an initial bigraph and set of reaction rules we can derive a non-deterministic transition system capturing the behaviour of the system. We have used this to encode the existing CAN language semantics in order to symbolically analyse BDI agent behaviour [16]. The encoding defines a bigraph equivalent for any CAN agent, and defines reaction rules that faithfully model the operational semantics (essentially a tree exploration).

Probabilistic bigraphs [18] allow reaction rules to be weighted, *e.g.* $\mathtt{t_1} = l_1 \xrightarrow{2}\!\!\blacktriangleright r_1$ and $\mathtt{t_2} = l_2 \xrightarrow{1}\!\!\blacktriangleright r_2$, such that if both (and only) $\mathtt{t_1}$ and $\mathtt{t_2}$ are applicable then $\mathtt{t_1}$ is twice as likely to apply as $\mathtt{t_2}$. In this case the transition system generated is a DTMC that can be analysed by probabilistic model checker, *e.g.* PRISM [19].

To execute (probabilistic) bigraphical reactive systems, we employ BigraphER [26], an open-source language and toolkit for bigraphs. It also allows exporting transitions systems, *e.g.* DTMCs, for analysis in specialised model checking tools. To aid writing logical formulas over the transition systems, states may be labelled using bigraph patterns that assign a state *predicate* label if it contains (a match of) given bigraph patterns.

[2] Similar to term rewriting lifted to graph structures.

3 Probabilistic Extension of CAN Semantics

In this section we detail how action outcomes, plan selection, and intention selection from CAN can be extended to support probabilistic reasoning.

3.1 Probabilistic Action Outcomes

Agents execute actions that both interact with an external environment (*e.g.* pick up an object), and in-turn revise the internal belief base (*e.g.* the agent believes it holds the object). Action execution is specified in CAN as:

$$\frac{act : \varphi \leftarrow \langle \phi^-, \phi^+ \rangle \quad \mathcal{B} \vDash \varphi}{\langle \mathcal{B}, act \rangle \rightarrow \langle (\mathcal{B} \setminus \phi^- \cup \phi^+), nil \rangle}\ act$$

This states that an action applies only if the precondition φ holds, and the outcome is to update the belief base by adding and removing the atoms specified by ϕ^+ and ϕ^-, respectively. Other than beliefs, the agent has no notion of the environment and these are assumed to be side-effects.

In practice, we know the outcomes of an action are uncertain (*e.g.* due to actuator malfunctions). For example, an agent may execute an action to pick up an object but fail to do so because a robotic arm fails. In this case, updating the beliefs that an object is held leads to misalignment between the true environment and the agent's representation of it. This form of uncertainty has been considered extensively in the planning literature and has led to, *e.g.* probabilistic planning domain definition languages (PPDDL) [27], that consider multiple outcomes with associated probabilities (*e.g.* estimated from the historic data).

We follow a similar approach and sample action outcomes from a probability distribution $\mu = [(\phi_1^-, \phi_1^+) \mapsto p_1, \ldots, (\phi_n^-, \phi_n^+) \mapsto p_n]$ with $\sum_{i=1}^{n} p_i = 1$. That is, we use actions in the form $a : \varphi \leftarrow \mu$ where the original action form is the special case of μ being a delta distribution (single outcome with probability 1). Defined using probabilistic transitions $\mathcal{C} \rightarrow_p \mathcal{C}'$ (*i.e.* move from $\mathcal{C}$ to $\mathcal{C}'$ with probability p) [28], we introduce a probabilistic action execution as follows:

$$\frac{act : \varphi \leftarrow \mu \quad \mu(\phi^-, \phi^+) = p \quad \mathcal{B} \vDash \varphi}{\langle \mathcal{B}, act \rangle \rightarrow_p \langle (\mathcal{B} \setminus \phi^- \cup \phi^+), nil \rangle}\ act^p$$

Importantly we do not expect programming language *implementations* based on these semantics to draw action outcomes probabilistically. Instead it is used solely for modelling, allowing us to capture environmental effects in a semantics where they are usually ignored.

3.2 Plan Selection and its Probabilistic Extension

BDI agents employ a user-provided plan library to respond to events. Each plan has i) a triggering event defining what event the plan can respond to, ii) a precondition defining what beliefs must hold for the plan to apply, and iii) a

plan-body defining what steps should be taken to execute the plan. To address a pending event (*e.g.* from the external environment), the agent retrieves a set of *relevant* plans: those with a matching triggering event, captured as follows:

$$\frac{\Delta = \{\varphi : P \mid (e' = \varphi \leftarrow P) \in \Pi \wedge e' = e\}}{\langle \mathcal{B}, e \rangle \rightarrow \langle \mathcal{B}, e : (\mid \Delta \mid) \rangle} \; event$$

Given a set of relevant plans, the agent then selects an *applicable* plan (one where the precondition is true) as specified by rule *select*:

$$\frac{\varphi : P \in \Delta \quad \mathcal{B} \models \varphi}{\langle \mathcal{B}, e : (\mid \Delta \mid) \rangle \rightarrow \langle \mathcal{B}, P \rhd e : (\mid \Delta \setminus \{\varphi : P\} \mid) \rangle} \; select$$

If there are no applicable plans a separate rule (unshown) propagates the failure.

Plan Selection Strategies

Notice that the preceding *select* rule does not specify which plan should be selected in case of multiple applicable plans, *i.e.* it is non-deterministic.

However, in practice, we often want more control over which plan is chosen, and different plans are likely to be more/less preferred based on domain-specific characteristics, *e.g.* costs. Therefore, in many implementations the choice is often made deterministically by a plan selection function of the following form:

$$\delta : 2^{\mathcal{B}} \times 2^{\Pi} \rightarrow \Pi \cup \{\bot\}$$

where $\mathcal{B}$ is the belief base and Π the plan library. Given a belief base and a set of (relevant) plans it returns an applicable plan or $\bot$, *i.e.* no applicable plan.

While a common heuristic is to select the plan with the highest order based on some characteristics (*e.g.* preference), it may not lead to globally optimal behaviours due to action side-effects. We argue that it should be possible to *prioritise* plan choice based on plan characteristics, but not assume a totally fixed ordering in order to allow exploration of non-highest order plans that might have better properties. This is akin to discrepancy search techniques [29] to go against the heuristic, and is particularly useful for declarative goals to avoid repeating the same plan obsessively.

To allow non-strict orderings we sample plans based on a probability distribution, *i.e.* with the following plan selection function:

$$\delta^p : 2^{\mathcal{B}} \times 2^{\Pi} \rightarrow Dist(\Pi) \cup \{\bot\}$$

where $Dist(\Pi)$ is the set of discrete probability distribution over the plan library and $\bot$ stands for no applicable plan available. Using δ^p we can define a probabilistic *select* rule as follows:

$$\frac{\varphi : P \in \Delta \quad \delta^p(\mathcal{B}, \Delta) = \mu \quad \mu \neq \bot \quad \mu(\varphi : P) = p}{\langle \mathcal{B}, e : (\mid \Delta \mid) \rangle \rightarrow_p \langle \mathcal{B}, P \rhd e : (\mid \Delta \setminus \{\varphi : P\} \mid) \rangle} \; select^p$$

where μ is the probability distribution returned from δ^p such that any non-relevant and non-applicable plans are being assigned the probability 0.

Trialling different distributions is possible by changing δ^p which could, for example, be extracted from historic data through machine learning. With our approach, it allows quantifying *exact* probabilistic effects of different δ^p choices.

3.3 Intention Selection and its Probabilistic Extension

BDI agents may pursue multiple intentions in parallel, allowing them to respond quickly to new events whilst continuing to handle existing events. As parallelism is interleaved (rather than simultaneous), at each step the agent must decide which intention to progress. Similarly to plan selection, the default CAN semantics specifies a non-deterministic choice for intention selection in following two cases:

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

$$\frac{P \in \Gamma \quad \langle \mathcal{B}, P \rangle \nrightarrow}{\langle E^e, \mathcal{B}, \Gamma \rangle \Rightarrow \langle E^e, \mathcal{B}, \Gamma \setminus \{P\} \rangle} A_{update}$$

That is, we can either select to progress *any* progressable intention (*i.e.* $\langle \mathcal{B}, P \rangle \rightarrow \langle \mathcal{B}', P' \rangle$) or drop *any* unprogressable intention (*i.e.* $\langle \mathcal{B}, P \rangle \nrightarrow$).

As expected, we also want more control over the order of intention execution in practice. This is critical as the wrong choice can cause failure to one or more events, for example, if deadlines are involved (real-time systems, *e.g.* in [30]).

Intention Selection Strategies

Many implementations provide a simple first-in-first-out strategy or round-robin scheduling (which ensures a notion of fairness between the intentions). Alternatively we may force a strict ordering on intentions based on the current situation, *e.g.* deadlines. Similar to plan selection we can express (deterministic) intention choice as a function which *chooses* the next intention to progress:

$$\eta = 2^{\mathcal{B}} \times 2^{\Gamma} \rightarrow \Gamma \cup \{\bot\}$$

where $\bot$ stands for no active intentions available for selection.

Again we argue that forcing a deterministic choice is not always appropriate and that you may require flexibility to choose from a distribution. We provide the following function to allow intention selection based on a distribution:

$$\eta^p = 2^{\mathcal{B}} \times 2^{\Gamma} \rightarrow Dist(\Gamma) \cup \{\bot\}$$

where $Dist(\Gamma)$ is the set of discrete probability distributions over Γ.

While the plan selection decides how to evolve a single intention (in terms of intention-level configuration $\langle \mathcal{B}, P \rangle$), the intention selection determines what it means to evolve an agent (in terms of agent-level configuration $\langle E^e, \mathcal{B}, \Gamma \rangle$). As such, agent-level transitions depend on the intention-level transitions and we need to account for this in the transition probabilities. To have a probabilistic agent step, we assume, for a chosen progressable intention $P \in \Gamma$, $\langle \mathcal{B}, P \rangle \rightarrow_{p'} \langle \mathcal{B}', P' \rangle$ holds, for example, if a plan selection for the given intention P is required based on $select^P$. For unprogressable intentions we have $\langle \mathcal{B}, P \rangle \nrightarrow_1$. We present the following probabilistic intention selection rules:

$$\frac{P \in \Gamma \quad \eta^p(\mathcal{B},\Gamma) = \mu \quad \mu \neq \bot \quad \mu(P) = p \quad \langle \mathcal{B}, P\rangle \rightarrow_{p'} \langle \mathcal{B}', P'\rangle}{\langle E^e, \mathcal{B}, \Gamma\rangle \Rightarrow_{p\cdot p'} \langle E^e, \mathcal{B}', (\Gamma \setminus \{P\}) \cup \{P'\}\rangle} A^p_{step}$$

$$\frac{P \in \Gamma \quad \eta^p(\mathcal{B},\Gamma) = \mu \quad \mu \neq \bot \quad \mu(P) = p'' \quad \langle \mathcal{B}, P\rangle \not\rightarrow_1}{\langle E^e, \mathcal{B}, \Gamma\rangle \Rightarrow_{p''} \langle E^e, \mathcal{B}, \Gamma \setminus \{P\}\rangle} A^p_{update}$$

where $\sum_{p''} p'' + \sum_{p,p'} p \cdot p' = 1$ and $p, p', p'' \in [0, 1]$. Finally, other than the four new probabilistic rules (act^p, $select^p$, A^p_{step}, and A^p_{update}), the other CAN rules (unshown) all transition with uniform probability to future states.

3.4 Situation-aware Distributions for Plan and Intention Selection

The plan and intention selection function δ^p and η^p are abstract and do not specify how to construct the resulting probability distributions in practice. In this section we give a declarative mechanism for calculating situation-aware distributions at *run-time*. In contrast, action outcomes are typically statically defined based on estimates of environmental effects at design time.

Our approach is to specify a situation value function for plans and intentions that assigns them a real-valued *weight* such that if $w_i < w_j$ then we should prefer the plan/intention with the weight w_j. Specific probabilities are then determined through normalisation. As intentions ultimately address external events, we measure the situation value of an intention by considering the characteristics of its related external event. As such, we adopt the notation of [15] and *extend* it to external events by annotating both plans and external events, namely $e : \varphi \leftarrow P[\theta]$ and $e'[\theta]$ where e is an event, φ the context condition, P the plan-body, $e' \in E^e$ an external event, and θ a situation value description. Importantly, each plan and external event can have a *different* situation value description. Same as in [15], we define θ to be $\langle d_0, \{(\varphi_1, d_1), \cdots, (\varphi_n, d_n)\}, f\rangle$ where d_0 is the default value and values d_i are aggregated using function f (*e.g.* to perform a sum) whenever $\mathcal{B} \models \varphi_i$ holds, $d_i \in \mathbb{R}_{\geq 0}$ $(1 \leq i \leq n)$, and $0 \leq n$. Details of the value description such as its expressivity and supported functions can be found in [15].

4 Evaluation

We demonstrate, using a smart manufacturing example and existing probabilistic model checking tools, how to quantitatively model BDI agent programs. Specifically, we evaluate our probabilistic, situation-aware, plan/intention selection against common strategies such as always selecting the most preferred plan. The results are promising, with the intention completion probability using situation-aware distributions being 97% higher than some strictly ordered plan and intention selection strategies. The models are freely available in BigraphER format online[3]. For quantitative analysis we use PRISM by importing the DTMC produced by BigraphER. While we only give details of a single case study, users of the executable semantics can employ BigraphER to "run" models with different settings, *e.g.* external events, plan libraries, custom situation value descriptions.

[3] `https://bitbucket.org/uog-bigraph/prob_bdi_models_sefm21/src/master/`

4.1 Smart Manufacturing Example

We consider a robotic packaging scenario, *extended* from [30], where a robot packs products and moves them to a storage area. Products have specific temperatures and must be packed in a suitable wrapping bag to prevent decay. If the product stays on the production line too long, the temperature increases and it is spoiled and lost. Given multiple waiting products the robot must *choose* which to handle first (intention selection). Once chosen, the robot must then decide which wrapping to use: either premium or standard (plan selection). Premium wrapping is expensive but always stops product decay and never breaks. On the other hand, standard wrapping is cheap, only works if the product temperature remains low, and has a risk of breaking (a negative action outcome).

Complexity arises from the following factors: (1) losses avoided depend on *when* a product is packed, (2) *when* a product is packed determines which wrappings are applicable – earlier packing means cheaper bags, (3) cheaper wrappings introduce uncertainty as they may break. A formal model of the agent system allows us to quantitatively reason about the robot's behaviours under this uncertainty and use these results as evidence, *e.g.* for regulatory certification, or to help improve the design of the robot, *e.g.* using a standard wrapping as often possible but within tolerable failure threshold.

4.2 Agent Design

We consider a simplified scenario with two products that are initially present on the production line, *i.e.* there are no dynamic events. Agent design is given in Fig. 1 and we assume propositional logic with numerical comparisons.

```
1  // Plans
2  e_product1 : true ← goal(success1,e_process_product1,failure1)
3  e_process_product1 : φ11 ← wrap_standard1; move_product_standard1 [θ11]
4  e_process_product1 : φ12 ← wrap_premium1; move_product_premium1 [θ12]
5  e_product2 : true ← goal(success2,e_process_product2,failure2)
6  e_process_product2 : φ21 ← wrap_standard2; move_product_standard2 [θ21]
7  e_process_product2 : φ22 ← wrap_premium2; move_product_premium2 [θ22]
8  // External events
9   e_product1 [θ13]
10  e_product2 [θ23]
```

Fig. 1. Agent design employing the syntax of Section 2.1 combined with the situation value descriptions given in Section 3.4.

Products awaiting processing are captured by external events shown in lines 9 and 10, *e.g.* `e_product1` with its situation value description θ_{13} (explained below). The agent responds to the events using a declarative goal on line 2 that states it wants to achieve the state `success1` (*i.e.* wrapped and moved) through addressing the (internal) event `e_process_product1`; failing if `failure1` (*i.e.* dropped or decayed) ever becomes true. Two plans (in lines 3 and 4), which represent the different wrappings, can handle the event `e_process_product1` each with different situation value descriptions. Event `e_product2` is handled in a similar way (in line 5–7).

There is a probabilistic outcome for the `move_product_standard1` action, such that it has a 10% chance of causing `failure1` by dropping the product accidentally, else it succeeds (adding `success1` to the beliefs), whereas `move_product_premium1` action always succeeds. In Section 4.5 we will investigate the effect with varying probability. To allow situational awareness, we encode (discrete) temporal information, for progress and deadline, as agent belief atoms. Progress determines how far (in terms of agent steps) an agent is through an intention, while deadline determines how many steps we can make before the product spoils. Mirroring implementations, we update beliefs based on timings in the background, without executing an explicit action. In this case, the progress increases whenever a specific intention is stepped, whereas deadline decreases after a step of *any* intention.

Table 1 gives the specifications for quantitative reasoning. A short commentary is as follows. $deadline_1 = 8$ and $deadline_2 = 12$ are the initial deadlines of two external events, namly `e_product1` and `e_product2`. The precondition $\varphi_{11} = deadline_1 \geq 3$ indicates that $deadline_1$ is greater than or equal to 3. The situation value description $\theta_{11} = \langle 1, \{\varphi_{11}, 1\}, sum \rangle$ indicates that if φ_{11} holds, then $\theta_{11}(\varphi_{11}) = 1 + 1 = 2$. The situation value description θ_{13} for the external event `e_product1` is defined as a function $(deadline_1 + progress_1)^{-3}$. Intuitively, if $deadline_1 + progress_1$ is smaller relative to other products, then it has been progressed less and the deadline is approaching, so it is more urgent. Importantly, the choice of situation value descriptions are made by the agent designer, *i.e.* $(deadline_1 + progress_1)^{-3}$ was their choice. Our approach enables the analysis of alternative functions quantitatively, before deploying the agent.

Table 1. Quantitative specifications with $x \in \{1, 2\}$.

Initial Deadlines	**Preconditions**	**Situation Value Descriptions**
$deadline_1 = 8$	$\varphi_{x1} = deadline_x \geq 3$	$\theta_{x1} = \{1, \{\varphi_{x1}, 1\}, sum\}$
$deadline_2 = 12$	$\varphi_{x2} = deadline_x \geq 0$	$\theta_{x2} = \{1, \{\varphi_{x3}, 1\}, sum\}$
	$\varphi_{x3} = 3 \geq deadline_x \geq 0$	$\theta_{x3} = (deadline_x + progress_x)^{-3}$

Table 2. Plan and intention selection strategies.

Plan Selection Strategies	Intention Selection Strategies
SMP: Select Most Preferred	**SMU**: Select Most Urgent
PSD: Preference Situational Distribution	**FIFO**: First-In-First-Out
	RR: Round Robin
	PUSD: Pure Urgency Situational Distribution
	LUSD: Layered Urgency Situational Distribution
	OLUSD: Optimised Layered Urgency Situational Distribution

4.3 Plan and Intention Selection Strategies

Table 2 lists the plan/intention selection strategies we analyse. We do not evaluate uniform random plans or intention selection strategies, as these do not capture any domain specific information (*e.g.* regarding preferences). Whereas **SMP** plan selection always selects the highest weighted plan, **PSD** selects a plan by sampling distribution based on preference. For intention selection, **SMU** always selects the intention closest to the deadline similar to **SMP**. **FIFO** and **RR** are fixed orders where the former always selects the intention which arrives first and the latter selects each intention in turn. **PUSD** selects an intention by sampling from distribution where situation value description is given by $(deadline + progress)^{-3}$. Unlike **PUSD**, **LUSD** only deems an intention urgent if the product is not packed or spoiled. As such, it will not select an intention in which the product is packed when there is another intention whose product is not packed. Finally, **OLUSD** selects an intention similarily to **LUSD** but the situation value description is revised to be $|deadline + progress - steps\_required|^{-3}$, which accounts for the steps remaining to pack a product (to avoid spoilage).

4.4 Plan and Intention Selection Analysis

To perform quantitative analysis, we use BigraphER to generate a DTMC—the underlying transition system of probabilistic bigraphs [18]—with bigraphs as states and probabilities as transitions. Each state is labelled by bigraph patterns [22]: if the pattern matches the current state then the predicate is true. In our example, we reason about the dynamic properties using Probabilistic Computation Tree Logic (PCTL) [31]. For example, the property $\mathcal{P}_{=?}\mathbf{F}[\phi]$ expresses the expected probability of ϕ holding *eventually* (in some state). We use S1, F1 (resp. S2, F2) to denote product 1 (resp. 2) successfully, or unsuccessfully being processed by the robot. For model analysis we use these as state-labels[4] in the transition system for all states where a particular product succeeds/fails.

[4] Implemented using bigraph patterns, where a specific match is constructed that only holds when that specific product was processed/failed.

Table 3 gives the probability of processing the products either successfully or with a failure, under the plan/intention selection strategies listed in Table 2. For example, $\mathcal{P}_{=?}\mathbf{F}[\text{S1} \wedge \text{S2}]$ is the probability both products being processed successfully.

Table 3 shows the necessity for good plan/intention selection, with the first 3 combinations *never* successfully processing both products, *i.e.* (S1, S2), and **PUSD** having very limited success ($p = 0.03$). In particular, the intention strategy of **RR** (which selects each intention in turn) is the worst, failing both products in all cases. Using **PUSD** has an almost 50% chance of succeeding with product 1 or failing both. This indicates the weighting function is skewed toward product 1 at the detriment of product 2, leading to the improved **LUSD** strategy. This is a key advantage of our approach: discovering potential pitfalls and trialling new strategies without changing the underlying agent programs and semantics. Similar reasoning, that now product 2 was succeeding more often, led to another strategy **OLUSD** being trialled with extremely good success rates, *i.e.* $p = 0.98$. We should never expect the probability of (S1, S2) = 1 due to the action outcome uncertainty (*e.g.* the wrapping bag breaks).

Table 3. Probability of product 1, product 2 for the properties, *e.g.* (S1, S2) with different plan and intention selection strategies listed in Table 2.

		Intention					
		SMU		**FIFO**		**RR**	
Plan	**SMP**	(S1,S2) 0	(S1,F2) 0.9	(S1,S2) 0	(S1,F2) 0	(S1,S2) 0	(S1,F2) 0
		(F1,S2) 0	(F1, F2) 0.1	(F1,S2) 0.9	(F1,F2) 0.1	(F1,S2) 0	(F1,F2) 1
	PSD	(S1,S2) 0	(S1,F2) 0.93	(S1,S2) 0	(S1,F2) 0	(S1,S2) 0	(S1,F2) 0
		(F1,S2) 0	(F1,F2) 0.07	(F1,S2) 0.93	(F1,F2) 0.07	(F1,S2) 0	(F1,F2) 1
		PUSD		**LUSD**		**OLUSD**	
Plan	**SMP**	(S1,S2) 0.03	(S1,F2) 0.48	(S1,S2) 0.510	(S1,F2) 0	(S1,S2) 0.97	(S1,F2) 0
		(F1,S2) 0.08	(F1,F2) 0.41	(F1,S2) 0.482	(F1,F2) 0.008	(F1,S2) 0.037	(F1,F2) 0
	PSD	(S1,S2) 0.03	(S1,F2) 0.49	(S1,S2) 0.513	(S1,F2) 0	(S1,S2) 0.98	(S1,F2) 0
		(F1,S2) 0.08	(F1,F2) 0.4	(F1,S2) 0.481	(F1,F2) 0.05	(F1,S2) 0.02	(F1,F2) 0

In this example, we find that plan selection has limited effect compared to intention selection, which is key to this application. This itself is a valuable insight. In general, probabilistic sampling that improves success rates, even marginally, should be used as it can result in great savings—particularly in large scale processes, *e.g.* an expected two-product successful behaviour tending to occur 98%

of the time instead of 97%. Given the complexity of agent behaviours, determining this expected probability precise, without such a model, would be difficult.

Table 4. DTMC generation: final size and timing.

Strategies	States	Transitions	Build time (s)	Rule Applications
(**SMP**, **SMU**)	31	30	66.57	217
(**SMP**, **FIFO**)	31	30	65.85	211
(**SMP**, **RR**)	19	18	52.13	143
(**PSD**, **SMU**)	36	36	92.26	273
(**PSD**, **FIFO**)	36	36	92.25	268
(**PSD**, **RR**)	19	18	51.72	143
(**SMP**, **PUSD**)	572	845	2447.37	5300
(**SMP**, **LUSD**)	323	478	1518.36	3116
(**SMP**, **OLUSD**)	323	478	1481.07	3116
(**PSD**, **PUSD**)	697	1039	17435.90	6836
(**PSD**, **LUSD**)	417	614	2106.64	4157
(**PSD**, **OLUSD**)	417	614	2098.51	4157

Table 4 details the DTMC that was used in the evaluation of each property: the number of states and transitions, build time, and rule applications. The last is 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, this includes belief revision, where we see only final output of a step of executing an action. As the internal steps still have to be generated, much of the build time is spent doing that—accounting for the low number of states, but large build time. We also have to check all required rules and, as bigraphs do not natively support numerical types, this includes many *generated* rules for different parameter values.

The build times for non-interleaved intention selection strategies, *e.g.* (**SMP**, **SMU**) and (**PSD**, **RR**) is in the order of minutes whereas the build times for interleaved selection strategies, *e.g.* (**SMP**, **PUSD**) and (**PSD**, **PUSD**), is significantly higher (up to 5 hours). This is expected due to the combinatorial nature of interleavings and the large number of rules that need to be checked for applicability in each state. Since our executable semantics is intended to be used at design time we do not believe this to be an issue in practice. Model optimisations may be possible, or statistical model checking used, for particularly large agent designs, and ultimately there may be a numerical plug-in for BigraphER.

4.5 Action Outcome Analysis

The effects of different action outcomes are shown in Figure 2 where the probability of standard wrapping failing is increased from 10% to 90% for two strategy pairs: (**PSD**, **SMU**) and (**PSD**, **OLUSD**).

We can see that negative action outcomes have a much larger effect on strictly ordered intention selection (**SMU**), *e.g.* the probability of (S1,F2) decreases from over 90% to below 40%. Meanwhile, (**PSD**, **OLUSD**) is more robust to action outcome changes. For example, the probability of (S1, S2) in **(PSD, OLUSD)** has a minor decrease of no more than 20%. This is due to increased interleaving of these two intentions, rendering the standard wrapping inapplicable more often.

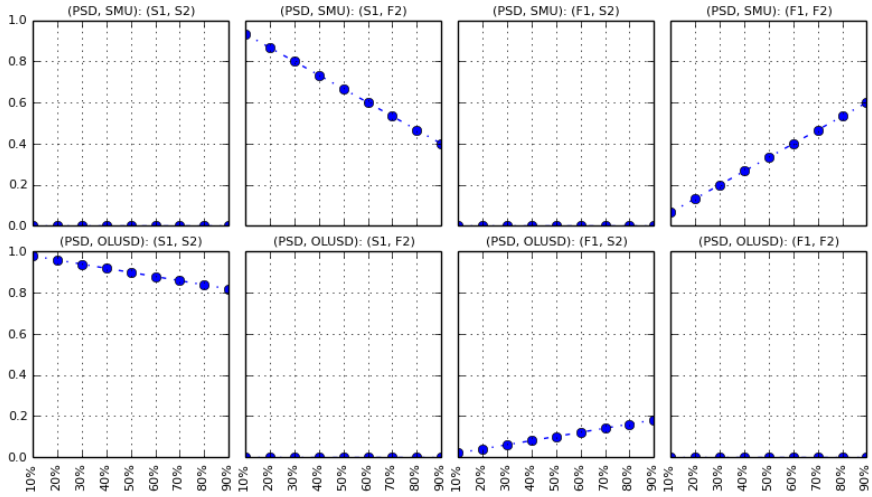

Fig. 2. Probability of reaching the end state (product 1, product 2) with increasing failure probability in (**PSD**, **SMU**) and (**PSD**, **OLUSD**).

When the cases become less complex, *e.g.* there is only one product with plenty time to process and all actions always succeed, the plan/intention choice of a BDI agent becomes trivial. In another words, our approach is particularly useful when situations are not straightforward and have complex domain information. Future work, however, is required to account for the *cost*, in terms of wrapping bags, of achieving different success rates and robustness to action outcomes while keeping the overall cost low, *i.e.* multi-objective optimisation.

5 Related Work

We are not the first to consider probabilistic verification of BDI agents. The work [32] uses a two-stage verification methods that first *generates* a model through program model checking (of a system implementation), and then converts this model to PRISM input format for analysis. However, unlike our focus on probabilistic extensions of the BDI semantics itself, the BDI agent used in [32] does not contain any probabilistic aspects. Instead, the *environment*

where the agent executes enables the probabilistic reasoning. Similarly, the work of [33] facilitates probabilistic verification of BDI agents by encoding them in PRISM. In this case, instead of generating the model based on an implementation, they implement a significantly simplified version of AgentSpeak directly in PRISM. The simplifications deviate from realistic BDI agents, *e.g.* enabling truly-concurrent intentions (and no intention selection) and treating plan selection as non-deterministic. Our approach captures an extension of the *full* CAN semantics while still providing PRISM verification capabilities.

Works studying plan and intention selection strategies have also been conducted within the BDI community. For example, the work of [30] compiles agent programs to TÆMS (Task Analysis, Environment Modelling, and Simulation) framework to represent the coordination aspects of problems such as "enables" and "hinders" relations between tasks. A Design-To-Criteria scheduler is then used for intention selection to determine the full set of decisions that the agent needs to perform. An increasingly popular topic in the BDI community is intention progression [34], *e.g.* the contest[5]. The intention problem includes the means (i.e. plan) to achieve a given event and which of the currently adopted plans (i.e. intentions) to progress at the current moment, when handling multiple intentions in parallel. Unlike our focus on automated quantitative analysis of BDI agents, their goal (same as [30]) is to help the agent to make better decisions, by modifying or replacing the original BDI reasoning entirely, through other advanced decision-making techniques such as automated planning techniques [35]. For example, the work [36] showed that many of the intention progress issues can be modelled in planning domain definition language (PDDL) [37] (the defacto standard planning language) and resolved through suitable planners. such as modern highly efficient (online) planner [38]. Finally, it is not a new idea to integrate advanced decision-making techniques into BDI agents to improve performance. There is a large amount of work (surveyed in [39]) to employ planning to dynamically synthesise new plans to achieve an event even when no pre-defined plan worked or exists. One work [40], for example, shows in detail how the integration of planning and BDI can be done at the semantic level.

Besides BDI agents, quantitative verification techniques have also applied to other types of agent systems. For example, the work of [41] considers uncertain communication channels between systems of interacting agents. For verification the multi-agent system is transformed to finite state Markov chains for establishing quantitative temporal properties of the system. Similar to our evaluation of plan/intention selection strategies, the work of [42] provides a quantitative assessment for a decentralised control policies in multi-vehicle scenarios. Specifically they study conflict resolution policies to ensure that a policy never causes collisions under some mild assumptions on the initial conditions. For an overview of general agent-based verification we refer to [43] for the interested readers.

[5] `https://sites.google.com/site/intentionprogression/home`

6 Conclusions

A quantitative evaluation and comparison framework can aid design-time specification, allowing us to reason about rational agents operating under uncertainty, for example due to uncertain environments or failure prone actuators, and inherently quantifiable agent characteristics such as plan preference.

We have extended the CAN language (which formalises the behaviour of a classical BDI agent) to a probabilistic setting, which allows both probabilistic action outcomes and probabilistic plan and intention selection. The extended semantics employs probabilistic bigraphs, which enable quantitative analysis with BigraphER and probabilistic model checking in PRISM. Importantly, our executable framework allows (non-expert) users to experiment with their own agent models without worrying about the underlying bigraph theory.

Through a smart manufacturing example we have shown that it is possible to reason about several plan and intention selection strategies, and that probabilistic plan and intention selection strategies can reduce the impact of undesirable outcomes, compared with ordered or fixed strategies. In this example, we found that plan selection has limited effect compared to intention selection, which is a valuable insight. In particular, due to the agent making smarter intention selection choices, the impact of action outcomes can be marginal—even when the failure probabilities are large.

Acknowledgements

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

References

[1] Rao, A.S.: 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) 42–55

[2] Winikoff, M., Padgham, L., Harland, J., Thangarajah, J.: Declarative and procedural goals in intelligent agent systems. In: the 8th International Conference on Principles of Knowledge Representation and Reasoning, Morgan Kaufman (2002)

[3] Sardina, S., Padgham, L.: A BDI agent programming language with failure handling, declarative goals, and planning. In: Autonomous Agents and Multi-Agent Systems. Volume 23., Springer (2011) 18–70

[4] Hindriks, K.V., Boer, F.S.D., Hoek, W.V.d., Meyer, J.J.C.: Agent programming in 3APL. Autonomous Agents and Multi-Agent Systems **2**(4) (1999) 357–401

[5] Dastani, M.: 2APL: a practical agent programming language. Autonomous agents and multi-agent systems **16**(3) (2008) 214–248

[6] Winikoff, M.: JACK intelligent agents: an industrial strength platform. In: Multi-Agent Programming, Springer (2005) 175–193

[7] Bordini, R.H., HüJomi, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason. Volume 8. John Wiley & Sons (2007)
[8] Pokahr, A., Braubach, L., Jander, K.: The Jadex project: programming model. In: Multiagent Systems and Applications, Springer (2013) 21–53
[9] Benfield, S.S., Hendrickson, J., Galanti, D.: Making a strong business case for multiagent technology. In: the 5th International Joint Conference on Autonomous Agents and Multiagent systems, ACM (2006) 10–15
[10] Braubach, L., Pokahr, A., Lamersdorf, W.: Negotiation-based patient scheduling in hospitals. In: Advanced Intelligent Computational Technologies and Decision Support Systems. (2014) 107–121
[11] McArthur, S., Davidson, E., Catterson, V., Dimeas, A., Hatziargyriou, N., Ponci, F., Funabashi, T.: Multi-agent systems for power engineering applications – part i: Concepts, approaches, and technical challenges. IEEE Transactions on Power systems **22**(4) (2007) 1743–1752
[12] Bordini, R.H., Fisher, M., Visser, W., Wooldridge, M.: Verifying multi-agent programs by model checking. Autonomous Agents and Multiagent Systems **12**(2) (2006) 239–256
[13] Dennis, L.A., Fisher, M., Lincoln, N.K., Lisitsa, A., Veres, S.M.: Practical verification of decision-making in agent-based autonomous systems. Automated Software Engineering **23**(3) (2016) 305–359
[14] Chen, H.: Applications of cyber-physical system: a literature review. Journal of Industrial Integration and Management **2**(03) (2017) 1750012
[15] Padgham, L., Singh, D.: Situational preferences for BDI plans. In: the 2013 international conference on Autonomous agents and multi-agent systems. (2013) 1013–1020
[16] Archibald, B., Calder, M., Sevegnani, M., Xu, M.: Modelling and verifying bdi agents with bigraphs. arXiv preprint arXiv:2105.02578 (2021)
[17] Milner, R.: The space and motion of communicating agents. Cambridge University Press (2009)
[18] Archibald, B., Calder, M., Sevegnani, M.: Probablistic bigraphs. arXiv preprint arXiv:2105.02559 (2021)
[19] Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: the 23rd International Conference on Computer Aided Verification (CAV'11). Volume 6806 of LNCS., Springer (2011) 585–591
[20] Sardina, S., Padgham, L.: Goals in the context of BDI plan failure and planning. In: the 6th International Joint Conference on Autonomous Agents and Multiagent Systems. (2007) 16–23
[21] Sevegnani, M., Kabác, M., Calder, M., McCann, J.A.: Modelling and verification of large-scale sensor network infrastructures. In: 23rd International Conference on Engineering of Complex Computer Systems, ICECCS. (2018) 71–81
[22] Benford, S., Calder, M., Rodden, T., Sevegnani, M.: On lions, impala, and bigraphs: modelling interactions in physical/virtual spaces. ACM Transactions on Computer-Human Interaction (TOCHI) **23**(2) (2016) 1–56
[23] Tsigkanos, C., Li, N., Jin, Z., Hu, Z., Ghezzi, C.: Scalable multiple-view analysis of reactive systems via bidirectional model transformations. In: 35th IEEE/ACM International Conference on Automated Software Engineering. (2020) 993–1003
[24] Bundgaard, M., Sassone, V.: Typed polyadic pi-calculus in bigraphs. In: ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming. (2006) 1–12
[25] Sevegnani, M., Pereira, E.: Towards a bigraphical encoding of actors. In: International Workshop on Meta Models for Process Languages. (2014)

[26] Sevegnani, M., Calder, M.: BigraphER: Rewriting and analysis engine for bigraphs. In: the 28th International Conference on Computer Aided Verification (CAV'16). (2016) 494–501
[27] Younes, H.L., Littman, M.L.: PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-162 **2** (2004) 99
[28] Di Pierro, A., Wiklicky, H.: An operational semantics for probabilistic concurrent constraint programming. In: the 1998 International Conference on Computer Languages, IEEE (1998) 174–183
[29] Prosser, P., Unsworth, C.: Limited discrepancy search revisited. Journal of Experimental Algorithmics (JEA) **16** (2011) 1–6
[30] Bordini, R.H., Bazzan, A.L.C., Jannone, R.D.O., Basso, D.M., Vicari, R.M., Lesser, V.R.: AgentSpeak (XL) efficient intention selection in BDI agents via decision-theoretic task scheduling. In: the first international joint conference on Autonomous agents and multiagent systems: part 3. (2002) 1294–1302
[31] Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal aspects of computing **6**(5) (1994) 512–535
[32] Dennis, L.A., Fisher, M., Webster, M.: Two-stage agent program verification. Journal of Logic and Computation **28**(3) (2018) 499–523
[33] Izzo, P., Qu, H., Veres, S.M.: A stochastically verifiable autonomous control architecture with reasoning. In: IEEE Conference on Decision and Control. (2016) 4985–4991
[34] Logan, B., Thangarajah, J., Yorke-Smith, N.: Progressing intention progression: A call for a goal-plan tree contest. In: AAMAS. (2017) 768–772
[35] Geffner, H., Bonet, B.: A concise introduction to models and methods for automated planning. Synthesis Lectures on Artificial Intelligence and Machine Learning **8**(1) (2013) 1–141
[36] Xu, M., McAreavey, K., Bauters, K., Liu, W.: Intention interleaving via classical replanning. In: 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), IEEE (2019) 85–92
[37] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D.: Pddl-the planning domain definition language. Technical report (1998)
[38] Keller, T., Eyerich, P.: Prost: Probabilistic planning based on uct. In: Twenty-Second International Conference on Automated Planning and Scheduling. (2012)
[39] Meneguzzi, F., Silva, L.: Planning in BDI agents: a survey of the integration of planning algorithms and agent reasoning. The Knowledge Engineering Review **30** (2015) 1–44
[40] Xu, M., Bauters, K., McAreavey, K., Liu, W.: A formal approach to embedding first-principles planning in bdi agent systems. In: International Conference on Scalable Uncertainty Management, Springer (2018) 333–347
[41] Dekhtyar, M.I., Dikovsky, A.J., Valiev, M.K.: Temporal verification of probabilistic multi-agent systems. In: Pillars of computer science. Springer (2008) 256–265
[42] Pallottino, L., Scordio, V.G., Frazzoli, E., Bicchi, A.: Probabilistic verification of a decentralized policy for conflict resolution in multi-agent systems. In: IEEE International Conference on Robotics and Automation. (2006) 2448–2453
[43] Bakar, N.A., Selamat, A.: Agent systems verification: systematic literature review and mapping. Applied Intelligence **48**(5) (2018) 1251–1274