Observable and Attention-Directing BDI Agents for Human-Autonomy Teaming

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

Human-autonomy teaming (HAT) scenarios feature humans and autonomous agents collaborating to meet a shared goal. For effective collaboration, the agents must be transparent and able to share important information about their operation with human teammates. We address the challenge of transparency for Belief-Desire-Intention agents defined in the Conceptual Agent Notation (CAN) language. We extend the semantics to model agents that are observable (i.e., the internal state of tasks is available), and attention-directing (i.e., specific states can be flagged to users), and provide an executable semantics via an encoding in Milner’s bigraphs. Using an example of unmanned aerial vehicles, the BigraphER tool, and PRISM, we show and verify how the extensions work in practice.

1 Introduction

Autonomous agents e.g., robots [27] are becoming increasingly. As the autonomy increases, so does the tasks that can be performed [28], and more emphasis is being placed on how agents can form teams, alongside humans, to achieve a common goal, a.k.a. human-autonomy teaming (HAT) [32, 19].

Transparency is especially important in HAT. While there is no single definition [38, 33], we draw on IEEE P7001 standard definition [15] that transparency is measurable, testable, and is “the transfer of information from an autonomous system or its designers to a stakeholder, which is honest, contains information relevant to the causes of some action, decision or behaviour and is presented at a level of abstraction and in a form meaningful to the stakeholder.” Here, we focus on two transparency requirements: observability of internal states and attention-directing of a human teammate.

To illustrate transparency, consider a HAT scenario where a human and an Unmanned Aerial Vehicle (UAV) survey a site together. The UAV performs site surveys through imaging and retrieves objects if needed, while the human assembles and interprets geographically related information (obtained from the UAV). To coordinate an effective exploration, it is essential to have the appropriate transparency into what tasks either is performing and the progress of the relevant tasks. For example, the human needs to know if the UAV has finished surveying an area, or how much is left, before assembling analysis for the same area. Transparency must also be supported to allow attention-direction during autonomy failure (e.g., the engine is malfunctioning) and to highlight changes in the environment, e.g., when a specific land condition is detected by the UAV the human should be notified immediately for inspection.

We tackle the challenge of transparency by modelling and verifying agents that are observable [5], i.e., the status of tasks is known, and attention-directing, i.e., specific states can be flagged to users. Both observability and attention-directing are recommended features in a practical HAT system engineering guide [17], though strictly this guide refers to the former as transparency and the latter as augmenting cognition, which we consider part of transparency.

To demonstrate these features, we extend a model of Belief-Desire-Intention (BDI) agents proposed in [2] that presents an executable semantics of the CAN agent language [40] based on Milner’s Bigraphs [18]. BDI agents [6, 22] consist of, (B)eliefs: 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 are chosen as they are represented in a declarative fashion (easy to log) with some self-explanatory concepts e.g. the beliefs of the agent. 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 [21], CAN includes advanced BDI agent behaviours such as reasoning with declarative goals, concurrency, and failure recovery, which are necessary for our UAV example modelled in Section 3. Importantly, although we focus on the 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. Besides the work of [7] that focuses on transparent ethical reasoning (i.e. why a decision has been made by an agent), we believe this is the first formal analysis of transparency applied to mainstream BDI agents.

We make the following research contributions: (1) an extension of CAN language semantics to support transparency, (2) an extension of the bigraph based executable semantics framework, (3) an evaluation based on Unmanned Aerial Vehicles (UAVs) to illustrate the framework.

2 Framework

In this section, we provide transparency mechanisms for observability, i.e. documenting the status of tasks and their progress toward completion, and attention-directing, i.e. flagging information about specific states—of the situation or agent itself—to a user. We start with some CAN language background.

2.1 CAN Background

The CAN language formalises a classical BDI agent consisting of a belief base $B$ and a plan library $\Pi$. The belief base $B$ is a set of formulas encoding the current beliefs and has belief operators for entailment (i.e. $B \models \varphi$), and belief atom addition (resp. deletion) $B \cup \{b\}$ (resp. $B \setminus \{b\}$) and any logic over the belief base $B$ is allowed providing entailment is supported. A plan library $\Pi$ is a collection of plans of the form $e : \varphi \leftarrow P$ with $e$ the triggering event, $\varphi$ the context condition, and $P$ the plan-body. Events can be either be external (i.e. from the environment in which the agent is operating) or internal (i.e. sub-events that the agent itself tries to accomplish). In the plan-body, we use $P_1; P_2$ for sequence and $\text{goal}(\varphi_s, P, \varphi_f)$ for the declarative goal failing if $\varphi_f$ holds and exiting successfully if $\varphi_s$ holds (see [25]).

A basic configuration $\langle B, P \rangle$, where $P$ is the plan-body being executed (i.e. the current intention), is used in rules that define the execution of a single intention. The agent configuration is defined as $\langle E^c, B, \Gamma \rangle$ where $E^c$ denotes the a set of pending external events and $\Gamma$ the current set of intentions (i.e. partially executed plan-body programs). The agent-level evolution is specified by the transitions over the agent configuration. For example, the agent-level transition to progressing intention which is progressable ($\langle B, P \rangle \rightarrow \langle B', P' \rangle$) or dropping any unprogressable intention ($\langle B, P \rangle \rightarrow$) can be given as follows:

$$
\text{Start} \quad P \in \Gamma \quad \langle B, P \rangle \rightarrow \langle B', P' \rangle \quad A_{\text{step}}
$$

$$
\text{Update} \quad P \in \Gamma \quad \langle B, P \rangle \rightarrow \langle E^c, B, \Gamma \setminus \{P\} \rangle \quad A_{\text{update}}
$$

We refer the reader to [40, 25] for a full overview of the semantics of CAN.

2.2 Observability

Observability captures what an agent is doing so that human teammates can coordinate their tasks for effective collaboration. As intention represents tasks, we focus on the status and progress of intentions.
2.2.1 Intention Status

As a high-level planning language, CAN is agnostic to many practical issues — including transparency. One issue of CAN is the inability to tell the status of an intention. For example, the rule $A_{\text{update}}$ in Fig. 1 discards an intention that cannot do any more intention-level transitions: both if it has already succeeded, or if it failed, and there is no way to determine which was the case. In practice, we may find it helpful, particularly in HAT, to have precise knowledge on the success or failure status of an intention.

We extend CAN semantics to allow intention status. Following work [13], we introduce four status values for an external event (that ultimately gives rise to an intention): pending, active, success, and failure along with an unique identifier $I$. These values indicate when an event is not addressed yet (pending), currently being addressed (active), successfully addressed (success), and addressed with a failure (failure). Unique identifiers enable the agent to track the means-end relations between the events and the related intentions, and differentiate intentions from others.

Figure 1 presents the original rules and our status-enabled rules where $E^e$ is a set of external events $E^e$ that the agent is required to respond, a belief set $B$, and intention base $\Gamma$ is a set of partially executed plan-bodies $P$ that the agent has committed to. While the original rule $A_{\text{event}}$ deletes events once they have turned into intentions, the new rule $A_{\text{new}_{\text{event}}}$ instead switches the status of the event to active. When intentions progress, the status of related external events remains active, until the intention is removed either with success (if we reached a nil) via $A_{\text{new}_{\text{update,suc}}}$ or failure via $A_{\text{new}_{\text{update,fail}}}$.

2.2.2 Intention Progress

Besides the intention status, it is also useful to estimate the progress of intentions. For example, the human teammate may want to know how long the UAV needs to finish surveying to plan their next tasks.

In BDI agents, goal-plan trees (shown in Fig. 2) are a canonical representation of intentions [34]. The root of the tree is an external event, and its children are plans that can handle this event. Plans may contain sub-events, giving rise to a tree structure that represents all possible ways of achieving a task. Traditionally, for simplicity, goal-plan trees often do not represent actions and contain only
goals/subgoals and plans e.g. in [34]. However, we need to include the actions—a crucial part in agent execution—to perform faithful intention progress estimation. Finally, executing a BDI program gives one execution trace, i.e. a path through the tree.

To illustrate goal-plan trees and execution traces, consider the following two plans as follows:

\[ P_1 = e_1 : \phi_1 \leftarrow a_1 ; a_2 ; \quad P_2 = e_1 : \phi_2 \leftarrow a_3 ; a_4 ; a_5. \]

These two plans are visualised as the goal-plan tree \( T \) given in Fig. 2. The goal-plan tree \( T \) expresses that, given an event \( e_1 \), it has two plan choices, namely plan \( P_1 \) and \( P_2 \), where the edges between the event \( e_1 \) and plan \( P_1 \) and \( P_2 \) are labelled with the relevant context condition \( \phi_1 \) and \( \phi_2 \). If plan \( P_1 \) is selected (given \( \phi_1 \) holds), it has two sequenced actions \( a_1 \) and \( a_2 \) to execute where the edge between the plan to its plan-body is labelled with the position number. As such, we can obtain one execution trace \( e_1 ; P_1 ; a_1 ; a_2 \) and the other one (due to the selection of plan \( P_2 \)) \( e_1 ; P_2 ; a_3 ; a_4 ; a_5 \).

For intention progress estimation, we use a compile-time process to obtain all possible execution traces. For each trace, we record the maximum length of the execution trace. During execution, the agent tracks its current execution trace so that after an agent-step we can determine the maximum length of the full trace this current trace belongs to. The progress of an intention is then estimated as the ratio of the position of the last element in the current trace and the length of the full trace this current trace belongs to. For example, given a trace \( e_1 ; P_1 ; a_1 ; a_2 \), its maximum length is 4 and the position for each element in this trace is 1 for \( e_1 \), 2 for \( P_1 \), 3 for \( a_1 \), and 4 for \( a_2 \). If the current execution trace is \( e_1 ; P_1 ; a_1 \) then the progress estimation is \( 3/4 = 75\% \) (as \( a_1 \) is the last element). In the case of failure recovery, for example, if action \( a_1 \) failed, the agent backtracked to event \( e_1 \) and subsequently selected the plan \( P_2 \). Then the current trace becomes \( e_1 ; P_2 \) (against the full trace \( e_1 ; P_2 ; a_3 ; a_4 ; a_5 \)) with the progress estimation \( 2/5 = 40\% \). The execution trace is maintained separately for each intention.

2.3 Directing Attention

Agents should be able to direct the attention of teammates when particular states become relevant, e.g. when adversarial situations are encountered. In BDI programming languages such as AgentSpeak, a change in the belief base generates an event, which subsequently requires an agent to select a plan. For example, if a belief atom \( b \) is true, an event in the form of \(+ b\) is added to the desires. While useful, this one-to-one mapping between belief and event is also limiting, especially in the context of directing attention e.g. it prohibits the generation of multiple events. Further, multiple events generated by a single belief change may require adjustment throughout the operational life of the agent.

---

1 For practicality, we do not allow recursive plans as this can lead to infinite traces.
2 To ensure each trace is uniquely identifiable, the same action occurring in a different plan is treated as a different action.
1  // Initial beliefs
2  ¬sensor_malfunc, ¬engine_malfunc
3  // External events
4  ⟨e_retrv, identifier1⟩
5  // Motivation library events
6  parked <- ⟨e_parked, identifier2⟩
7  // Plan library
8  e_retrv : φ <- take_off; goal(at_destination, e_path1, fc); retrieve
9  e_retrv : φ <- take_off; goal(at_destination, e_path2, fc); retrieve
10  e_retrv : φ <- take_off; goal(at_destination, e_path3, fc); retrieve
11  e_retrv : sensor_malfunc <- return_base
12  e_retrv : engine_malfunc <- activate_parking
13  e_path1 : true <- navigate_path_1
14  e_path2 : true <- navigate_path_2
15  e_path3 : true <- navigate_path_3
16  e_parked : true <- send_GPS
where φ = ¬sensor_malfunc ∧ ¬engine_malfunc,
f ∗ = sensor_malfunc ∨ engine_malfunc

Figure 3: UAV BDI agent design.

We take an approach to directing attention that employs the motivation library of [26]. This allows new (and multiple) intentions (e.g. to communicate with a human) to be created dynamically when a particular state is recognised (i.e. the correct beliefs hold). To be precise, a motivation library (specified by the agent designers) \( \mathcal{M} \) is a set of rules of the form: \( \psi \rightsquigarrow ⟨e, I⟩ \) where \( \psi \) is a world state, \( e \) an event, and \( I \) an identifier. The semantics of motivation execution is specified as follows:

\[
\psi \rightsquigarrow ⟨e, I⟩ \in \mathcal{M} \iff \psi, ⟨e, I⟩ \notin Γ \\
(E^σ, \mathcal{B}, Γ) \rightarrow (E^σ \cup \{⟨e, I, \text{active}⟩\}, \mathcal{B} \cup \{⟨e, I⟩\}) \quad A_{\text{motive}}
\]

Informally, if the agent believes \( \psi \), it should adopt the event \( ⟨e, I⟩ \) (if it has not adopted it before). As such, the programmers can specify (and revise) multiple rules \( \psi \rightsquigarrow ⟨e_1, I_1⟩, \ldots, \psi \rightsquigarrow ⟨e_n, I_n⟩ \) to ensure a correct set of events to be generated at any time when \( \psi \) holds. From our experience, this approach also benefits from the modularity principle by separating the dynamics of desires and design of plan library.

3 UAV Example

To demonstrate our new transparency mechanisms, we give a simple Unmanned Aerial Vehicles (UAVs) example, where UAVs are used for object retrieval tasks, e.g. package delivery, and might be subject to engine or sensor malfunction. The agent design is in Fig. 3 where we heavily rely on declarative goal and failure recovery features in CAN. There is one main retrieve task, initiated by external event \( e_{\text{retrv}} \) (line 4), which can be handled by five relevant plans (lines 8–12). The first 3 plans provide different flight paths after take-off in which they all have a declarative goal. For example, the declarative goal
Observable and Attention-Directing BDI Agents for HAT

goal(at_destination, e_path1, fc) says that it is achieved if it believes at_destination holds and failed if fc holds. When fc holds, the other two plans (lines 11 and 12) perform safe recovery in the event of engine or sensor malfunction. In the case of engine malfunction, instead of returning to base (when sensor is faulty), the plan in line 12 instructs the UAV to land and park itself. Once the UAV is landed and parked, the human teammate should be notified of the situation and respond accordingly. Therefore, for directing attention, the motivation library is given in line 6, so that if the belief that the UAV parked itself (i.e. parked) holds after the action activate_parking, the agent should adopt the event e_parked to send GPS coordinates (line 16) to a human teammate to retrieve the UAV. For succinct presentation, we do not show the encoding of actions such as take_off and retrieve (which can found in the online model[3]).

3.1 Analysis

We encoded the CAN semantics in bigraphs [18, 2], which has been used previously to encode the semantics of process calculi [8, 30]. Our bigraph encoding permits execution and symbolic analysis and we employ BigraphER [29]—an open-source language and toolkit for bigraphs—to generate (and export) a transition system for analysis with model checking tools e.g. PRISM [16]. The example above is available in BigraphER format in the online model. For reasoning, states are labelled with bigraph patterns [4], predicates that indicate if there is match of a bigraph in that state. Temporal properties are expressed using linear or branching time temporal logics e.g. Computation Tree logic (CTL) [10].

For observability, we want to check 1) if an intention is being progressed, its status should never be pending, 2) if an intention becomes a completed empty program, its related event will eventually succeed, and 3) if an intention becomes blocked, but is not an empty program, its related original event will eventually fail. For progress, we label each step in the execution trace with its true progress estimation and check that there always exists a path where these match. For attention-directing, we check that if the belief atom parked is believed, then eventually the event e_parked will be added to the agent desires.

As an example, consider observability and properties 2) and 3) for the event, namely ⟨e_retrv, identifier1⟩ given in Fig. 3, which we can express in CTL as follows:

\[ \text{A} \left[ \varphi_1 \right] \implies \text{F} \left( \varphi_2 \land \neg \varphi_4 \right) \]  
for property 2)

\[ \text{A} \left[ \varphi_3 \right] \implies \text{F} \left( \varphi_4 \land \neg \varphi_2 \right) \]  
for property 3)

The specification for property 2) checks that along all paths, if an intention is completed (i.e. \( \varphi_1 \) holds), this implies that eventually the original event will succeed (and not fail), i.e. \( \text{F} \left( \varphi_2 \land \neg \varphi_4 \right) \). Similar logic applies when an intention is blocked for property 3). We do not give full details, but for the interested reader, the state formulae can be represented by the following bigraph patterns:

\[ \varphi_1 \equiv \text{Intent}_e.(\text{Identifier}.\text{Identifier1} \mid \text{Nil} \mid \text{id}) \]

\[ \varphi_2 \equiv \text{Event}_e.(\text{Identifier}.\text{Identifier1} \mid \text{Success} \mid \text{id}), \]

\[ \varphi_3 \equiv \text{Intent}_e.(\text{Identifier}.\text{Identifier1} \mid \text{ReduceF} \mid \text{id}) \]

\[ \varphi_4 \equiv \text{Event}_e.(\text{Identifier}.\text{Identifier1} \mid \text{Failure} \mid \text{id}), \]

where Identifier1 is the identifier of the event \( \langle e\_retrv, \text{identifier1} \rangle \), the symbol id (called a site in bigraphs) stands for the part of model that is abstracted away such as the execution trace, and the subscript e (called a link) maintains the mean-end relation between an original event and the related intention. The transition system for the agent in Fig. 3 has 44 states and 44 transitions[4] and all the above mentioned properties are shown to hold.


[4] There are numerous internal states, that do not appear in the final transition system, but add to build time.
4 Related Work

We are not the first to label intention status \cite{13, 14} (as pending/active), however like the original CAN semantics, these approaches do not make intention success/failure explicit. We believe having this information is much more important to human users than simply knowing an intention is dropped. Intention estimation has also been explored \cite{35, 36}, but in the context of scheduling where they wish to choose the most-completed intention first. The approach is based on completeness measures that use the resource consumption and effects of achieving intentions as an estimate, however this requires domain knowledge on the pre-conditions/post-effects of events that can be hard to obtain. For the attention directing feature, we adopted the motivation library approach from \cite{26}. There are similar approaches such as conditional goals in \cite{23, 24} and automatic events \cite{39} (adopting a goal or an event that is conditionalised by beliefs).

Verifying BDI agents through model checking has been well explored, e.g. \cite{12} verifies the decision making part (modelled in BDI agents) of a hybrid autonomous system, and \cite{11} uses model checking to reason about agent programs written in Agent Programming Languages. Bremner \textit{et al.} verify that BDI-based decision making (e.g. plan selection) in robotic systems adhere to ethical rules (e.g. save human) \cite{7}. For transparency, they propose a recorder that produces human readable logs consisting of information such as belief base and the plans executed. Though we agree on the use of logging as a means of transparency, their aim is to provide logs for future forensic analysis whereas ours is to provide run-time logs for effective HAT.

5 Conclusion and Future Work

We presented an executable framework for verifiable BDI agents supporting observability (showing task status) and attention-directing (flagging relevant information), to facilitate effective human-autonomy teaming (HAT). Observability allows comprehending what an agent is doing and how much progress has been made, while attention directing ensures communication of critical information to human teammates.

Observability is implemented by enabling the agent to show the status of intentions (e.g. pending or active), and to provide a quantitative estimation of progress at each step. Attention-directing is achieved through a motivation library that allows agents to adopt multiple new events when a particular world state is believed to hold. Using a UAV example, we have demonstrated these features in practice.

Full scale observability and attention directing is beyond this initial framework. Future work includes an in-depth survey to determine what are the most important aspects to be made observable (to humans). This might require domain-dependent observability, for example, sometimes why an agent decides to do something is more important than progress estimation. Our current intention progress is a metric that measures the distance to the end of (a possible) execution trace. However, in practice actions do not take the same length of time and further domain knowledge annotation for agent programs may be required for more realistic progress estimation. Transparency encompasses more than observability and attention directing and determining: what to log, how to log, and how to use these logs, e.g. future forensic analysis or run-time decision making, will be crucial to tackle emerging HAT scenarios. Finally, a long-term goal is analysis of the accuracy of the intention progression predictions in different scenarios.

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


