CIT. Journal of Computing and Information Technology, Vol. 27, No. 1, March 2019, 13–24
doi: 10.20532/cit.2019.1004396

Application of the GORITE BDI Framework to Human-Autonomy Teaming: A Case Study

Salma Noorunnisa[1], Dennis Jarvis[1], Jacqueline Jarvis[1] and Marcus Watson[2]

[1]Central Queensland University, Rockhampton, Australia
[2]The University of Queensland, Brisbane, Australia

Human-Autonomy Teaming (HAT) is of growing interest in the military sector, particularly in its application to war gaming using semi-automated computer generated forces (CGF). In these applications, one or more operators manage multiple semi-autonomous game entities. If effective collaboration (teaming) is to occur between operators and entities, then having effective interaction models is essential if the levels of trust and explanatory capability required for military operations are to be delivered. The Situation Awareness-Based Agent Transparency (SAT) Model has been identified as providing a suitable conceptual framework for such models. However, while the SAT model is informed by the Belief-Desire-Intention (BDI) model of agency, to date there has been no implementation of an interaction model at the level of desires and intentions, *i.e.* goals. In this paper, we propose that GORITE, a novel BDI framework that employs explicit goal representations and a shared data context for goal execution, provides a suitable platform for the development of SAT-enabled agents. The feasibility of this proposition is demonstrated through the development of a simple but representative CGF case study.

ACM CCS (2012) Classification: Computing methodologies → Artificial Intelligence → Distributed Artificial Intelligence → Multi-agent systems

Keywords: human-agent collaboration, BDI, multi-agent systems

1. Introduction

At a 2017 workshop on Human-Autonomy Teaming (HAT), having a shared mental model was identified as being essential if HAT systems are to deliver the levels of trust and explanatory capability required for military operations [1]. Furthermore, the Situation Awareness-Based Agent Transparency (SAT) Model developed by Chen *et al.* [2] was identified as providing a suitable conceptual framework for future HAT research. SAT-enabled visualisation agents have been demonstrated to provide operators with improved situation awareness of evolving mission environments [2], [3] by providing operator support at three levels:

1. What's going on and what is the agent trying to achieve?
2. Why does the agent do it?
3. What should the operator expect to happen?

In addressing these questions, a SAT-enabled agent would draw on its desires and intentions at Level 1 and its beliefs at Level 2. However, while the SAT model is inspired by the Belief-Desire-Intention (BDI) model of agency, the creation of SAT agents grounded in the BDI model of agency has not been pursued by Chen and her colleagues. Rather, the focus of Chen's work has been on the visualisation of information pertaining to the SAT levels and the demonstration through controlled experimentation that operator performance and trust in automation is enhanced through such visualisation. Chen has proposed that the research community should continue with that agenda and we are in agreement. However, there is also a need for a complementary research program with a focus of developing a BDI software framework that explicitly supports human-autonomy teaming through the use of the SAT model. This will require a framework that provides explicit representation of beliefs, desires and intentions in order to enable the agent to reflect on and explain its actions and to enable humans to dynamically modify agent behaviour. This represents a significant departure from traditional BDI agent frameworks and we propose to use GORITE, a novel open-source BDI framework developed by Rönnquist [4] as our starting platform. GORITE itself is a mature, open source[1] and fully functional software framework, as evidenced by the case studies presented in [4]. GORITE differs from traditional BDI frameworks in that goals are explicitly represented. However, it retains a focus on autonomous behaviour, so extensions to the framework are required if a richer teaming model is to be supported.

Our intent is to tackle this research program iteratively, with each iteration involving model extension, framework realisation and application development. This paper represents the second iteration of that process. In the first iteration [5], human-autonomy teaming was restricted to the user initiated inspection and modification of beliefs. In this iteration, teaming is extended to include both the inspection and modification goals and agent initiated collaboration. Furthermore, in the first iteration, a very simple CGF example was used. This iteration is now grounded in vignettes extracted from the CGF scenario employed in [6], [7], which was concerned with a company attack on an enemy platoon.

In the remainder of this paper, we first review the BDI model and identify why, even though the SAT model is informed by the BDI model, the traditional BDI execution model provides a poor starting point for the provision of the functionality required for SAT-enabled agents. In Section 3, an alternative BDI framework, namely GORITE, that better supports the required SAT functionality, is presented. The case study and its implementation using GORITE are presented in Sections 4 and 5. The paper concludes with a discussion and a conclusion.

2. SAT and the BDI Model

The BDI model is concerned with how an agent makes rational decisions about the actions that it performs through the employment of

1. Beliefs about its environment, other agents and itself,
2. Desires that it wishes to satisfy and
3. Intentions to act towards the fulfilment of selected desires.

The model has its origin in Bratman's theory of human practical reasoning [8]. Bratman's ideas were first formalised by Rao and Georgeff [9] who subsequently proposed an abstract architecture in which beliefs, desires and intentions were explicitly represented as global data structures and where agent behaviour is event driven. However, while this conceptualisation faithfully captured Bratman's theory, it did not constitute a practical system for rational reasoning. In order to ensure computational tractability, they proposed the following representational changes [10]:

- Only beliefs about the current state of the world are represented explicitly
- Desires are referred to as goals, which are represented as events. Goals have only a transient representation, acting as triggers for plan invocations.
- Information about the means of achieving certain future world states (desires) are represented procedurally as plans.
- Intentions are represented implicitly by the collection of currently active plans.

Furthermore, while a particular goal may be realisable via multiple plans, an agent must commit itself to (select) a single plan for its realisation. However, if that plan fails, the means for achieving the goal can be reconsidered. An agent can pursue multiple goals concurrently.

These considerations led to the following execution model for BDI agent goal deliberation:

```
repeat
   wait for the next goal event;
   select (on the basis of current
      beliefs) a plan to achieve the
      current goal;
   execute the selected plan;
   update beliefs;
end repeat
```

[1]GORITE is available under an LGPL licence. Contact the second author for further details.

This execution model (which we refer to as the traditional BDI execution model) has provided the conceptual basis for all major research and commercial BDI frameworks, in particular PRS [11], dMARS [12] and JACK [13].

The traditional BDI execution model is well suited to the realisation of situated autonomous behavior – in response to both internal and external goal events, an agent selects and commits itself to a course of action (plan) on the basis of its current world view. If that course of action fails, then the current goal can be reconsidered or pursuit of the goal can be terminated. As a consequence, the BDI model of agency has underpinned many successful agent applications [4] and has been identified as one of the preferred vehicles for the delivery of industry strength, knowledge rich, intelligent agent applications [14]. The BDI model has been extended in JACK Teams [13] to accommodate teams of agents (such as platoons and manufacturing cells) as distinct entities with their own beliefs, desires and intentions and in CoJACK [13] to provide agents with an explicit cognitive architecture to ground agent reasoning. However, these extensions retain the essence of the traditional BDI execution model, namely that the goals are not represented as explicit, persistent entities, but rather as transitory events.

At a particular point in time, a BDI agent may be pursuing multiple intentions (plans) and it will have a current set of beliefs. If the agent is a member of a human-autonomy team, a human team member should be able to

1. pause some (or all of) the agent's current intentions (plans) prior to inspection or modification of intentions and/or beliefs and to
2. resume paused intentions on completion of inspection or modification.

Additionally, the agent should be able to pause some (or all) of its own intentions if it determines that assistance from a human team member is required. These considerations give rise to the following set of functional requirements for a SAT-enabled BDI agent.

Requirements R0, R1 and R5 have been designated as foundational (SAT Level 0), as they underpin the SAT requirements (R2-R4). Additional functionality such as goal replay and goal re-execution with modified context may be beneficial in some circumstances and particularly at SAT Level 3. However, as our immediate focus is SAT Levels 1 and 2, such functionality is deemed to be out of scope.

Table 1. Mapping of requirements to SAT levels.

	Description	**SAT Level**
R0	Initiate goal execution (user)	0
R1	Pause and resume a particular goal execution (user initiated).	0
R2	Inspect current beliefs relevant to a particular goal execution and if appropriate, make modifications.	1
R3	Inspect historical beliefs associated with a particular goal execution	2
R4	Inspect the goals that an agent has committed to pursue and, if necessary, add new goals, delete existing goals or modify the execution order.	1
R5	Pause goal execution (agent initiated).	0

While the extensions to the traditional BDI execution model embodied in JACK Teams provides support for teams of agents and for team goals, team goal achievement remains a primarily autonomous activity, but with the flexibility of being able to dynamically change team membership. If the human-autonomy teaming involves only delegation, then the teaming model provided by JACK Teams will suffice. However, a more comprehensive teaming model is required if the functionality detailed in Table 1 is to be supported. The provision of such a model is problematic for frameworks that employ the traditional BDI execution model for the following reasons:

- Interruption of plans is not supported.
- Goal representation is implicit and transient, with goals modelled as events that are not persisted. Consequently, goals are not inspectable.
- Depending on how beliefs are stored, they may be inspectable. However, no distinction is made in the traditional BDI execution model between individual agent be-

liefs, shared agent beliefs and beliefs that are shared by agents that are collaborating on a particular goal execution.

Consequently, if SAT functionality for BDI agents is required, it may be better to employ a framework such as GORITE [4], in which intentions are explicitly represented.

3. GORITE

GORITE is a Java BDI framework that provides class level support for the development of agent applications that involve teams of BDI agents. Note that, unlike the traditional BDI frameworks mentioned earlier (PRS, dMARS and JACK), a separate plan language is not required. Instead, agent and team behaviour is specified in the form of goal-based process models. These models are similar in concept to the functional flow diagrams employed in the functional analysis activity of systems engineering [15]. However, in a process model, behaviour is decomposed using goal/sub-goal objects rather than functions. Furthermore, a more extensive set of control nodes (including choice, sequence, loop and parallel nodes) is employed. These nodes, as well as the process model itself, are also represented as goal objects. All goal objects in a process model are instantiations of the GORITE framework's `Goal` class and its sub-classes. If required, goal-specific behaviour can be specified by overriding the `Goal.execute()` method – default behaviours are provided for all GORITE control goal classes.

While a process model is associated with a particular agent or team, in the latter case, the binding of sub-goals to team members is not hardwired into the process model. Rather, a separate construct called a task team maintains a mapping between team members (which may in turn be teams) and goals through the concept of a role, which is defined as a set of goals. A task team can be associated with multiple process models and its structure can be changed dynamically. A default allocation strategy for task team formation is provided by GORITE, but this strategy can be overridden to provide team formation strategies of arbitrary complexity.

Process models are executable, and as such, an alternative execution model to that employed by traditional BDI frameworks (where an agent initiates plan execution in response to the arrival of goal events) is required. Rather than explicitly managing its own behaviour, a GORITE agent delegates that responsibility to an executor object, which then initiates goal execution on behalf of the agent. This execution involves the traversal of the process model and the invocation of the `execute()` m methods of each of the component goals. During this traversal, the executor makes available to the participants in the execution (*i.e.* the task team members) a shared data context, thus providing for a clear separation between an agent's individual beliefs and those that it shares with other agents involved in the goal execution. In the GORITE execution model, BDI execution semantics is preserved, with the agent still able to choose between courses of action to achieve a goal or to reconsider how a goal might be achieved.

Process model execution can be initiated synchronously from the application's main thread via the `performGoal()` method provided by the `Performer` class. However, GORITE also supports an alternative asynchronous execution framework. Central to this framework is the ToDo Group, which is a list of the intentions (goals) that an agent or team is currently pursuing or intending to pursue. Asynchronous execution is time sliced, and during a time slice, only one intention is progressed – that is, the intention that is at the top of the list. However, prior to the executor progressing this intention, meta-level reasoning can be invoked to determine which intention is to be progressed in the next time slice, thereby enabling scheduling strategies such as round-robin to be implemented.

Actions on the environment (either virtual or physical) are performed by the leaf nodes of the (dynamically) expanded process model. In the implementations discussed in [4], the environment is virtual and actions are modelled as goals whose execute() methods invoke time delays. However, case studies are presented in [16], [17] where GORITE agents interact with a physical manufacturing system. In this regard, GORITE provides two specialized `Goal` classes – namely the `Action` class and the `RemoteGoal` class to better support physical execution.

The `execute()` method of an action goal, rather than having a data context as its single argument, takes two arguments – a set of input values and a set of output values. Furthermore, execution can block until all input values are set. On the other hand, a remote goal is wrapper for a remote execution, which may or may not involve a GORITE execution on the remote process. However, the data context for the local execution is shared with the remote execution.

The key (and novel) features of GORITE that enable the collaboration requirements R0-R5 identified in the previous section to be realized are the concepts of

1. the ToDo group
2. the perceptor and
3. the data context

With respect to ToDo groups, note that in traditional BDI frameworks, an agent (or agent team) selects a plan to achieve its next goal from a set of plans that are applicable to the goal in question. This determination is made on the basis of the agent's current beliefs, but these beliefs do not include any explicit representation of current intentions. In GORITE, the agent's current intentions are accessible via the agent's ToDo group to inform such reasoning. ToDo groups can also be used to model reactive behaviour, including user interaction. In this respect, GORITE provides a `Perceptor` class that can be used by a performer to add goals to its ToDo group when particular events occur. In [4], perceptors were used to model incoming manufacturing orders and requests for sensor team reformation. However, they also provide a convenient mechanism to support user-initiated goal execution.

The data context provides, inter alia, associative access to a table of named, multi-valued data elements. As noted above, a data context is made available by a GORITE executor object to all participants of a goal execution as the executor object traverses a process model. Consequently, a primary use of data context elements is to update the values of objects involved in the goal execution. Note that, because the elements are multi-valued, all changes to an element can be recorded. There are, also no restrictions on the data type of a data context entry. Consequently, the history of actual goals executed (and by which team/agent) during the execution of a process model could be modelled as a named object in the data context. This evolving history could then be inspected by a user and provide a starting point for understanding why particular behaviours were observed.

In terms of the SAT requirements of Table 1, the mapping of those requirements to GORITE constructs is as shown in Table 2.

Table 2. Mapping of requirements to GORITE concepts.

	Description	GORITE Concept
R0	Initiate goal execution (user).	Perceptor
R1	Pause and resume goal execution (user).	ToDo group
R2	Inspect and/or modify current beliefs.	Data context
R3	Inspect historical beliefs.	Data context
R4	Inspect and/or modify goal execution.	ToDo group
R5	Pause goal execution (agent).	ToDo group

Note that while the concepts presented in Table 2 will directly support the corresponding concepts, additional (new) framework concepts are required in order to facilitate the effective provision of teaming functionality. These new concepts are discussed in Section 4.

4. The Case Study

The case study is based on the war-gaming scenario presented in [6], [7], which is concerned with the deliberate attack by a mounted infantry company on an enemy formation. In this scenario, a company agent produces a plan and courses of action to carry out the four phases for an attack, namely the preparatory, assault, exploitation and reorganisation phases. The company contains multiple platoons. Each platoon is comprised of three sections, each of which has nine soldiers and an armored personnel carrier (APC). To prosecute the attack, the company agent identifies a fire support platoon and two assaulting platoons. It then plans the routes, form-up positions and coordination parameters for the attack. Once the mission has started, the company agent monitors the location and status of the platoons involved and, if necessary, changes the plan if circumstances dictate. In the planning and execution of the attack, the agent applies standard military doctrine and makes appropriate use of terrain. In the absence of the agent, an operator (known as a puckster) would need to perform the role of the company agent under the direction of the military personnel playing the game. In this regard, note that the company (and enemy) actions are played out and visualized in a simulated environment that employs an accurate and realistic terrain model, as shown in Figure 1.

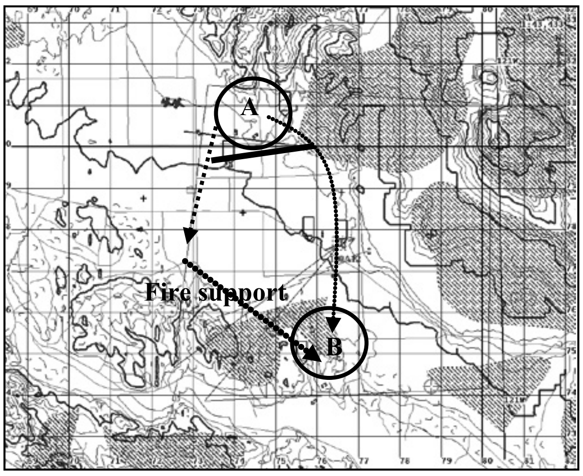


Figure 1. The scenario for demonstrating the agent capabilities. The objective is for a mounted infantry company located at point A to attack an enemy formation occupying a position in the vicinity of point B [6].

Table 3. Mapping of vignettes to requirements.

	Description	Requirements
V1	Path traversal by a single platoon	R0-R3
V2	Path traversal by multiple platoons	R0-R3
V3	Detection by a platoon of enemy movement	R0, R4, R5

In this paper, only vignettes from the scenario which are specifically concerned with the SAT requirements R0-R5 are considered. These vignettes are summarized in Table 3. Furthermore, these vignettes are considered independently of the broader scenario and of the simulated environment, as the intent of this project is to demonstrate the feasibility of using GORITE to construct SAT-enabled BDI agents. Experimental studies using realistic simulated environments (as in [2], [3]) will need to be performed, but that is out of scope for this project.

In V1, a single platoon is traversing a specified set of waypoints. The operator intervenes and directs the platoon to follow a new path. In V2, three platoons are traversing different paths concurrently. The operator intervenes and directs one of the platoons to follow a new path. In V3, the platoon in V1 detects enemy movement and seeks advice from the user. The operator directs the platoon to execute a waiting goal and upon completion of this goal, the traversal goal is resumed.

5. Implementation

The starting point for our discussion of the GORITE implementation of the case study is the achievement of R0, namely user initiated goal execution. In this regard, all three vignettes involve the execution of one or more traverse path goals. The Java code for the traverse path goal is presented in Algorithm 1.

Observe that the path traversal goal specifies both the activities required to achieve the goal and the coordination requirements for those activities. Both facets are specified explicitly and uniformly using GORITE goal class instances (*e.g.* `Goal`, `SequenceGoal`, `LoopGoal` in the method above). In this regard, note that traverse path is a sequence goal that contains two goals – a process percept goal and a visit waypoints goal. The second of these goals is a loop goal, whose body consists of three goals that are performed in sequence.

The resulting traverse goal instance can then be executed on behalf of the goal owner by a separate executor object. As noted in Section 3, this object will traverse the goal instance graph and at each node (which is an object of type `Goal`) invoke the node's `execute()` method.

Algorithm 1. Path traversal method.

```
Goal traversePath() {
    return new SequenceGoal(TRAVERSE_PATH, new Goal[]{
        new Goal("process percept") {
            public Goal.States execute(Data d) {
                System.err.println("Execution started");
                // Set PATH in the data context
                Path p = (Path) d.getValue(PERCEPT);
                d.setValue(PATH, p);
                // Initialise the execution object for this goal
                String ename = (String) d.getValue(EXECUTION);
                Execution e = etable.get(ename);
                e.state = State.RUNNING;
                //Record that goal execution has started
                record(Request.START, e);
                return Goal.States.PASSED;
            }
        },
        new LoopGoal("visit waypoints", new Goal[] {
            //Move to next waypoint
            traverseSegment(),
            //Is this the final destination?
            trackProgress(),
            //Has a pause been requested?
            checkpoint(),
        )
    });
}
```

This method has a single parameter of type `Data`, which is the data context. Values can be retrieved from the data context for use/update via `Data.getValue()` and new values can be added via `Data.setValue()`.

The behaviour for the traverse segment goal is simpler – its `execute()` method blocks for a specified period of time, Algorithm 2.

Traverse goal execution can be initiated by adding a traverse goal instance to the company agent's ToDo group. In the case study, this is achieved via the `Company.start()` method, Algorithm 3.

The percept object contains the waypoints that the platoon is to visit. This object is added to the data context `(d)` for the goal execution by the `perceive()` method of the `Perceptor`

Algorithm 2. Segment traversal method.

```
Goal traverseSegment() {
    return new Goal(TRAVERSE_SEGMENT) {
        public Goal.States execute(Data d)
            //Extract delay from data context
            int n = (int) d.getValue(DURATION);
            System.err.println("Waiting for " + n + " time units");
            //Wait for n msecs
            if (TimeTrigger.isPending(d, "deadline", n * 1000)) {
                return Goal.States.BLOCKED;
            }
            System.err.println("Waiting finished");
            return Goal.States.PASSED;
        }
    };
}
```

Algorithm 3. Initiation of goal traversal.

```
public void start(String ename,String gname,Object percept,Data d){
    //Create an execution object for this goal and add it to the
    //execution table
    Execution e = new Execution(ename);
    e.request = Request.START;
    etable.put(ename, e);
    //Record that goal execution has been requested
    record(Request.START,e)
    d.setValue(EXECUTION, ename);
    //Initiate goal execution
    Perceptor perceptor = perceptors.get(gname);
    perceptor.perceive(percept, d);
}
```

class and is given the default name of `PERCEPT`. Note that multiple goals can be added to the ToDo group and that these goals can be executed either sequentially, or through the use of meta-goals, concurrently. For a more complete description of the GORITE execution model, the reader is referred to [4].

Prior to the initiation of goal execution, an object of type `Execution (e)` is created and stored in an associative lookup table `(etable)`. The role of an execution object is to manage the teaming aspects of the goal execution – the two key aspects being the last request (`START`, `PAUSE`, `CONTINUE`, ...) and the current state of the goal execution (`RUNNING`, `PAUSED`, `IDLE`). Note that the execution state is separate from the `Goal.States` value returned to the executor object by a goal's `execute()` method.

The `start()` method is invoked by a method chain originating in the action listener for the Start button in the application GUI, which is illustrated in Figure 2.

Note that as required for V2, multiple goal executions can be initiated. Also for reasons of convenience, when initiating a goal, only the number of waypoints is specified. The actual waypoint values are generated automatically.

Having initiated goal execution, all three vignettes require the ability to suspend and resume goal execution (R1 and R5). Furthermore, both humans (V1) and agents (V3) need to be able to initiate goal suspension. In this regard, when a goal execution is initiated via the Start button in Figure 2, a GUI for the management of that particular goal execution is created, as in Figure 3. Each goal execution has a separate GUI that is bound to its execution object.

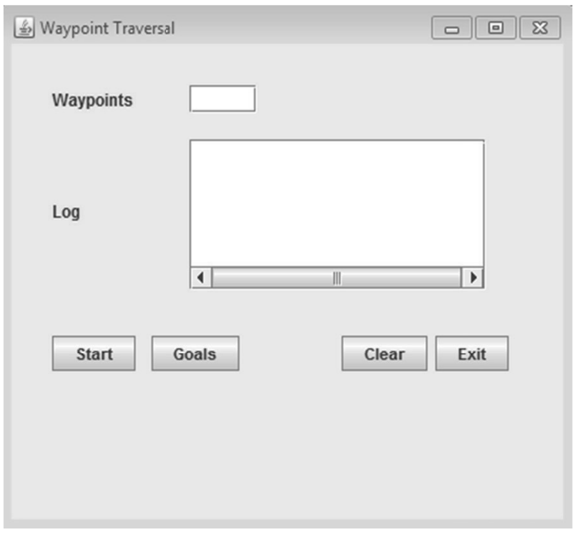
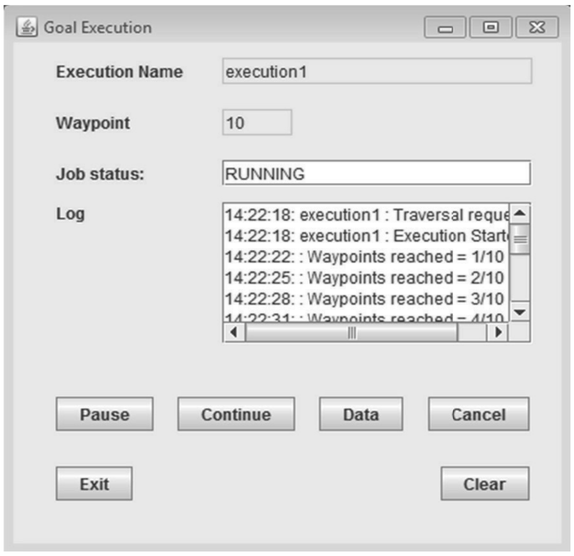


Figure 2. The GUI for the waypoint traversal application.


Goal Execution

Execution Name: execution1

Waypoint: 10

Job status: RUNNING

Log

```
14:22:18: execution1 : Traversal reque
14:22:18: execution1 : Execution Start
14:22:22: : Waypoints reached = 1/10
14:22:25: : Waypoints reached = 2/10
14:22:28: : Waypoints reached = 3/10
14:22:31: : Waypoints reached = 4/10
```

Figure 3. An execution GUI.

For user initiated goal suspension, the preference is for goal execution to be interrupted at well-defined points which we refer to as checkpoints, Algorithm 4. This is the approach that has been employed in the traverse goal definition in the previous section – a checkpoint goal is performed whenever a waypoint is reached. This goal passes if there are no outstanding user requests. If there is a suspension request, then the goal execution is blocked until a resumption request is issued by the user.

While the goal execution is blocked, the user is able to inspect and modify the data context for the goal execution via the Data button of Figure 3. New goal executions can also be added and the existing (blocked) goal execution removed via the goal manipulation GUI (illustrated in Figure 4) that is displayed when the Goals button in Figure 2 is clicked. As noted in the previous section, if more flexibility is required in terms of the application of checkpoint reasoning, a checkpoint goal can be attached to the ToDo as a meta-goal. GORITE's meta-level reasoning infrastructure is discussed in [4].

Agent initiated suspension will arise when the agent encounters a situation that it is unable to deal with. At this point, the behaviour is similar to that for user initiated suspension – the agent

1. Sets the request field of its execution object to `PAUSE`
2. Updates the goal execution GUI
3. Adds a checkpoint goal to the top of its ToDo group and
4. Organises for `BLOCKED` to be returned by the goal's execute() method until its execution is resumed or the goal is removed.

In V3, this behaviour is initiated in response to the observation of enemy troops nearby. In GORITE, there are various ways in which observation can be modelled. For the sake of simplicity, we have employed the failure handling approach described in [4] for the sensor

Algorithm 4. Goal execution interruption at checkpoints.

```
Goal checkpoint() {
    return new Goal( "checkpoint" ) {
        public Goal.States execute(Data d) {
            //Extract execution object from the execution table
            String ename = (String) d.getValue(EXECUTION);
            Execution e = etable.get(ename);
            //Handle the request
            if (e.request == PAUSE) {
                if (e.state == State.RUNNING) {
                    e.state = State.PAUSED;
                    //Record state change
                    record(PAUSE,e);
                }
                //Pause goal execution
                return Goal.States.BLOCKED;
            }
            if (e.request == CONTINUE) {
                if (e.state == State.PAUSED) {
                    e.state = State.RUNNING;
                    //Record state change
                    record(CONTINUE,e);
                }
                //Resume goal execution
                return Goal.States.PASSED;
            }
            //Should not happen
            return Goal.States.FAILED;
        }
    });
}
```

network application. In this approach, the traverse segment goal that appears in the traverse path goal code fragment presented earlier is modelled as a loop goal whose body consists of a GORITE fail goal and a fail handler goal. The fail handler goal implements the four steps outlined above. The fail goal has two sequential sub-goals – an observe goal and a move goal. When an observe goal selects an observation of interest, it fails, the fail goal succeeds and the fail handler goal handles the observation. Other modelling approaches are possible, including the use of GORITE's parallel goal construct.

In addition to the implementation of the three foundational requirements described above, realisation of the three vignettes also required implementation of the following functionality:

1. Inspect and modify an existing data context (V1, V2)
2. Inspect and modify the goals in the ToDo group (V3)

Note that for V3, as a new waiting goal needed to be added by the user, a new data context for that goal had to be dynamically created. Rather than using Java reflection to generically construct a GUI for the goal, the simpler expedient of manually constructing a goal-specific GUI was employed. In addition to entering the data context, the GUI can also be used for the inspection and modification of an existing data context. A new framework class called `GoalInfo` was created to maintain this (and other) information for each agent goal that can be involved in teaming. This information, together with the goals in the ToDo group, was used to populate the goal manipulation GUI, as illustrated in Figure 4. This GUI was created in response to the clicking of the Goals button in Figure 2.

When a goal is selected from the list of available goals in Figure 4, clicking the Create button will result in a data context entry GUI being displayed for that goal. When data entry has been completed, an instance of the selected goal (together with the newly generated data context) will be added to the top of the ToDo group. The goal manipulation GUI also allows for a selected goal in the ToDo group to be either removed or promoted to the top of the list. Code for the complete application is available from the authors.

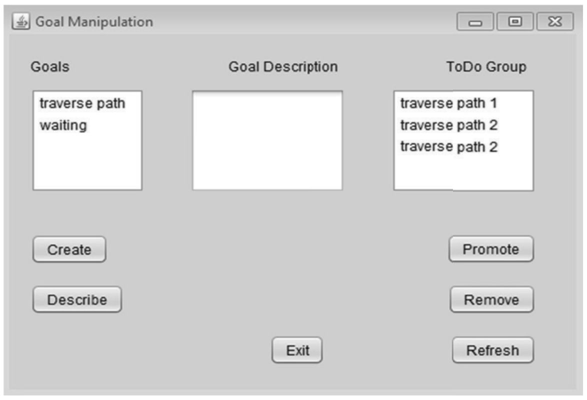


Figure 4. The goal manipulation GUI.

6. Discussion

The motivation of this work has been to demonstrate that the GORITE BDI framework, through its explicit goal representation and corresponding execution model, can be used as a platform to develop SAT-enabled BDI agents. Through the use of a simple but representative case study, mechanisms have been demonstrated whereby the foundational requirements human-autonomy teaming – namely the ability for humans to initiate, suspend and resume GORITE agent activity and for GORITE agents to suspend agent activity – have been demonstrated.

The ability to provide this functionality in a generic manner is dependent on the underlying execution model and the representations employed for intentions. In the case of GORITE, goal execution is achieved through an executor object orchestrating the execution of explicitly represented team goals contained in a ToDo group. This enables execution to accommodate both controlled and uncontrolled suspension. With controlled suspension, a process model is augmented with generic checkpoint goals, so that suspension occurs at well defined points within in the process model execution. The operator can then inspect and manipulate both the intentions contained in the ToDo group and the elements in the data context. Uncontrolled suspension refers to the association of meta-goals with a ToDo group. A meta-goal is executed at every time step and whenever intentions are added to or removed from a ToDo group. If the meta-goal is a checkpoint goal, then suspension of the process model execution can occur. Uncontrolled suspension was not employed in this case study. With traditional BDI frameworks, checks for operator-initiated suspension could be incorporated into individual agent plans. However, with no explicit representation of intention, inspection and modification of intentions is problematic.

From a modelling perspective, this work has identified concepts that would benefit from framework support, in particular, the notion of an execution object. An execution object manages the collaboration state (`RUNNING`, `PAUSED`, `IDLE`) of its associated goal and, as such, operates at a higher level than GORITE's execution state (`Goal.States`), which controls executor object behaviour. The `Execution` class, together with the `CheckpointGoal` and `GoalInfo` classes, provides a starting point for a generic human-autonomy teaming capability grounded in the SAT model. Note also that, as indicated in Section 4, there are numerous ways in which agent behaviour for the case study can be modelled in GORITE. The purpose of this paper was to demonstrate the feasibility of using GORITE to provide effective human-autonomy teaming and, as such, not all behaviour modelling options for the case study were explored. This exploration will be ongoing as we apply GORITE to new HAT domains.

7. Conclusion

Using GORITE as a platform to achieve effective human-autonomy teaming through the deployment of SAT-enabled BDI agents will be an ongoing activity. Chen *et al*. have demonstrated that the transparency provided by SAT-enabled agents is beneficial in terms of human operator effectiveness. The tasks involved in those studies were relatively straightforward; a key challenge, we believe, will be in the scaling up of human-autonomy teaming to address more complex problems. In particular, while GORITE may provide a basic set of building blocks for creating SAT-enabled agents, what is not clear is how these agents should be constructed and what additional support should be provided at the framework level. The goal collaboration concept has proven useful both in this work and in other related applications, in particular manufacturing, which suggests that such an abstraction is generally useful and should be supported at the framework level. Visualisation is another example where generic support could be provided – for instance, using interactive Gantt charts as a vehicle for goal management rather than the conventional GUI-based approach employed in this project could be beneficial in terms of the user experience. Also, we would see integration with simulation as a key element in the delivery of functionality at SAT Level 3.

References

[1] S. G. Hill and M. F. Ling, *Human-Robot Bidirectional Communication and Human-Autonomy Teaming Workshop Summary, DST-Group-GD-0965*, Defence Science and Technology Group, Fishermans Bend Australia, 2017.

[2] J. Y. C. Chen *et al.*, *Situation Awareness-Based Agent Transparency*, ARL-TR-6905 Army Research Laboratory, Aberdeen Proving Ground, Maryland, April 2014. Available from https://www.arl.army.mil/arlreports/

[3] J. Y. C. Chen *et al.*, "Situation Awareness-Based Agent Transparency and Human-Autonomy Teaming Effectiveness", *Theoretical Issues in Ergonomics Science*, vol. 19, pp. 259–282, 2018. http://dx.doi.org/10.1080/1463922X.2017.1315750

[4] D. Jarvis *et al.*, *Multiagent Systems and Applications. Volume 2: Development Using the GORITE BDI Framework*, Springer, 2012. http://dx.doi.org/10.1007/978-3-642-33320-0

[5] S. Noorunnisa *et al.*, *Human-Agent Collaboration: A Goal-Based BDI Approach*, in *Proc. of Agent & Multi-Agent Systems: Technologies & Applications (KES-AMSTA-18)*, Gold Coast, 2018. http://dx.doi.org/10.1007/978-3-319-92031-3_1

[6] F. Lui *et al.*, "An Architecture to Support Autonomous Command Agents in OneSAF Testbed Simulations", in *Proc. of SimTectT 2002*, Melbourne, 2002.

[7] R. Connell *et al.*, "The Mapping of Courses of Action Derived from Cognitive Work Analysis to Agent Behaviours", in *Proc. of Agents at Work: Deployed Applications of Autonomous Agents and Multi-agent Systems Workshop, 2nd. Int. Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2003)*, Melbourne, 2003.

[8] M. E. Bratman, *Intention, Plans, and Practical Reason*, Harvard University Press, 1987.

[9] A. S. Rao and M. P. Georgeff, "Modeling Rational Agents within a BDI Architecture", in *Proc. of the Second International Conference on Prin-*

ciples of Knowledge Representation and Reasoning, Morgan Kaufman, 1991.

[10] A. S. Rao and M. P. Georgeff, "BDI Agents: from Theory to Practice", in *Proc. of the 1st International Conference on Multi-Agent Systems (ICMAS 1995)*, San Francisco, 1995.

[11] M. P. Georgeff and A. I. Lansky, "Procedural Knowledge", *Proceedings of the IEEE*, vol. 74, pp. 1383–1398, 1986. http://dx.doi.org/10.1109/PROC.1986.13639

[12] M. d'Inverno *et al.*, "A Formal Specification of dMARS", Lecture Notes in Artificial Intelligence vol. 1365, pp. 155–176, 1998. http://dx.doi.org/10.1007/BFb0026757

[13] AOS Group, "AOS Group – Products", 2018. http://www.agent-software.com

[14] R. Jones and R. Wray, "Comparative Analysis of Frameworks for Knowledge-Intensive Intelligent Agents", *AI Magazine*, vol. 27, pp. 57–70, 2006. http://dx.doi.org/10.1609/aimag.v27i2.1880

[15] S. Blanchard and W. Fabrycky, *Systems Engineering and Analysis: Fifth Edition*. Pearson Education Inc., 2011.

[16] D. Jarvis *et al.*, "PROSA/G: An Architecture for Agent-Based Manufacturing Execution", in *Proc. of the IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA 2018)*, Torino, 2018. http://dx.doi.org/10.1109/ETFA.2018.8502598

[17] A. Kalachev *et al.*, "Intelligent Mechatronic System with Decentralised Control and Multi-Agent Planning", in *Proc. of the IEEE 44th Annual Conference of the IEEE Industrial Electronics Society (IECON 2018)*, Washington DC, 2018. http://dx.doi.org/10.1109/IECON.2018.8591390

Received: September 2018
Revised: February 2019
Accepted: February 2019

Contact addresses:
Salma Noorunnisa
Central Queensland University
Rockhampton
Australia
e-mail: s.noorunnisa@cqu.edu.au

Dennis Jarvis
Central Queensland University
Rockhampton
Australia
e-mail: d.jarvis@cqu.edu.au

Jacqueline Jarvis
Central Queensland University
Rockhampton
Australia
e-mail: j.jarvis@cqu.edu.au

Marcus Watson
The University of Queensland
Brisbane
Australia
e-mail: m.watson2@uq.edu.au

Salma Noorunnisa received a Bachelor in Software Engineering degree from the University of Canberra, Australia in 2005. After working in the IT Industry, in 2015 she took a Master of Informatics degree as a part-time student in the School of Engineering and Technology at Central Queensland University. In 2018, she transferred to the PhD program. Her research is concerned with the development of both conceptual and framework support to enable BDI agents to collaborate effectively with humans.

Dennis Jarvis has a BSc (Hons) degree from Flinders University and a PhD degree from the University of Queensland. Since gaining his PhD, he has worked as a computer scientist in academia, government research organisations and in private industry. Dr. Jarvis is currently an Associate Professor in the School of Engineering and Technology at Central Queensland University. Prior to joining CQU in 2007, he was a software architect at Agent Oriented Software for six years and before that, a principal research scientist for CSIRO. His primary research interest is the BDI model of agency and how BDI software frameworks can be extended to address challenging problems that are of practical importance.

Jacqueline Jarvis has a BSc (Hons) degree from Flinders University, an MSc in Software Development and Analysis from Heriot-Watt University and a PhD degree from the University of South Australia. Since gaining her MSc degree, she has worked as a computer scientist in academia, government research organisations and in private industry. Dr. Jarvis is currently a senior lecturer in the School of Engineering and Technology at Central Queensland University. Prior to joining CQU in 2007, she was a software engineer at Agent Oriented Software for six years and before that, a senior research scientist for CSIRO. Her primary research interest is the BDI model of agency and how BDI software frameworks can be extended to address challenging problems that are of practical importance.

Marcus Watson has a BSc (Hons) degree from Latrobe University and PhD in Human Factors from Swinburne University of Technology. From 2006-2016, he was the Executive Director of Queensland Health's Clinical Skills Development Service. At CSDS, in addition to his management activities, he conducted research on the use of simulations for the design and evaluation of technology to support clinical decision-making and the development of an evidence-based training and assessment program. Dr. Watson is currently an Associate Professor in the School of Psychology at the University of Queensland, where he conducts research in the general area of human factors.