Singh, Dhirendra and Padgham, Lin and Logan, Brian (2016) Integrating BDI agents with Agent-based simulation platforms. Autonomous Agents and Multi-Agent Systems . pp. 1-22. ISSN 1387-2532

Autonomous Agents and Multi-agent Systems manuscript No.
(will be inserted by the editor)

Integrating BDI agents with Agent-Based Simulation Platforms

Dhirendra Singh · Lin Padgham · Brian Logan

Received: date / Accepted: date

Abstract Agent-Based Models (ABMs) is increasingly being used for exploring and supporting decision making about social science scenarios involving modelling of human agents. However existing agent-based simulation platforms (e.g., SWARM, Repast) provide limited support for the simulation of more complex cognitive agents required by such scenarios. We present a framework that allows Belief-Desire-Intention (BDI) cognitive agents to be embedded in an ABM system. Architecturally, this means that the "brains" of an agent can be modelled in the BDI system in the usual way, while the "body" exists in the ABM system. The architecture is flexible in that the ABM can still have non-BDI agents in the simulation, and the BDI-side can have agents that do not have a physical counterpart (such as an organisation). The framework addresses a key integration challenge of coupling event-based BDI systems, with time-stepped ABM systems. Our framework is modular and supports integration of off-the-shelf BDI systems with off-the-shelf ABM systems. The framework is Open Source, and all integrations and applications are available for use by the modelling community.

Keywords BDI · Agent-Based Modelling · Simulation · Integration

Supported by ARC Discovery DP1093290, ARC Linkage LP130100008, and Telematics Trust grants

D. Singh
School of Computer Science and Information Technology, RMIT University, Australia
E-mail: dhirendra.singh@rmit.edu.au

L. Padgham
School of Computer Science and Information Technology, RMIT University, Australia
E-mail: lin.padgham@rmit.edu.au

B. Logan
School of Computer Science, University of Nottingham, UK
E-mail: bsl@cs.nott.ac.uk

1 Introduction

Agent-Based Models (ABMs) are increasingly being used for exploring and supporting decision making about social science scenarios involving modelling of human agents (e.g. [22,16,1]). This paper presents work to enable developers to easily combine cognitive agent modelling platforms with more traditional ABM systems which typically use simpler agents.

Agent-based simulations are often built using toolkits such as Repast [34] or NetLogo [56], which provide a graphical development interface and a suite of tools to assist in the analysis of simulation results. In such toolkits, the agents are relatively simple entities that respond reactively to their environment. This approach has been very successful in the ecological domain. However, for social science simulations involving humans, it can be challenging to represent human-like behaviour using simple reactive agents. For example, our work with social science researchers and with the emergency services in modelling residents in emergency situations [49,36,46], has highlighted the need for a rich cognitive model for agents in such simulations. Informal validation by emergency services personnel often involves determining how believable the modelled behaviours of residents are, based on what they typically see during actual bushfires. We have found the Belief, Desire, Intention (BDI) model to be a good candidate for this, as it is able to capture complex behaviours in a compact way using a goal-plan hierarchy, while at the same time being intuitive for non-programmers. A recent paper [2] surveyed a substantial number of approaches to modelling humans for simulation, of which BDI systems were one. While we focus here on BDI systems, our integration framework could equally well be applied to whichever cognitive framework implementation is preferred, so long as that system can meet some basic requirements such as input of environmental percepts, specification of actions within the environment, and an ability to manage failed and durative actions. However our own experience has been only with different BDI platforms, and so we refer to the BDI cognitive platform, rather than a more generic cognitive platform. The incorporation of some form of cognitive agents beyond production rules seems essential if we are to capture human behaviour in an appropriate manner for many social simulations involving humans.

BDI [39] agents are based on a simplified model of human behaviour, which balances pro-active goal seeking behaviour, and reactive responses to changes in the environment. BDI programming languages and platforms, such as JACK [12], Jadex [11] or JASON [7] facilitate the high level specification of complex human behaviour, and have been demonstrated to be very efficient for building complex applications [5]. In addition to the "core" AgentSpeak family of languages exemplified by JACK, Jadex and JASON, a wide variety of extensions have been developed, encompassing aspects such as social norms and organisational structures (e.g. [15]), learning (e.g. [50]), planning (e.g. [45]), teams (e.g. [52]), etc.

Rather than attempt to build yet another simulation platform, we have taken the approach of developing an infrastructure which supports integra-

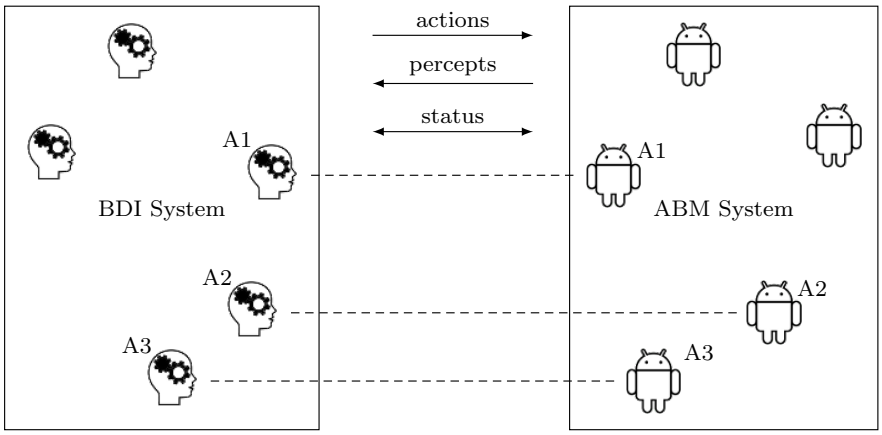

Fig. 1 Conceptual BDI-ABM integration architecture

tion of a wide range of cognitive agent modelling platforms, with a similarly wide range of agent-based simulation platforms or applications.[1] In this paper we describe in some detail this infrastructure and a number of the different applications developed.

Conceptually the infrastructure provides a mechanism whereby some agents in the simulation have a "brain", the decision making component, in the BDI system, while the "body" carries out actions in the ABM system. These operate synchronously as described further in Section 4 passing information about actions, percepts and the status of actions. This is shown in Figure 1. Some agents may not require a BDI "brain" while others may influence the simulation by communication with other agents in the BDI system, but not require embodiment in the ABM.

We first describe briefly the BDI agent paradigm and ABMs in the next section. We note the requirements regarding each of these that are necessary for using our infrastructure. Then, in Section 3, we present the conceptual framework for the BDI-ABM integration, followed in Section 4 by the technical integration we have developed. Section 5 covers some specific applications we have developed. Finally, in Section 6 we briefly describe the key similarities and differences between our approach and related work from the literature, and conclude.

[1] The support infrastructure, along with the code required for a number of specific systems and several example applications, is freely available at `http://tiny.cc/bdi-abm-integration`.

2 Background

2.1 Agent-Based Modelling

Early work in individual-based modelling demonstrated that complex group behaviours such as flocking and following can emerge from the application of simple rules by individuals in a population (e.g., [42]). Agent-based modelling and simulation is an extension of this approach, in which each individual retains information about its current and past states, and its behaviour is controlled by an internal decision process. In agent-based modelling (ABM), the system of interest is conceptualised in terms of agents and their interactions. Agents often correspond to natural kinds, such as people, vehicles, etc. The agents in an ABM are situated in an environment, and are able to sense the state of the environment (e.g., via smell, hearing, vision), and perform actions that change the state of the environment or the perceptible characteristics of the agent (e.g., moving from one location to another, communicating with other agents, etc.). The environment may contain passive objects (e.g., topography) and active objects and processes which change spontaneously during the course of the simulation (e.g., weather) and/or in response to the actions of the agents (e.g., the amount of water available in a reservoir). This focus on interaction through the medium of an environment means that agent-based models are often *spatially explicit*, i.e., the individuals are associated with a location in geometrical space. Such spatially explicit agent systems are sometimes called *situated* in the agent literature, e.g., [17].[2]

The evolution of an agent-based model is determined by the current internal state of each individual agent (which may include an internal world model, or a history of previous events) and the sensory information it receives. At each simulation step, each agent chooses an action based on its state and the percepts it has received at this step. These actions are then executed, resulting in changes to the environment which are perceived by the agents at the next simulation step, which may in turn change their state and influence the decisions they make, and so on. This reliance on individual choice to drive the simulation means that ABM is most suited to *bottom up modelling*—when individual entity behaviour is known (or assumed) and emergent properties are of interest. It can also be used when the entities comprising the system are heterogeneous, or when the behaviour of entities is discrete or stochastic in a way that can't easily be aggregated. It is especially useful when the choice of action to be performed is based on the local environment and/or individual variation across members of a population, for example where the choice of action is strategic, since it is likely that the optimal strategy for an individual depends on the strategies adopted by others in the group.

Agent-based models can be used to explore the effects of model parameters: factors such as income distribution, social network, climate, etc., can all be

[2] In some ABMs, the environment consists solely of other agents and the percepts and actions available to the agents are limited to the exchange of messages. However in this paper, we focus on spatially explicit ABMs.

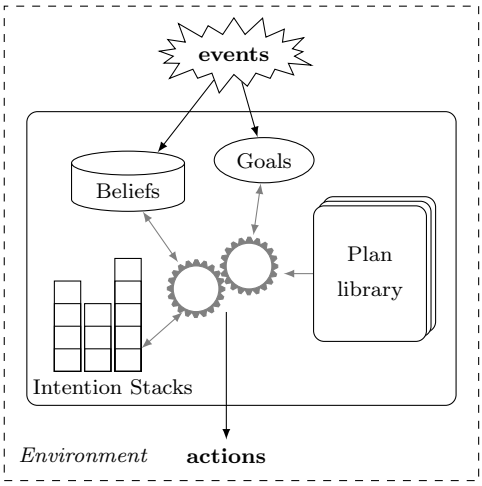

Fig. 2 The BDI architecture

altered and the effects on the agents' behaviour can be observed. If we are confident that the decision procedure is robust, then we can use the behaviour of the agents to predict the emergent behaviour of real populations. As a result, agent-based modelling and simulation is increasingly being used for exploring and supporting decision making about social science scenarios involving modelling of human agents. However, while existing agent-based simulation platforms (e.g., SWARM, Repast) allow the development of relatively simple reactive agents in which the choice of action is based on a few simple rules, they provide only limited support for simulating the more complex, cognitive agents required by such scenarios. One way of modelling such cognitive agents is by drawing on the tools developed in the cognitive agents community, and in particular the Belief-Desire-Intention paradigm, and we outline this work in the next section.

2.2 BDI Agents

The Belief Desire Intention paradigm is a cognitive framework with its roots in philosophy [10,14], which has been extensively used in describing and developing "intelligent agents". While any cognitive agent framework based on beliefs, goals and intentions is part of the broader BDI paradigm, here we focus on systems and languages in the AgentSpeak family, as it comprises a large group of languages and implemented development environments, and these are the languages with which we are most familiar.

BDI agent programs are essentially a set of plan rules of the form $G : \psi \leftarrow P$, meaning that plan P is a reasonable plan for achieving the goal (or responding to the percept) G when (context) condition ψ is believed true. The body of P is made up of sub-goals, which have associated plans, and actions.

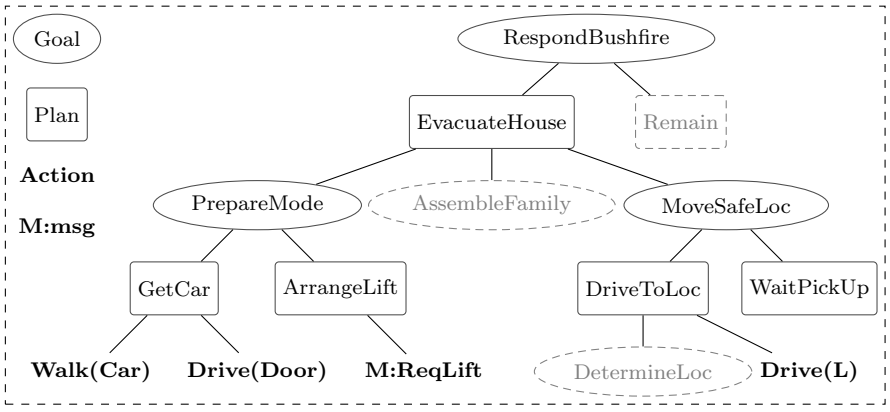

Fig. 3 Example BDI goal-plan hierarchy for a resident agent in the bushfire situation

A plan rule can be chosen for instantiation, and its plan body executed, if the context condition ψ is True according to the agent's beliefs. The plan trigger G may be an internally generated goal such as a sub-goal that is part of the programmed behaviour of a plan, an external event from the environment such as a percept, or a message from another agent.

BDI languages in the AgentSpeak [40] family, such as PRS [21], JACK [58] and JASON [9], follow the general architecture shown in Figure 2. In this setup, an agent is situated in an environment that it perceives (via events) and in which it acts (via actions). Here, beliefs are typically represented within some form of database system, and desires are implemented as goals (for various technical reasons). The programmed plan rule templates exist within a *plan library*, from which plan *instances* are created at run time, whenever the programmed context conditions hold. These plan instances, or intentions, are managed internally using FIFO stacks (intention stacks) where active higher level plans progressively end up at the bottom, while the more immediate lower level plans (sub-goals) at the top are executed first. The execution engine (represented with gears) that manages the BDI execution loop–to continuously perceive, reason, and act–follows Rao and Georgeff's abstract interpretor [41].

Given a plan library consisting of a set of plan rules, the corresponding collection of goals and plans can be represented as a goal-plan tree as shown in Figure 3, which is a part of the design of a simple resident agent in a bushfire situation. This is basically an AND/OR tree where goals have links to some number of plans, one of which must be chosen (OR), and plans have links to some number of (sub)goals/actions, all of which must be accomplished (AND) for the goal to succeed.

In the program of Figure 3, the agent has two programmed choices (plans) for responding to a bushfire warning (`RespondBushfire`) event. These are to either `EvacuateHouse`, or `Remain` at home (not detailed for the sake of brevity). If the agent chooses to evacuate (the context condition evaluates to True and the plan is selected), then it must do three things: prepare a mode of trans-

portation (`PrepareMode`), assemble the family members (`AssembleFamily`), and then move to a safe location (`MoveSafeLoc`), all of which are in themselves complex tasks (sub-goals). Eventually the deliberation leads to leaf plans where the agent actually performs an action in the environment, such as contacting a neighbour to arrange a lift (by sending message `M:ReqLift`), or driving to the safe location (`Drive(L)`).

BDI programs assume that environments are dynamic and can change rapidly. Therefore, the agent does not form a complete plan of action upfront, as is the case in classical planning [43]. Instead, deliberation over a choice is left to the last possible moment. For instance, when the agent decides to evacuate (selects the `EvacuateHouse` plan), it does not know exactly how it will do so (whether it will end up driving or arranging a lift with a neighbour). Further, BDI programs assume that actions they perform in the environment can fail. For instance, the plan to get the car (`GetCar`) could fail if the car turned out to have a flat battery. When a plan fails, the BDI failure recovery mechanism allows the agent to reconsider its options for its most immediate goal (it may still be possible to achieve the `PrepareMode` goal by arranging a lift with a neighbour). This way of reasoning is not dissimilar to how humans operate. In fact, we have found the BDI representation to be intuitive also for non-programmers, such as social scientists and emergency services personnel, making informal validation of behaviours by domain experts feasible.

3 Conceptual Framework

In this section we describe the conceptual framework we have introduced in Figure 1, where the agent is split between two systems – the cognitive part in the BDI system, and a physical part in the ABM system.

3.1 BDI Agent View

In the BDI world view, an agent is *situated* in an environment that it perceives, via *percepts*, and in which it acts, via *actions*. In the integrated setting, perceiving and acting, in relation to the environment, happens inside the ABM, where the physical agent interacts with the physical world of the domain. It is the ABM agent that obtains environmental information available to it, such as about its surroundings, and performs environmental actions that make sense in the given situation. The percepts/actions arrows in Figure 1, on the other hand, are communications between the BDI-ABM counterparts, and need not be the same as the percepts/actions in the ABM. More than likely in fact, these are one-to-many mappings to environment level percepts/actions, that capture higher-level concepts.

A percept going from the ABM agent to its BDI counterpart, is typically a higher-level percept composed from lower-level observations in the environment. For example, in the bushfire evacuation scenario, a `road-congestion`

percept may be composed by the ABM agent whenever it observes several cars queued up ahead and its own speed has dropped below a certain threshold. This setup is not dissimilar to that found in robotic agent systems, where the vision subsystem processes lower-level video frames to produce object perception, for consumption by higher-level reasoning subsystems.

In the same way, an action going from the BDI agent to its ABM counterpart must typically be decomposed into a sequence of lower-level environment actions that the ABM agent knows how to perform. Note that BDI systems in general allow running several intentions in parallel, which can lead to an agent performing multiple compatible actions at any given time. For instance, in response to a fire alert, a parent agent could be driving to the school to pick up their children, while trying to contact the school on the phone at the same time.

A central tenet in BDI systems, in contrast to classical planning systems, is that environments are dynamic. Combined with the view that executing plans take time to complete, i.e., actions are *durative*, this means that the enabling conditions of a plan can change during its (long) execution. For this reason, BDI agents do not commit to a full plan of action upfront. Instead, plan selection for sub-tasks (sub-goals) is deferred to the time when the agent is actually in the situation when that choice has to be made. Since a BDI agent always evaluates the situation before committing to the next sub-task of the plan, it also has the ability to *suspend* or altogether *abort* its current plan of action based on any new information it has received (via percepts). Even actions it performs in the environment are allowed to *fail*, for reasons beyond its control. From a simulation point of view, all this means that actions sent by the BDI agent may be executed by the ABM agent for the duration of several simulation cycles; the `drive-to(location)` action being a case in point. Our integration framework also allows for executing actions to be suspended, aborted, or failed. We cover the technical details of this in Section 4.

3.2 BDI Programming View

In the integrated BDI-ABM setting, the BDI system retains autonomy. It can proactively start a new course of action, and/or suspend or abort an existing course of action. That is to say that the BDI agent is not restricted to merely reacting to events in the ABM environment (captured by percepts coming from the ABM). Technically, this is possible because the BDI system is invoked on each ABM cycle, even if there are no percepts or actions statuses to deliver from the ABM side. This gives the BDI agents the opportunity to initiate new actions in any simulation cycle.

From a BDI programming point of view, the developer has flexibility in the level of abstraction used for the BDI-ABM percepts/actions, and how they are composed/decomposed into environment level percepts/actions in the ABM. For instance, in the bushfire evacuation simulation, the decision to drive to a given evacuation shelter is put into action by the BDI agent by sending a

`drive-to(shelter)` action, which results in the ABM agent (MATSim agent in this case) planning a route (a series of links to traverse on the road network) from its current position to the destination, and inserting this new driving leg into its travel plan. The route planning in this case happens at the ABM level, simply because the ABM (MATSim) already has advanced route planning routines along with all the necessary information about the road network and current traffic conditions to produce a reasonable travel plan for the agent.

Overall, the workflow for the BDI programmer remains unchanged in this integrated setting. The key addition is the added responsibility of implementing the BDI level percepts/actions in terms of the ABM level percepts/actions. We will describe this in more detail in Section 5. Once the application-specific BDI actions and percepts are agreed upon, development can typically be done in parallel on the BDI and ABM application code.

3.3 Integrated BDI-ABM View

The conceptual framework of Figure 1 realises the integration of an off-the-shelf BDI system with an off-the-shelf ABM system. In this integrated setting, the two systems run independently, and our framework provides the "glue" that manages the synchronisation and data passing between them.

A key issue at the system level is that the ABM is time-stepped and operates on a simulation clock, whereas the BDI system is event-based, and therefore does not model time explicitly. Conceptually we resolve this by making the ABM the "master" that drives the integrated simulation, and making the BDI system the "slave" that gets called by the ABM system once on each simulation cycle. This master-slave relationship does not preclude the BDI system from pro-actively initiating actions, since the the BDI system is called at every simulation cycle, as mentioned earlier.

An interesting conceptual question here is how to recognise if the corresponding amount of time has elapsed in the event-based BDI system which does not have an equivalent to the ABM clock function. In other words, what is a suitable "reasoning cycle" in the BDI system, corresponding to each simulation cycle, after which control can be passed back to the ABM? We use the BDI system state where all agents are idle, to signify the end of the BDI reasoning cycle. This is reasonable, since in this state, the BDI system can only be activated by an external event, which in this case can only come from the ABM. So upon reaching this state, it is safe to return control back to the ABM which can then progress to the next simulation cycle. One nuance here is that one could end up in the situation where a BDI agent becomes idle after sending a message to another agent that is already idle. For the duration that the second agent has not received the message, it is conceivable that the BDI system becomes momentarily idle if all other agents are idle. To avoid such cases, we explicitly keep track of agent-to-agent messages, incrementing the counter whenever a message is sent, and decrementing it when a message is

received. We declare the system as idle only when all agents are idle *and* the message counter equals zero.

Agent-to-agent communication in this setting also brings up an interesting question. Should two BDI agents be allowed to communicate directly with each other, or should all agent-to-agent communication be done only between two agents in the ABM? A reasonable option is to enforce the latter, since, after all, what does it mean for two brains to talk to each other without a physical medium. That said, all BDI systems directly support agent-to-agent communication, and if the way in which the medium impacts communication is not being modelled, then it is certainly more efficient to bypass the overhead of communicating via the ABM. Moreover, since our framework supports BDI agents that do not necessarily have a physical embodiment, such as an organisation, then restricting communication to ABM agents alone does not suffice. Finally, there may be good reason to model agent-to-agent communication at a finer granularity than what is offered by the simulation clock, for instance when one time step represents one simulated year and agent reasoning involves agent-to-agent interactions over several days within the year. Therefore, we do not impose any restrictions on how the agents communicate, and both ways are supported.

In our setup of Figure 1, the responsibility of controlling and progressing the simulation lies with the ABM. In this regard it is also worth discussing the value of an external controller. For instance, one could conceive of an alternative architecture where both the BDI and ABM systems are slaves, and the master is some external controller that pulls together and orchestrates the integrated simulation. This use case is not unusual, and in fact several general-purpose integration frameworks exist, that could be considered.

The High Level Architecture (HLA) standard [30, 24], for instance, allows a geographically distributed simulation (federation) to be constructed by linking together a number of simulation components (federates). The federates may be written in different languages, and can run at different time steps. A Federation Object Model (FOM) defines the types of and the relationships between the data exchanged by the federates in a federation. Each federate must typically translate from its internal notion of simulated entities to HLA objects and interactions as specified in the FOM. A federate declares its interest in objects and attributes at the beginning of a simulation by publishing any attributes it may update during the simulation (via the RTI Ambassador) and subscribing to attributes which it would like to receive updates for (via the Federate Ambassador). To update the value of a particular attribute instance, a federate must first acquire ownership of (i.e., write access for) that instance. All communication between the federates in a federation is accomplished via middleware called the Runtime Infrastructure (RTI). From the BDI-ABM integration perspective, while technically possible, use of HLA for control has several drawbacks. Firstly, the HLA standard is rather complex. For simple integration scenarios this can result in a significant implementation overhead, as each federate must implement at least the core HLA functionality. As a result, considerable work may be required to integrate an existing ABM or BDI

platform into an HLA federation. Secondly, the inter-operability supported by the HLA comes at a computational cost, as every interaction in the simulation must happen via the RTI. For MAS simulations which consist of many lightweight components that access the simulation state frequently (relative to the CPU they consume), the computational overhead of the RTI can be significant. Lastly, HLA standard does not mandate an underlying network protocol, which is left to the implementer of the RTI. This can severely limit inter-operability in practice, as each federate must implement RTI and Federate ambassadors compatible with the RTI used by a particular federation.

In terms of data transfer between the BDI and ABM systems, we perform two key optimisations. First, a single data container is passed between the systems in each simulation cycle, as opposed to the BDI-ABM counterparts sending individual messages. The data container bundles the messages for all agents and delivers them all together to the other system. This simplifies the synchronisation between the systems, and is also typically faster, specially if the communication link between the BDI and ABM systems is network based.

Second, and importantly, not every percept that an agent may ever need is computed and pushed to the BDI system on every cycle. This is because the BDI agent processes information contextually, as it performs its course of action, and so only certain information is useful in certain situations. For instance, an evacuating agent may be interested in knowing that its elderly neighbours have arranged for someone to pick them up, only when it is about to leave. Once it is on the road, this information may no longer play a part in its decision making. In such cases, it is more economical, for data transfer, if the BDI agent pulled this percept from the ABM environment as needed. Our framework supports this kind of information retrieval using *BDI queries*, which we describe further in the next section.

4 Integration Architecture

In this section we describe the technical aspects of our integration framework. Overall, the framework consists of three distinct layers as shown in Figure 4: a *generic layer*, which realises the conceptual model shown in Figure 1; a *system layer*, which provides the code necessary for linking a particular BDI or ABM system into the generic layer; and an *application layer*, which provides the application-specific code including agent behaviour and reasoning. The pseudo code for the BDI-ABM runtime execution is shown in Algorithm 1. The layer responsible for each step of the process is also indicated. The BDI and ABM systems execute once per simulation time step, with the BDI application providing action decisions to the ABM, while the ABM provides observations and environmental information of interest, to the BDI module.

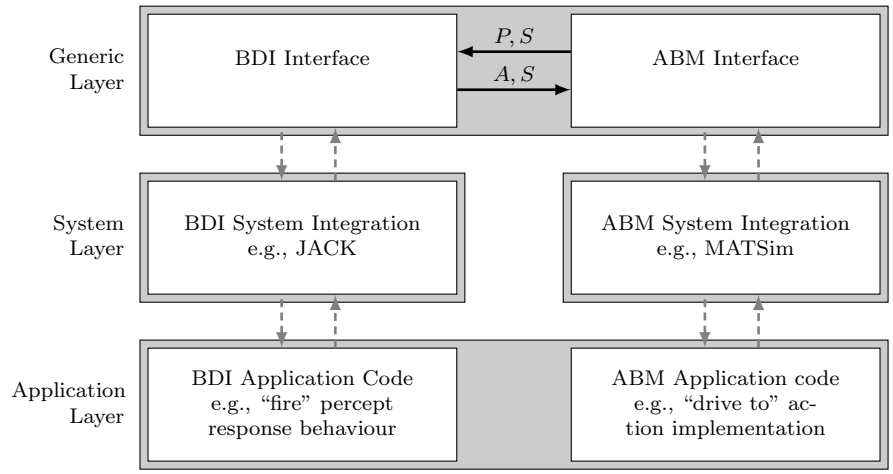

Fig. 4 Three-tiered BDI-ABM integration architecture showing information flow in each synchronised simulation step; ABM system sends percepts list P to the BDI system and receives back actions list A; S is the list of actions' status'

Algorithm 1: Algorithm for the BDI-ABM runtime execution; comments indicate the integration layer (G: Generic, S: System, A: Application) in which the code executes

```
1  BDI and ABM applications initialised ;                          // A
2  Links established between BDI-ABM agent counterparts ;          // A
3  for each time step do
4      ABM sends packaged information to BDI module ;              // G
5      Information is distributed to individual BDI agents ;       // S
6      for each BDI agent do
7          execute until idle (possibly querying ABM counterparts) ;  // S
8          notify new actions or action status changes ;           // A
9      BDI sends packaged information on agent actions to ABM ;    // G
10     Information is distributed to individual ABM agents ;       // S
11     for each ABM agent do
12         execute one time step ;                                 // S
13         package percepts and action status ;                    // A
```

4.1 The Generic Layer

The generic layer provides the high level interface between the BDI and ABM systems. It defines functions that allow the ABM to initialise/terminate the BDI system and create/destroy BDI counterpart agents on the one hand, and the BDI system to pull percepts from the ABM on the other. It ensures that control is passed between the two systems in a synchronised manner.

The generic layer includes data structures for passing information between the two systems. Table 1 shows the structure of the agent data container. Each such container is tagged with the ID of the agent it belongs to, and a list of

Table 1 Structure of an agent data container—a list of these containers is passed between the ABM and BDI systems on each simulation cycle

Data Type	Values
BDI action	$< instance\_id, action\_type, parameters, status >$ where *status* is one of: INITIATE—Initiated by BDI agent and to be executed RUNNING—Being executed by the simulation agent PASS—Completed as expected FAIL—Aborted/failed by the simulation agent DROPPED—Aborted by the BDI agent SUSPENDED—Temporarily suspended by the BDI agent
Percept	$< percept\_type, value >$ where *value* may be a complex type
Query	$< query, response >$

these containers is passed between the two systems, inside a single composite container, in each simulation cycle. The data in these containers represents a *BDI Action*[3], a *Percept* or a *Query*:

- **BDI Actions** are the high level actions initiated by BDI agents, that are executed as a sequence of environment level actions by the ABM agent. Note that BDI actions are durative, i.e., can execute for several simulation cycles on the ABM. Action *status* is used by the ABM system to indicate the status of active actions, such as success/failure. On the BDI side, action status is used to initiate new actions, or suspend[4]/abort current actions.
- **Percepts** refer to the percepts that are pushed from the ABM to the BDI system, providing information which conceptually should automatically be noticed, or *perceived*, by each reasoning (BDI) agent. These percepts may trigger some response from the BDI agent.
- **Queries** refer to the percepts that are pulled by the BDI system, accessing information from the ABM, as and when needed. These queries are typically sent by the BDI agent during the evaluation of the context condition of a plan. Queries only extract information from the ABM, and do not change its state in any way.

4.2 The System Layer

The system layer contains platform-specific code for integrating particular BDI and ABM platforms into the framework. It provides implementations for the generic level functionality, as well as providing the code for packing and unpacking of the generic layer data container message.

[3] We call these actions BDI Actions to distinguish them from actions in the ABM which may include lower-level actions.

[4] Suspension is not yet fully implemented in the publicly available software.

4.2.1 BDI System Layer

The BDI system layer has three main functions, to: provide a transparent programming model for durative actions (explained below); collect actions from, and distribute percepts to, individual agents (this is fairly straightforward, and we do not discuss this any further here); and detect the end of a BDI reasoning cycle (this we have covered already in Section 3).

As explained earlier, BDI actions may take several execution cycles to complete on the ABM side. Where such an action is a synchronous step in an agent intention (plan), the agent reasoning for that particular intention (plan) must wait (block) until the action has completed with success or failure, unless something happens such that the BDI agent wishes to abort. We do this by providing a generic goal-plan pair: `actionGoal` and `actionPlan`. In order to initiate the execution of a BDI `action` the BDI agent posts the `action`, such as `drive-to(loc))`, as the goal `actionGoal("drive-to",loc)` from within a plan body. This goal is then handled by the plan `actionPlan` as shown in Algorithm 2. Here `actionList` is a list that contains all instances of actions which are in the states described in Table 1.

Algorithm 2: Algorithm for `actionPlan`

```
1 Add action to actionList indexed on agentID, with action status=INITIATE
2 Wait for status to change // due to ABM message, or BDI decision making
3 if status==PASS then
4 |   finish, indicate that the goal actionGoal was achieved, i.e. action was executed
  |   successfully, and remove action from actionList
5 if status==FAIL or DROPPED then
6 |   finish, indicate that the goal actionGoal failed, i.e. action was not executed
  |   successfully, and remove action from actionList
7 if maintenance or parallel violation then
8 |   set status of action to DROPPED
```

A common reason in agent systems for aborting plan execution, is when a *maintenance condition*, which in normal operation should hold for the duration of plan execution, fails. Another reason is if there are parallel sub-goals being pursued, and the programmer has indicated that if one parallel clause fails, all should fail. Different BDI systems provide slightly different mechanisms for this, but we basically handle such events by setting the relevant action status to `DROPPED` (lines 7–8), which results in the action being aborted by the ABM, and on the next cycle being removed from the action list by the BDI system.

So far we have developed BDI layer integrations for the following platforms: JACK [12], Jadex [11], and GORITE [26].

4.2.2 ABM System Layer

THE ABM system layer serves three key purposes, to: update actions status'; perform an ABM time step; and distribute actions to, and collect percepts from, individual agents. It also provides access to the step function of the simulation in some manner, as this is used to maintain the synchronisation between the two systems. Code must be integrated with the step function of the ABM to manage the `actionList` that is passed between the two systems.

Typically an ABM agent contains a series of single time-step rules with execution schedules or trigger conditions defining at which time-steps they should run. As our BDI actions often need to be executed over multiple time-steps our system level manages application level rules such that the sensor-actuator agent will perform an appropriate portion of the action. For example if the BDI action is to move to a given destination, the rule will move some distance towards the destination in each step. There can be multiple rules within an agent, each to perform a different action. These may be able to be performed in parallel. In order for an ABM agent to execute the actions specified in the `actionList` we provide a new system level rule, executed at each time-step, which operates as follows:

Algorithm 3: Algorithm for handling BDI actions in the ABM

```
1 for each action in actionList do
2     if action.status == DROPPED then
3         remove action from the actionList and continue
4     if action.status == INITIATE then
5         read action parameters and change action.status to RUNNING
6     if action.status == RUNNING then
7         call the appropriate rule, indexed on agentID, to take one
          (time)step towards the action's completion
```

Note that an action will begin execution in the same time-step that its status is set to `RUNNING`. When an action successfully completes, or fails, the system level changes the status of the action to `PASS` or `FAIL` in the message to be passed to the BDI agent, and then removes the action from its `actionList`.

The ABM layer integrations we have developed, provide support for the following platforms: the general-purpose Repast [34] and GAMS [18] platforms, the domain specific MATSim [3] traffic simulator, as well as an application-specific Python-based simulator [20].

4.3 The application layer

The application layer is where a particular BDI system layer and a particular ABM system layer is composed, together with all the specific logic for the given domain, to form an executable program. Building application level code requires,

- identifying the application-specific BDI actions (e.g., `drive-to`) and percepts (e.g., `road-congestion`),
- making design decisions, given that the agent is split between the BDI and ABM systems (Figure 1), as to what reasoning is performed in which system,
- writing functions on the ABM side to implement BDI actions and to collect BDI percepts,
- writing BDI code that performs deliberation, taking into account its goals, as well as incoming percepts from the ABM side, to perform BDI actions using the generic action goal infrastructure of the BDI system layer, and
- creating an application scenario in the ABM (e.g., for a given number of agents, distributed spatially in a certain configuration).

Once the application-specific BDI actions and percepts are agreed on, development can typically be done in parallel on the BDI and ABM application code. On the BDI side, this involves writing a program in the usual way (for example, the resident behaviour in Figure 3), but using the system level generic action goal for BDI actions and handling the ABM percepts coming in. Application code on the ABM side implements BDI actions in terms of agent actions in the ABM, as well as building the agent-specific BDI percepts by querying ABM state.

5 Integrations and Application Examples

The three-tiered integration architecture we have described in Section 4 lends itself to putting together simulation applications with BDI cognitive agents in a modular and decoupled manner. The various components in the architecture of Figure 4 are built as standalone Java libraries, packaged in JAR (`.jar`) files. Building a new application typically involves pulling together the generic and platform level JAR files, and writing the application level code for the problem domain. Altogether, we have built five applications for very different domains in this manner, as outlined in Table 2. In this section we describe two of these in some detail – a bushfire evacuation application, and a conservation ethics model.

5.1 Bushfire Evacuation Model

The bushfire evacuation model is a proof-of-concept decision support tool that we have developed over several years with guidance from the Country Fire Authority (CFA) in Australia. The tool aids the planning and preparation for evacuations of regional towns in the event of imminent bushfires. It is a

[5] `http://repast.sourceforge.net/docs/RepastJavaGettingStarted.pdf`

Table 2 Application examples available in the BDI-ABM integration repository

Application	Platforms Used	Description
Bushfire Evacuation	MATSim, JACK	A GIS simulation of a bushfire evacuation in regional Australia; uses OpenStreetMap for road network data; Census and government databases for residential and households data; Phoenix RapidFire [57] for simulating fire spread; MATSim for simulating the traffic; JACK for modelling residents' responses to a bushfire warning
Zombies	Jadex, Repast	Extends the Repast Zombies and Human demonstration simulation[5] to build "smarter" zombies (programmed in Jadex) that coordinate their moves when looking for human prey
Taxi Service	GORITE, MATSim	A GIS based simulation of a taxi company; uses MATSim for traffic simulation; taxi drivers are BDI agents (GORITE); a taxi operator (exists only in GORITE, with no physical representation in MATSim), manages jobs broadcasts and assigns jobs based on requests from taxi drivers; taxis have autonomy in which jobs they bid for
Conservation Ethics	JACK, GAMS	Builds on work by [25] that models (in GAMS) an auction-based system where landholders bid on a range of work packages aimed at conservation targets, such as increasing the population of certain species, in an area; adds BDI reasoning to the landholder agents, to capture considerations other than financial, such as social and ethical "barometers"
Vaccination of Children	JACK, Python-sim	Builds on work by [19] that models (in a standalone Python-based simulation) the spread of diseases, such as measles, in a population (up to a few hundred thousand agents) over several decades; adds BDI reasoning (JACK) to the decision making of parents w.r.t. vaccination of their children; BDI reasoning is invoked only for females and only during the years when vaccination decisions have to be made

GIS-based model of a local area with its road network (extracted from OpenStreetMap[6]), buildings (extracted from government datasets[7]), and population (constructed using demographics data from census[8]). Residents are modelled as BDI agents in JACK. Traffic flow simulation is done in MATSim [37]. The simulator can inject time-stamped bushfire progression for the area, modelled in Phoenix RapidFire [57]. A custom built visualiser allows user-interaction with the simulation (Figure 5).

Community evacuation in response to bushfires is a relatively new policy measure introduced after the devastating Black Saturday fires of 2009 in which 179 lives were lost. The first real evacuation was performed in 2014, for the regional town of Halls Gap, VIC. As such, little or no evacuation data exists, to aid evacuation planning. In this regard, a simulation tool that can be configured for individual communities, and that gives fine grained understanding

[6] `www.openstreetmap.org`

[7] `http://services.land.vic.gov.au/landchannel/content/productCatalogue`

[8] `http://www.abs.gov.au/websitedbs/censushome.nsf/home/communityprofiles`

Fig. 5 Planning community evacuation for Halls Gap VIC, Australia: a custom visualiser allows a user to interact with the simulation, here influencing the behaviour of an agent via breakpoints at decision points in the BDI program.

of the nuances of evacuation in that community (such as local bottlenecks in traffic flow) can be invaluable.

Our interactions with emergency services personnel over the past few years have highlighted a key concern they have with any kind of modelling – that people behave in complex ways, and unless this variation can be captured in a convincing way, modelling of community response can have only limited value. To this end we have found the BDI agent model to be a very powerful tool. First, the BDI goal-plan hierarchy can capture incredible variation in behaviour in a compact representation. Second, and importantly, BDI concepts are intuitive to non-programmers, which allows emergency services personnel to engage in the development and refinement of behaviour profiles, such as the response of parents to a bushfire threat during school hours. We have found that this kind of informal validation of individual behaviours by emergency services personnel, leads to better confidence in and acceptance of the overall outputs of the model.

This bushfire evacuation problem domain has been a driving force in the development of our integration framework, since from the onset it was clear that we needed a nice way to integrate BDI agent modelling with agent-based modelling and simulation.

Figure 5 shows a snapshot of the interactive simulation which gives a bird's eye view of the local region. In this snapshot, cars, representing residents, can be seen stationed at homes. Each resident is internally modelled as a BDI agent in JACK and a physical agent in MATSim. In the event of a bushfire warning, the BDI agents make the high level decisions about what to do (such as pick

up kids) and where to go (such as to the school first), while the MATSim agent carries out the actions by figuring out a route and driving to the particular destinations. Overall, individual residents (cars) can be inspected during the interactive simulation to find out where they are going, and importantly, why.

The simulation also consists of an Evacuation Controller agent, that models an incident controller person who makes decisions about which regions of the town to evacuate in which order. When it is time to evacuate based on a progressing fire, the Controller orders the evacuation of the regions, as per the decided schedule. Within the simulation, the Controller agent lives only in the BDI side, and has no physical counterpart in the ABM.

Users can interact with the Controller to influence its behaviour, via the custom visualiser. This is done by setting breakpoints at particular goals that the agent has (corresponding to decisions about which region to evacuate to which relief center, at what time, and using which route), giving the user the opportunity to force different plan choices to what the agent would otherwise select.

The actions and percepts in the system are as follows:

- Percept `fire-alert`: This percept is generated by the fire module, when the developing fire has progressed enough to pose a risk to the town. The Controller agent acts on this percept by broadcasting messages to all residents in each region, at appropriate times as determined by the schedule.
- Percept `road-congestion`: This percept is generated by MATSim and issued to all evacuating residents who have been on a link for longer than a factor of β times the free speed travel time, where $\beta > 1$. This allows encoding of BDI behaviours where residents are known to take risks when stuck in traffic jams.
- Action `drive-to(loc)`: In both systems where we have used MATSim (see Table 2), this is always one of the BDI actions. The MATSim counterpart executes this high level BDI-action by planning the route from the current position to the required location and inserting the relevant legs into the MATSim plan. Standard MATSim behaviours will then cause the agent to follow this route to the destination. The MATSim counterpart reports when the action succeeds, or when it fails.
- Action `take-route-to(loc, route)`: This action is used for sharing the route-planning decision making between the BDI and ABM systems. Here `route` is a list of way-points that map out a high-level plan of reaching the destination `loc`, as determined by the BDI agent-based on reasoning about the fire behaviour. This is then translated by the ABM side into a series of legs (what would be computed by a sequence of `drive-to(waypoint)` actions) that are inserted together into the travel plan for the agent.

Evacuating resident agents exhibit some of the known behaviours of residents in bushfires, such as driving to pick up children from school and/or loved ones from nearby locations first, before driving to the designated evacuation centres.

5.2 Conservation Ethics Model

Market-based incentive schemes are increasingly being used worldwide to increase landholder participation in local conservation and biodiversity management. One such scheme is the *conservation auction*–a competitive reverse-auction mechanism where regional landholders submit a set of environmental projects (bids) from which the conservation agency (auctioneer) selects suitable ones to satisfy its environmental objectives. In a multi-objective auction model, the agency has a fixed target for conservation outcomes which it wishes to secure at minimum cost, and allocates budget (selects projects) accordingly. Landholders then act to achieve the contracted outcomes by the end of the specified period, through on-ground actions on their land (the projects). In Australia, there have been many conservation auction trials in recent years [25].

These monetary incentive schemes, however, do not take into account intrinsic motivation factors, or social norms, that play an important role in landholders' decisions to participate. For instance, motivation crowding theory suggests that monetary incentives can have a negative impact on intrinsic motivation in voluntary participation settings.

We have developed a model that we compare against a previously developed economic model of conservation auctions [25]. This work was done in collaboration with authors of the original work, together with social science experts. In the original simulation the landholder agents learn (using machine learning techniques) to make successful bids in an iterated auction procedure. The landholders are assumed to be rational economic agents. In the comparator model we have four groups of landholders, with different bidding strategies based on their *conservation ethic*–the extent to which a person cares about conservation issues–and profit motive as follows:

- high conservation ethic and low profit motive: relatively many bids, at a cost that is slightly over or slightly under actual cost recovery.
- high conservation ethic and high profit motive: a moderate number of bids with a moderate profit.
- low conservation ethic and high profit motive: only those bids which give a substantial amount of profit (actual amount, not simply percentage of cost recovery).
- low conservation ethic and low profit motive: a small chance of random participation.

Conservation targets are specified, similar to the original model, as the population sizes of three threatened species (Red-Tailed Phascogale, Carpet Python and Malleefowl) found in Western Australia. The auctioneer has a target population size at the end of a specified contract period, however landholders do not know what the target is. Landholders bid on one or more of 26 available projects, for conservation efforts delivering different combinations of population size increases in the three species, for different costs. Each landholder has the same cost model, which allows for economies of scale. The auctioneer determines the bid winners such that the target can be met,

while minimising the overall cost. The auction outcomes modify (potentially) each landholder's conservation ethic and profit motive for subsequent auction rounds, resulting in emergent behaviour.

Here, landholders are modelled as BDI agents in JACK. The ABM is the auction environment, and is the original model developed in GAMS [25]. The auctioneer agent only exists in the ABM. The BDI actions and percepts in this setting are as follows:

- `call-for-bids` : this percept is received by all BDI agents, and signals the start of the auction process to the BDI landholders;
- `bid(params)` : this is a BDI action sent by a landholder agent, and contains a list of all bids for that agent, for that auction round
- `auction-result(params)` : this is a percept provided by the ABM auctioneer to all BDI agents, and contains the results of the auction round, including information about the winning bids.

We do not claim to have an accurate model of landholders; rather we have implemented a simplified version of some of the social science findings regarding motivational crowding out, and show that a landholder model which takes account of non-monetary factors can result in significantly different outcomes with respect even to cost effectiveness, as compared to the model that considers only monetary incentives.

6 Discussion and Conclusion

There are several strands of work in the Agent-Based Modelling and BDI literature that consider the problem of integrating BDI agents with an ABM. In this section, we briefly describe the key similarities and differences between our approach and related work from the literature, and conclude with some directions for future work.

One strand of work has focused on extending either standard (non agent-based) simulation formalisms or ABM platforms to support BDI agents. For example, Mittal and Douglass [33] present a formalisation of the ACT-R architecture for cognitive agents in the Discrete Event System Specification (DEVS) formalism [59], and Zhang and Verbraeck [60] present a formalisation of PRS [21] agents in DEVS. The DEVS formalism is widely used in the discrete event simulation community, allowing inter-operability with other simulation models formalised in DEVS. The resulting models are platform (simulator) independent, and can be readily parallelised. However the approaches in those works [33,60] provide limited support for translating high-level ACT-R or PRS agent programs into the DEVS formalism, and current DEVS simulation platforms do not provide development or debugging support at the BDI agent level, rendering agent development more difficult. Shendarkar et al. [47] describe an implementation of BDI agents using the AnyLogic[9] commercial

[9] `www.anylogic.com`

simulation package. AnyLogic provides extensive support for developing agent based models, however the BDI agents in the system of Shendarkar et al. [47] are targeted specifically at crowd simulation. Sakellariou et al. [44] present NetLogo [56] libraries to support the development of BDI agents and FIPA-compliant communication between agents. However while their approach is general in allowing a wide range of scenarios to be simulated, the BDI library described is limited to simple intention structures, and lacks the features of a fully-fledged BDI platform.

Another strand of work involves the use of existing BDI platforms for (agent-based) simulation. For example, Bordini and Hübner [8] describe how the JASON BDI platform can be used for social simulation, however most BDI platforms (e.g., Brahms [48], 2APL [13], Goal [23], JaCaMo [6]) provide some degree of support for developing simulations, as the default assumption underlying such platforms is that the agent's environment is simulated. This approach provides limited support for the non-agent parts of the simulation compared to, agent-based simulation platforms such as Repast.

A third strand of work seeks to couple a mature agent platform with a particular fully-fledged ABM simulation. A number of special purpose ABM simulations have been developed that are designed to allow the integration of BDI agents, e.g., RoboCup [28], RoboCup Rescue [29], Gamebots [27], and the Multi-Agent Programming Contest [4]. Communication with the BDI platform is typically based on message passing (sockets), and so neutral with respect to the choice of BDI platform used to implement the agents. However these simulators support a single, parameterised simulation model, e.g., robot soccer in the case of RoboCup. There has also been work on combining an ABM simulation with a particular cognitive agent platform [53,55,54]. Similar to the previous systems [28,29,27], these simulators support a single (parameterised) simulation model; however in addition they are limited to a single agent platform. For example, TacAir Soar [53] and SWARMM [55] are specific to a particular domain (air combat) and particular agent platform (Soar and dMARS respectively).

Finally, there is also a strand of work on extending ABM and BDI platforms to make them compliant with the High Level Architecture (HLA) and thus inter-operable. For example, HLA-Repast [32] is an HLA-compliant extension of the Repast agent-based simulation platform, and HLA-Agent [31] is an HLA-compliant implementation of the SIM AGENT [51] agent simulator. However this work targets federated, distributed, simulations and requires additional Run-Time Infrastructure to manage the execution of the composed simulators.

To the best of our knowledge, there has been very little work on integrating a fully-fledged BDI platform with a fully-fledged ABM platform in a way that leverages the features of both. The majority of the work described above starts either from a BDI agent platform and provides basic support for implementing a simulation, or from an ABM simulation platform and provides some support for more complex agents. The previous work that does provide a high level of support for both BDI and ABM is limited to simulating fully-fledged BDI agents in a single simulation scenario, or a narrow range of scenarios.

One piece of work which is very related to our infrastructure is that of van Oijen in his PhD thesis [35] where he explores and describes his CIGA middleware system that seeks to connect a game engine with a BDI system. His high level communication is very similar to that of our generic layer, which was also described in [38], being one of our first implementations of the BDI-ABM connection which we have now fully generalised. CIGA is focused specifically on games, and as such makes demands of the simulation graphics which adds complexity that we did not attempt to deal with in our infrastructure. Our applications have had agents which while they may be cognitively somewhat complex, do not require the graphical realism of characters in games. For example our evacuation simulation graphics is quite satisfactory with cars as dots following roads. Nevertheless some of the issues mentioned by van Oijen were also seen and addressed in our system. An example is the requirement for durative actions, which in their case study caused issues. Our system level BDI mechanism for managing this addresses this issue, but does need to be developed at the system level if durative actions are not natively supported. CIGA aims to provide many more generic services than does our system. In order to accomplish many of the things addressed by such services (such as percept processing into more usable percepts), we rely on guiding the developer on how to develop the necessary functions within the simulation engine/application chosen. We do expect that for a particular domain and simulator, re-usable libraries of percept processing functions will be developed, and we have started to see this in our repeated use of MATSim. Overall the goals of CIGA are quite similar to our system, though more extensive. At the same time it would seem that our system is perhaps more complete, within its more limited scope, and has been used for more complete applications.

Overall, the work presented here, supports integration of a wide range of cognitive agent modelling platforms, with a similarly wide range of agent-based simulation platforms or applications. The only requirement is that the percepts (or environmental observations/events) of interest to each agent, and the actions that the agent may execute in the simulation environment can be identified. The support infrastructure we have developed along with the code required for a number of specific systems, and several example applications is freely available at `http://tiny.cc/bdi-abm-integration`.

Acknowledgements We would like to thank Kai Nagel, Sarah Bekessey, Fiona Fidler, Ascelin Gordon, Sayed Iftekhar, Nic Geard, Carole Adam, Todd Mason, Sewwandi Perera, Edmund Kemsley, Oscar Francis, Daniel Kidney, Thomas Wood, Andreas Suekto, Qingyu Chen, Arie Wilsher, Sarah Hickmott, and Dave Scerri for their contribution to the various platform integrations and applications discussed in this paper. We thank AOS for supporting this work through the provision of their JACK agent system for research purposes.

References

1. Axelrod, R.M.: The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration. Princeton Univeristy Press (1997)

2. Balke, T., Gilbert, N.: How do agents make decisions? A survey. Journal of Artificial Societies and Social Simulation (JASSS) **17**(4), 13 (2014)
3. Balmer, M., Rieser, M., Meister, K., Charypar, D., Lefebvre, N., Nagel, K., Axhausen, K.: MATSim-T: Architecture and simulation times. Multi-agent Systems for Traffic and Transportation Engineering pp. 57–78 (2009)
4. Behrens, T.M., Dastani, M., Dix, J., Hübner, J., Köster, M., Novák, P., Schlesinger, F.: The multi-agent programming contest. AI Magazine **33**(4), 111–113 (2012)
5. Benfield, S.S., Hendrickson, J., Galanti, D.: Making a strong business case for multiagent technology. In: Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 10–15. Hakodate, Japan (2006)
6. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with JaCaMo. Science of Computer Programming **78**(6), 747–761 (2013)
7. Bordini, R.H., Hübner, J.F.: BDI Agent Programming in AgentSpeak Using Jason. In: International Workshop on Computational Logic in Multi-Agent Systems (CLIMA), pp. 143–164. London, UK (2005)
8. Bordini, R.H., Hübner, J.F.: Agent-Based Simulation Using BDI Programming in Jason. In: Multi-Agent Systems: Simulation and Applications, pp. 451–471. CRC Press (2009)
9. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-agent Systems in AgentSpeak Using Jason. Wiley (2007). Wiley Series in Agent Technology, ISBN: 0470029005
10. Bratman, M.E.: Intentions, Plans, and Practical Reason. Harvard University Press (1987)
11. Braubach, L., Pokahr, A., Lamersdorf, W.: Jadex: A BDI agent system combining middleware and reasoning. In: Software Agent-Based Applications, Platforms and Development Kits, pp. 143–168 (2005)
12. Busetta, P., Rönnquist, R., Hodgson, A., Lucas, A.: JACK intelligent agents - components for intelligent agents in Java. Tech. rep., AOS Pty. Ltd, Melbourne, Australia (1998)
13. Dastani, M.: 2APL: a practical agent programming language. Autonomous Agents and Multi-Agent Systems **16**(3), 214–248 (2008)
14. Dennett, D.C.: The Intentional Stance. MIT Press (1987)
15. Dignum, V., Vázquez-Salceda, J., Dignum, F.: OMNI: introducing social structure, norms and ontologies into agent organizations. In: Programming Multiagent Systems Languages, Frameworks, Techniques and Tools workshop (PROMAS), pp. 181–198 (2004). Selected Revised and Invited Papers
16. Epstein, J.: Generative Social Science - Studies in Agent-Based Computational Modeling. Princeton University Press (2006)
17. Ferber, J.: Multi-Agent Systems. Addison Wesley Longman (1999)
18. GAMS Development Corporation: General algebraic modeling system (gams) website. `http://www.gams.com/` (2015). Accessed: March 20, 2015
19. Geard, N., McCaw, J.M., Dorin, A., Korb, K.B., McVernon, J.: Synthetic population dynamics: A model of household demography. Journal of Artificial Societies and Social Simulation **16**(1), 8 (2013)
20. Geard, N., Singh, D., McVernon, J., Padgham, L.: A model of parental decision making and behaviour about childhood vaccination. In: Epidemics 4: Fourth International Conference on Infectious Disease Dynamics, pp. 19–22. Amsterdam, The Netherlands (2013)
21. Georgeff, M., Ingrand, F.: Decision making in an embedded reasoning system. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 972–978. Detroit, MI, USA (1989)
22. Gilbert, N., Troitzsch, K.: Simulation for the social scientist. McGraw-Hill International (2005)
23. Hindriks, K.V.: Programming rational agents in GOAL. In: Multi-Agent Programming: Languages, Tools and Applications, pp. 119–157. Springer US (2009)
24. IEEE Standard for modeling and simulation (M&S) High Level Architecture (HLA) — Framework and rules. IEEE (2000). (IEEE Standard No.: 1516-2000)
25. Iftekhar, M., Hailu, A., Lindner, R.: Does it pay to increase competition in combinatorial conservation auctions? Canadian Journal of Agricultural Economics/Revue canadienne d'agroeconomie **62**(3), 411–433 (2014)

26. Jarvis, D., Jarvis, J., Rnnquist, R., Jain, L.C.: Multiagent Systems and Applications - Volume 2: Development Using the GORITE BDI Framework, *Intelligent Systems Reference Library*, vol. 46. Springer (2013)
27. Kaminka, G.A., Veloso, M.M., Schaffer, S., Sollitto, C., Adobbati, R., Marshall, A.N., Scholer, A., Tejada, S.: GameBots: A flexible test bed for multiagent team research. Communications of the ACM **45**(1), 43–45 (2002)
28. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E.: RoboCup: The robot world cup initiative. In: First International Conference on Autonomous Agents (Agents'97), pp. 340–347. Marina del Rey, CA, USA (1997)
29. Kitano, H., Tadokoro, S.: Robocup rescue: A grand challenge for multiagent and intelligent systems. AI Magazine **22**(1), 39–52 (2001)
30. Kuhl, F., Weatherly, R., Dahmann, J.: Creating Computer Simulation Systems: An Introduction to the High Level Architecture. Prentice Hall (1999)
31. Lees, M., Logan, B., Theodoropoulos, G.: Distributed simulation of agent-based systems with HLA. ACM Transactions on Modeling and Computer Simulation **17**(3), Article 11 (2007)
32. Minson, R., Theodoropoulos, G.K.: Distributing repast agent-based simulations with hla. Concurrency and Computation: Practice and Experience **20**(10), 1225–1256 (2008)
33. Mittal, S., Douglass, S.A.: Net-centric ACT-R-based cognitive architecture with DEVS unified process. In: Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium, TMS-DEVS'11, pp. 34–44. Boston, MA, USA (2011)
34. North, M.J., Howe, T.R., Collier, N.T., Vos, J.R.: Advancing Social Simulation: The First World Congress, chap. A Declarative Model Assembly Infrastructure for Verification and Validation, pp. 129–140. Springer Japan, Tokyo, Japan (2007)
35. van Oijen, J.: Cognitive agents in virtual worlds: A middleware design approach. Ph.D. thesis, Utrecht University (2014)
36. Padgham, L., Horne, R., Singh, D., Moore, T.: Planning for sandbagging as a response to flooding: A tool and case study. Australian Journal of Emergency Management (AJEM) **29**, 26–31 (2014)
37. Padgham, L., Nagel, K., Singh, D., Chen, Q.: Integrating bdi agents into a matsim simulation. Frontiers in Artificial Intelligence and Applications **263**(ECAI 2014), 681–686 (2014)
38. Padgham, L., Scerri, D., Jayatilleke, G.B., Hickmott, S.L.: Integrating BDI reasoning into agent based modeling and simulation. In: Winter Simulation Conference (WSC), pp. 345–356 (2011)
39. Rao, A., Georgeff, M.: Modeling rational agents within a BDI-architecture. In: Principles of Knowledge Representation and Reasoning (KR), pp. 473–484. Cambridge, MA, USA (1991)
40. Rao, A.S.: AgentSpeak (L): BDI agents speak out in a logical computable language. In: Agents Breaking Away, pp. 42–55. Springer (1996)
41. Rao, A.S., Georgeff, M.P.: BDI agents: From theory to practice. In: International Conference on Multi-Agent Systems (ICMAS), pp. 312–319. San Francisco, CA, USA (1995)
42. Reynolds, C.W.: Flocks, herds and schools: A distributed behavioral model. In: 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '87, pp. 25–34. New York, NY, USA (1987)
43. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 2 edn. Pearson Education (2003)
44. Sakellariou, I., Kefalas, P., Stamatopoulou, I.: Enhancing NetLogo to simulate BDI communicating agents. In: Artificial Intelligence: Theories, Models and Applications, *Lecture Notes in Computer Science*, vol. 5138, pp. 263–275. Springer Berlin Heidelberg (2008)
45. Sardiña, S., Padgham, L.: A BDI agent programming language with failure handling, declarative goals, and planning. Autonomous Agents and Multi-Agent Systems **23**(1), 18–70 (2011)
46. Scerri, D., Hickmott, S., Bosomworth, K., Padgham, L.: Using modular simulation and agent based modelling to explore emergency management scenarios. Australian Journal of Emergency Management (AJEM) **27**, 44–48 (2012)

47. Shendarkar, A., Vasudevan, K., Lee, S., Son, Y.J.: Crowd simulation for emergency response using BDI agent based on virtual reality. In: Winter Simulation Conference (WSC), pp. 545–553. Monterey, CA, USA (2006)
48. Sierhuis, M., Clancey, W.J., van Hoof, R.J.J.: Brahms: A multiagent modelling and simulation environment for work processes and practices. International Journal of Simulation and Process Modelling **3**(3), 134–152 (2007)
49. Singh, D., Padgham, L.: Community evacuation planning for bushfires using agent-based simulation (demonstration). In: Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 1903–1904. Istanbul, Turkey (2015)
50. Singh, D., Sardina, S., Padgham, L., James, G.: Integrating learning into a BDI agent for environments with changing dynamics. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 2525–2530. Barcelona, Spain (2011)
51. Sloman, A., Poli, R.: SIM AGENT: A toolkit for exploring agent designs. In: Intelligent Agents II: Agent Theories Architectures and Languages (ATAL-95), pp. 392–407. Springer–Verlag (1996)
52. Tambe, M.: Towards flexible teamwork. Journal of Artificial Intelligence Research **7**, 83–124 (1997)
53. Tambe, M., Johnson, W.L., Jones, R.M., Koss, F.V., Laird, J.E., Rosenbloom, P.S., Schwamb, K.: Intelligent agents for interactive simulation environments. AI Magazine **16**(1), 15–39 (1995)
54. Taylor, G., Frederiksen, R., III, R.R.V., Waltz, E.: Agent-based simulation of geo-political conflict. In: Conference on Innovative Applications of Artificial Intelligence (IAAI), pp. 884–891. San Jose, CA, USA (2004)
55. Tidhar, G., Heinze, C., Selvestrel, M.C.: Flying together: Modelling air mission teams. Applied Intelligence **8**(3), 195–218 (1998)
56. Tisue, S., Wilensky, U.: Netlogo: A simple environment for modeling complexity. In: International Conference on Complex Systems (ICCS), pp. 16–21. Boston, MA, USA (2004)
57. Tolhurst, K., Shields, B., Chong, D.: Phoenix: development and application of a bushfire risk management tool. Australian Journal of Emergency Management (AJEM) **23**(4), 47–54 (2008)
58. Winikoff, M.: JACK intelligent agents: An industrial strength platform. In: Multi-Agent Programming: Languages, Platforms and Applications, *Multiagent Systems, Artificial Societies, and Simulated Organizations*, vol. 15, pp. 175–193. Springer (2005)
59. Zeigler, B.P., Praehofer, H., Kim, T.G.: Theory of Modeling and Simulation, 2nd edn. Academic Press (2000)
60. Zhang, M., Verbraeck, A.: A composable PRS-based agent meta-model for multi-agent simulation using the DEVS framework. In: Symposium on Agent Directed Simulation, ADS'14, pp. 1:1–1:8 (2014)