Proceedings of the 2011 Winter Simulation Conference
S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, eds.

INTEGRATING BDI REASONING INTO AGENT BASED MODELING AND SIMULATION

Lin Padgham
David Scerri
Gaya Jayatilleke
Sarah Hickmott

School of Computer Science and Information Technology
RMIT University
GPO Box 2476
Melbourne 3001, VIC, Australia

ABSTRACT

Agent Based modeling (ABM) platforms such as Repast and its predecessors are popular for developing simulations to understand complex phenomenon and interactions. Such simulations are increasingly used as support tools for policy and planning. This work takes a Belief Desire Intention (BDI) agent platform and embeds it into Repast, to support more powerful modeling of human behavior. We describe the issues faced in integrating the two paradigms, and how we addressed these issues to leverage the relevant advantages of the two approaches for real world applications.

1 INTRODUCTION

Simulation using Agent Based modeling (ABM) is being increasingly used for exploring and supporting decision making about social science scenarios (e.g., Epstein 2006, Axelrod 1997). These simulations are typically built using toolkits such as Repast (North, Howe, Collier, and Vos 2007), where the agents are relatively simple entities, that respond reactively to things in the environment. These widely used toolkits typically provide a graphical development interface and a suite of tools to assist in analysis. This paradigm has been very successful in the ecological domain, but for social science simulations involving modeling of humans, it is not always straightforward to represent human-like behavior.

Belief, Desire, Intention (BDI)(Rao, Georgeff, Institute, Department of Industry, and Commerce 1991) agents on the other hand are based on a simplified psychological/philosophical view of how people behave, particularly in balancing between pro-active goal seeking behavior, and reactive response to the environment. BDI programming languages and platforms, such as JACK (Busetta, Rönnquist, Hodgson, and Lucas 1998), Jadex (Braubach, Pokahr, and Lamersdorf 2005) or Jason (Bordini and Hübner 2005) facilitate a high level specification of complex human behavior and have been demonstrated to be very efficient for building complex applications (Benfield, Hendrickson, and Galanti 2006).

In this work we integrate BDI agents using the commercial JACK platform (AOSGroup 2011) into the established simulation framework provided by Repast to provide an improved modeling capability for simulations involving some complex human behaviors. We allow the BDI system to represent (and execute) the agent's reasoning and decision making, while the acting within the environment, and the observations of the world state remain within the ABM simulation. We describe some of the issues that arise when integrating these two paradigms, and how we have resolved them to give us a platform suited to some of the real world requirements in areas such as modeling of emergency management in bushfire situations that we are working in. For example we want to model CFA (Country Fire Authority) agents that have a variety of specific plans that they use to attempt to influence the behavior of residents, depending on the details of the situation. Some of these plans incorporate a number of different steps with subgoals and alternative ways of achieving these. BDI modeling provides a natural and efficient approach to programming these agents.

To the authors' knowledge there is very limited work done on integrating an existing BDI platform with an existing ABM platform. There are several examples of work where cognitive reasoning agents are used in simulation, but this is done by building a customized extension to either the reasoning platform or the simulation platform, which doesn't have the same years of research and community support as an existing platform. The work in Bordini and Hübner (2009) provides a limited, custom simulation environment for BDI agents which lacks many of the features of a more developed simulation platform such as Repast. Parunak, Nielsen, Brueckner, and Alonso (2007) examines cases where it may be advantageous to combine *heavyweight agents*, which are cognitive agents capable of more advanced reasoning, and *lightweight agents*, which are simple agents which only react to their surroundings. Again, this is built into a custom solution, which lacks many of the standard features expected in an ABM platform. There are also many systems which incorporate cognitive agents into a black-box simulator, such as Paquet, Bernier, and Chaib-draa (2004) and Dastani, Dix, and Novák (2007). We are not aware of any work that integrates an existing reasoning system with an existing modeling and simulation platform in a way that leverages the features of both.

1.1 Example

BDI agents are programmed using *goals* (or events) representing what the agent wants to achieve or respond to, which then trigger *plans* describing (possibly at an abstract level), different ways to achieve these goals. Plans are then made up of sub-goals, which have associated plans, and *actions*. Agent *beliefs* are used to select which plan to instantiate in order to achieve a goal in a particular situation. The collection of goals and plans can then be represented as a goal-plan tree as shown in Figure 1.

This example shows a set of plans available for an agent to achieve the top level goal to get a present. Buying the present is a plan with subgoals to get some money, followed by the actions go to the shop, and purchase a present. The goal to get some money has two alternative plans (or rules for achieving this goal). The agent can go to the ATM and withdraw money, or go to a friend and borrow money. The preferred approach is to go to the ATM, but if this is not applicable (or fails) because there is no money, then an alternative plan is to go to a friend's house and borrow money. In selecting the plan to go to a friend's house, the agent must also determine a particular friend to go to. If this plan fails, then the same plan type(with different bindings) can be used to try a different friend.

We have chosen this example simply to illustrate a particular BDI agent, so that we can discuss the principled integration into Repast, with reference to a concrete example. An application will typically involve multiple agents and their interactions, on both the BDI side and the ABM side. However, as we are synchronizing the separate systems, these interactions will not be affected, and will play out within both the BDI system and the ABM system.

Some important aspects of the BDI representation are: (a) it is straightforward and natural, (b) pursuit of the goal occurs across multiple timesteps, but there is no need to explicitly manage where things are up to, and (c) if something fails there is a local attempt to recover using an alternative plan. While these can be achieved in Repast they must be specifically encoded by introducing data structures, counters, etc.

In our simulation of this example, the BDI agent has been integrated with a slightly modified, existing Repast model called RepastCity (Malleson 2011). This provides agents which are able to walk around a

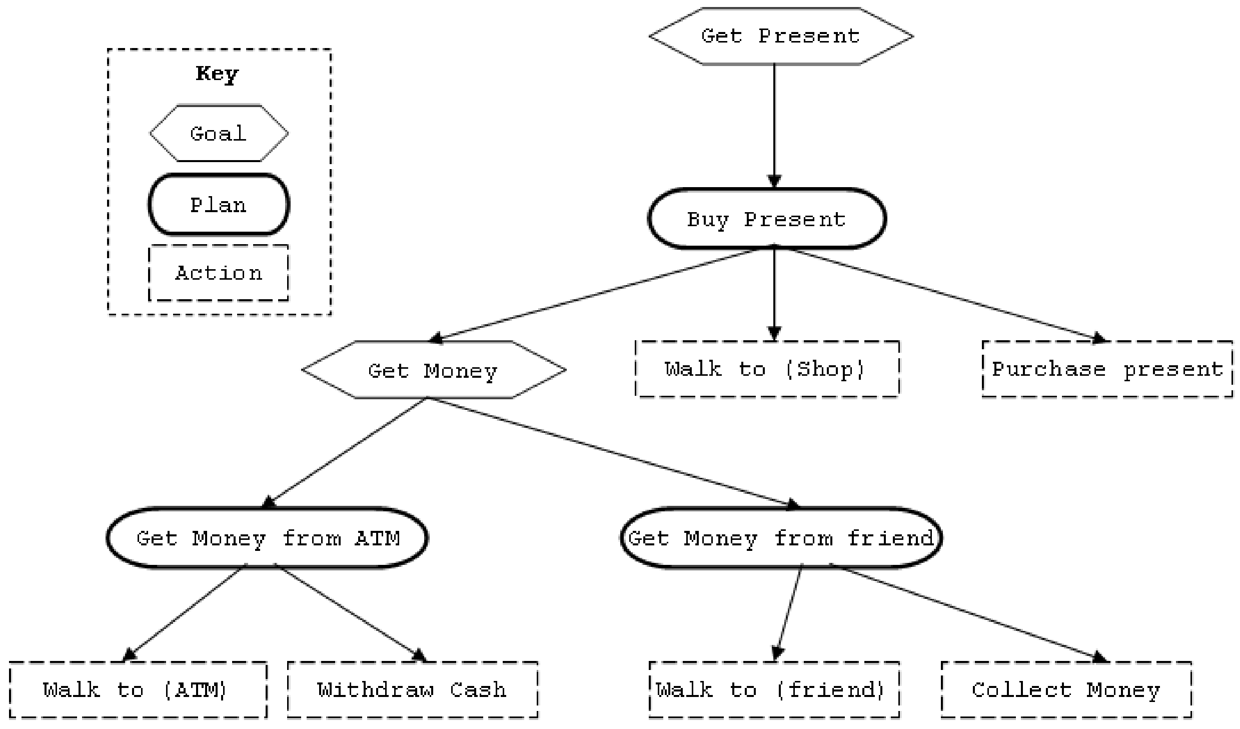

Figure 1: Goal-plan tree for Party Scenario

small city, going from a current location to a destination. We have added some stores, ATMs and houses at specific addresses.

We use this simple example to assist in describing the framework we have developed for connecting the BDI system JACK, with Repast. We explore particular issues around synchronization of the BDI system with the Repast system, in a way that will allow us to do reliable simulation and modeling. We also describe some important details of the percept-action interface we have developed between Repast and JACK. We close with an efficiency evaluation, discussion and future work.

2 FRAMEWORK OVERVIEW

Our simulation framework consists of the ABM model and platform, which includes representation of the environment, and the integrated BDI model which contains *reasoning* agents to provide high-level decision making. The ABM model contains *sensor-actuator* agents which handle low-level processing and direct execution of actions in the environment. Entities that are part of the overall simulation may require only a BDI representation (reasoning only), only an ABM representation (reactive only), or both. For example agents which are goal oriented and do reasoning tasks, but whose purpose is only to influence other agents, with no direct actions in the environment, may have only a BDI representation while entities which have relatively simple reactions at each timestep, but act in the environment, may exist in the ABM model only. Some agents however will require modeling in both the BDI and the ABM models and it is these agents which are the primary focus of this paper. We note also that our aim is to integrate an existing BDI platform into an existing ABM platform, not to implement a completely new modeling and simulation tool which has both aspects, which would perhaps be approached somewhat differently. Many years of research and development effort have gone into developing state of the art systems in both these areas, so our aim is to leverage these, largely as they are.

A standard part of modeling in BDI systems is to continually obtain information from the environment (via percepts) and to affect the environment (via actions). If the BDI system is a robotic system, the percepts may be obtained from sensors, and the actions may be executed via actuators. If it is an electronic system, then it is likely that both percepts and actions are some form of message or method call. When operating in a simulated environment it is natural for relevant events or observations in that environment to form the percepts, and for actions to be the way the BDI agent affects the simulated environment. ABM simulators however are generally closed systems, which do not take information from nor affect outside the simulation system.

To provide a communication interface between the BDI representation of an agent and its ABM counterpart we define what we call BDI actions and percepts.

- **BDI actions**. These are calls from the BDI reasoning agent, to its ABM sensor-actuator counterpart acting within the (simulated) environment. BDI actions may well require further processing by the sensor-actuator agent and may also involve ABM activity over multiple timesteps. The important point is that they are specified at the level at which the reasoning agent passes management of the action to its ABM counterpart.
- **BDI percepts**. These are inputs from the ABM system, to the BDI system, providing information perceived by sensor-actuator agents which should be noticed by their reasoning counterparts. The ABM system pro-actively notifies the BDI system of all generated percepts.

In addition we introduce a third interface component, which we call a *BDI sensing action*.

- **BDI sensing actions**. These provide the BDI system with on demand information from the ABM system, as needed for reasoning *within a particular timestep*. Typically these are used to access state information needed for choosing between different available plans. Clearly these could be modeled as percepts, sent whenever there is a relevant change in state, and maintained as beliefs in the BDI system. However, to minimize the number of percepts pushed from the ABM system, we provide on demand sensing actions.

Looking at Figure 1: when the BDI agent has determined that it will choose the action `go-to(ATM)`, this will be provided to the ABM system, and the BDI agent (if this is its only current intention) will be paused until that action has completed. The ABM system, using the existing RepastCity model, has a rule for moving an agent from its current location to a destination, using the city map that is the environment. This rule includes planning a route to the destination if one does not exist, storing it, and moving an appropriate distance along the route for the given timestep (Note that the route planning could be put in either the BDI model or the ABM model, and this is a design decision.)

Completion of the action `go-to(ATM)` may take multiple timesteps, depending on the distance to the destination. The BDI model will not continue execution (at least on this intention, and assuming no incoming percepts need to be addressed) until the ABM system completes the action.

We note here that it is possible for the action to fail in the ABM model. Although not modeled in the original version of RepastCity, a modified version which we used in a bushfire evacuation simulation, allowed for road blockages due to blockades or fire. If there is a blockage such that no new route can be planned to the destination then the ABM system will be unable to complete the action. In this case such information is passed back to the BDI model, which will fail the action. Similarly, there may be something that happens in the environment, conveyed to the BDI model via a percept, which causes the BDI agent to decide it no longer wishes to pursue its current action. This is then conveyed to the ABM model. In the following sections we explore the details of how these interactions between the two models are accomplished.

The BDI execution and the ABM execution each run in separate threads which need to be synchronized in some way. Because we are interested in simulation, rather than reasoning for control of a real-time system, we base synchronization on the ABM model timesteps. Essentially, at the end of each timestep, the BDI system is provided with any new percepts generated during that timestep. The BDI model is then allowed to run, sending across any desired actions (which will be run by the ABM model in its next timestep). When the BDI execution becomes idle, meaning that there is no further reasoning required until new percepts arrive, or requested actions are completed, then control is handed back to the ABM system which executes its next timestep. BDI reasoning happens only at the start of (or rather prior to) the ABM timestep. Conceptually the agents reason, then execute actions (possibly with some further processing), then reason again with whatever information has resulted from those actions, and so on. If there is no change to the action states, and no new percepts are generated, the BDI model will not run, which means that the ABM model may run many cycles to each cycle of the BDI model. We note that sensing actions, unlike the BDI actions which affect the environment, are sent and processed during the BDI execution, without halting the system. Sensing actions directly query the state of the ABM system and receive an immediate response.

The following two sections explore the details of how we have managed the interactions between the BDI and ABM systems around actions, percepts, sensing actions and time-stepping of the two systems in a synchronized manner.

3 INTERACTION

Here we describe in detail how our framework supports interaction between the ABM and BDI models, via actions, percepts and sensing actions. These structures are passed between the BDI and ABM models using a defined message passing protocol.

3.1 Actions

An *action* `act` is a tuple `<identifier, parameters, state>` where `identifier` is a unique name given to the action type, `parameters` are input variables for this action, and `state` is the current state of this action. Actions are the means by which a reasoning agent in the BDI model requests its sensor-actuator counterpart to affect the simulated environment in the ABM model and involve low level processing by the sensor-actuator agent.

Table 1: Action states.

State	Description
INITIATE	Initiated by BDI agent and to be executed
RUNNING	Being executed by the simulation agent
PASS	Completed as expected
FAIL	Aborted by the simulation agent
DROPPED	Aborted by the BDI agent
SUSPENDED	Suspended by the BDI agent

Both the BDI and ABM models maintain a list of actions which we refer to as the `actionList`. At any point in the simulation these lists will contain all instances of actions which are in the states described in Table 1.

When a reasoning agent wants its ABM counterpart to execute an action `act`, it sets its state to `INITIATE`, adds it to its own `actionList`, then monitors the action for any updates. In order to reduce the effort required by the modeler we have chosen to provide a generic goal-plan pair to support this process. We refer to these as the `actionGoal` and `actionPlan`. In order to initiate the execution of `act` the BDI agent posts the goal `actionGoal(act)` from within a plan body. This goal is handled by the plan `actionPlan`, which does the following:

1. Add `act` to `actionList` with `state=INITIATE`, which triggers a message to be sent to the ABM counterpart;
2. If `act` is updated, either due to a message from the ABM counterpart, or a decision by the BDI agent, respond as follows:
   - if `act.state==PASS` then finish, indicate that the goal `actionGoal` was achieved, i.e. `act` was executed successfully, and remove `act` from `actionList`.
   - if `act.state==FAIL` or `DROPPED` then finish, indicate that the goal `actionGoal` failed, i.e. `act` was not executed successfully, and remove `act` from `actionList`.
3. If `MaintenanceViolation` or `ParallelException` then set status of `act` to `DROPPED` (triggering a message to the ABM agent and removal of `act` from the `actionList`.

The `MaintenanceViolation` and `ParallelException` referred to here are discussed below in the context of dropped actions. In our implementation the `actionPlan` provides a consistent and generic way to (1) monitor and respond to changes in the state of an action (2) handle any exceptions arising elsewhere in the BDI program, requiring dropping of an action, e.g. `MaintenanceViolation`. The one-place specification of `actionPlan` facilitates extending the framework with new action states and implementing any additional control mechanisms required by the BDI model as deemed necessary.

The ABM model also maintains an `actionList` which is updated as relevant messages are received from the BDI model. In general, an ABM agent contains a series of single timestep rules with execution schedules or trigger conditions defining at which timesteps they should run. If the modeler requires an action to be executed over multiple timesteps then (s)he must construct rules such that the sensor-actuator agent will perform an appropriate portion of the action. For example if the action is to move to a given destination, the rule will move some distance towards the destination 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 the ABM agent to execute the actions specified in its `actionList` we provide a new rule, executed at each timestep, which operates as follows:

1. For each action in the `INITIATE` state, read the action parameters and change the state to `RUNNING`;
2. For each action in the `DROPPED` state, remove from the `actionList`
3. For each action in the `RUNNING` state, call the appropriate rule to take one (time)step towards the action's completion.

Note that an action will begin execution in the same timestep that its state is set to `RUNNING`. When an action successfully completes, or fails entirely, the sensor-actuator agent changes the state of the action to `PASS` or `FAIL` respectively to trigger the BDI agent response as described, and then removes the action from its `actionList`.

3.1.1 Synchronous and Asynchronous Actions

BDI systems support synchronous and asynchronous generation of goals, where a *synchronous* goal is a subtask that must complete before the next step is taken in the current intention, whilst an *asynchronous* goal starts a new intention which executes concurrently with the generating intention. However there is often no concept of synchronous or asynchronous *actions* in BDI systems, since actions are usually regarded as instantaneous. In our (combined) simulation system however, where actions may take time, it makes sense to be able to generate actions both synchronously and asynchronously. We have made synchronous actions the default, with a plan simply suspending until the action completes. However, if the developer wants the agent to initiate an action and continue with its current intention, then this is straightforward to achieve. As described earlier, a BDI reasoning agent executes an action `act` by generating a goal `actionGoal(act)`; if `act` is an asynchronous action, we can simply post `actionGoal(act)` as an asynchronous goal. In this case a new intention will be started, containing only the plan that passes the action to the ABM agent, and will continue concurrently with the original intention thread. When the action finishes this new intention will finish.

3.1.2 Dropping Actions

There are a variety of reasons that the BDI reasoning agent may decide that it wishes to drop actions that are in the process of being executed by the sensor-actuator counterpart. It may be the case that a condition under which a particular intention or partial intention is being pursued is violated, or that there is some alternative action the agent wishes to pursue which conflicts with a currently executing action, or some other reason for aborting. A common programming construct used in agent systems for supporting aborting under some condition, is a *maintenance condition*, which if it becomes false, will fail the plan contained within. In JACK this triggers a Java exception called `MaintenanceViolation` that is propagated to all subsidiary plans. We catch this exception within `actionPlan` and remove the associated action from the action list. Another typical reason for aborting actions is if there are parallel sub-goals being pursued, and the programmer has indicated that if one parallel clause fails, all should fail. This situation also generates in JACK an exception, called `ParallelException`, which is handled in the same way as the maintenance violation exception. An arbitrary plan may also reason that a currently executing action should be aborted, perhaps because a higher priority conflicting action needs to execute. In this case the plan making this decision can modify the state of the action in the action list to `DROPPED`. This will be observed by the `actionPlan`, which will then remove it and fail. We also provide an additional flexibility of allowing actions to be `SUSPENDED` by the BDI agent. In some situations, the BDI agent may prefer to simply suspend a currently executing action, returning to it when the reason for suspension changes. This is accomplished by modifying the state of the action to `SUSPENDED`, causing the ABM sensor-actuator to ignore it, without alerting the relevant `actionPlan` on the BDI side. When the reason for suspension has passed, the plan setting `SUSPENDED` simply sets the state back to `RUNNING` and the sensor-actuator will then continue its execution. This provides a useful mechanism for the BDI reasoning agent to reason about potential conflicts at the level of actions, based perhaps on declarative information regarding which actions are conflicting.

3.2 Percepts

A *BDI percept* is a tuple `<identifier, value>` where `identifier` is a unique name given to the percept and `value` is a (possibly multi-valued) object. Percepts are provided to a reasoning agent by its sensor-actuator partner in response to a change in the environment detected by the sensor. Percepts are not specifically requested by the reasoning agent during simulation, this is decided at design-time.

During each timestep, the ABM agent senses changes in the environment, generates relevant percepts, and passes them to the BDI system as messages. The incoming messages are stored in a message queue at the BDI end in the same order they arrive. At the conclusion of the ABM timestep, the BDI system processes the messages in the queue and pushes them to the relevant BDI agents for processing.

Consider again the running example and the possibility of extending this to enable the BDI agent to receive phonecalls. This could be implemented by the sensor-actuator agent sensing a phonecall and generating <`'phonecall', 'Joe'`>. At the end of the timestep, the BDI agent would process the percept and may decide to answer the phonecall by asking the sensor-actuator agent to execute the action <`'talk', 'Joe'`> at the next timestep.

We do not place any restrictions on the format of percepts, requiring only that both the ABM agent and the BDI agent manage the relevant structure. It is a standard aspect of BDI modeling to respond to percepts either by updating beliefs, or by selecting a plan.

3.3 Sensing Actions

A *BDI sensing action* is a tuple `<identifier, parameters, value>`, where `identifier` is a unique name given to the sensing action, `parameters` are the input variables required by the action, and `value` is the requested information. Sensing is a conceptually instantaneous action which provides a reasoning agent with on-demand information about the environment, via its sensor-actuator agent partner. A sensing action has no effect on the environment but may involve some low level processing by the sensor-actuator agent. In our example, to allow the BDI agent to reason about which friend to (attempt to) borrow money from based on his current location it could send the ABM agent `<‘location’, [self], coordinate>`. The ABM agent would reply with `coordinate` set to the agent’s current location.

4 SYNCHRONIZATION

As discussed, our aim is to integrate BDI and ABM models in a manner that respects these paradigms, leverages existing platforms for each, and requires minimal changes to any existing ABM model that we want to extend by adding BDI reasoning. One of the issues that arises from this integration is how to synchronize the two models, given the differences in the execution style. A typical BDI agent is event-driven and does not act on a timestep basis. ABM systems on the other hand are generally timestep driven; the modeling paradigm requires that all agents are synchronized at each timestep, in order to depict a realistic progression of the simulated environment. For example, if during a simulation two ABM agents are each executing actions to walk from location A to location B, at different speeds, it is important that the simulation advances the state of the system accurately at each timestep to reflect this difference. However in most BDI based systems, the synchronization occurs at the end of each action or at points marked by pre-defined events, e.g. when the agents have already reached location B.

These different execution styles are managed by running the ABM and BDI models separately and sequentially, where one simulation timestep consists of the BDI model executing until idle, followed by the ABM model executing one ABM timestep. The following rules then ensure synchronization of the BDI and ABM model executions:

1. When the ABM timestep is being performed, the BDI model must remain idle;
2. Both systems put all messages (changes to an action state, new actions or percepts) into an outgoing message queue, and send them as a package at the end of their execution cycle;
3. Messages are processed in the same order as they were placed in the queue;
4. The next ABM timestep can not be started until the BDI; model execution is idle pending an action state change or a new percept; and
5. Sensing actions may be performed outside of the ABM timestep, within the BDI execution cycle by being sent directly, without waiting in the outgoing message queue.

This approach to synchronization does however introduce the limitation that the BDI model can not perform more than one sequential action in a timestep. (Multiple parallel actions are of course possible.) This suggests that the modeler should consider making the simulation timestep duration as short as the smallest BDI action. Note that this limitation is somewhat reduced by the sensing action mechanism, which provides BDI agents with instantaneous access to information held by the ABM model during BDI execution.

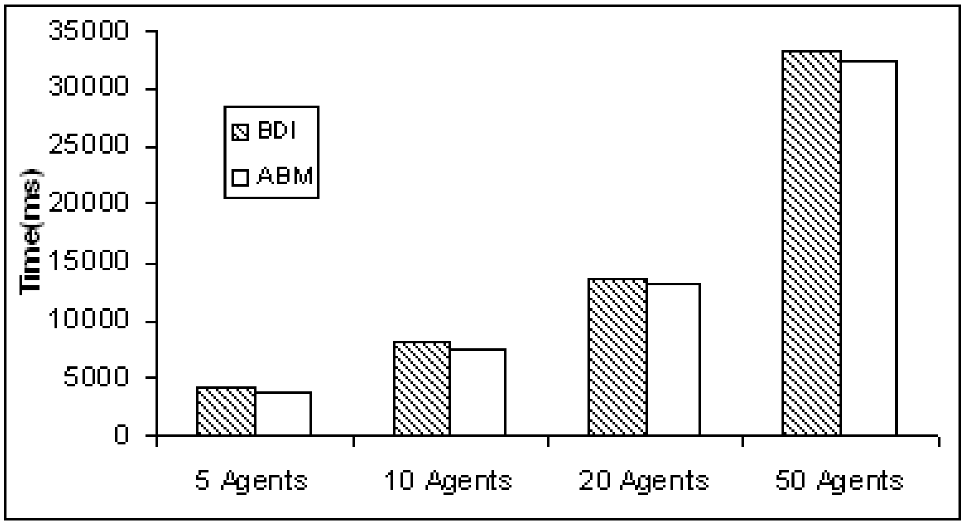

Figure 2: Performance of BDI integrated system.

The order in which events occur may be important and should be maintained. For example if an agent's action succeeds because of a change in the environment, it is important that its reasoning counterpart processes the environment change via a percept before it processes the action's success. This is facilitated by ensuring that when the message list is sent, the order they were added is maintained, and they are then processed in the same order. Thus the BDI simulation loop involves: processing each of the messages in the message queue; waiting for all the BDI agents to become idle; then sending any action changes or new actions to the ABM. The ABM simulation loop involves: receiving message list of new actions or changes; running one (time)step of the ABM model, storing percepts and actions changes in outgoing list; then sending outgoing list to BDI agent.

4.1 Identifying the BDI Execution as Idle

In order to know when to execute the next ABM step, we need to identify when all the BDI agents have finished reasoning and are waiting for input from the next ABM cycle in the form of action state change or percepts. The manner in which this control is implemented will depend on the BDI system used, particularly the available access to the BDI execution cycle. The approach described here in the context of JACK however does address some more general issues.

JACK does not provide access to the execution cycle, but allows agents to be monitored via the agent method `isIdle()`. Thus by monitoring each agent, and maintaining a list of active agents, we can determine when the entire BDI executor is idle. It is possible however to detect false positives where an agent becomes idle due to something external to the system as opposed to waiting for a percept or action update. We use a counter to keep track of the plans which are idle but not waiting for an action to complete nor an incoming percept.

Another instance where a false positive idle state can be detected is when sending messages between agents. In JACK messages are sent asynchronously where a sending agent can become idle before an already idle receiver agent reacts to the message. In order to prevent the system going into an idle state in this manner we provide a synchronous handshaking mechanism for message passing between agents.

5 EVALUATION

To evaluate performance overhead of the BDI integration, we compared performance of the original RepastCity model with the BDI integrated model. In the original model agents move randomly from one destination to the next. In our integrated evaluation, the BDI portion of the agent receives a percept when the agent has nowhere to go (either because the simulation has just begun or because it has reached its destination). The BDI portion then performs some arbitrary reasoning, which involves progressing through a plan hierarchy of depth and width 4, evaluating context conditions to select plans. Each of the leaf plans sends the sensor-actuator an action with a new, random destination. We ran each system for 1000 timesteps on a Dual Core 3.16Ghz machine running Windows XP and recorded the time taken. Each test was repeated 10 times and the average taken. Figure 2 shows that having the BDI integration adds a very small constant overhead to the system and does not significantly reduce performance. As the overhead is not affected by number of agents we conclude it is due to the messages between systems, rather than the agent reasoning.

6 CONCLUSIONS AND FUTURE WORK

In this paper we have described the successful integration of the two popular but different paradigms of BDI programming and Agent Based modeling, accomplished by interfacing a state of the art system of each type. The motivation for this is to allow more precise modeling and simulation of real-world systems in order to inform areas such as policy development and enhance theoretical understanding of social phenomena.

ABM platforms provide support for ease of modeling by domain experts by including sophisticated tools. However specifying agents with complex reasoning over multiple timesteps, with varying levels of reactive and goal directed behavior, can be difficult. BDI programs are well suited to capturing human behavior/reasoning which balances goal directed activity over time, with reactivity to the environment.

In order to cleanly integrate the two paradigms it was necessary to address some specific issues regarding synchronization of the two systems, and how to interface between the BDI system and the ABM environment. To synchronize the two systems we used the ABM timestep as the basic unit and employed interleaved execution cycles between the two systems. We first allow the BDI system to reason about what to do, pausing it when there is currently no further reasoning to do. We then allow the ABM system to execute a timestep, including doing whatever actions have been determined by the BDI system. We have defined an interface between a BDI reasoning agent and its sensor-actuator counterpart, consisting of actions (as requested by the BDI reasoner), percepts (as generated within the ABM system) and sensing actions (on demand observations of the state of the ABM system). We use message passing between the two systems to notify of actions and percepts.

To our knowledge this is the first work that explores in detail, and resolves the issues inherent in combining these two paradigms, using existing platforms. If it is to be possible to develop complex simulations in a modular and incremental manner, this is an important step. In future work we plan to develop a graphical interface for defining the BDI reasoning agents that integrates seamlessly with ABM graphical interfaces used by modelers within systems such as Repast.

REFERENCES

AOSGroup 2011. “JACK”. Accessed March 30, 2011. http://www.aosgrp.com/.

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

Benfield, S. S., J. Hendrickson, and D. Galanti. 2006. "Making a strong business case for multiagent technology". In *Proc. of AAMAS'06*, edited by G. W. Hideyuki Nakashima, Michael Wellman and P. Stone, 10–15.

Bordini, R. H., and J. F. Hübner. 2005. "BDI Agent Programming in AgentSpeak Using Jason". In *CLIMA VI*, edited by F. Toni and P. Torroni, 143–164: Springer.

Bordini, R. H., and J. F. Hübner. 2009. "Agent-Based Simulation Using BDI Programming in Jason". In *Multi-Agent Systems: Simulation and Applications*, 451–471. CRC Press.

Braubach, L., A. Pokahr, and W. Lamersdorf. 2005. "Jadex: A BDI Agent System Combining Middleware and Reasoning". In *Software Agent-Based Applications, Platforms and Development Kits*, edited by M. C. Rainer Unland, Matthias Klusch, 143–168.

Busetta, P., R. Rönnquist, A. Hodgson, and A. Lucas. 1998. "JACK Intelligent Agents - Components for Intelligent Agents in Java". Technical report, AOS Pty. Ltd, Melbourne, Australia.

Dastani, M., J. Dix, and P. Novák. 2007. "The second contest on multi-agent systems based on computational logic". *Computational Logic in Multi-Agent Systems* 2:266–283.

Epstein, J. 2006. *Generative Social Science - Studies in Agent-Based Computational Modeling*. Princeton University Press.

N. Malleson 2011. "RepastCity". Accessed March 30, 2011. http://portal.ncess.ac.uk/access/wiki/site/mass/repastcity.html.

North, M., T. Howe, N. Collier, and J. Vos. 2007. "A Declarative Model Assembly Infrastructure for Verification and Validation". In *Advancing Social Simulation: The First World Congress*, edited by D. S. S. Takahashi and J. Rouchier, 129–140: Springer.

Paquet, S., N. Bernier, and B. Chaib-draa. 2004. "DAMAS-Rescue description paper". *RoboCup-2004: Robot Soccer World Cup VIII* 3276:12.

Parunak, H. V. D., P. Nielsen, S. Brueckner, and R. Alonso. 2007. "Hybrid multi-agent systems: integrating swarming and BDI agents". In *ESOA'06*, edited by M. J. Sven A. Brueckner, Salima Hassas and D. Yamins, 1–14.

Rao, A., M. Georgeff, A. A. I. Institute, T. Department of Industry, and A. Commerce. 1991. *Modeling rational agents within a BDI-architecture*. Australian Artificial Intelligence Institute.

AUTHOR BIOGRAPHIES

LIN PADGHAM is Professor of Artificial Intelligence in the School of Computer Science and I.T. at RMIT University, Melbourne, Australia. She has a PhD from University of Linkoping, Sweden, 1989. Lin's research interests are in modeling, building and understanding intelligent agents for complex application areas requiring a balance between goal directed long-term behavior and reactive response to a dynamic environment. Her email address is lin.padgham@rmit.edu.au.

DAVID SCERRI is a PhD candidate in the Intelligent Systems group at RMIT University, Australia. He received a Bachelor in Computer Science at RMIT University and is currently researching the validation and analysis of Agent Based Models. His email address is david.scerri@rmit.edu.au.

GAYA JAYATILLEKE is a post doc in the School of Computer Science and IT at RMIT University, Australia. He received his doctorate in Computer Science from RMIT University. His research interests are in agent oriented software engineering and model driven development. His email address is gaya.jayatilleke@rmit.edu.au.

SARAH HICKMOTT is a post doc in the Intelligent Systems group at the RMIT, Australia. She received a Doctorate in Engineering from the University of Adelaide and National ICT Australia. She is interested in supporting decision making around sustainable futures and climate adaptation, with the use of agent oriented modeling and simulation, and automated planning. Her email address is sarah.hickmott@rmit.edu.au.