Going beyond BDI for agent-based simulation

Larsen, John Bruntse

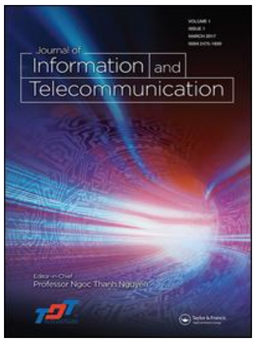
Published in:
Journal of Information and Telecommunication

Link to article, DOI:
10.1080/24751839.2019.1620024

Publication date:
2019

Document Version
Publisher's PDF, also known as Version of record

Citation (APA):
Larsen, J. B. (Accepted/In press). Going beyond BDI for agent-based simulation. Journal of Information and Telecommunication, 1-19. https://doi.org/10.1080/24751839.2019.1620024



Journal of Information and Telecommunication

ISSN: 2475-1839 (Print) 2475-1847 (Online) Journal homepage: https://www.tandfonline.com/loi/tjit20

Going beyond BDI for agent-based simulation

John Bruntse Larsen

To cite this article: John Bruntse Larsen (2019): Going beyond BDI for agent-based simulation, Journal of Information and Telecommunication, DOI: 10.1080/24751839.2019.1620024

To link to this article: https://doi.org/10.1080/24751839.2019.1620024

Published online: 03 Jul 2019.

JOURNAL OF INFORMATION AND TELECOMMUNICATION
https://doi.org/10.1080/24751839.2019.1620024

Going beyond BDI for agent-based simulation

John Bruntse Larsen

DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark

ABSTRACT
Research in multi-agent systems has resulted in agent programming languages and logics that are used as a foundation for engineering multi-agent systems. Research includes reusable agent programming platforms for engineering agent systems with environments, agent behaviour, communication protocols and social behaviour, and work on verification. Agent-based simulation is an approach for simulation that also uses the notion of agents. Although agent programming languages and logics are much less used in agent-based simulation, there are successful examples with agents designed according to the BDI paradigm, and work that combines agent-based simulation platforms with agent programming platforms. This paper analyses and evaluates benefits of using agent programming languages and logics for agent-based simulation. In particular, the paper considers the use of agent programming languages and logics in a case study of simulating emergency care units.

ARTICLE HISTORY
Received 30 June 2018
Accepted 14 May 2019

KEYWORDS
Multi-agent systems; logic; simulation

1. Introduction

Agent-Oriented Programming (AOP) is a programming paradigm where programs are composed of agents. Similar to objects in Object-Oriented Programming (OOP), agents maintain a mental state and react to input by performing actions and changing their mental state. Some agents are also assumed to be intelligent agents, meaning that they pursue goals and exhibit social behaviour by communicating with other agents. Agent programming languages are programming languages that are designed for development of multi-agent systems with AOP. Examples of platforms that use agent programming languages include Agent-0 (Shoham, 1993), 3APL (Hindriks, De Boer, Van Der Hoek, & Meyer, 1999), 2APL (Dastani, 2008), Jason (Bordini, Hübner, & Wooldridge, 2007), JACK (Busetta, Ronnquist, Hodgson, & Lucas, 1999; Winikoff, 2005) and GOAL (Hindriks, 2009). The notions of belief, desire and intention (BDI) are key components in these languages, as they respectively denote what the agent believes, what the agent would like to achieve, and what the agent is currently working towards achieving. Formalizations of a BDI model in modal logics provide syntax and semantics for the model. Thus logic provides a theoretic framework for specification and verification of agent programs. In particular, work in the AOP community has resulted in frameworks and meta-models for Multi-Agent-Oriented Programming (MAOP).

CONTACT John Bruntse Larsen jobla@dtu.dk DTU Compute, Technical University of Denmark, 2800 Kongens Lyngby, Denmark

The BDI paradigm has also been used in agent-based simulation (ABS). The purpose of ABS compared to multi-agent systems is to gain insight into how global properties emerge from a system of local interacting processes. Examples of ABS platforms include Mason (Luke, Cioffi-Revilla, Panait, Sullivan, & Balan, 2005), Repast (North et al., 2013) and GAMA (Amouroux, Chu, Boucher, & Drogoul, 2009). ABS platforms generally do not use above mentioned agent programming languages but some of them provides a framework for making models with the BDI paradigm (Kravari & Bassiliades, 2015). A BDI model allows agents to exhibit more complex behaviour than purely reactive models but without the computational overhead of cognitive architectures. It is generally also easier for domain experts to specify their knowledge in terms of a BDI model compared to an equations-based model, and a BDI model supports explainable behaviour. Adam and Gaudou (2016) present an extensive analysis and evaluation of approaches to integrating BDI models in ABS. They highlight the previously mentioned benefits of BDI models as a way to implement *descriptive agents* which use richer and more complex models than reactive agents.

This paper is an extension of Larsen (2018) presented at ACIIDS 2018, which presents an analysis and evaluation of using recent advances in agent programming languages and logics, in particular frameworks for implementing social behaviour, in ABS. Our objective is to highlight inherent limitations of using BDI for social simulation that can be adressed by using the AORTA framework for organizational reasoning. Compared to the original paper, we have extended the case study with an implementation of a BDI-based ABS for a case study with emergency care. In order to achieve our objective, we include code for the implementation. We evaluate the implementation and discuss how it could be improved with some of the frameworks and meta-models we discuss in this paper. The paper first presents a summary of AOP, ABS platforms and work on integrating BDI models in simulation platforms based on Adam and Gaudou (2016). It then describes research in frameworks and meta-models for implementing virtual environments and social behaviour in agent programming languages. It then presents the case study mentioned earlier and finally discusses further use of MAOP in ABS. The criteria used in the evaluation are in terms of:

(1) How the framework supports descriptive agents.
(2) How reusable the framework or meta-model is.
(3) How useful the framework or meta-model is for analysis.

We have chosen these criteria in order to evaluate from different perspectives that we consider important in ABS: a modelling perspective, a software engineering perspective and a simulation perspective. The evaluation is based on previous work on using the agent organization framework AORTA (Jensen & Dignum, 2014) to create a simulation model for an emergency care unit (Larsen & Villadsen, 2017).

2. AOP, logic and agent-based simulation

AOP was originally proposed by Shoham (1993) as a specialization of OOP. Shoham motivated AOP with cases in which multiple entities interacted with each other in order to manufacture cars and reserve plane tickets. In AOP, each entity (now called an agent) maintains a mental state of beliefs, capabilities and decisions that have dedicated terms with a formal syntax. Communication with other agents occurs through speech-act inspired messages. Some of the approaches to programming languages designed for AOP include:

- AgentSpeak(L) (Rao, 1996) in which an agent has a database of plans or rules for choosing actions that match its current mental state. The agent programming platform Jason (Bordini et al., 2007) implements AgentSpeak(L).
- Languages based on logic programming such as 3APL (Hindriks et al., 1999), 2APL (Dastani, 2008), and GOAL (Hindriks, 2009).
- Jack (Busetta et al., 1999; Winikoff, 2005) which extends Java with agent programming keywords.
- A combination of XML and Java. This approach is used in the agent programming platform Jadex (Pokahr, Braubach, & Lamersdorf, 2005).

These programming languages use BDI as a common paradigm for a mental model but as it can be seen, they have very different approaches to implementing it. The BDI paradigm comes from philosophy and the mental model can be given formal syntax and semantics with epistemic logics. Other logics such as first-order logic and temporal logics can be used to specify world models of concepts and dynamics. Given a specification it is then possible to use logic reasoning to verify properties of the specification. Thus logic provides a theoretical framework for specifying and verifying properties of agent programs. In the programming languages AgentSpeak(L), 3APL, 2APL and GOAL, the agents also use logic to do reasoning in their decision making.

ABS is an approach to simulation that takes the perspective of the individuals that inhabit the simulated system. ABS is useful in cases where it is easier to describe a system in terms of interacting agents rather than as a global process (Siebers, Macal, Garnett, Buxton, & Pidd, 2010). A critical part of ABS is a scheduling mechanism which ensures that all agents are synchronized in a finite sequence of time steps. ABS platforms provide frameworks for ABS and typically features tools for visualizing the simulation, data extraction tools and analysis tools. Commonly used ABS platforms include:

- Mason (Luke et al., 2005) which is a Java-based discrete-event simulation platform that has been extended with ABS.
- Repast (North et al., 2013) which is a suite of tools in multiple programming languages for implementing ABS.
- GAMA (Amouroux et al., 2009) which features an XML based language GAML for implementing agents. GAMA also features tools for using GIS data in the simulation.

The ABS platforms typically have tools for implementing reactive agents but little support for implementing proactive behaviour. This works well for many cases but as argued by Adam and Gaudou, there are also cases, often those involving human agents, where more descriptive agent models are useful for gaining insight into the decision making. The BDI paradigm provides a framework for implementing descriptive agents that are still fairly efficient. There have been three general approaches to implementing BDI in ABS:

- Extending agent programming platforms with ABS features. Bordini and Hübner (2009) does this with Jason.
- Extending ABS platforms with BDI modelling features. Caballero, Botia, and Gomez-Skarmeta (2011) does this with Mason .
- Combining ABS platforms with agent programming platforms. Padgham, Scerri, Jayatilleke, and Hickmott (2011) does this with Repast and JACK, and Singh, Padgham, and Logan (2016) designs a framework for integrating any two platforms with each other.

The benefit of the last approach is that it leverages features from both platforms but with a cost of computational power in keeping agents synchronized between the platforms. Besides mental models for the individual agents, there is also work on implementing meta-models for the environment and social behaviour such as the MASQ meta-model by Dignum, Tranier, and Dignum (2010). Meta-models for environments and social behaviour are covered further in the following section.

3. From AOP to MAOP

Much of the early research in agent programming languages has been focused on the internal agent architectures with different approaches to programming languages based on the BDI paradigm and speech-act communication. Both the environment that the agents inhabit and the social skills of the agents have been designed and programmed for specific domains. Recent research has gone into making more reusable frameworks and meta-models for creating environments and agent societies (van Riemsdijk, 2012; Weiss, 2013). Notable examples include:

- CArtAgO (Common Artefact Infrastructure for Agent Open environment) (Ricci, Viroli, & Omicini, 2006) which is a Java-based framework for developing and running virtual environments based on the Agents & Artefacts meta-model. In this meta-model, the agents use artefacts to communicate with other rather than only by speech-acts. The artefacts provide an interface for the communication that allows for also non-BDI agents to communicate with BDI agents. The framework has been integrated with Jason, 2APL and JADEX (Piunti, Ricci, Braubach, & Pokahr, 2008; Ricci et al., 2008).
- EIS (Environment Interface Standard) (Behrens, Hindriks, & Dix, 2011) which is a Java-based framework for connecting agent programming platforms with environments. It is not a meta-model for environments but it acts as an interface for agent programming platforms to environment platforms such as CArtAgO-based platforms.
- OperA (Dignum, 2004) which is a meta-model for agent organizations. In agent organizations, the agents are assigned roles that puts a structure on how the agents can use their abilities to communicate and carry out actions. The Eclipse plugin Operetta (Aldewereld & Dignum, 2011) is a tool for design, verification and simulation of OperA models.
- $\mathcal{M}$OISE$^+$ (Hübner, Sichman, & Boissier, 2007) which is also a meta-model for implementing agent organizations. $\mathcal{M}$OISE$^+$ is integrated with CArtAgO and Jason in the JaCaMo platform (Hübner, Boissier, Kitio, & Ricci, 2010).
- AORTA (Jensen & Dignum, 2014; Jensen, Dignum, & Villadsen, 2017) which is a meta-model that enables individual agents to reason about organizations described in

OperA. It is designed for adding organizational reasoning capabilities to BDI agents and has been integrated with Jason (Jensen, Dignum, & Villadsen, 2014).

Table 1 summarizes the main characteristics and advantages of these frameworks and meta-models. A common feature of the examples is that they support more open heterogeneous systems of agents: agents can enter and exit the system freely even though they use different internal mechanisms for decision making. The frameworks and meta-models put an emphasis on Multi-Agent-Oriented Programming (MAOP) with system level frameworks rather than traditional AOP with agent-level mental models and speech-act communication. Use of MAOP is not common in ABS literature. A possible reason for this might be that openness is less important in simulation where the purpose is to gain insight in a given system. A potential benefit of MAOP though is that it can offer reusable tools for implementing environments and social behaviour. Using MAOP with a foundation in logic would also allow for specification and validation of simulation models similar to the work presented by Jensen (2015) on verification of organization-aware agents in AORTA.

4. Case study: emergency care units

To illustrate the need for going beyond BDI models in agent-based simulation, we describe, analyse and evaluate a simulation of emergency care units we have implemented by using the BDI framework of the ABS platform GAMA. Emergency care units are responsible of providing care to acute patients. Hospitals often have an entire department dedicated to emergency care and the number of incoming patients has been increasing in recent years. Simulation could assist management staff in the decision making by computing expected outcomes of the decisions.

4.1. Scope of the implementation

For our simulation we follow the description of emergency care from Mercuur, Larsen, and Dignum (2018) as a base for the implemented model of an emergency care unit. The model is based on observations from a real hospital but is still very simple compared to reality. We limit the model to cover only the parts ending with the *triage* but as we shall see, these parts already necessitate a careful agent design and implementation with the BDI paradigm. We implement the following process:

(1) A patient arrives at the emergency care unit with some symptoms.
(2) The patient waits in a designated waiting area until retrieved by a nurse.
(3) The nurse performs triage on the patient.
(4) The patient leaves the care unit.

Table 1. Summary of main characteristics and advantages of MAOP frameworks and meta-models.

Framework/meta-model	Main characteristic	Main advantage
CArtAgO	Virtual environments	Integrated with Jason
EIS	AOP platform/environment interface	Integrated with Jason
OperA	Agent organizations	Implemented in Operetta
$\mathcal{M}$OISE$^+$	Agent organizations	Integrated with Jason
AORTA	Agent organizations	Formally extends BDI reasoning

The treatment would continue after step 3 but these parts are outside the scope of our model. Depending on what kind of symptoms the patient has, the triage is carried out by one of three teams: damage team, medical team or surgical team. The choice of team is made by an overall head nurse, and the choice of nurse is made by a team head nurse.

4.2. GAMA BDI framework

GAMA is an agent-based simulation platform that is originally designed for implementing reflexive simulation agents using the GAML language (Caillou, Gaudou, Grignard, Truong, & Taillandier, 2017; Taillandier, Bourgais, Caillou, Adam, & Gaudou, 2017). The *simple-bdi* module extends agents with BDI-based behaviour and is designed with efficiency and ease-of-use for simulation creators in mind. As such it is in the category of ABS platforms that have been extended with BDI modelling features. We chose to use GAMA, as the BDI framework is developed actively and allows us to easily create a visual representation of the emergency care simulation. The main features of the platform and the *simple-bdi* module we used were:

- Graphs – GAMA features import of geodata that describe a network of polygons connected by lines, which are then transformed into a graph structure that agents can move along. We use a graph to model the environment.
- Perception – An agent has a set of **Perception** statements that defines which parts of the environment the agent perceives and under what conditions. We implement most the belief updates as **Perception** statements.
- Rules – An agent has a set of **Rule** statements that is used for revising beliefs, desires and intentions of the agent. We use rules to revise the desires of an agent when it gains a new belief.
- Agent properties – An agent can have properties with values, similar to that of an object. We use properties to model things we assume known such as what team a nurse belongs to.

The next sections describe the implementation in more detail. We write … to denote passages of code omitted to save space.

4.3. Implementing the environment

The agents are situated in a virtual environment that represents the different parts of the emergency care unit. We create a simple graph-based model where the environment consists of areas connected by hallways. The benefit of using a graph-based model is that the areas and hallways can be represented by nodes and edges in a straight-forward manner. Figure 1 shows the emergency care environment. It consists of eight areas:

- An entrance/exit area for patients
- A waiting area for patients assigned to the damage team.
- A waiting area for patients assigned to the medical team or the surgical team.
- A triage area for patients assigned to the damage team.
- A triage area for patients assigned to the medical team or the surgical team.
- An office for each of the three teams.

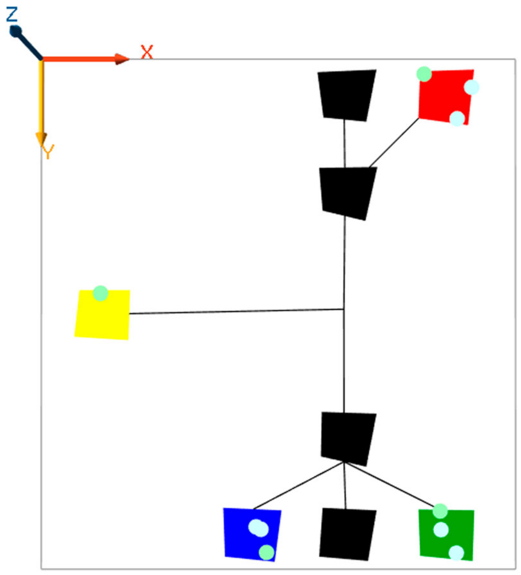


Figure 1. The emergency care unit simulation environment consists of eight areas. The entrance/exit (yellow), damage team (red), medical team (blue), surgical team (green), and waiting areas and triage rooms (black).

We encode the environment as three shapefiles: areas, hallways and a boundary file.

```
global {
   file shape_file_areas <- file("Areas.shp");
   file shape_file_routes <- file("Routes.shp");
   file shape_file_bounds <- file("Bounds.shp");
   geometry shape <- envelope(shape_file_bounds);
   graph the_graph;
   …
   init {
      create area from: shape_file_areas with: [name::string(read("name"))];
      create route from: shape_file_routes ;
      the_graph <- as_edge_graph(route);
      …
      }
   …
   }
}

species area
   {
   …
}
```

```
species route {
   ...
}
```

4.4. Implementing basic agent code

We continue with the implementation of the agents. In this section we focus on the code that implements basic abilities of the agents. We define the following helper functions to handle steps of the treatment process. The functions `finishPlan` and `focusSubintention` are used to respectively execution of a plan and start working on subgoals. The `believes` function is used to check if an has a given fact in its belief base.

```
action finishPlan {
      do remove_belief(get_current_intention());
      do remove_intention(get_current_intention(),true);
}

action focusSubintention(string predName, map args) {
      predicate pred <- new_predicate(predName,args);
      do add_subintention(get_current_intention(), pred, true);
      do current_intention_on_hold();
}

bool believes(string pred, map args) {
      list<predicate> preds <- get_beliefs_with_name(pred);
      return preds first_with (each.values = args) != nil;
}
```

The following rules and plans enable an agent to follow other agents and go to a named area. The rule makes it so that when an agent is asked to follow someone or go somewhere, adding a predicate to its belief base, the agent turns the belief into a desire and eventually an intention. When going to an area, the agent follows the graph of edges but when following another agent, it moves in a straight line ignoring the graph. It assumes that the agent it follows is moving on the graph.

```
rule belief:new_predicate(“follow")
      new_desire:(get_belief_with_name(“follow"));
rule belief:new_predicate(“gotoArea")
      new_desire:(get_belief_with_name(“gotoArea"));

plan gotoArea intention:new_predicate(“gotoArea"){
      string targetName <- get_current_intention().values[“name_value"];
      area target <- area first_with (each.name = targetName);
      if (self.location = target.location) {
         do finishPlan();
      }
      else {
            do goto target:target on:the_graph;
        }
}

plan follow intention:new_predicate(“follow") {
        string targetName <- get_current_intention().values[“nurseName_value"];
        nurse target <- nurse first_with (each.name = targetName);
        do goto target:target;
}
```

Nurses need to be able to locate specific patients and we assume that a nurse has free vision of everyone in the same area as the nurse. We implement this by having each nurse perceive the area they are currently in and maintain a list of patients which are inside that area.

```
list<patient> perceivable_patients update: patient inside currentArea;

perceive target:area in: 1 {
        myself.currentArea <- self;
}
```

4.5. Implementing the treatment process

Next we focus on the implementation of the treatment process as described earlier. The treatment process is implemented through the plans that the agents have. These are more problem specific, and would thus have to be redesigned and implemented to new cases.

We create the agents at a global level, stating how many and where they are created.

```
global {
        …

    init {
        …
        area medical_office <- area first_with (each.name = “medical_office");
        area damage_office <- area first_with (each.name = “damage_office");
        area surgical_office <- area first_with (each.name = “surgical_office");
        area exit <- area first_with (each.name = “exit");

        create species:nurse number:2
                with:(team:“medical", location:any_location_in(medical_office));
        create species:nurse number:2
                with:(team:“surgical", location:any_location_in(surgical_office));
        create species:nurse number:2
                with:(team:“damage", location:any_location_in(damage_office));
        create species:overallHeadNurse number:1
                with:(location:any_location_in(exit));
        create species:teamHeadNurse number:1
                with:(team:“medical",location:any_location_in(medical_office));
    create species:teamHeadNurse number:1
        with:(team:“surgical",location:any_location_in(surgical_office));
    create species:teamHeadNurse number:1
        with:(team:“damage",location:any_location_in(damage_office));
    }
    …
}
```

Patients are created by random chance in each simulation step

```
reflex createDamagePatient when:flip(damage_patient_prop){
    area exit <- area first_with (each.name = “exit");
    create species:patient number:1
      with:(disease:“crushed hand",location:any_location_in(exit));
}
```

Step (1) of the process is the arrival and registration of the patient. When the overall head nurse perceives a newly arrived patient they must assign the patient to one of three teams depending on the symptoms of the patient. In this case study we assume that the symptoms of the patient are already known, meaning that they know which team to assign the patient to. Assigning the patient to a team consists of two tasks: instruct the patient to go to the waiting area of the team and instruct the team head nurse about the arrival of the new patient. Assuming that they can message the team head nurse remotely, we can implement this step as a plan for the overall head nurse using the following code.

```
plan assignPatient intention:new_predicate(“arrivedPatient"){
    string disease <- get_current_intention().values[“disease_value"];
    string patientName <- get_current_intention().values[“name_value"];
    patient target_patient <- patient first_with (each.name = patientName);
    string medical_wa <- “medical_wa";
    string damage_wa <- “damage_wa";
    if (target_patient != nil) {
      string wa;
      teamHeadNurse thn;
      if (disease = “poisoning") {
        thn <- teamHeadNurse first_with (each.team = “medical");
        wa <- “medical_wa";
      } else if (disease = “stomach") {
        thn <- teamHeadNurse first_with (each.team = “surgical");
        wa <- “medical_wa";
      } else {
        thn <- teamHeadNurse first_with (each.team = “damage");
        wa <- “damage_wa";
      } ask thn {
        do add_belief(new_predicate(“newPatient",
          [“patientName_value"::patientName]) with_priority 1);
      } ask target_patient {
            do add_belief(new_predicate(“gotoArea",
               [“name_value"::wa]) with_priority 1);
         }
         focus enrolledPatient var:patientName priority:1;
      }
      do finishPlan();
}
```

Step (2) of the treatment process is having a nurse attend the waiting patient. When the overall head nurse tells the team head nurse about the new patient, the team head nurse assigns an available nurse to the patient. We implement the assignment of a nurse as a plan for the team head nurse.

```
plan assignTriageNurse intention:new_predicate(“newPatient"){
      string patientName <- get_current_intention().values[“patientName_value"];
      patient targetPatient <- patient first_with (each.name = patientName);
      nurse freeNurse <- nurse first_with (each.get_current_intention() = nil and
         each.team = team);
      if (freeNurse != nil){
         predicate triage <- new_predicate(“triage",
            [“patientName_value"::patientName]) with_priority 1;
         ask freeNurse{
            do add_belief(triage);
         }
         do finishPlan();
      }
}
```

Having found a free nurse, the nurse then continues the treatment process by doing triage.

Step (3) of the treatment is the triage process which involves: locating the patient, bringing the patient to a triage room and then performing the triage there. We implement the process using subgoals, which gives flexibility to the execution of the triage process. A subgoal for getting the nurse to the same location as the patient and, once that is the case, a subgoal of doing the triage in a triage room. The plan ensures that the nurse only uses the room of their own team for the treatment.

```
plan doTriage intention:new_predicate(“triage"){
      string patientName <- get_current_intention().values[“patientName_value"];
      patient targetPatient <- patient first_with(each.name = patientName);
      if (believes(“triaged", [“name_value"::patientName])) {
         do finishPlan();
      }
      else if (targetPatient.location = self.location)
      {
         string trRoom;
         if (team = “damage") {
            trRoom <- “damage_trRoom";
            } else {
                trRoom <- "medical_trRoom";
            }
            do focusSubintention("bringToTR",
                ["roomName_value"::trRoom,"patientName_value"::patientName]);
        } else {
            patient foundPatient <-
                perceivable_patients first_with (each.name = patientName);
            if (foundPatient = nil) {
                string wa;
                if (team = "damage") {
                    wa <- "damage_wa";
                } else {
                    wa <- "medical_wa";
                }
                do focusSubintention("gotoArea",["name_value"::wa]);
            } else {
                do goto target:foundPatient;
            }
        }
}
```

Having located and made contact with the patient, the plan for bringing the patient to a triage room and doing the triage is as follows. The nurse asks the patient to follow them and they then go to the triage room. Having triaged the patient in the triage room, the nurse asks the patient to leave and adds a belief about the patient being triaged so that the `doTriage` plan can finish.

```
plan bringPatientToTR intention:new_predicate("bringToTR") {
        string roomName <- get_current_intention().values["roomName_value"];
        area trRoom <- area first_with (each.name = roomName);
        string patientName <- get_current_intention().values["patientName_value"];
        patient p <- patient first_with (each.name = patientName);
        if (self distance_to trRoom <= 1) {
            predicate leave <- new_predicate("leave");
            ask p {
                do add_belief(leave);
            }
            do add_belief(new_predicate("triaged",["name_value"::patientName]));
            do finishPlan();
        }
        else
        {
            string nurseName <- name;
            ask p {
                predicate follow <-
                    new_predicate("follow",
                [“nurseName_value"::nurseName]) with_priority 1;
            do add_belief(follow);
          }
          do focusSubintention(“gotoArea",[“name_value"::roomName]);
        }
}
```

Step (4) of the treatment process is the patient leaving the care unit. The nurse asks the patient to leave (by adding the belief `leave` to its belief base), and the patient then applies a rule that both removes the desire to follow the nurse and adds a desire to leave. The patient leaves the unit by going to the exit and then removing itself from the simulation (by calling `die()`).

```
rule belief:new_predicate(“leave")
        remove_desire:(get_belief_with_name(“follow"))
        new_desire:new_predicate(“leave");

plan leave intention:new_predicate(“leave") {
        area exit <- area first_with (each.name = “exit");
        if (self distance_to exit <= 1) {
          do die();
        } else {
          do focusSubintention(“gotoArea",[“name_value"::“exit"]);
        }
}
```

4.6. Evaluation

Although the simulation only covers a small part of the actual treatment process in an emergency care unit, we can evaluate it in terms of the criteria proposed in the introduction and discuss the general limitations of the BDI-based approach.

4.6.1. Support for descriptive agents

Using the BDI paradigm we made agents in the emergency care with rich and complex models. Agents are modelled in terms of beliefs, desires and intentions, which they use when selecting what plans to execute and how to execute them. The agents communicate with each other and perceive each other, updating their mental state according to their inference rules. Although the implemented mental models were rather simple, they could be extended with more rules and hence make more complex agents.

From a process perspective though, it is quite difficult to understand and extend the simulation to cover more of the treatment process in an emergency care unit. Implementing the process using goals and plans requires careful design and it is easy to introduce errors which can halt the entire simulation.

4.6.2. Support for reusability

Different parts of the implemented code can be reused to varying degrees. The basic agent code for implementing agent movement abilities presented in Section 4.4 can be reused for other simulations. The code pattern for perceiving other agents in a local area can be reused but needs to be reimplemented for other simulations. The code that implements the treatment process however can not be reused as it is very specific to the implemented process. The goals and plans of the agents are designed to implement the process in this case study and can not easily be modified to simulate another process.

4.6.3. Support for analysis

We can at any time in the simulation inspect the mental state of the agents and see what beliefs, desires and intentions they have, and what their currently selected plan is. In this way the BDI model allows us to analyse the system from an agent perspective: how agents revise their mental state and decide on plans. However we can not easily analyse the system from a system perspective and answer questions such as: what stages the implemented process consists of, or what roles and responsibilities the agents have.

4.6.4. Generality

The limitations we have highlighted in the evaluation above follow from the complexity of the use case. The combination of interaction between individuals and an overall work process they carry out is difficult to capture in a model. The strength of BDI is that it provides a simple paradigm for descriptive agents, reusable basic behaviour models and analysis of agent reasoning. It is less useful for defining and simulating an event process.

4.7. Perspective on using AORTA

As the use case shows, using only the BDI paradigm has some limitations in terms of both support for descriptive agents, reusability and analysis. The reason is that BDI is a primarily a paradigm for designing agent reasoning and not processes or agent organizations. We could overcome this limitation by using an organizational meta-model such as AORTA. In previous work we presented an AORTA meta-model of the acute patient treatment process (Larsen & Villadsen, 2017). In the AORTA meta-model, we encode organizational knowledge in terms of roles, objectives and sub-objectives, role dependencies and conditions. Each agent then maintains two knowledge bases: one with personal knowledge and one with organizational knowledge. The organizational knowledge base describes the stages that the patient goes through, which staff members are involved in each stage and a selection of conventions that the agents are expected to follow. When deliberating which action to perform, an agent can then reason about if an action complies or violates any obligations of the agent. Updating the knowledge base is done accordingly to general rules of the meta-model when the agents perform actions. The agents can also perform organizational actions, such as enacting roles, which will update their knowledge base accordingly. In addition, the explicit representation of organizational knowledge supports specification and verification of the organizational agent model. To summarize using AORTA in ABS could give:

(1) Descriptive agents that have a mechanism to include organizational reasoning in their decision making. We can encode system processes as an organization and then agents can then use general rules to decide actions. An organizational model would support

complex social and explainable behaviour, and we can extend the model without having to add much more code.

(2) A reusable meta-model that can be integrated in any agent programming platform that supports the BDI paradigm.

(3) Formal syntax and semantics that can be used for specification and verification of the organizational agent model. Logic reasoning can provide insight into social relations which are otherwise hard to identify or reason about.

5. Discussion

The BDI paradigm on its own only provides generalized methods for implementing internal agent reasoning. It does not provide generalized methods for implementing important aspects of multi-agent systems such as organizations and environments. The previous section analysed potential benefits that the AORTA meta-models can provide for ABS in the emergency care unit scenario. In this section we recap that analysis and discuss potential benefits of applying the other frameworks and meta-models listed in Table 1 for ABS.

CArtAgO provides a framework for implementing agent environments in Java, which is commonly used in ABS platforms, using the Agents & Artefacts meta-model. In domains where people interact through physical objects such as whiteboards or telephones, CArtAgO would provide a generalized framework for encoding these objects. In the case study with emergency care units, the physical location and availability of information communication technologies can have a major influence on the workflow. CArtAgO has been implemented in Jason and has been used to an increasing extent in MAS. As it is Java based, it could potentially also be implemented for dedicated ABS platforms that support BDI models. The Agents & Artefacts meta-model also provides theoretical foundation for specification and verification of agent environments.

EIS provides a Java framework for integrating agent programming platforms with environments. This is useful for implementing systems where the internal agent reasoning logic and the environment logic are separated from each other. The separation allows for more openness, as agents can then be integrated in the environment no matter how their internal reasoning works like. As mentioned earlier, openness is less of a concern in ABS than MAS so, although the framework is reusable, we do not see an immediate benefit of using EIS in ABS.

OperA provides a meta-model for designing and analysing agent organizations. As the evaluation in the previous sections shows, there are clear benefits of applying organization meta-models to domains with human organizations. Making a model of the organization in OperA would provide a basis for implementing ABS with AORTA agents that perform organizational reasoning. $\mathcal{M}$OISE$^+$ provides an alternative meta-model for agent organizations. Its integration with CArtAgO and Jason in JaCaMo could provide a framework for implementing ABS with both environment and organization models.

The AORTA meta-model, which was evaluated in the previous section, provides a basis for implementing organizationally aware agents in ABS platforms. Doing so would give ABS that supports descriptive agents that replicate organizational behaviour in terms of roles and norms. In domains with human organizations, such as in the hospital case, simulation with organizationally aware agents should provide more accurate outcomes than with only the BDI paradigm. There are already implementations of AORTA in Jason, which to some degree supports ABS, and since AORTA has well defined semantics and operational rules, it can be implemented in dedicated ABS platforms that support BDI models. The formal syntax and semantics in logic also supports specification and verification of the organizational agent model.

In ABS of social systems, there is also a growing interest in frameworks and meta-models for social values. A social value represents a concept that an agent cares about and it will generally perform actions that promotes its social values. Simulation with social value models have gained interest as a way to implement social behaviour that agents do exhibit without explicitly reasoning about them. Although there is work on meta-models for social values, there still remains much to be done in terms of formalization and implementation in ABS platforms. Finally, it is worth noting that current MAOP platforms, including the one based on AORTA (Jensen et al., 2014), introduce an overhead that may not be practical for ABS in practice. Practical ABS typically involves a large number of agents so while introducing frameworks and meta-models can be useful for modelling the agents, the overhead of doing so must be kept low.

6. Conclusion and future work

There is active research into providing better frameworks for implementing BDI models in ABS. They generally use one of the methods:

- Implementing simulation features in agent programming platforms (Bordini & Hübner, 2009).
- Implementing BDI models in ABS platforms (Caballero et al., 2011).
- Combining ABS platforms with agent programming platforms (Padgham et al., 2011; Singh et al., 2016).

The third method has the advantage that it can make use of advances in tools for both ABS and AOP platforms. As argued by Adam and Gaudou (2016), the cost of high computational power might also become negligible as computers get more powerful. Research in agent programming languages and logics has given frameworks and meta-models for implementing environments and social behaviour. These are designed to be reusable and their logical foundation can be used for specification and verification of ABS models. We have given an analysis and evaluation of using agent programming languages and logics in a case study based on emergency care. We have presented a simulation of emergency care made with BDI agents in GAMA and highlighted limitations in terms of support for descriptive agents, reusability and analysis. The case study motivates going beyond BDI for ABS and we have given perspective on how the AORTA meta-model allows agents to include organizational reasoning in their decision making, is reusable, and has a formal syntax and semantics that can be used for specification and verification. We also discussed potential benefits of using some of the other MAOP frameworks shown in Table 1 for ABS.

To the author's knowledge, there are still few reusable frameworks and meta-models for implementing social behaviour in ABS. Future work include implementing AORTA for ABS in an extended emergency care unit scenario.

Acknowledgments

This work is part of the Industrial PhD project *Hospital Staff Planning with Multi-Agent Goals* between PDC A/S and Technical University of Denmark. I am grateful to Innovation Fund Denmark for funding and the governmental institute Region H, which manages the hospitals in the Danish capital region, for being a collaborator on the project. I would also like to thank Jørgen Villadsen, Rijk Mercuur and Virginia Dignum for comments on the ideas described in this paper.

Disclosure statement

No potential conflict of interest was reported by the author.

Funding

This work was supported by Innovationsfonden [grant number 5189-00075B].

Notes on contributor

John Bruntse Larsen is employed at PDC A/S as an Industrial PhD student with a grant from Innovationsfonden for his PhD project. He holds an MSc from Technical University of Denmark. He has also worked at PDC A/S as software developer.

ORCID

John Bruntse Larsen http://orcid.org/0000-0003-2500-9944

References

Adam, C., & Gaudou, B. (2016). BDI agents in social simulations: A survey. *Knowledge Engineering Review*, *31*(3), 207–238.

Aldewereld, H., & Dignum, V. (2011). OperettA: organization-oriented development environment. *Lecture Notes in Computer Science*, *6822*, 1–18.

Amouroux, E., Chu, T.-Q., Boucher, A., & Drogoul, A. (2009). GAMA: an environment for implementing and running spatially explicit multi-agent simulations. *Lecture Notes in Computer Science*, *5044*, 359–371.

Behrens, T. M., Hindriks, K. V., & Dix, J. (2011). Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, *61*(4), 261–295.

Bordini, R. H., & Hübner, J. F. (2009). Agent-based simulation using BDI programming in Jason. In A. M. Uhrmacher & D. Weyns (Eds.), *Multi-agent systems: Simulation and applications* (pp. 451–476). Boca Raton: CRC Press.

Bordini, R. H., Hübner, J. F., & Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason*. Chichester: John Wiley and Sons.

Busetta, P., Ronnquist, R., Hodgson, A., & Lucas, A. (1999). JACK intelligent agents – Components for intelligent agents in Java. *AgentLink News Letter*, *2*, 2–5.

Caballero, A., Botia, J., & Gomez-Skarmeta, A. (2011). Using cognitive agents in social simulations. *Engineering Applications of Artificial Intelligence*, *24*(7), 1098–1109.

Caillou, P., Gaudou, B., Grignard, A., Truong, C. Q., & Taillandier, P. (2017). A simple-to-use BDI architecture for agent-based modeling and simulation. *Advances in Intelligent Systems and Computing*, *528*, 15–28.

Dastani, M. (2008). 2APL: A practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, *16*(3), 214–248.

Dignum, V. (2004). *A model for organizational interaction: Based on agents, founded in logic* (PhD thesis). Utrecht University.

Dignum, V., Tranier, J., & Dignum, F. (2010). Simulation of intermediation using rich cognitive agents. *Simulation Modelling Practice and Theory*, *18*(10), 1526–1536.

Hindriks, K. V. (2009). Programming rational agents in GOAL. In A. El Fallah Seghrouchni, J. Dix, M. Dastani, & R. H. Bordini (Eds.), *Multi-agent programming: Languages, tools and applications* (pp. 119–157). Boston, MA: Springer.

Hindriks, K. V., De Boer, F. S., Van Der Hoek, W., & Meyer, J.-J. C. (1999). Agent programming in 3APL. *Autonomous Agents and Multi-agent Systems*, *2*(4), 357–401.

Hübner, J. F., Boissier, O., Kitio, R., & Ricci, A. (2010). Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, *20*(3), 369–400.

Hübner, J. F., Sichman, J. S., & Boissier, O. (2007). Developing organised multiagent systems using the MOISE+ model: Programming issues at the system and agent levels. *Int. J. Agent-Oriented Softw. Eng.*, *1*(3/4), 370–395.

Jensen, A. S. (2015). Model checking AORTA: Verification of organization-aware agents. *CoRR*, abs/1503.05317.

Jensen, A. S., & Dignum, V. (2014). AORTA: Adding organizational reasoning to agents. *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014)*, *2*(3), 1493–1494.

Jensen, A. S., Dignum, V., & Villadsen, J. (2014). The AORTA architecture: Integrating organizational reasoning in Jason. *Lecture Notes in Computer Science*, *8758*(3), 127–145.

Jensen, A. S., Dignum, V., & Villadsen, J. (2017). A framework for organization-aware agents. *Autonomous Agents and Multi-Agent Systems*, *31*(3), 387–422.

Kravari, K., & Bassiliades, N. (2015). A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, *18*(1), 11.

Larsen, J. B. (2018). Agent programming languages and logics in agent-based simulation. In A. Sieminski, A. Kozierkiewicz, M. Nunez, & Q. T. Ha (Eds.), *Modern approaches for intelligent information and database systems* (pp. 517–526). Cham: Springer.

Larsen, J. B., & Villadsen, J. (2017). An approach for hospital planning with multi-agent organizations. In *Rough sets, IJCRS 2017* (pp. 454–465). Cham: Springer.

Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., & Balan, G. (2005). MASON: A multiagent simulation environment. *Simulation*, *81*(7), 517–527.

Mercuur, R., Larsen, J. B., & Dignum, V. (2018, July). *Modelling the social practices of an emergency room to ensure staff and patient wellbeing*. Presentation at Socio-Cognitive Workshop 2018 (SCS18@FAIM2018), 15 pp.

North, M. J., Collier, N. T., Ozik, J., Tatara, E. R., Macal, C. M., Bragen, M., & Sydelko, P. (2013). Complex adaptive systems modeling with Repast Simphony. *Complex Adaptive Systems Modeling*, *1*(1), 3.

Padgham, L., Scerri, D., Jayatilleke, G., & Hickmott, S. (2011). Integrating BDI reasoning into agent based modeling and simulation. In *Proceedings of the Winter Simulation Conference* (pp. 345–356).

Piunti, M., Ricci, A., Braubach, L., & Pokahr, A. (2008). Goal-directed interactions in artifact-based MAS: Jadex agents playing in CARTAGO environments. *2008 International Conference on Intelligent Agent Technology* (Vol. 2, pp. 207–213).

Pokahr, A., Braubach, L., & Lamersdorf, W. (2005). Jadex: A BDI reasoning engine. In R. H. Bordini, M. Dastani, J. Dix, & A. El Fallah Seghrouchni (Eds.), *Multi-agent programming: Languages, platforms and applications* (pp. 149–174). Boston, MA: Springer.

Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In W. Van de Velde & J. W. Perram (Eds.), *Agents breaking away* (pp. 42–55). Berlin, Heidelberg: Springer.

Ricci, A., Bordini, R. H., Piunti, M., Hübner, J. F., Acay, L. D., & Dastani, M. (2008). Integrating heterogeneous agent programming platforms within artifact-based environments. *Proceedings of the international joint conference on autonomous agents and multiagent systems* (Vol. 1, pp. 222–229).

Ricci, A., Viroli, M., & Omicini, A. (2006). CArtAgO: A framework for prototyping artifact-based environments in MAS. *Third International Workshop on Environments for Multi-agent Systems, E4MAS 2006. Selected revised and invited papers* (Lecture notes in artificial intelligence Vol. 4389, pp. 67–86).

Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence*, *60*(1), 51–92.

Siebers, P.-O., Macal, C. M., Garnett, J., Buxton, D., & Pidd, M. (2010). Discrete-event simulation is dead, long live agent-based simulation! *Journal of Simulation*, *4*(3), 204–210.

Singh, D., Padgham, L., & Logan, B. (2016). Integrating BDI agents with agent-based simulation platforms. *Autonomous Agents and Multi-Agent Systems*, *30*(6), 1050–1071.

Taillandier, P., Bourgais, M., Caillou, P., Adam, C., & Gaudou, B. (2017). A BDI agent architecture for the GAMA modeling and simulation platform. In *Multi-agent based simulation xvii* (pp. 3–23). Cham: Springer.

van Riemsdijk, M. B. (2012). 20 years of agent-oriented programming in distributed AI: History and outlook. In *Proceedings of the 2nd edition on programming systems, languages and applications based on actors, agents, and decentralized control abstractions* (pp. 7–10). ACM.

Weiss, G. (2013). *Multiagent systems* (2nd ed.). Cambridge, MA: MIT Press.

Winikoff, M. (2005). Jack intelligent agents: An industrial strength platform. In R. H. Bordini, M. Dastani, J. Dix, & A. El allah Seghrouchni (Eds.), *Multi-agent programming: Languages, platforms and applications* (pp. 175–193). Boston, MA: Springer.