Scientific Journal of Riga Technical University
Computer Science. Applied Computer Systems
2011
Volume 47

Identification of Relations Between BDI Logic and BDI Agents

Arvids Grabovskis[1], Janis Grundspenkis[2], [1-2]*Riga Technical University*

***Abstract* – BDI (Beliefs, Desires, Intentions) is one of the most popular intelligent agent architectures which was inspired by multi-modal BDI logics. The main idea behind BDI is to implement system's behaviour by specifying it as a set of mental objects. This allows designing systems at a high level of abstraction which come closer to a human-like thinking. Although this architecture has been rapidly developing for about 20 years, its relevance to BDI logic is still arguable. This paper describes the basics of modal logic and main inference algorithms. Main concepts of BDI agents are presented and their relationships with BDI logic are discussed. Finally advantages and disadvantages of implementing BDI interpreter as a theorem prover are discussed.**

***Keywords* – Modal Logic, Intelligent Agents, BDI Agents, Agent Architectures, Example Application**

I. Introduction

The notion of intelligent agent is one of the main concepts in artificial intelligence which broadly is defined as "An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators" [1]. Although definition of this term differs among authors, some key properties that should be possessed by such an entity can be identified. These key characteristics are reactivity, autonomy, goal existence and working in favour of the user [2]. Some other properties like mobility, cooperation, rationality, proactivity, learning capabilities, etc. are mentioned by researchers, but not so commonly accepted [2]. Stronger notion of agency according to [3] says that besides having previously stated properties in addition they are conceptualised or implemented using concepts that are generally applied to humans. This involves knowledge, plans, intentions, desires, beliefs and even emotions [4]. BDI architectures are probably the most popular agent architectures [5]. And their main data structures are beliefs, desires and intentions. The basic assumption in BDI is that intelligent agents should be rational and this rationality can be logically proven [6].

Knowledge (true beliefs [1]) is one of the main notions, because it describes things that an agent knows about its environment. Besides, inference algorithms can be applied in order to derive new facts which are not known directly (see [1] for simple examples), therefore reducing lack of environment information that can be perceived through sensors. There is well established theory and practice in using first-order logic as knowledge representation language (see [7] and [1] for concise description), but it is not always suited, because knowledge of agents just like human knowledge might be incomplete or uncertain, which must be taken into account while selecting actions.

One of the easiest ways of thinking about uncertainty is in terms of possibility and necessity [8]. These modalities are incorporated into formal language known as modal logic. This formalization should help creating algorithms of selecting appropriate actions even under uncertainty. Theory of formal modal logic ([7], [8], [9], [10]) has been developed for more than a century [1], but its practical application in computer sciences lacks concrete examples even though in [12] it is stated that modal logic is widely used in intelligent agents. Some extended versions of inference algorithms have been developed, such as labelled tableaux (which according to [13] is the most popular one) and modal resolution [14], to suite the needs of modal logic. So in theory practical use of modal logic is possible in order to infer actions in BDI agents.

BDI agent development must be supported by agent development platforms, because development itself involves only data structure definition, but it is up to platform to execute them. For example in PRS (Procedural Reasoning System) BDI architecture data structure consists of beliefs, desires, intentions and plans. They must be combined using BDI interpreter which selects appropriate desires as intentions based on current beliefs. Further interpreter is responsible for plan selection and execution in order to fulfil its intentions. Some of the most popular BDI agent development platforms are dMARS [15], JADEX [16], Jason [17] and JACK [18]. Although a research like [19] tries to maintain the link between theory and practice, it is still a question whether they use theoretic foundations established by BDI logics. The main goal of this paper is to find out how BDI agent development platforms use BDI logic internally and to evaluate the benefits and disadvantages of implementing BDI interpreter as a theorem prover.

The paper is organized as follows. In the second section the basics of modal logic are reviewed. Its syntax, semantics, axiom systems, types of modal logic and inference algorithms are described. In the third section focus moves to BDI agents – their main concepts, development platforms, example application, concept usage in different platforms, the main possible advantages and drawbacks of implementing BDI interpreter as theorem prover are pointed out. In conclusion the main reasons of the existence of weak link between theory and practical applications are given.

II. Logic

First-order logic is based on 3 assumptions [7]: all required knowledge is accessible in knowledge base (KB), KB does not

contain contradictions and KB is monotonic, i.e. set of entailed sentences can only increase as information is added to KB [1]. Modal logic tries to eliminate those restrictions. Another reason of using modal logic is sentences like "John knows that money is the root of all evil". Truth value of this sentence depends upon properties of John's knowledge and facts about the world. Modal logic captures the essence of this by using modal operators to denote certain modalities like necessity, possibility, knowledge, beliefs, obligation, etc. That allows writing the previous example of sentence in the form "$K_{John}(\alpha)$", where "K_{John}" is the modal operator denoting "John knows that…" and "α" denotes "money is the root of all evil". In this way syntax of first-order logic is supplemented by modal operators which follow the same induction rules as a negation. For example, if a modal operator is K_j and sentence of first-order or modal logic is α_i, then the complex sentence can be constructed as follows:

$$K_1(K_2(\alpha_1)) \vee \neg(K_3(\alpha_2)). \quad (1)$$

In the same way the predicate logic can be extended with modal operators to form the modal predicate logic.

Semantics of modal logic is a lot more complex than semantics of first-order logic. Basically there are two main modalities – necessity and possibility denoted by *L* and *M* accordingly. One can be expressed through another by the following equation:

$$L\alpha \Leftrightarrow \neg M\neg\alpha. \quad (2)$$

Equation (2) says that "necessary α" is logically equivalent to "not possible not α". For example, "it is necessary that the sun is shining" is equal to "it is impossible that the sun is not shining".

Further explanation of modal logic semantics is based on Kripke structures [8]. Kripke structure S is 3-tuple S=(W, R, π), where:

- W is a set of possible worlds $\{w_1, \ldots, w_n\}$;
- R is a subset of W×W, which denotes world accessibility from each other i.e. if a world *w'* is accessible from a world *w*, then $(w,w') \in R$ or *wRw'*;
- π is the interpretation function, which maps truth value of each sentence in each possible world i.e. $\pi(w_i)(\alpha) = \{\text{true, false}\}$.

If a sentence α is valid in a structure S, it is denoted by $(S,w) \models \alpha$. So truth values of complex sentences are [8]:

- $(S,w) \models \alpha$ iff (denotes "if and only if") $\pi(w)(\alpha)=true$;
- $(S,w) \models \alpha \wedge \beta$ iff $(M,w) \models \alpha$ and $(M,w) \models \beta$;
- $(S,w) \models L\alpha$ iff $(M,w') \models \alpha$ for all *wRw'*;
- $(S,w) \models M\alpha$ iff exists such *w'* that $(S,w') \models \alpha$ and *wRw'*.

Truth values of other operators like implication, logical equivalence and disjunction can be derived using standard tautologies (see [8] or [9] for in-depth explanations). Let us illustrate this with a simple example:

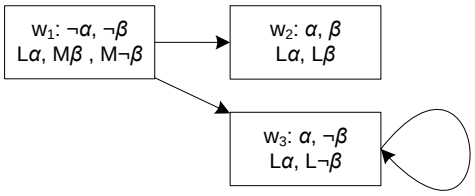

Fig. 1. Graphical representation of the example

- $W=\{w_1, w_2, w_3\}$;
- w_1Rw_2, w_1Rw_3, w_3Rw_3;
- $\pi(w_1)(\alpha)$=false, $\pi(w_1)(\beta)$=false, $\pi(w_2)(\alpha)$=true, $\pi(w_2)(\beta)$=true, $\pi(w_3)(\alpha)$=true and $\pi(w_3)(\beta)$=false.

Semantics can be represented by a directed graph, where nodes represent possible worlds, and edges – accessibility relation (see Fig 1). In this example Mα is true in a world w_1, because α is true in both accessible worlds (w_2 and w_3). Mα is also true in w_2 (because no other world is accessible from w_2) and in w_3 (because w_3 is accessible from itself, where α is true). So MMα is also true, and so is MMMα and so on. In a world w_1 sentences Mβ and M¬β both are true according to semantic rules.

A. Axiom systems

Axiom systems are an important part of modal logic. They are describing modes of truth, i.e. modalities. Using axioms is one way of describing accessibility relation in R.

The basic axiom system is called K. This system is interesting, because it is contained in all other systems, which are used to describe beliefs, desires or intentions. It includes 3 transformation rules and the axiom K. Transformation rules are as follows:

1. Sentences can be uniformly substituted;
2. Modus Ponens – if α and α→β are theorems of system S, then also β is a theorem of S;
3. If α is a theorem of S, then so is Lα;

The axiom K is:

$$L(\alpha\rightarrow\beta)\rightarrow(L\alpha\rightarrow L\beta). \quad (3)$$

The system D is the first of direct interest, because it is used for reasoning about desires and intentions. The main purpose of this system is to state that facts are not in contradiction. This is done by supplementing the system K with the axiom D:

$$L\alpha\rightarrow M\alpha \quad (4)$$

The meaning of this axiom is that "if something is necessary true then it is also possible". Generally this system is used for reasoning about desires and intentions because they are not always true, but they should be consistent i.e. it would be false to desire achieving state where "the sun must be shining" despite the fact that "it is impossible that the sun is shining". The same goes for intentions – it would be false to intend achieving state knowing that this state is not possible.

The second system of special interest is weak-S5, which is used for reasoning about beliefs. It includes axioms K, D, 4

and 5. The first two have already been described above, but axioms 4 and 5 are represented as equations (5) and (6) respectively:

$$L\alpha \rightarrow LL\alpha \quad (5)$$

$$M\alpha \rightarrow LM\alpha \quad (6)$$

Axiom 4 in context of epistemic logic, which is used for reasoning about knowledge, states that "agent knows what it knows", i.e. it knows all facts that it has in its KB. Axiom 5 states that "agent knows, what it does not know". Because of their meaning these two axioms are also called positive and negative introspection axioms.

Weak-S5 includes all known axioms except the knowledge axiom also known as axiom T. This system is usually applied when reasoning about beliefs (known as doxastic logic). Axiom T states that "every fact that is believed to be true, is true". This is not the case with beliefs, because beliefs can be false.

Here are given only the most relevant axioms, but complete list is accessible in [10].

B. BDI Logic

Although it is stated that desires and intentions use system D and beliefs use weak-S5, it is more like a suggestion than a requirement. There is no need for complete and unique aximatization that covers all BDI agent concepts, because different axiomatization allows modelling various types of agents for different purposes [20].

BDI logic is multi-modal logic, because it operates with 3 modalities – beliefs, desires and intentions. The most commonly used relationship types among these (belief-, desire- and intention-) accessible worlds are [20]:

- Weak realism – if an agent desires to achieve a proposition it will not believe the negation of the proposition to be inevitable;
- Realism – if an agent believes a proposition, it will desire it;
- Strong realism – if an agent desires to achieve a proposition, it will believe proposition to be an option.

Some other properties, such as consistency between beliefs, desires and intentions, can be necessary, but final decision is up to a designer to choose appropriate relationships between modalities and axiom systems of each modality.

C. Theorem Proving

So far syntax and semantics of BDI logic have been described. Now it will be shown that these languages can be used to automate reasoning and infer appropriate course(s) of action(s) in a formal way. As it can be seen, truth value of each sentence in addition to interpretation (which is also used in first-order logic) depends upon possible world and its relation to other worlds (i.e. axiom system). It is easy to conclude that modified versions of first-order inference algorithms must be used to take into account possible worlds and their relations.

There are two main inference algorithms that are used in modal logic: labelled tableaux calculus [21] and modal resolution [14]. Tableaux procedures are based on refutation. This algorithm constructs a tree graph with propositions at nodes. Each branch represents conjunction, but different branches – disjunction. If there are contradictions in a branch, then it is closed. If all branches are closed, then so is the tree. Therefore proving the truth of a query involves showing that KB would become inconsistent, if supplemented by negated query proposition. Labelled tableaux calculus is extended version of this algorithm, where each vertex also has a label denoting a world in which a given proposition is true. This algorithm is very easy to extend, because it is only necessary to add a successor creator, branch extending and structural rules, which determine how branches are extended, created and how new worlds (i.e. labels) are created, respectively. Structural rules depend on selected axioms. Some implemented theorem provers are mentioned in [21] and in [13]. From both sources it can be seen that this algorithm is successfully implemented.

Unlike the previous algorithm, the modal resolution is not used so widely. Although its theory is well described in [14], it requires modifications of logic language to bring it closer to syntax of predicate logic by adding a path, which indicates the world, where a given predicate is true. For example, the predicate *MLMp(x)* should be transformed to the form given in (7) or (8) depending on the chosen style:

$$\forall Wp[0gWh](x) \quad (7)$$

$$\forall W(p(h(W(g(0))),x)) \quad (8)$$

In (7) square brackets contain the path to the world in which the predicate p(x) is true, i.e. in this case the path is 0→g→W→h, where 0 denotes w_0 (current world), lower case letters denote some accessible world just like Skolem constants, when eliminating existential quantifiers (see [1]), but upper case letters denote all accessible worlds, just like variables can represent different constants. Therefore $p[0](x)$ is semantically equal to $p(x)$ while $p[0g](x)$ is equal to $Mp(x)$ ("possible that p(x)") and $\forall Wp[0W](x)$ is equal to $Lp(x)$ ("necessary that p(x)"). In (8) the path to the world is given via function, where 0 is the current world, g(0) is the function of accessible world, while W(g(0)) represents all accessible worlds from g(0) and so on. These two types of syntax modifications can be perceived as alternative ways of specifying possible worlds, where sentence is true (i.e. modalities). Unification algorithm [1] is used in all first-order inference algorithms. It takes two sentences and returns unifier (i.e. substitution) which can make these sentences identical. For example, substitution of *father(X)* and *father(john)* would be θ={X/father}. The goal of path unification is the same – to find the substitution which makes two paths look identical. For example, in system K the path

[0XYa] can be unified with the path [0Xba] with substitution θ={Y/b}, where upper-case letters denote variables (i.e. every accessible world) and lower-case letters denote constants (i.e. exact accessible world). But in system T the path [0] can be unified with the path [0X] by using substitution θ={X/Ø}.

This modified unification algorithm can be applied to forward chaining, backward chaining or other first-order inference algorithms.

So here it is shown that it is possible to adapt first-order theorem provers to handle modal logic and therefore infer new sentences.

III. BDI AGENTS

BDI architecture is based on interpreter and 4 key data structures – beliefs, desires, intentions and plans [5]:

- Beliefs are facts that agent knows about the world;
- Desires are possible goals;
- Intentions are selected goals that agent is capable to achieve (e.g. state of affairs allows to achieve them);
- Plans are instructions on how to achieve goals.

As previously stated, there is no single notion of agency, but BDI agents support all currently best known definitions very well. While the concepts which make the "BDI" acronym explicitly support the definitions mentioned in [3], the accordance to definitions which list mandatory characteristics is clarified in further paragraphs.

Let us divide key properties mentioned in the introduction in two categories, where the first category includes requirements for data structure and the second category includes behaviour requirements. In the first category goals and events (to support reactivity) are included, while the remaining properties are included in the second category. Goals in this architecture are called desires and hence are explicitly supported. Events are also important part of BDI, because they trigger belief updates and intention revision.

Other properties (i.e. behaviour requirements) like autonomy are not strictly defined and hence can not be functional requirements. For example, in space systems it might be important to continue following the plan even if signal from the Earth is lost, but this is only part of autonomy, because it is composed of many other properties. The same holds for flexibility (i.e. if some part of the system fails), rationality, etc. So this must be achieved by selecting appropriate system design which consists of its components, component structure and communication protocols.

If BDI data structures are domain specific, then an interpreter should be of general purpose with capabilities of extending it, for example, to modify plan selection or add learning capabilities.

A. Agent development platforms

In order to review possible BDI logic applications in development platforms, let us look at abstract algorithm of interpreter (Fig. 2). This abstract interpreter shows the sequence of actions while agent is running.

```
1  initialize-state();
2  repeat
3      options ← option-generator(event-queue);
4      selected-options ← deliberate(options);
5      update-intentions(selected-options);
6      execute();
7      get-new-external-events();
8      drop-successful-attitudes();
9      drop-impossible-attitudes();
10 end repeat
```

Fig. 2. Abstract algorithm of BDI interpreter offered in [2]

At first agent must define possible goals and initial beliefs. Then, in the course of action, agent continuously updates options using new events and adopting some of those options as new intentions. Further, agent executes an action from its intention structure and new events must be gathered. Finally, intention structure must be updated in order to drop satisfied and unreachable intentions.

The main points where inference can be used are option selection (line 3) and calculating successful and impossible attitudes (lines 8 and 9 respectively). In option selection inference can be used to select only those desires as intention candidates, whose context declares them as attainable. Successful attitude dropping can be achieved by querying beliefs for achieved states. Impossible attitude dropping should be done when inconsistencies are found.

Currently the best known BDI agent development platforms are PRS, dMARS, AgentSpeak and Jason, JADEX, 3APL and 2APL, GOAL, Jack, SPARK and JAM [22]. Unfortunately, only JADEX, Jack and Jason are platforms which still are under active development, so it makes sense to focus on their inspection. All 3 platforms employ some kind of belief base, which is used to store beliefs and mechanism to select intentions. So it is worth examining those two parts of platforms.

In order to more thoroughly inspect the application of modal logic in agent development platforms, an example application has been implemented. It will be described in the next section, while the use of concepts will be discussed in the following sections.

B. Application Example

Example of multi-agent system was developed using JADEX platform. Application domain is a forest, where fire breaks out from time to time. Fire scale increases in time. In order to report the place of fire, there are several watcher agents, which wander around and report on locations of fire. Then it is up to firemen to extinguish the fire. Each fireman has a limited amount of water. If fire scale is greater than the water amount at the time of arrival, a fireman can not extinguish the fire completely, but only reduce the scale of the fire. When all water is poured out, a fireman must return to the base, where he fills the tank and returns to the fire.

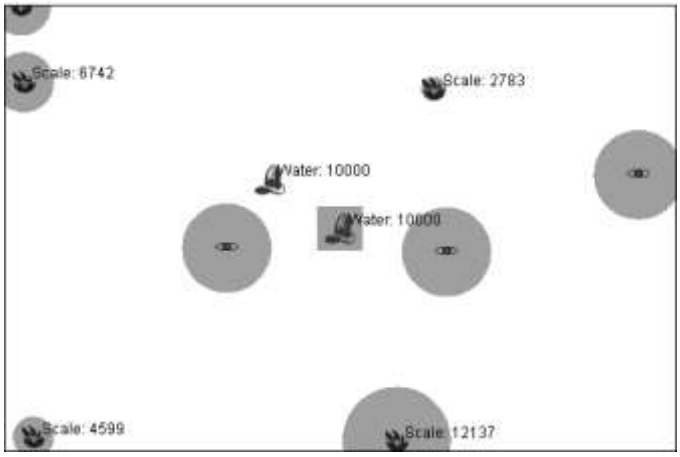

Fig. 3. Environment of example application. Watcher agents are wandering around and are depicted by eye icon with circle showing their vision distance. Places of fire are depicted as fire icons and circle shows the scale of it. Firemen agents are depicted by fireman hat also showing the available water amount.

As movement is a common ability of both types of agents, i.e. firemen and watchers, it was defined using capabilities which are used to denote an encapsulated agent module composed of beliefs, goals and plans. Movement capability is defined as follows:

```
<capability name="Movement" package="…">
 <imports>…</imports>
 <beliefs>
  <belief name="environment" class="…"><fact>
(ContinuousSpace2D)((IApplicationExternalAccess)$sco
pe.getParent()).getSpace("myForestSpace")
  </fact></belief>
  <beliefset name="objectsSeen" class="ISpaceObject"
exported="true" />
  …
 </beliefs>
 <goals>
  <achievegoal name="moveTo" exported="true">
   <parameter name="destination" class="IVector2" />
  </achievegoal></goals>
 <plans>
  <plan name="moveToDestination">
   <parameter class="IVector2" name="destination">
    <goalmapping ref="moveTo.destination" />
   </parameter>
   <body class="MoveToDestinationPlan" />
   <trigger><goal ref="moveTo" /></trigger>
  </plan></plans>
</capability>
```

As it can be seen, there are beliefs, goals and plans. The only difference in capability definition from agent definition is absence of initial goals. This example demonstrates that beliefs are simple Java objects or a set of objects. The goal definition includes the goal name and required parameter list. It can be perceived as a function with arguments. Plans specify the actual sequence of actions in order to achieve the goal or handle an event (e.g. fact adding, message receiving, etc.). In this example actions are specified by *MoveToDestinationPlan* agent class, which is defined as the subclass of JADEX *Plan* class. Plan execution is invoked by its *body()* method where movement is implemented as small steps of agent transition from one point of the environment to another towards destination.

A fireman agent is defined by importing movement capability, *waitForFire* and *fillTank* goals and *waitForAlarmPlan*, *extinguishFirePlan* and *fillTankPlan* plans. Plan *extingiushFirePlan* is invoked when new facts about fire locations are added. These facts are added by *waitForFirePlan* upon receiving message which contains fire alarm. Each agent needs an initial goal. Here it is *waitForFire*.

A watcher agent is implemented by importing movement capability, *wander* goal and *wanderPlan* and *initiateAlarmPlan* plans. Wandering involves choosing a random spot in the environment, selecting it as a destination, moving to it and watching fire in the field of vision along the path. The second plan is invoked each time, when a new fact of seen fire object is added. This fact is added by *wanderPlan*. The initial goal for watchers is wandering.

Application descriptor files include environment specification and component (agent) importing. JADEX 2.0-rc2 version was used for implementation of this example. This version supports easy configuration of environment by XML files which allows decoupling environment from agents.

Environment definition includes visual representations of its objects (e.g. fire base, firemen, fire and watchers), processes, percepts, views and executors. In this example processes to fire creation and fire increase processes were employed. Each process is a class which implements *ISpaceProcess* interface where main methods are *start()* and *execute()* for initialization and execution respectively. View definition contains the class which must be used to show objects. Executors are responsible for executing environment in a certain manner. In this case *DeltaTimeExecutor* is used to update environment after short periods of time to imitate real-time action. Finally the application descriptor specifies the number of each agent type acting in the environment. In this case there are 3 watcher and 20 firemen agents.

A screenshot of application is shown in Figure 3. There are 5 places of fire, 3 watcher agents wandering around and one fireman agent going to extinguish the fire. All remaining 19 firemen agents are at their base in the centre of map. Considering the direction of active fireman agent it can be seen that he is going to extinguish the fire in the top left corner of the map.

C. Beliefs

A belief base is called differently in each platform. Jack uses the term “beliefset”, JADEX uses the term “beliefbase” and Jason uses the term “beliefs”, but the concept is the same – belief storage.

It turns out that currently JADEX and Jack employ a belief base as simple fact storage. Let us see a simple example of belief definition in JADEX:

```
<beliefs>
 <belief name="env" class="ContinuousSpace2D">
  <fact>
```

```
(ContinuousSpace2D)((IApplicationExternalAccess)
$scope.getParent()).getSpace("myForestSpace")
  </fact>
 </belief>
 <belief name="me" class="ISpaceObject">
  <fact>
   $beliefbase.environment.getAvatar($scope
.getComponentIdentifier())
  </fact>
 </belief>
</beliefs>
```

Here beliefs are nothing else but passive data store, where common Java objects can be stored and expression languages can be used to manipulate those objects. That allows monitoring a belief base and triggering goal creation, dropping procedure or rendering the context of a plan invalid leading to a plan cancelation.

In Jack beliefs are defined using beliefsets, where callbacks and triggers can be defined which are actions that must be executed in response to changes. This allows attaining the same functionality as in JADEX, but in a different way:

```
beliefset Fireman extends ClosedWorld {
  #key field String number;
  #value field int waterAmount;
  void postEvent(Event e) {/* event handling */}
  add/remove/(String number, double balance) {}}
```

Data structures in JADEX and Jack are defined differently, but they keep their tight relationship with Java programming language. Jason, on the other hand, uses completely different programming language called AgentSpeak. For example, the belief definition in Jason is as follows:

```
has(fireman, water, 10000).
tankFull() :- has(fireman, water, Q) & Q==10000.
```

So beliefs are stored as predicates and Prolog-like expressions can be used to create complex logical sentences. This is a significant difference from previous platforms where such functionality can be achieved only by coding these expressions in Java.

So far in literature we have not been able to find applications of modal logics in belief bases and there is no difference between beliefs and facts (recall that beliefs may be false). JADEX states that even contradictory beliefs can be stored [23], which is confusing. This is possible, because there is no logical inference involved, which would be broken in such case.

D. Intention Selection

Next part, which could involve inference, is intention selection. Each goal (desire) has a context description, which describes under what circumstances it can be achieved. For example, a fireman would intend to extinguish the fire only if its tank is full.

So theoretically intention selection is another part of BDI interpreter, where BDI logic can be applied, because there are:

- Appropriate formal languages to specify beliefs, desires, intentions and their relationships;
- Inference algorithms that suit the needs of modal logics.

Each platform implements desires (i.e. goals) as an entity which consists of:

- Context which defines under what circumstances a goal must be selected for achievement;
- Plan specification or reference to plan specification.

But it turns out that none of the examined products implements it using logic. Jack does not employ concepts of desires or intentions – it uses events and plans to handle those events. Therefore it is arguable whether Jack can be called BDI agent development platform, even though the same behaviour can be achieved as using other platforms.

JADEX does not have such shortcoming – goals are explicitly stated in agent definition file. For example:

```
<goals>
  <performgoal name="waitForFire" retry="true"/>
  <achievegoal name="fillTank" />
</goals>
```

Below there is an example of a fireman agent, whose main goals are waiting for the fire, extinguishing the fire and filling the tank. Further plan definitions are given, which determine the reference to plan execution code and its trigger conditions.

```
<plans>
 <plan name="waitForAlarmPlan" priority="0">
  <body class="WaitForAlarmPlan" />
  <trigger><goal ref="waitForFire" /></trigger>
 </plan>
 <plan name="extinguishFirePlan" priority="1">
  <body class="ExtinguishFirePlan" />
  <trigger><factadded ref="fire" /></trigger>
 </plan>
 <plan name="fillTankPlan" priority="2">
  <body class="FillTankPlan" />
  <trigger><goal ref="fillTank" /></trigger>
 </plan>
</plans>
```

Just like JADEX, Jason explicitly uses goals that can be achieved. For example:

```
+!fireExtinguished (Fire) : tankFull()
    <- goto(Fire);
       pourWaterOnFire(Fire);
       …
       goto(Station);
```

This goal example starts with target state definition – goal will be achieved when fire is extinguished. Then context definition follows – this goal can be achieved only when tank is full. And finally actions which must be executed in order to extinguish the fire are stated.

As it can be seen, no applications of modal logic can be found, when reasoning about beliefs or in intention selection in the most popular BDI agent development platforms.

E. BDI Interpreter as a Theorem Prover

Of course, the performance of theorem prover is worse than simple rule-matching mechanism in option selection and belief

retrieval. But whether it could give some kind of benefit, if BDI interpreter is implemented as a theorem prover therefore allowing the use of modal logic in agent specifications and BDI interpreter configuration?

First, BDI logics must be reviewed. Let us assume that modal operator B stands for "agent believes that …" Although it is possible to create expressions like "B¬Bα", so far there is no sound evidence of their possible applications in real-world situations. Besides, such complex expressions are not self-explanatory and therefore it is hard to debug such system in case of problems. As a result from the theory of modal logic one can use only application of axiom systems to point out that believed facts can be false, beliefs should be consistent (which is not the case with JADEX) and agent knows its beliefs and disbeliefs (axiom systems D, 4 and 5, respectively) and create a new formal logic with simplified execution.

The same problem of expressiveness may occur dealing with desires and intentions. Although it is possible to combine modal operators to form complex sentences, there is almost no chance that such expressions would be needed, because there is no use of desiring to desire something or intend to desire to intend something. But all this expressiveness comes at a price and makes theorem provers very complex.

Also implementing BDI interpreter as a theorem prover will eliminate the possibility of storing contradictory facts and goals, which can become an important shortcoming. Of course, at first it is hard to find rational systems where such behaviour would be required. On the other hand, it is worth to keep as few restrictions as possible since contradictions in human knowledge and goals are very common, but they tend to handle those contradictions very well. For example, one might intend to repair his teeth in the same time avoiding pain which is impossible (contradictory).

IV. Conclusions

In this paper it has been pointed out that BDI logic is not used in practical implementations of BDI agent development platforms. This can be explained by the fact that agent is a resource bounded entity. Even in highly experimental applications responsiveness is a highly desirable property. Therefore, implementing BDI interpreter based on theorem prover is possible, but not worthwhile. That is why BDI logic in context of BDI agents can be used only for conceptual system discussion and specification.

Another aspect is that while specifying system behaviour even at purely theoretical level only limited expressiveness of modal logic is used. For example in BDI logics the same modal operator is rarely applied more than once in a row to construct sentences like "agent believes that he does not believe to believe to …".

Current BDI platform implementations have proven themselves successful in real-life applications – they are high-performance implementations of algorithms which must choose the correct goal to be achieved under given circumstances (beliefs). Development platforms are continuing their steady development to improve extensibility and flexibility.

Another reason in favour of current BDI agent development platforms is possibility to handle contradictory knowledge and goals (as JADEX manual states). That would not be possible using theorem provers since currently most wide-spread algorithms (such as tableaux calculus and modal resolution) are based on refutation procedures.

It is convenient to use BDI agents when implementing goal-oriented systems. Example implementation proved BDI agent development platform to be a handy tool for modelling purposes. Using JADEX makes environment definition especially easy because it offers standard environment components – view type (e.g. grid view and continuous space), agent representations, environment background processes and environment evolution logic.

Also it must be noted that software development must be simplified as much as possible in order to improve platform usage. Direct use of formal modal logic requires a lot of background knowledge which is acceptable in academic research, but not welcomed in software development industry, unless there are sound benefits of this additional complexity.

In future work separation of BDI logics from modal logics should be done since only limited expressiveness in BDI logic is needed. Limited expressiveness would allow simplifying inference engines thus improving their performance.

References

[1] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*, 2nd ed. Upper Saddle River, NJ: Prentice hall, 2009.
[2] C. Chira, "Software agents," IDIMS Report, vol. 21, 2003.
[3] M. Wooldridge and N. Jennings, "Intelligent agents: Theory and practice," The knowledge engineering review, vol. 10, no. 02, pp. 115–152, 1995.
[4] C. Adam, A. Herzig, and D. Longin, "A logical formalization of the OCC theory of emotions," Synthese, vol. 168, no. 2, pp. 201–248, 2009.
[5] B. Luigi, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*. Wiley, 2007.
[6] R. Jones and R. Wray, "Comparative analysis of frameworks for knowledge-intensive intelligent agents," AI Magazine, vol. 27, no. 2, p. 57, 2006.
[7] G. Luger, Intelligence *Structures and Strategies for Complex Problem Solving*, 6th ed. Reading, Massachusetts: Addison Wesley, 2009.
[8] J. Halpern, *Reasoning about uncertainty*. Cambridge, Massachusetts: The MIT Press, 2003, 483 lpp.
[9] N. Cocchiarella and M. Freund, *Modal logic: an introduction to its syntax and semantics*. Oxford: Oxford University Press, 2008.
[10] G. Hughes and M. Cresswell, *A new introduction to modal logic*. London: Burns & Oates, 1996, 432 lpp.
[11] R. Goldblatt, "Mathematical modal logic: A view of its evolution," Handbook of the History of Logic, vol. 7, pp. 1–98, 2006.
[12] W. Michael, *An introduction to multiagent systems*. Wiley, 2002.
[13] R. Schmidt, "Advances in Modal Logic," 2010. [Online]. Available: http://www.cs.man.ac.uk/~schmidt/tools/ [Accessed June 03, 2010].
[14] H. Ohlbach, *A resolution calculus for modal logics: 9th International Conference on Automated Deduction*. Springer, 1988, pp. 500.–516.
[15] M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge, "A formal specification of dMARS," *Intelligent Agents IV Agent Theories, Architectures, and Languages*, pp. 155–176, 1998.
[16] A. Pokahr, L. Braubach, and W. Lamersdorf, "Jadex: A BDI reasoning engine," *Multi-Agent Programming*, pp. 149–174, 2005.
[17] R. Bordini, J. Hübner, and M. Wooldridge, *Programming multi-agent systems in AgentSpeak using Jason*. Wiley-Interscience, 2007.
[18] "Autonomous Decision-Making Software", http://www.aosgrp.com.au, 2010. [Online]. Available: http://www.aosgrp.com.au [Accessed: May 30, 2010].
[19] M. Móra, J. Lopes, R. Viccari, and H. Coelho, *BDI models and systems: Reducing the gap: Intelligent Agents V. Agent Theories, Architectures,*

and Languages: 5th International Workshop, ATAL'98, Paris, France, July 1998. Proceedings. Springer, 2000, pp. 629–629.
[20] A. Rao and M. Georgeff, *BDI agents: From theory to practice: Proceedings of the first international conference on multi-agent systems (ICMAS-95).* San Francisco, 1995, pp. 312–319.
[21] Z. Li, "Efficient and Generic Reasoning for Modal Logics," Ph.D. dissertation, University of Manchester, 2008.
[22] S. Sardina and L. Padgham, "A bdi agent programming language with failure handling, declarative goals, and planning," *Autonomous Agents and Multi-Agent Systems*, pp. 1–53, 2010.
[23] "Jadex User Guide". [Online]. Available: http://-garr.dl.sourceforge.net/¬project/¬jadex/¬jadex/¬0.96/¬userguide-0.96.pdf [Accessed: Sept. 30, 2010].

Arvids Grabovskis is a PhD student at Riga Technical University. He received his Bc.sc.ing degree in 2008 and his Mg.sc.ing. degree in 2010 from the same university. His major field of study is computer science,, computer systems in particular.
He is currently working at Riga Technical University as a software developer and he is the leader of a developer group. In addition he is a research assistant at Riga Technical University. His research interests include artificial intelligence, knowledge representation using logic, intelligent agents and their development.

Janis Grundspenkis graduated from Riga Politecnical Institute (now Riga Technical University) in 1965. His major was electrical engineer of automation and telemechanics. He received his Dr.sc.ing. degree from Riga Politecnical Institute in 1972 and his Dr.habil.sc.ing. degree in 1993 from Riga Technical University.
He is a professor of systems theory at Riga Technical University. He is also the Dean of the Faculty of Computer Science and Information Technology, the Director of the Institute of Applied Computer Systems, and the head of the Department of System Theory and Design. His research interests are agent technologies, knowledge engineering and management, structural modeling for diagnostics of complex systems and development of intelligent tutoring systems.
He is a member of Institute of Electrical and Electronics Engineers (IEEE), Association of Computer Machinery (ACM) and International Association for Development of the Information Society (IADIS). He is a full member of Latvian Academy of Science.

Arvīds Grabovskis, Jānis Grundspeņķis. BDI loģikas un BDI aģentu saistības identificēšana
Rakstā ir izpētīta BDI loģika un BDI aģenti ar galveno mērķi noteikt to saistību. Viena no nozīmīgajām aģentu spējām ir zināšanu formalizēšana un spriešana, izmantojot šīs zināšanas, tādējādi mazinot informācijas trūkumu. Taču arī pašas zināšanas var būt nedrošas vai pat pretrunīgas. Lai šos aspektus ņemtu vērā, dažādos avotos iesaka izmantot modālo loģiku. Modālās loģikas sintakse būtībā ir pirmās kārtas loģikas sintakse, kas papildināta ar modālajiem operatoriem. Taču galvenās atšķirības ir semantikā, jo modālie operatori apraksta katra izteikuma patiesumvērtības veidu, piemēram, zināšanu nepretrunīgumu, patiesumu, citu zināšanu apzināšanos, kā arī neziņas apzināšanos. Šo izmaiņu dēļ bija jāpārbauda modālo secināšanas algoritmu esamība un izmantojamība. Rezultātā var secināt, ka visplašāk izmantotais algoritms ir iezīmēto pozīciju rēķini, bet retāk – modālā rezolūcija. Viens no modālās loģikas paveidiem ir BDI loģika, kuras trīs galvenie koncepti ir pārliecības, vēlmes un nolūki. Līdz ar to intuitīvi šķiet, ka BDI aģentiem, kas izmanto tādas pašas datu struktūras, ir ļoti ciešs sakars ar šo loģiku. BDI loģikas iespējamais pielietojums ir meklējams BDI aģentu izstrādes ietvaros. BDI aģentu izstrādē ir jāizmanto ietvari, jo izstrādes laikā definētās datu struktūras ir jānodod BDI interpretatoram (daļa no ietvara), kas tās kombinē un izpilda ar tām saistītos plānus (reizēm plānus arī min kā ceturto jeb „aizmirsto" BDI konceptu). No BDI ietvariem ir izvēlēti JADEX, Jason un Jack, kuros ir apskatīta zināšanu izmantošana un nolūku izvēles mehānismi. Papildus ir izstrādāta neliela sistēma JADEX izstrādes ietvarā. Rezultātā atklājās, ka pārliecību bāze ir vienkārša datu glabātuve, bet nolūku atlasē nekādā veidā netiek izmantoti modālās loģikas secināšanas mehānismi. Papildus secinājums ir tas, ka modālās loģikas pilnas izteiksmes spējas reti tiek izmantotas aģentu izstrādē un ir grūti atrast modālo loģiku reālus pielietojumus, izņemot sistēmu konceptuālā izstrādē.

Арвидс Грабовскис, Янис Грундспенкис. Идентификация связи между BDI логикой и BDI агентами
В этой статье представлены исследования BDI логики и BDI агентов с целью определения их связи. Одна из важнейших способностей, присущих агентам, это формализация знаний и суждений, использующих эти знания, таким образом уменьшая нехватку информации. Но и сами знания могут быть ненадежными или даже противоречивыми. Чтобы учитывать эти аспекты, в разных источниках предлагают использовать модальную логику. В сущности, синтаксис модальной логики - это синтаксис логики первого порядка, дополненный модальными операторами. Но главные отличия в семантике, так как модальные операторы описывают вид истинного значения каждого высказывания, например, непротиворечивость знаний, правдивость, осознание других знаний, а также осознание неизвестности. Из-за этих отличий было необходимо проверить существование и возможность использования алгоритмов для модального заключения. В результате можно сделать вывод, что чаще всего используемый алгоритм - исчисления обозначенных позиций, а реже – модальная резолюция. Один из видов модальной логики - BDI логика, три главных принципа которой: убеждения, желания и умыслы. Поэтому на интуитивном уровне кажется, что у BDI агентов, которые используют такие же структуры данных, тесная связь с этой логикой. Возможное применение BDI логики может быть найдено в рамках разработки BDI агентов с использованием каркасов. В процессе разработки BDI агентов определенные структуры данных (часть каркаса) необходимо передать BDI интерпретатору, который комбинирует их и выполняет связанные с ними планы (иногда планы упоминаются как четвертый или «забытый» концепт). Из BDI каркасов выбраны JADEX, Jason и Jack, в которых рассмотрены использование знаний и механизмы выбора умысла. Дополнительно, используя BDI каркас JADEX, разработана небольшая система. В результате оказалось, что база убеждений - это простое хранилище данных, и в отборе умыслов механизмы заключения модальной логики не принимают никакого участия. Дополнительное заключение таково, что возможности полных высказываний модальной логики редко используются в разработке агентов, а также трудно найти реальное применение модальной логике, кроме как в концептуальной разработке систем.