ASP-Driven BDI-Planning Agents in Virtual 3D Environments

André Antakli, Ingo Zinnikus, and Matthias Klusch

German Research Center for Artificial Intelligence, DFKI
Agents and Simulated Reality Department
Saarland University Campus, Bldg. D3.2
Stuhlsatzenhausweg 3, 66123 Saarbruecken, Germany
{andre.antakli,ingo.zinnikus,matthias.klusch}@dfki.de
http://www.dfki.de

Abstract. This paper introduces the agent platform HumanSim, a combination of the BDI-paradigm and Answer Set Programming (ASP), to simulate entities in three-dimensional virtual environments. We show how ASP can be used to (i) annotate a virtual three-dimensional world and (ii) to model the goal selection behavior of a BDI agent. Using this approach it is possible to model the agent domain and its behavior – reactive or foresighted – with ASP. To demonstrate the practical use of HumanSim, we present a three-dimensional planning and simulation application, in which worker agents are driven by HumanSim in the shop floor domain. Furthermore, we show the results of an evaluation of HumanSim in the former mentioned simulation application.

Keywords: BDI Agents, Intelligent Virtual Agents, Virtual Environments, Answer Set Programming, IVA architecture

1 Introduction

Applications of virtual three-dimensional environments have seen a strong growth in the last few years. One main reason is the increasing number of affordable and easy to use hardware and software by which it is possible to create and display such environments in less time, with a more realistic graphics and intuitive interaction. These technologies are not only used in the consumer section, for example in games. Moreover, those technologies enable new opportunities in research or manufacturing. Examples are the evaluation of ergonomic aspects in cars[1] or the simulation of a shop floor configuration[2] which require the (as far as possible) realistic simulation of autonomous entities. In order to use those entities in a daily routine, for example in different evaluation settings, a flexible way to model their behavior is necessary.
An often used technology to drive three-dimensional entities in virtual environments is the BDI-agent architecture (see [1][2]). This paradigm is a established

[1] RAMSIS Automotive http://www.human-solutions.com/
[2] FlexSim Simulation Software, https://www.flexsim.com/flexsim/

and well researched agent model, which has its origins in the research done by [3]. It combines aspects of reactive and deliberative agent models. A BDI-agent can adapt its behavior in non-deterministic environments in a resource efficient way. The main components of a BDI-agent are *Beliefs*, *Desires* and *Intentions*. *Beliefs* represent the agent knowledge about its environment and its current situation. *Desires* are long-term goals which have to be achieved by the agent. *Intentions* are the actions which have to be performed next to reach the current goal. Agent frameworks which are using this approach are Jack [4] or Jadex [5]. An agent can only interact with its environment in an autonomous and realistic way, if it 'understands' what it is 'seeing' by dint of its sensors. The virtual environment and thus its objects need to be annotated with semantic information, which describe in a – for the agent – meaningful way what they are standing for. Usually, the semantic information about the world state, the agent actions and their potential effects are expressed in heterogeneous formalisms and languages. In [1] for example, three-dimensional objects in a simulated virtual environment are semantically annotated with OWL2[3]. In [6], RDF[4] is in use to describe the 3D-scene instead. The perceived semantic information can be used by the agent to deliberate and reason about the current world state to execute specific agent plans for reaching goals.

In this paper we present our approach to describe the BDI agent environment and its reasoning process with a uniform declarative logical formalism, answer set programming (ASP). The application areas of ASP, which is based on the stable model semantics [7], are NP-hard search problems [8], typically used for model-checking, scheduling, diagnostics and decision-making [9]. By using ASP, on the one hand we are able to annotate the agent environment and model its ontology. We are also in the position to endow agents with commonsense reasoning [10], to simulate reliable foresighted acting virtual humans. Furthermore, ASP allows extending the specification of a knowledge-intensive problem domain with additional features such as e.g. non-deterministic effects, indirect effects of events, default reasoning or background knowledge in a uniform way. E.g. it is possible to naturally specify and reason about transitive relations necessary for agents in 3D environments such as reachability of locations which can be formalized in PDDL only by using *derived predicates* which are supported only by a few PDDL planners[5]. A further motivation and requirement for using a declarative formalism is the possibility to modify and change the agent's knowledge during the design phase of a simulation scenario without the need to recompile the agent code after each modification. Simulation environments are used e.g. for rapid prototyping, investigating and evaluating properties of production scenarios before they are put in place etc. In this case, the behavior of human avatars needs to be adapted to various settings. We have integrated the ASP-based BDI agent approach into a design and editing framework for 3D environments which allows changing agent behaviour in a flexible way. Our approach enables *elabora-*

[3] Web Ontology Language, `http://www.w3.org/TR/owl2-overview/`
[4] Resource Description Framework, `https://www.w3.org/RDF/`
[5] For a discussion of the problems involved in derived predicates in PDDL cf. [11]

tion tolerant[6] ways of specifying human actions. If e.g. objects are duplicated in a 3D environment, logical rules can immediately be applied to these new objects without reconfiguring the agent behavior specification (which would be the case in e.g. automata based behaviour representations).
The remainder of this paper is organized as follows. In Section 2 and 3, we describe our layered approach and introduce the HumanSim architecture. In section 4 and 5 we show, with reference to a shop floor use case example, how HumanSim can be embedded and used in a simulation environment. Section 6 describes the evaluation results and section 7 shows the related work in the areas of agent description and simulation in 3D environments.

2 HumanSim: a Layered Approach

For simulating human actions in virtual worlds, we structure agents representing human avatars with a three layer architecture. Each layer is responsible for dealing with specific aspects, i.e. related to navigation and animation (bottom layer), basic actions (middle layer) and deliberation (upper layer). The navigation layer handles path finding and animation, the middle layer provides routines and recipes for executing ground actions such as e.g. walking, picking and placing objects, etc. The deliberation layer finally contains a representation of the agent's beliefs about the environment, the possible actions it might perform and their consequences, as well as the decision procedures to form an intention based on this knowledge.

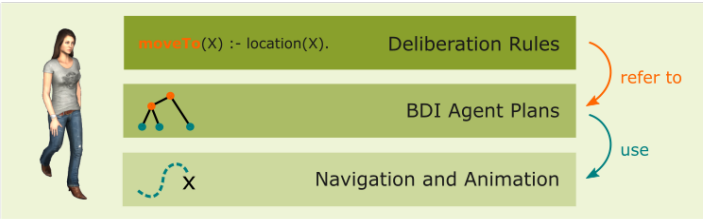

Fig. 1. The HumanSim three layer architecture.

2.1 Layered Architecture

Navigation and Animation layer: This layer handles the basic tasks of path finding and motion generation of agents in 3D environments. Both are standard tasks in such environments supported by several game engines[7]. This layer in

[6] "Elaboration tolerance is the ability to accept changes to a person's or a computer program's representation of facts about a subject without having to start all over." [12]
[7] Example of an engine which supports both tasks is unity3d: `http://unity3d.com/`

HumanSim represents the 3D scene, the objects as well as generating paths and motions which allow avatars to interact with the 3D scenario by avoiding collisions with objects or other avatars. The navigation and animation components are accessible for all agents and handle the motion requests and the execution of the agent motions in the 3D scene. During execution, the agents receive constant updates of their motion to control the accomplishment of a for example movement or picking task.

BDI Agent Plans: The BDI agent plans provide routines for ground actions the agents can execute in 3D environments. Ground actions include movement (walking, running, etc.), picking and placing objects as well as more fine grained tasks such as pushing buttons, locking and unlocking (doors, storages, etc.). Ground actions such as walking use the bottom layer for navigation, whereas other actions require specific animation functionality depending on the rendering environment used. Agents comprise of a set of these ground plans which according to the BDI approach are triggered when respective events are received. For selecting the ground plans, deliberation rules are used.

Deliberation Rules: HumanSim agents possess an explicitly represented, symbolic model of the world as well as knowledge about their capabilities to act and the effects of these actions. Both knowledge is expressed in ASP rules. The world model includes facts about the state of the world as well as knowledge about relationships between classes, types of objects, terminological knowledge etc. which is usually represented in ontologies. The combination is used to perform the process of intention formation, i.e. the selection of plans which allow the agent to accomplish its goals.

2.2 Logic Programs: Basic Concepts and Terminology

Due to space reasons, we refrain from a detailed introduction of the syntax and semantics of logical rules in ASP (the reader is referred e.g. to [13]). Instead we try to highlight several aspects which characterize ASP and are necessary for the understanding in the context of this paper.
A logical program P in ASP consists of a set of rules with (possibly negated) literals in the body and at most one literal in the head of a rule. Literals may be ground or contain variables. Facts are rules with empty body. The stable model semantics for a program P is based on the *Gelfond-Lifschitz* transformation [7]. Given P and a set of ground atoms S, the *reduct* P_S is obtained by deleting

1. each rule that has a negated literal **not** L in its body with $L \in S$, and
2. negated literals of the form **not** L in the bodies of the remaining rules.

The set S is a stable model of P, if the least Herbrand model $MM(P_S) = S$ (note that P_S is a definite program, i.e. does not contain negated literals). Thus, a stable model or *answer set* is a consistent set of ground literals. Intuitively, a stable model consists of the ground literals which are consistent with the rules in P and are justified by the rules (i.e. appear in the head of at least one rule). It is important to note that usually a logic program has several stable models.

Answer set programming is based on the stable model semantics extended with rules for expressing e.g. choice, weights, cardinality constraints and optimization statements [14]. In the context of planning, a choice rule can be used to express that a rule may be optionally applied in a specific situation. Thus, this semantics allows expressing non-determinism in a natural way. For generating the stable models of a program, a number of efficient solvers have been developed.

2.3 Deliberation Rules for Reactive Agents

Extending the BDI reasoning cycle with deliberation rules, the logical rule set representing the agent's knowledge about the current state of the world is used for selecting the agent's intention. If a relevant event is received from the environment the belief state of the agent is changed and the reasoning process for the updated rule set is started. With respect to the usual BDI reasoning cycle, this provides a filter which interprets events in the light of the knowledge rules and as a result leads to potential intentions. Since the rule set can have several stable models, each of these models provides an option the agent might choose to achieve his goal(s). The options refer to 'ground' plans which the agent is able to perform. One implemented BDI plan is e.g. the *moveTo plan*, which will be triggered by a *moveTo(X)* predicate contained in a stable model. In the following, a random walk behavior, implemented with one rule, is shown:

```
intention(moveTo(L)) :- location(L), not agentPosition(L,T), actualTime(T).
```

This rule has the following meaning: *"Go to location L, if there is a location L and you are not at location L at time T and T is the current time."*

2.4 Look-ahead and Planning with the Discrete Event Calculus

A general limitation of BDI agents is the limited support when accounting for the possible effects of an action while selecting an intention. The Discrete Event Calculus (DEC) is a logical formalism for expressing commonsense knowledge and enabling the reasoning about effects of events and actions, see [10]. DEC uses a sorted logic with events, fluents and timepoints. *Events* represent events or actions that can occur in a world. *Fluents* represent properties of the world which can vary over time, which is represented by *timepoints*. DEC provides axiomized predicates which allow to express the effect of events (e.g. agent actions) on fluents. Reasoning about effects of events and actions can be used to endow agents with look-ahead capabilities and planning. DEC has initially been formalised using first-order logic with circumscription (which is a second order feature) in order to cope with the frame problem. Lee and Palla [15] have shown that DEC can be represented in ASP[8] which allows the usage of state-of-the-art ASP solvers in new contexts (which indicates the close relationship between circumscription and the stable model semantics). The usage of DEC allows agents

[8] DEC ASP Rules: http://reasoning.eas.asu.edu/ecasp/examples/foundations/DEC.lp

to project the knowledge about the current situation into the future, in particular to plan the course of actions in order to achieve their goals.

The following example shows a specification of the action *moveTo* as DEC event with effects of performing the action at timepoint T. *moveTo* can be applied to all known locations (line 1). The rule in line 2-4 specifies that a *moveTo* action can be performed under certain circumstances (e.g. the agent believes that the location is reachable at timepoint T). The DEC predicates *initiates* and *terminates* (lines 5-8) state which fluents are affected if the action is performed (the fluent *agentPosition* is modified). Finally, an ASP constraint (a rule with empty head, line 9) specifies which property must not hold after executing agent actions, i.e. the goal the agent wants to achieve (in the example: a dispenser has to be filled with units). Note that *moveTo* does not affect the fluent *needUnits*, hence by itself is not sufficient to bring about the goal condition.

```
01| event(moveTo(L)) :- location(L).
02| {happens(moveTo(L), T)} :-
03|        holdsAt(agentPosition(M),T), location(L),
04|        reachable(L, T), M!=L, time(T), T<maxtime.
05| initiates(moveTo(L),agentPosition(L),T) :- location(L),
06|        holdsAt(agentPosition(M),T), time(T), M!=L.
07| terminates(moveTo(L),agentPosition(M),T) :- location(L),
08|        holdsAt(agentPosition(M),T), time(T), M!=L.
09| :- holdsAt(needUnits(dispenser), maxtime).}
```

3 Extended BDI Architecture

The layered approach described in the previous section is embedded into a BDI-agent model and reasoning cycle in order to simulate humans in virtual three-dimensional environments by coupling several AI technologies, see figure 2.

Our agent model uses the BDI-model to describe the interior agent state, hence the agent model is divided into the components *Beliefs*, *Desires* and *Intentions*. The agent *Beliefs* respectively its knowledge base KB includes two different kinds of knowledge: the knowledge $K_\sigma \in KB$ of received information about the world state σ; and the knowledge $K_\beta \in KB$ about how to react to these states to reach specific goals. The set $D \subseteq Des$ represents these goals, where Des are all possible desires. An agent also has a subset $I \subseteq Int$ of BDI plans, where Int designates all possible intentions. I provides routines how to execute an atomic action $\alpha \in \Gamma$ in the agent 3D environment, where Γ denotes the set of all possible agent actions. An agent $A \subseteq Agt$ of all possible HumanSim agents has also a function φ which uses the KB of A as input to select a set of intentions I_σ at specific agent world states to reach $D \in K_\beta$. Each time φ selects I_σ, the corresponding plans out of I become the next intentions of A and atomic actions out of Γ were executed. Hereby K_σ will be updated and after processing all I_σ, φ will be executed again with the updated KB. We call φ also *reasoning cycle*, because of this cyclical execution. While receiving information δ of updated entities E in the agent environment, the old information of E will be overwritten with δ in K_σ.

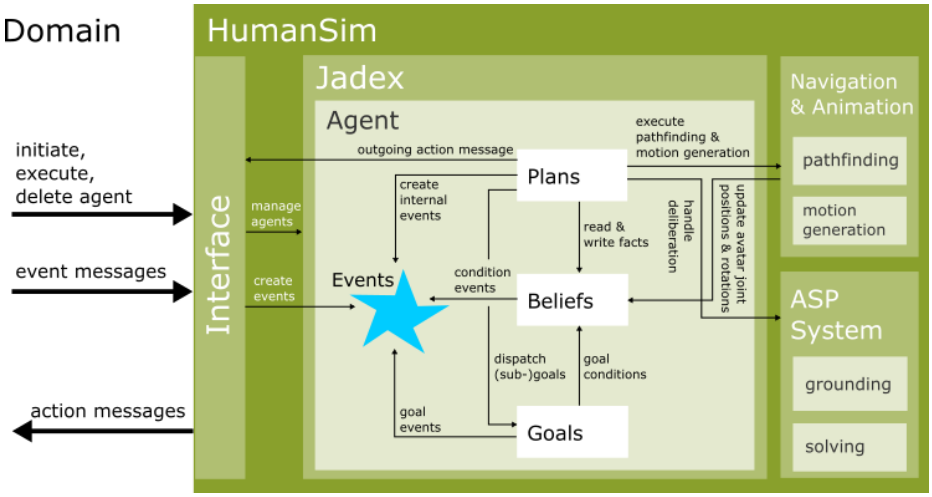

Fig. 2. HumanSim architecture overview.

3.1 Reasoning Cycle

The *reasoning cycle* φ is shown below as pseudocode by using an ASP system in our BDI-agent framework. This function will be executed if domain updates were received by the agent and no plan is currently executed.

Pseudocode of the HumanSim reasoning cycle.

reasoningCycle*(event, K_σ, K_β, I, Γ, maxtime)*

```
01 |   run := true
02 |   parallel_while true
03 |       K_σ := receiveDomainInformation()
04 |   endparallel_while
05 |   while run = true
06 |       r := getRulesets(K_σ,K_β)
07 |       for i := 1 to maxtime do
08 |           m := GetStableModelsWithASPsystem(r,i)
09 |           if |m| ≥ 1 then break
10 |       endfor
11 |       if m = null or |m| = 0 then run := false
12 |       m_x := chooseModelOutOf(m)
13 |       m'_x := coverIntentionPredicates(m_x,I)
14 |       if |m'_x| = 0 then run := false
15 |       for s := 1 to |m'_x| do
16 |           I_σ := selectIntentionBySequenceNumber(m'_x,s)
17 |           success := executeActionsByIntention(I_σ,Γ,K_σ)
18 |           if success = false then break
19 |       endfor
20 |   endwhile
```

φ starts with collecting all beliefs from KB^A of agent A which is updated in a parallel process each time a updated domain information is received (line 2-4). In the next step, the ASP system will be executed (line 8) in a loop with the collected information (line 6). This loop will be passed until the ASP solver output contains a stable model or the loop number reaches $maxtime \in \mathbb{N}$. *maxtime* is a number which can be set from outside to control the loop but also the resulting stable model. While executing the ASP system a variable defined in the rulesets is set with the current loop number, e.g. to define the maximal search depth. If no stable model has been found after *maxtime* was reached, the *reasoning cycle* will be canceled and the agent listens to future belief changes. Otherwise one of the stable models will be chosen to cover all defined predicates which refer to available BDI plans (line 12-13). These predicates will be then transformed into agent intentions I_σ (line 16) and applied depending on the sequence number s in their predicates (line 17). While achieving a current goal, action messages were sent to perform ground actions in the agent domain. Belief updates resulting from this achievement are used to evaluate the success of the executed plan. If all intentions in the sequence are processed or the plan fails while execution, the *reasoning cycle* will be passed again (line 18).
Note that while generating the stable models, the knowledge of the agent is treated under the closed world assumption according to the stable model semantics. Nevertheless, since the environment is dynamic, the agent can acquire new knowledge (facts, but also general rules) and e.g. deliberately explore the environment to get more information[9].

4 3D Simulation Environment

The main emphasis of HumanSim is the simulation of human beings in three-dimensional virtual worlds. To create and simulate such a three-dimensional scenario we integrated our layered approach into a collaborative, web-based framework with components for creating and simulating 3D scenarios for training and evaluation issues. These components are categorized in two phases: *designtime* and *runtime*. In the *designtime* phase, a three-dimensional scenario with animated characters and the 3D models as well as their semantic information are modeled. In the *runtime* phase, the scenario designed is simulated and executed.

Designtime: To simulate a scenario it has to be created in the COMPASS web editor[10] (figure 3). In COMPASS it is possible to drag and drop predefined entity definitions into a scene. To define entities as agents, we use the *agent component*. With this component it is possible to model the agent behavior with ASP rules. In figure 4, a component which defines an agent with a 'random walk' behavior, is shown. To make ASP annotations other components are available.

[9] Arguably, in 3D environments considered in the context of this paper, the closed world assumption is more appropriate.

[10] COMPASS (Collaborative Modular Prototyping And Simulation Server): https://github.com/dfki-asr/compass

Fig. 3. COMPASS editor to model the key chain scenario.

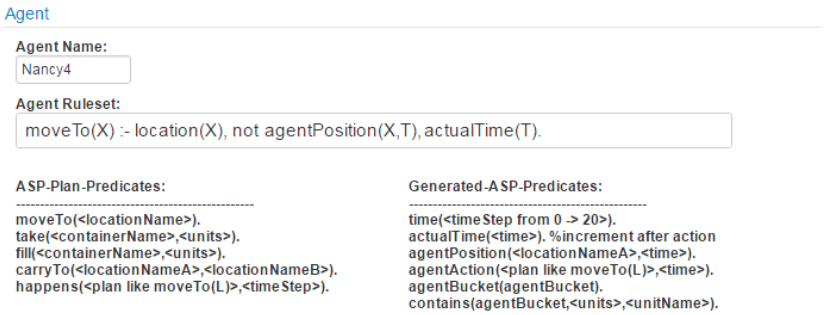

Fig. 4. Agent component to define an entity as an agent with a 'random walk' behavior.

Runtime: In the runtime phase the annotated 3D scene, modeled in COMPASS, is simulated with the execution environment FiVES, a highly scalable synchronization platform[11]. FiVES initiates agents in HumanSim over its interface with agent information previously defined in an *agent component*. While executing agents, HumanSim sends agent information to FiVES, to interact with the simulated environment. An agent has to perceive its environment, the world state represented in FiVES, to reason about its domain. Such information is received by HumanSim (figure 5) through a plugin which listens to create, update or delete events of FiVES-entities. Upon receiving these events, HumanSim updates the belief base of its agents with the new information. After updating

[11] FiVES (Flexible Virtual Environment Server): https://github.com/fives-team

the belief base the HumanSim *reasoning cycle* will be performed. While performing this cycle, all 'perceived' information about the simulation environment and the agent behavior is used to decide which actions have to be performed subsequently in the simulated environment.

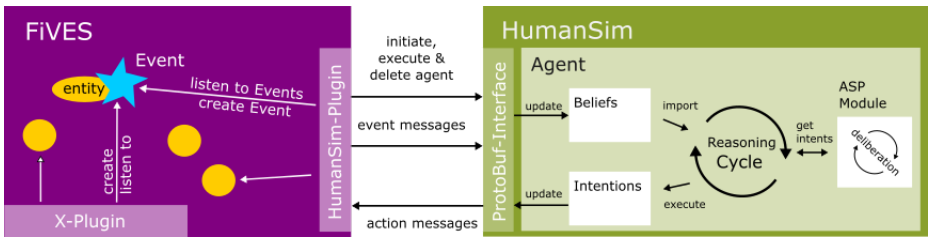

Fig. 5. FiVES-HumanSim Interaction.

5 Use Case Example

In the use case scenario, a key chain manufacturing process is simulated in a three-dimensional virtual world. In this process a single manufacturing module, consisting of presses, two dispensers and a pick and place (PnP) arm, produces key chains. To produce a key chain, the PnP arm has to pick all required parts stored in the dispensers and places them into a press which finally presses all parts together. Thus, the filling level of the dispensers decreases until their minimum filling level is reached. After reaching this level, the production stops.

A worker is present in this scenario which is in charge of maintaining the production. The agent monitors the key chain production to react to possible errors. The considered manufacturing fault is a reached minimum filling level of the dispensers, while producing multiple key chains. To avoid the *key chain* production termination, the simulated worker has to go to a storage location to fetch new material and walk to a under filled dispenser to refill it.

5.1 Scenario Modeling

To model and simulate the *key chain* scenario, we edit the 3D scene using the C3D-framework, in particular the COMPASS editor. Assets contained in the scene are a *manufacturing module*, with its presses and dispensers; a *factory model*; several *storage locations*; and also a *worker* asset. Figure 6 shows the created 3D scene in which the *key chain* scenario will be simulated.

To annotate the three-dimensional scene, we use the *ruleset component*. With this component, it is possible to annotate an entity with ASP rules, describing to which classes it belongs and its initial state at the execution time. Figure 7 shows three different ASP rules which describes the minimum and maximum filling level of an entity and the fact that entities contained are withdrawable.

Fig. 6. 3D Scenario overview.

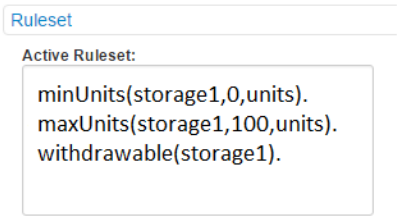

Fig. 7. Component to annotate entities by ASP rules.

5.2 Agent description.

For the *key chain* scenario we implemented in addition to the *moveTo* ground plan further BDI plans in the HumanSim framework, like the *Take* and *Fill* plan. These plans are recipes which enable an agent to interact with its environment to refill underfilled dispensers. To realize this behavior, we modeled the 'refill' behavior in two different strategies: the 'reactive' and the 'foresighted' look-ahead planning strategy.

Reactive Behavior. With this strategy, an agent assesses only its current situation and acts depending on it. This means that a 'reactive' agent has for one situation one or more action goals to reach another situation. The *take* and *fill* action rules of the 'reactive' refill agent, which are holding if the agent with no units stands beside a storage location in the first case and in the latter case, if it has enough units and stands beside the underfilled dispenser, are shown below:

```
01| intention(take(C,U)) :- contains(C,S), withdrawable(C),
02|         needUnits(D,U), C != D, S >= U, agentPosition(C,T),
03|         agentHasUnits(B), actualTime(T), B < U, container(D).
04| intention(fill(C,U)) :- container(C), agentPosition(C,T),
05|         agentHasUnits(B), actualTime(T), needUnits(C,U), U <= B.
```

The BDI *take* goal will be triggered if the situation from line 1-5 holds. This rule can be read as: *"If the current agent position is container C and C is withdraw able and container D needs units U and the agent does not have enough units R in its bucket B, then the agent has to take U units out of C."* Instead, the BDI *fill* goal will be triggered if the situation from line 4-5 holds.

Foresighted Behavior. As mentioned in section 2, there are different ways to plan with ASP, and therefore to model foresighted agent behavior with our

approach. We use the DEC axiomatisation and specify further ground actions (e.g. for *take* and *fill*) with execution conditions and effects. Unlike the 'reactive' strategy, apart from the execution conditions and effects of ground actions, only the initial situation and goal situation have to be present as rules. By defining general conditions for actions, this strategy is more flexible than the 'reactive' strategy. Moreover, it is possible to get multiple stable models and therefore multiple possible plans to solve a problem, by using this kind of strategy.

5.3 Scenario Simulation

In the *runtime* phase, the annotated *key chain* scene with the agent definition were simulated by FiVES. After initiating HumanSim, a simulated worker is driven by an agent, while receiving scenario information out of FiVES. With this information and the agent behavior modeled in ASP, the *reasoning cycle* will be performed. The execution of ground agent actions in the simulated environment depends i.a. on the used agent behavior and therefore on the predicates containing in a resulting stable model[12]. Following, the output of the ASP module at world state σ_t using the 'reactive' behavior is shown:

```
Answer: 1
intention(moveTo(storage1)) intention(moveTo(storage2))
```

We get one stable model with two *intention(moveTo(X))* predicates, if a dispenser is underfilled and the agent has to move to one of two possible storage locations to take needed units. In the next *reasoning cycle* step, one of those predicates will be chosen and a *moveTo* intention, with location X present in this predicate, apply. After achieving this goal in the simulated environment, and therefore after updating the agent belief base with the new world state `agentPosition(X)`, the *reasoning cycle* will be executed again. It is performed until no action rule is present in a stable model. This is the case when the former underfilled dispensers have enough units, after: `intention(take(X,10))`, `intention(moveTo(dispenser1))`, `intention(fill(dispenser1,5))`, `intention(moveTo(dispenser2))` and `intention(fill(dispenser2,5))` showed up in the ASP module output. The following output of the 'planning' behavior shows two of four possible stable models which hold if two dispensers are underfilled.

```
Answer: 1
intention(moveTo(storage1),1) intention(take(storage1,10),2)
intention(moveTo(dispenser1),3) intention(fill(dispenser1,5),4)
intention(moveTo(dispenser2),5) intention(fill(dispenser2,5),6)

Answer: 2
intention(moveTo(storage2),1) intention(take(storage2,10),2)
intention(moveTo(dispenser1),3) intention(fill(dispenser1,5),4)
intention(moveTo(dispenser2),5) intention(fill(dispenser2,5),6)
```

[12] To minimize the output of the ASP module, we use *gringo* filter statements `#show`.

In these models, the performing sequence of the BDI plan E in the *intention(E,T)* predicate, is denoted by the increasing number T. These sequences propose a way and therefore different possibilities to reach the goal state in which no dispenser is underfilled. After (e.g. randomly) selecting one sequence respectively stable model, the selected one will be performed in the virtual environment by executing the BDI intention specified for time T.

6 Evaluation

We simulated several settings for each behavior strategy: the 'reactive' and the 'foresighted' strategy. We considered multiple situations which affect the runtime of our agent system in the presented scenario domain, see section 5. We assume that the execution time of selected plans is constant, thus we measured only the runtime of the agent deliberation process. To do so, we first tested two different domain situations: a 'Critical' situation where the worker has to refill two dispensers and a 'Idle' situation in which every dispenser has enough units and the worker has the goal to rest. Further aspects which influences the former mentioned process are the number of placed objects in the scene like: locations (defined by one rule), storages (defined by six rules) and dispensers (defined by five rules). We also measured the increase of the logical program complexity by defining the deliberation rules more general. While evaluating the 'foresighted' strategy, we calculate all stable models to get every possible intention for each plan step, as we receive while using the 'reactive' strategy. The evaluations were carried out with the ASP system Clingo 4.4.0 on a Windows 7 system with 16 GB memory and an Intel i7-3770k CPU.

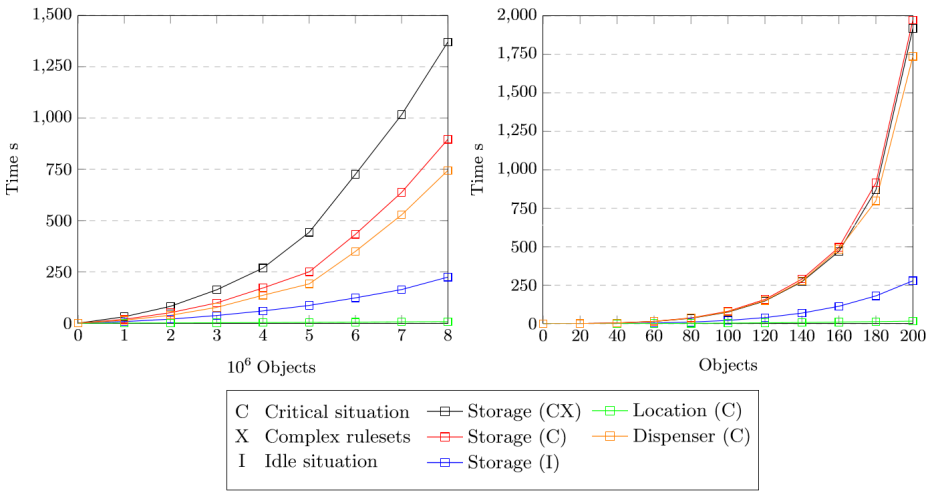

Fig. 8. Strategy runtime evaluation: 'reactive' (left); 'foresighted' (right).

In the evaluations (see figure 8) the number of the placed objects were increased from 1 to 8×10^6 while using the 'reactive' strategy and for the 'foresighted' strategy, 1 to 200 objects were placed. The runtime to find stable models depends i.a. on the number of evaluated rules. This relationship becomes visible while comparing both strategies for the 'Idle' setting: The ASP system needs 224.25s to examine $\approx 48 \times 10^6$ rules to find a stable model for the 'reactive' situation. For the 'foresighted' strategy it needs 278.78s to examine $\approx 57 \times 10^6$ rules instead. Considering the other results of the 'foresighted' program, its runtime exponentially increases, the other 'reactive' results increase polynomial instead. The strong runtime growth while using the 'foresighted' strategy depends on the search depth needed to find a stable model. In all 'Critical' situations a search depth of six is needed which is defined by *maxtime* in the agent function φ. Using the 'reactive' strategy instead, in all situations a stable model can be found with a *maxtime* of one. If placed objects do not influence the deliberation process as in the 'Location (C)' situation, their runtime effect is negligible.

7 Related Work

We use the agent paradigm to simulate human models with HumanSim. This paradigm is not just used to simulate virtual humans, as in [16], but also to control robotic systems as in [17] or in the shop floor domain as in [18]. The agent architecture used in HumanSim is the BDI approach implemented with the Jadex framework. Another BDI-agent framework is Jason[13]. Jason uses AgentSpeak(L) to describe the BDI-agent beliefs, goals and to model the decision-making process [19]. AgentSpeak(L), is a logic based programming language for modeling BDI agents such like APL3 [20] or DALI [21]. In [22] a BDI framework uses ASP to support agent belief operations. In [16] instead, the decision-making process of social virtual agents is extended with ASP. In [23] ASP modules are introduced which can be integrated in systems to describe agent "capabilities". Closer to our approach for enhancing BDI-agents with deliberation are [24], [25] and [26]. While [25] focusses on incorporating plans generated from first principles into the plan library, [26] provides a detailed formal account of an extended BDI agent language and proposes to use an HTN planner to incorporate lookahead planning. Apart from the restriction on HTN planning (which again requires a different formalism and language), their theoretical focus is not oriented towards usage in practical environments.

8 Conclusion and Future Work

In this paper, we presented HumanSim, a layered BDI-agent framework for simulating human avatars in 3D environments. For intention selection, HumanSim uses logical rules expressed in ASP which incorporate knowledge about the environment as well as knowledge about the capabilities of the agent. We detailed

[13] Jason: `http://jason.sourceforge.net/`

the extended BDI reasoning cycle with reactive and lookahead features based on the Discrete Event Calculus and described the usage of HumanSim in the context of a simulation environment for production scenarios. Finally, we evaluated the reactive and foresighted reasoning behavior with respect to solving time for different domain sizes.

One may argue that for planning purposes, PDDL could be used since highly optimised software exists. Apart from the fact that this would reintroduce another formalism as opposed to our goal of using a uniform approach, PDDL only provides the specified language constructs which are in addition not supported by all planners available. As indicated in the introduction, ASP is much more versatile and allows expressing a wide range of commonsense knowledge in an intuitive way, admittedly at the expense of efficiency.

The approach is not limited to one agent but can be applied to an open number of agents, under the condition that goals are not conflicting among agents. To coordinate agents in case of conflicting goals, standard mechanisms (protocols) could be applied. As future work, we will explore how the logical formalism used for deliberation within one agent can be used to support decisison making among a group of agents. Together with the reactive and the deliberation layer, the resulting framework would be an instantiation (with up-to-date technology) of the InteRRaP architecture [27] in which tradition we situate our approach.

Acknowledgments. The research described in this paper has been funded by the German Federal Ministry of Education and Research (BMBF) through the projects Collaborate3D and INVERSIV.

References

1. Nesbigall, S., Warwas, S., Kapahnke, P., Schuboltz, R., Klusch, M., Fischer, K., Slusallek, P.: ISReal: A Platform for Intelligent Simulated Realities. In: Agents and Artificial Intelligence: 2nd Int. Conf. (ICAART2010), pp. 201 –213. Springer (2011)
2. Davies, N. P., Quasim, 2M.: BDI for Intelligent Agents in Computer Games. In: Pro. of CGAMES2006, The University of Wolverhampton (2006)
3. Bratman, M. E.: Intentions, Plans, and Practical Reasoning. Cambridge University Press (1999)
4. Busetta, P., Ronnquist, R., Hodgson, A., Lucas, A.: JACK Intelligent Agents - Components for Intelligent Agents in Java. (1999)
5. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: A BDI reasoning engine. In: Multi-Agent Programming, Languages, Platforms and Applications, vol. 15, pp. 149–174. Springer (2005)
6. Radkowski, R., Weidemann, F.: Semantic Web-Techniques and Software Agents for the Automatic Integration of Virtual Prototypes. In: Virtual and Mixed Reality - Systems and Applications, LNCS, vol. 6674, pp. 387–396. Springer (2011)
7. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: 5th Int. Conf. on Logic Programming (ICLP), pp. 1070–1080. MIT Press (1988)
8. Lifschitz, V.: What Is Answer Set Programming? Department of Computer Sciences University of Texas at Austin (2008)

9. Brain, M., De Vos, M.: Answer Set Programming a Domain in Need of Explanation. In: Proc. of 3rd Int. Workshop on Explanation-aware Computing (ExaCat), pp. 391–403. CEUR-WS.org (2008)
10. Mueller, E. T.: Commonsense Reasoning. Elsevier Science, (2006)
11. Thiebaux, S., Hoffmann, J., Nebel, B.: In defense of PDDL axioms. Artif. Intell. 168, 1-2, pp. 38-69, (2005)
12. McCarthy, J. 1998. Elaboration Tolerance. Common-Sense 98.
13. Gelfond M.: Answer sets. In van Harmelen, F.; Lifschitz,V.; and Porter, B., eds., Handbook of Knowledge Representation, Elsevier (2007)
14. Simons, P., Niemela, I. Soininen, T.: Extending and implementing the stable model semantics. In: Artificial Intelligence Journal, vol. 138, pp.181–234, (2002)
15. Lee, J., Palla, R.: Reformulating the Situation Calculus and the Event Calculus in the General Theory of Stable Models and in Answer Set Programming. In: Journal of Artifcial Intelligence Research, vol. 43, pp. 571–620, (2012)
16. Lee, J. H., Li, T., De Vos, M., Padget, J.: Using Social Institutions to Guide Virtual Agent Behaviour. In: The AAMAS Workshop on Cognitive Agents for Virtual Environments (CAVE-2013) (2013)
17. Ryuh, Y. S., Moon, J. I.: Multi-agent control and implementation of Bio-inspired underwater robots for mariculture monitoring and control. In: Robotics and Biomimetics, pp. 777–783, IEEE (2012)
18. Barenji, R. V. Barenji, A. V., Hashemipour, M.: A multi-agent RFID-enabled distributed control system for a flexible manufacturing shop. The Int. Journal of Advanced Manufacturing Technology, vol. 71, pp. 1773–1791, Springer (2014)
19. Rao, A. S.: AgentSpeak(L): BDI Agents speak out in a logical computable language. In: 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, LNCS, vol. 1038, pp. 42–55, Springer (1996)
20. Dastani, M., van Riemsdijk, B. Dignum, F., Meyer, J. J. C.: A Programming Language for Cognitive Agents: Goal Directed 3APL. In: Programming Multi-Agent Systems, LNCS, vol. 3067, pp. 111–130, Springer (2004)
21. Constantini, S., Tocchio, A.: DALI: An Architecture for Intelligent Logical Agents. In: Proc. of Int. Workshop on Architectures for Intelligent Theory-Based Agents (AITA08), AAAI (2008)
22. Krümpelmann, P., Thimm, M., Ritterskamp, M., Kern-Isberner, G.: Belief Operations for Motivated BDI Agents. In: Proc. of Autonomous Agents and Multiagent Systems (AAMAS 2008), pp. 421–428, (2008)
23. Constantini, S.: Answer Set Modules for Logical Agents. In: Proc. of Int. Conf. on Datalog Reloaded, LNCS, vol. 6702, pp. 37–58, Springer (2011)
24. Walczak, A., Braubach, L., Pokahr, A., and Lamersdorf, W.: Augmenting BDI agents with deliberative planning techniques. In Proc. of the Programming Multiagent Systems Workshop (PROMAS), pp. 113–127, (2006)
25. de Silva, L., Sardina, S., Padgham, L.: First principles planning in BDI systems. In: Proc. of Autonomous Agents and Multi-Agent Systems (AAMAS), vol. 2, pp. 1001–1008, (2009)
26. Sardina, S., Padgham,L.: A BDI agent programming language with failure recovery, declarative goals, and planning. In: Proc. of Autonomous Agents and Multi-Agent Systems (AAMAS 2011), vol. 23, pp. 18–70, (2011)
27. Müller, J.P. and Pischel, M.: The Agent Architecture InteRRaP: Concept and Application. Research Report RR-93-26, German Artificial Intelligence Research Center (DFKI), Saarbrcken, June 1993.