The Project ENTs: Towards Modeling Human-like Artificial Agents

Ondřej Bojar[1], Cyril Brom[1], Milan Hladík[1], Vojtěch Toman[1]

[1]Charles University, Faculty of Mathematics and Physics
Malostranské nám. 2/25, Prague, Czech Republic
{ obo@ruk, brom@ksvi.mff, vtoman@ksi.ms.mff }.cuni.cz
milan_hladik@centrum.cz

Abstract. The Project ENTs aims at providing a universal high-level tool for prototyping human-like artificial agents. The main features of this tool are a virtual test-bed environment and E language, which enables description of human-like agents' behaviours using various techniques, in particular reactive planning and BDI-architecture. In this paper, we present first generation of this tool together with an example agent—an artificial gardener that acts in a virtual family house. We then briefly discuss applicability of this tool and introduce some requirements for its second generation.

1 Introduction

An *agent* is an encapsulated computational system, that is situated in some environment, and that is capable of flexible, autonomous behaviour in order to meet its design objective [25]. *Human-like agents* (h-agents in following text) are agents designed to simulate human behaviour in a virtual environment similar to natural world. Such agents are used in educational applications, interactive drama, computer games or cognitive science.

H-agents typically consist of a simulated *body*, *sensors*, *efectors* and a *control unit* (mind). One of the key problems in the field of h-agents simulation is how to design their mind, and thus their behaviour. It is because environments of h-agents are typically large and dynamic, h-agents must carry out complex tasks, and are *persistent* (they can not simply stop acting when a sub-task fail). Various approaches to control h-agents have been used so far; such as BDI-architecture [15] or subsumption architecture [5], hierarchical or any-time planning [8, 21], hierarchical rule-based system [16], finite state machines [26] or even neural networks [12]. Various techniques can be also used to create subsidiary components of h-agents' mind—for example linguistic module or emotional block discussed in this paper.

Although there are many individual applications featuring h-agents and special programming languages used to control them, it is hard to find any high-level tool that would simplify the development. In fact, not only computer experts create h-agents. From the development process point of view, designers and testers participate on the design, or debugging and parameterization of h-agents, respectively. From the academic point of view, students or non-computer researchers (like artists, librarians,

psychologists etc.) also create h-agents from time to time. Some pragmatic reasons, why these people find it difficult to develop h-agents, follow:

a) Non-computer researchers usually cannot code in C++ or Java, therefore they are not able to build new h-agent or their environments from scratch.
b) Students are often not able to see "the gap" between theory (*i.e.*, artificial intelligence algorithms) and practice (*i.e.*, implementation). Students cannot test their knowledge in practice and later they do not know how to use their knowledge.
c) Due to complexity of h-agents' mind, programmers or designers sometimes need to prototype it (it means to test several approaches and choose the best one). Languages like C++ or Java are not a good tool for prototyping.
d) Researchers and programmers must often focus on supplementary low-level issues instead of their goals, repeatedly reinventing what has been already invented (for example how to design behaviour using reactive planning or how to program path-finding).

To address these issues we aimed at creating a universal high-level tool for prototyping human-like agents. The main goals were 1) to enable prototyping, thus to simplify and speed up the design (to solve a), b) and c)), 2) to let users focus on their main objective, not to supplementary issues (addressing d)). We did not attempt to integrate existing tools and libraries, preferring to create a high-level test-bed for artificial intelligence. We focused mainly on a control unit of h-agents.

In this paper we present the first generation of this tool. Its main features are 1) a neat virtual environment with a graphical user interface (GUI), 2) a universal high-level E language for prototyping h-agents' mind, 3) a library of predefined behaviours, and 4) a linguistic and memory module. Later we discuss how we have met our goals and suggest requirements on second generation of the tool.

The rest of the paper proceeds as follows: We recall some theoretical background and related works on h-agents simulation in section 2. Section 3 details the first generation of project ENTs. Section 4 presents a prototyped h-agent—a gardener—controlled by reactive planning. In section 5, we evaluate the results and present some requirements for second generation of the tool.

2 Related work

In this section we present an overview of h-agents' mind architectures and summarize some related works comprising both tools and applications featuring h-agents. We also compare presented applications with project ENTs.

Reactive behaviour is a term used for reflexive behaviour responsible for quick reactions to unpredictable events, while *deliberative behaviour* involve deliberative reasoning about how to commit tasks. According to these criteria we will divide h-agents in this section.

For h-agents created by our application we will use the term *ent* (this world comes from *entity*).

2.1 Deliberative agents

Deliberative h-agents are such h-agents that act not only reactively, but also intentionally. These h-agents are typically based on cognitive paradigm.

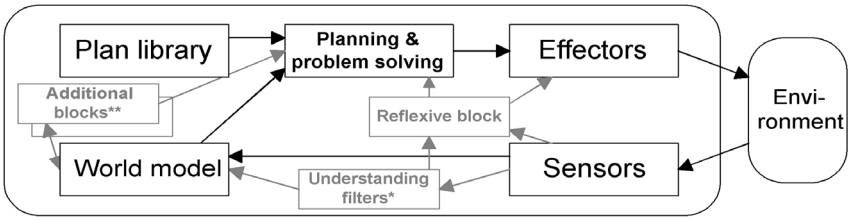

Fig. 1. Elementary block scheme of deliberative h-agent. Gray blocks are optional. (*)Understanding filter is for example a linguistic module, (**)an example of additional block is an emotional module or vegetative system block.

In both continuous and discrete time environments, it is usually not possible to deliberately plan behaviour of h-agents using STRIPS-like method [20], because the simulation can not be paused until the plan is finished, and an agent must also be reactive in some way; in such case the plan would have to be recomputed.

Therefore plans how to achieve goals must be somehow *explicitly* prescripted and pure planning can be allowed only at times. The key function of a mind would be then an opportunistic switching among prescripted plans to find the one, which corresponds the best with the current situation in the environment. Another alternative is to use any-time planning.

Well-known approach is *Belief-Desire-Intention architecture* (BDI) [25]. The mind of BDI-h-agents updates the internal world model (*i.e.*, beliefs), generates opportunities accordingly (*i.e.*, desires) and chooses among them (*i.e.*, commits intentions). System JAM [13], which is a second-generation descendant of Procedural Reasoning System [10], provides an interpreter for BDI-architecture. H-agents like synthetic actors for interactive improvisational plays [15] are created in JAM. Expression power of JAM language is similar to E language.

Another example of interpreter based on BDI-architecture paradigm is Jason [3]. Based on similar techniques, Mateas presents ABL/Hap language [19] for designing h-agents in virtual storytelling. Comparing with project ENTs, JAM, Jason and ABL/Hap are architectures or high-level programming languages, rather than complex tools with virtual environments.

An alternative to BDI is a *hierarchical rule-based system*. An example is Soar—the implementation of unified theory of cognition [16]. Soar offers not only rule-based system, but also associative memory, meta-level reasoning and learning, as well as debugging tools. The disadvantage of Soar is that it is "low-level" and highly unintuitive for non-computer experts and therefore not applicable for prototyping (what is one of the goals of project Ent). An example of Soar-h-agent is Steve [23] or Quakebot [17].

Also some advanced planning techniques can be used to control h-agents. *Hierarchical planning* is used in Cavazza's h-agents for virtual storytelling [8] and

anytime planning is used in Nareyek's project Excalibur [21] that is aimed at h-agents operating in computer games. Both projects seem very promising, but they are not available in present time—therefore no additional comparison is possible.

Reactive planning represents a trade-off between planning and reactive techniques. Based on it, Bryson developed a methodology for behaviour-oriented design [7], and proved it to be successful in primates agents simulation. As she suggests, her method can be also applied in JAM or Soar, and as we show in section 4, also in E.

2.2 Reactive agents

Pure *reactive h-agent* is an h-agent that reacts only to external or internal events (caused by an environment or an agent's body, respectively). If an h-agent follows some goals, they are *implicitly* hidden in its architecture. Architecture of these h-agents is often based on connectionist paradigm or on a so-called novel approach in artificial intelligence. H-agents behaviour is typically created in "bottom-up" manner.

Various types of *networks* can be used to control reactive h-agents. Neural network controls animals in the computer game Creatures [12], Tyrrell uses hierarchical networks [24], Maes uses flat networks [18]. Unlike ents, agents controlled by networks are not h-agents in the sense that they do not carry out any complex tasks, but animal-like predator avoidance, eating or mating. Therefore their application domain is different than the one of ents. In addition, Tyrrell suggested that flat networks are much more complicated to design then hierarchical ones, and Bryson proved her reactive planning to be easier to design than Tyrrell's hierarchical networks [6].

Novel approach to artificial intelligence came with Brooks and his robotic *subsumption architecture* [5]. Brooks opposed to the cognitive approach, because of the complexity of the external environments to be represented in a central manner. He and his followers do not use central world representation, instead they decompose behaviour into independent blocks that operate concurrently, and each block holds its own information, if it needs it. Because of complexity of h-agents environments, it seems meaningful to take inspiration from this approach, as proven by project Fear [9], a platform for prototyping h-agents in first-person shooters games. It combines subsumption architecture with several different techniques like neural networks or genetic algorithms. Fear is a well-documented complex tool, but serves only programmers of computer games—it uses 3D Quake environment and its h-agents are programmed in C++. Another example of a tool based on subsumption architecture is InViWo toolkit [22].

Main disadvantage of novel approach is that overall behaviour *emerges* from interaction of individual blocks and therefore it is difficult to design h-agents to fulfill complex tasks (see [25] for discussion). It is problem both in Fear and InViWo—these tools are too "low-level" for non-computer experts. This is also reason why we follow cognitive approach. Nevertheless, cognitive approach has also some disadvantages, and in the section 5 we will discus possibilities of combination of several approaches.

3 Project ENTs

This section describes the first generation of project ENTs. We will start with an overall description, then we will focus on the environment and the body of h-agents, and finally on the mind of h-agents. The whole project is detailed in [2].

3.1 Application architecture

Project ENTs consists of three parts: 1) environment server (ES), 2) graphical user interface (a GUI) and 3) simulator of individual ents. All three parts have been developed as independent applications for Linux platform, so a user can use them separately. ES is a server to which several GUIs and ents can be connected. ES can instantiate different world models. Each GUI is able to simulate one user avatar.

Simulation time goes in discrete time steps. Space is divided into rooms separated by doors, each room is divided to square tiles. Every tile can contain various objects.

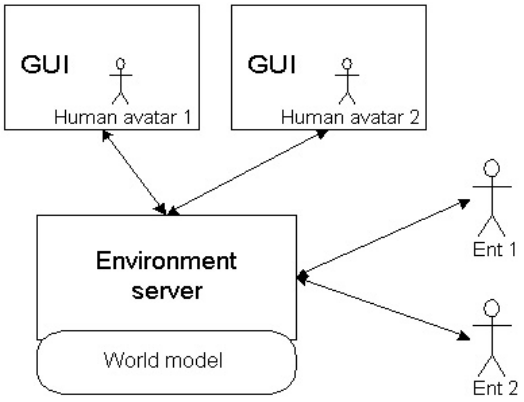

Fig. 2. Architecture of the first generation of project ENTs

3.2 Virtual environment and the h-agents' body

The purpose of ES is to simulate artificial environment, it means to simulate all objects, and acting and moving of ents' bodies. A user can instantiate different world-models. As an example, a model of family house is included in the current release.

ES recognizes about 100 types of objects. This set has been chosen to enable modeling of worlds similar to natural human environment. The set is fixed. The ent can take or manipulate with most objects.

Bodies of ents are described just by several attributes (like position, hunger, thirst or sleepiness). All ents are autonomous, embodied h-agents, subject to constraints of

the environment, and therefore all their attributes are controlled by ES. Attributes like hunger or thirst that represent the needs of the ent are automatically increased in time and the ent can satisfy them just by acting in the world. The same rules stands for human avatar.

Ents have two hands and they face no particular direction in their environment (the illusion of orientation is caused by the GUI—see Fig. 3).

Fig. 3. A world of ents. From the left: a user-avatar and an ent holding a book.

Sensing and acting. The ent (or the human avatar) senses events and objects only from the room it is located in (it can not see objects in closed containers). Sensing is passive. Whenever the ent comes into a room, it receives complete information about position and the state of objects in the room from ES. During next simulation steps it receives only changes from previous state—we call this information *delta*.

Atomic instruction (a-instruction) is a basic action the ent can do. All a-instructions last one time step. In every time step, the ent sends one a-instruction to ES. ES returns information about its success together with the delta in the next time step.

An example of an a-instruction is:

```
aWatering( who, withWhat, whichBed )
```

Each parameter uniquely identifies an actor (an ent), an object, or a subject, respectively. In this example `who` identifies the ent, `withWhat` identifies the can the ent is holding (*i.e.*, the subject), and `whichBed` identifies the bed which is being watered (*i.e.*, the object).

ES recognizes about 70 predefined a-instructions.

3.3 The mind of an ent

The idea behind architecture of ents mind is based on cognitive paradigm. That means the environment state is mapped into a symbolic representation transmitted to the ent as percepts (in a delta). When percepts are received, they are stored in a central memory of the ent to be later used by a control unit. A-instructions are generated from scripts written in E language in accordance with the information stored in ent's memory (*i.e.*, actual and past percepts, and internal state of the ent). The overall architecture is based on Fig. 1.

Memory. Central memory consists of a list of records called *memory-sentences* (m-sentences). They represent facts about the world and the ent, such as:

```
to_be_what_where_since( object, position, time )
```

Each m-sentence has its *plausibility*; the information about the relevance of the m-sentence. Memory module automatically decreases the plausibility in time; the ent can forget the m-sentence eventually. All m-sentences are based just on percepts or are created by an internal component of the ent—no secret connection from the memory to outer world is provided.

E language. The core of ent's mind is an interpreter of E language. The basic construct of the language is a *behaviour-script* (b-script). Highest-level b-scripts are called *top-level* and have *priorities*. Priorities are functions of time and are either constant or so-called *trapezoidal* (see Fig. 5 for the example). Some b-scripts (including top-level ones), can have *prerequisites*—we call these b-scripts *interrupts*. The purpose of interpreter is to decompose top-level b-scripts to a-instructions.

At one time, more top-level b-scripts can be active, but only one can be executed at a time—the one with the highest priority. A top-level b-script became active *iff* (a) its prerequisites became valid and (b) its trapezoidal priority is increased to more than zero (thus, top-level b-scripts without prerequisites and with constant priority is active all the time—see "bumming around" at the Fig. 5). Top-level b-scripts are tied up by a simple scheduler written in E.

In each time-step, the interpreter performs an internal cycle. In each loop of the cycle the interpreter can perform one of the following: activate an interrupt, instantiate a sub-b-script, ask the memory, run a computational script, or send an a-instruction. By sending an a-instruction the current time-step ends and the cycle is finished.

Sub-b-scripts can have more variants—the interpreter performs "a meta-level reasoning" by computing a *utility function* for each of the variant. Active top-level b-scripts with variants of sub-b-scripts can be seen as a set of AND-OR trees.

Statements of E language include: if-then construction, for-cycles, RERUN, COMMIT and FAIL statements and optional Prolog-like backtracking. Custom C-extensions are also possible. Denotation of RERUN is to restart a b-script and to reinitialize local variables, while denotation of COMMIT and FAIL is to interrupt a b-script as satisfied or failed, respectively.

Basic data type of E is a *handle* and a *list*. The handle is an integer number or identifies uniquely an object (*e.g.*, the ent, the carrot etc.) or a group of objects (*e.g.*, any carrot or all carrots). The list is a list of more objects, with their properties optionally.

By means of b-scripts, various cognitive techniques can be implemented; including BDI-architecture and reactive planning. Library of low-level behaviours like path-finding or object-finding is provided, as well as a debugger of b-scripts.

Linguistic module. The user (it means his or her avatar) can talk with ents in a simplified variety of Czech language. Although only sentences from predefined set of templates can be used, the conversation can be more complicated than Eliza-chats. For example, declension is implemented, as well as a *concretization*—that means finding out which object is exactly the current conversation topic (see Fig. 4).

The purpose of linguistic module is to map natural language sentences to the m-sentences (and vice versa), and to manage a communication, in particular a process of concretization. The linguistic module is an optional extension and custom modules can be provided.

4 Example of an ent—a gardener

The current implementation includes a model of a family house with prescripted ents—a gardener and a musician. This section briefly describes some top-level tasks of a gardener, in particular a task of watering a garden based on reactive planning.

In the Fig. 5 are depicted priorities of morning tasks of the gardener. Trapezoidal priority of watering causes that the gardener, after waking up, goes to water all beds. When the timeout expires at about 8 a.m., the gardener switches to bumming around. If it wants to eat or to go to the toilet (or is asked by a user to perform a command), watering or bumming is interrupted by appropriate goal with higher priority.

To evaluate E universality, we have written b-scripts using several techniques, namely BDI, reactive planning and hierarchical planning. B-script for watering presented here is inspired by reactive planning methodology developed by Bryson [7].

The final goal of watering is: *there are no known dry beds*. Its subtasks are: *to find a watering-can*, *to fill the can*, *to find a bed*, and *to water the bed*. They can appear in any order or even with a cycle (due to more beds and the fact that the can may be emptied). In addition, watering may be interrupted by a goal with higher priority (like going to eat). For path-finding or object-funding subtasks are used predefined scripts from E-library.

After the task is finished, the ent must perform cleaning: *to empty and put down the can*. The code-schema for the watering is on the Fig. 6. Notice local interrupts, which restart the watering (by RERUN statement). They are activated in order given by their local priorities (which do not influence the priorities of top-level interrupts). Thanks to restarting technique, cycles can appear in the behaviour.

User: Take the blue can.
Ent: Which one: the one in the garden, or the one in the cellar?
User: The one in the garden.
(Ent goes for can and takes it.)
User: Put it to the case.
Ent: Which case: the one in...?
User: To <this one>.*(User points at the case by mouse.)*
(Ent puts the can to the case.)

Fig. 4. Conversation with an ent—concretization process (translated from Czech language).

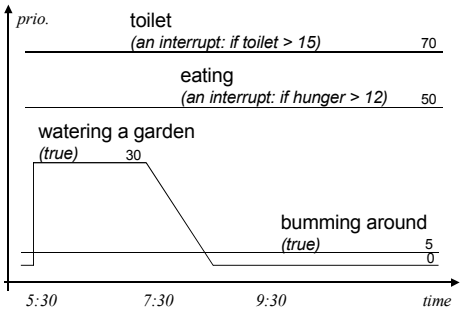

Fig. 5. Priorities of morning top-level tasks of the gardener. Values of priorities are written on the left. Notice trapezoidal priority of watering and prerequisities of interrupts.

```
top_levelGoal_WaterAllBeds :-
      // if everything is watered, try to put the can and commit
  if GOAL_COND then { try sgPutCan, COMMIT } fi,
      // if you are not holding a can, find it and take it; then activate a local interrupt
      // that tests whether the can is still at hands -- if not, restart the watering
  if ! holdCan then sgFindAndTakeCan fi,
  localHook( ! holdCan, localPrioMax-1, { RERUN }, id1 ),
      // if you are not holding an empty can, fill it; then activate a local interrupt that
      // tests whether can is not empty -- if it is, restart the watering
  if holdCan and canInHandEmpty then sgFillCan fi,
  localHook( holdCan and canInHandEmpty, localPrioMax-2,
             { RERUN }, id2 ),
      // follows the same for other subgoals...
  ...
```

Fig. 6. Schema of a b-script for watering a garden. Subgoals start with `sg`. Local interrupts are activated by `localHook` statements and their prerequisities are checked in every time-step.

5 Evaluation and future work

The project ENTs has proven to be successful in deliberative h-agents prototyping, however some final goals (see section 1) have been achieved only partially. The main point is the lack of tool universality. In this section we go through some issues the critics may point to, and suggest solutions for second generation of the tool.

Cognitive paradigm and robustness of E. Imagine what happens when an ent comes to the garden. It receives an update of 2197 carrots and the speed of the system goes down. This problem (and other similar ones) could be solved by techniques not based on cognitive paradigm (such as active sensing or distributed memory)—but these techniques are not supported in our tool, in particular by E language and memory module.

Solution: When we perceive each individual h-agent as a multi-agent system [25]*, various different techniques and paradigms can be combined, for example every component could have its own memory. Such a distributed system with no central element could profit from advantages of subsumption architecture* [5]*, while avoiding its disadvantages. In addition, each component can theoretically be coded in different language (not only in E). See* [4] *for more detailed discussion on this topic.*

Environment flexibility. Even though a user of ENTs can instantiate different world models, the world topology and the set of objects and a-instructions remain fixed.

Solution: Imagine a user who wants to add new object and new a-instruction to an environment. How an h-agent recognizes them? A theory of affordances *is applied here—each object "manifests" in the environment its attributes and actions the agent can do with it* [11, 14]. *For topology we aimed at following* way-point *approach—the ent follows nodes (way-points) in a graph* [9].

Reusability. Although every ent is an independent program, it is hard to connect it to an external environment. Thus, ents cannot be used in another projects.

Comment: This is a never-ending problem with interfaces, and in fact, our tool is focused on prototyping, and not as a universal interface. Nevertheless we plan to use world-agent standards for artificial intelligence in computer games [1].

Considering all the above-mentioned problems, we come to the following conclusion: The reason, why project ENTs achieves some of its goals only partially, lies in its relatively small flexibility and robustness. Specifically, it utilizes on purely cognitive approach (neither connections, nor novel approach) and its three main components (ES, simulator of individual ents and a GUI) are designed as monolithic blocks, rather than multiple-component systems.

It is quite clear that a *fully universal* tool for prototyping all types of h-agents will probably never exist, but we believe the universality of our tool will increase significantly with solutions suggested above.

6 Conclusion

In this paper we have presented a first generation of project ENTs—a tool for human-like artificial agents prototyping. Modular virtual environment and a universal E language for cognitive behaviour-design have been introduced as main parts of this tool. The contribution of this tool is twofold. First, students and researchers can use it as a test-bed for cognitive human-like agents. Second, the whole tool serves as a large

case-study for more general application. Requirements for the second generation of the tool would not have been available without previous experience. These requirements have been also presented.

Acknowledgments. The application ENTs was developed as a student project at Faculty of Mathematics—Physics, Charles University, Prague. Thanks to Vladislav Kuboň for supervising the project and to David Voňka and Mikuláš Vejlupek for their contribution. Thanks also to Rudolf Kryl, Tomáš Bureš and Kamamúra-Group for their suggestions and comments.

References

1. Artificial Intelligence Interface Standards Committee (AIISC): `www.igda.org/ai`
2. Bojar, O., Brom, C., Hladík, M., Toman, V., Vejlupek, M., Voňka, D.: *Documentation of project ENTs*. `http://www.enti.lit.cz`. Faculty of Mathematics—Physics, Charles University. Prague (2002) (in Czech)
3. Bordini, R. H., Hübner J. F.: Jason: A Java-based agentSpeak interpreter; project homepage: `http://jason.sourceforge.net/`
4. Brom, C.: Towards Architecture of Human-like Artificial Agent: Project ENTs. In: *WDS'04 Proceedings of Contributed Papers*, Prague, Matfyzpress (to appear)
5. Brooks, A. R.: Intelligence without reason. In: *Proceedings of the 1991 International Joint Conference on Artificial Intelligence*, Sydney (1991) 569-595
6. Bryson, J.: Hierarchy and Sequence vs. Full Parallelism in Action Selection. In: *The Sixth International Conference on the Simulation of Adaptive Behavior (SAB00)*. MA. MIT Press, Cambridge (2000) 147-156
7. Bryson, J.: The Behavior-Oriented Design of Modular Agent Intelligence. In: Müller, J. P. (eds.): *Proceedings of Agent Technologies, Infrastructures, Tools, and Applications for E-Services*, Springer LNCS 2592 (2003) 61-76
8. Cavazza, M., Charles, F., Mead, S. J.: Developing Re-usable Interactive Storytelling Technologies. In: *IFIP World Computer Congress 2004*, Toulouse, France (to appear)
9. Champandard, A.J.: *AI Game Development: Synthetic Creatures with learning and Reactive Behaviors*. New Riders, USA (2003)
10. Georgeff, M., Lansky A. L. Reactive Reasoning and Planning. In: *Proceedings of the 6th National Conference on Artificial Intelligence*, Seattle, Washington (1987) 677-682
11. Gibson, J.J., The Theory of Affordances, In R. Shaw, J. Bransford (eds.): *Perceiving, Acting and Knowing*. Hillsdale, NJ: Erlbaum (1977)
12. Grand, S., Cliff, D., Malhotra, A.: Creatures: Artificial life autonomous software-agents for home entertainment. In: Johnson, W. L. (eds.): *Proceedings of the First International Conference on Autonomous Agents*. ACM press (1997) 22-29
13. Huber, M. J.: JAM: A BDI-theoretic mobile agent architecture. In: *Proceedings of the 3rd International Conference on Autonomous Agents* (Agents'99). Seatle (1999) 236-243
14. Kallmann., M., Thalmann, D.: Modeling Objects for Interaction Tasks. In: *Proceedings of EGCAS 98* (1998) 73-86
15. Klesen, M., Szatkowski, J., Lehmann, N.: A Dramatised Actant Model for Interactive Improvisational Plays. In: *Proceedings of Intelligent Virtual Agents* (IVA 01), Springer LNAI 2190, Berlin (2001) 181-194
16. Laird, J. E., Newell, A., Rosenbloom, P. S.: SOAR: An Architecture for General Intelligence. In: *Artificial Intelligence*, 33 (1) (1987) 1-64

17. Laird, J. E.: It Knows What You're Going To Do: Adding Anticipation to a Quakebot. AAAI Technical Report SS-00-02 (2001)
18. Maes, P.: A bottom-up mechanism for behaviour selection in an artificial creature. In: Meyer J.-A., Wilson, S. (eds.): *From Animals to Animats*. Cambridge, MA, MIT Press (1991) 478-485
19. Mateas, M.: *Interactive Drama, Art and Artificial Intelligence.* Ph.D. Dissertation. Department of Computer Science, Carnegie Mellon University (2002)
20. Lažanský, J.: Plánování. In: Mařík, V., Štěpánková, O., Lažanský, J.: *Umělá inteligence (1)*, **8**. Academia, Prague (1993) 184-216 (in Czech)
21. Nareyek, A.: Intelligent agents for computer games; Project Excalibur homepage: `http://www.ai-center.com/projects/excalibur/`
22. Richard, N., Codognet, P., Grumbach. A.: The InViWo Toolkit. In: *Proceedings of Intelligent Virtual Agents* (IVA 01), Springer LNAI 2190, Berlin (2001) 195-209
23. Rickel, J., Johnson, W. L.: Animated Agents for Procedural Training in Virtual Reality: Perception, Cognition, and Motor Control. In: *Applied Artificial Intelligence*, 13. (1999) 343-382
24. Tyrrell, T.: *Computational Mechanisms for Action Selection*. Ph.D. Dissertation. Centre for Cognitive Science, University of Edinburgh (1993)
25. Wooldridge, M.: *An Introduction to MultiAgent Systems*. John Wiley & Sons (2002)
26. Waveren, J. M. P. van: *The Quake III Arena Bot*. Master thesis. Faculty ITS, University of Technology Delft (2001)