Issues and Approaches in Artificial Intelligence Middleware Development for Digital Games and Entertainment Products

Börje Felipe Fernandes Karlsson
Centro de Informática
Universidade Federal de Pernambuco (UFPE)
Av. Professor Luis Freire s/n
Recife, PE, Brazil
CEP 50740-540
+55 81 3271-8430
bffk@cin.ufpe.br

ABSTRACT

This work presents issues and approaches regarding the creation of artificial intelligence (AI) middleware to aid the development of digital games and entertainment products in general. It starts with a discussion of the concept and context of an AI middleware (emphasizing the relations of traditional AI areas with computer games).

Then, some approaches to the problem of creating an AI middleware are presented, followed by a taxonomy regarding design methods and componentization, and related research. Finally, we discuss the impact of such middleware, open issues to be addressed and future directions.

Keywords

Artificial Intelligence, AI middleware, computer games.

INTRODUCTION

As the market of digital entertainment products (especially digital games) grows, these products get more and more complex and their users present higher and higher expectations, requiring quality and believability in the character behaviours. Because of these facts, artificial intelligence (AI) functionalities are no longer held in a secondary level during development. Despite of this, AI applied to entertainment products development, remains a complex domain and relatively unexplored. A great source of solutions and techniques is the academic community. Almost every game genre can benefit from what the many AI areas have to offer in the creation of more realistic and interesting environments and NPCs (Non Player Characters) or even in the control of the narrative flow.

Besides the previously cited, the application of artificial intelligence in digital games offers very interesting challenges and advantages from a research point of view. [10], for instance, presents as advantages for the AI community, the growing realism in today games and the fact that game developers create support for others to easily modify the games (building the so called *mods*); what turns games in a highly attractive alternative to the creation of complex simulation environments. The complexity in current games is reaching a level comparable to the real world, but on the same time allowing the researchers to focus on the AI issues and forget about real world motors and sensors. In other words, digital games provide a great test-bed for artificial intelligence techniques and models (as shown for example in, [9] and [1]).

As challenges for the AI research community, digital games offer dynamic environments that require complex decisions (often based on incomplete knowledge of the world), require real-time responses (limiting the available time for reasoning), and may have to handle resource management [16]. Also, games provide an environment where it is possible to develop and test machine learning techniques, human-computer interfaces and interaction, knowledge representation and intelligent architectures [12].

With the ever growing necessity of AI functionalities and the fact that some techniques are already in use in game development, supporting tools were created in order to help with these tasks but these tools presented little flexibility. Without a certain level of support, a developer will spend a great part of his time struggling with low-level details or re-implementation of common functionalities. It is believed that the next big step in the quality of AI game techniques depends on the creation of AI middleware, to alleviate developers and allow them to concentrate on creative tasks related to AI.

In the following sections this work presents a discussion of the concept and context of an AI middleware (emphasizing the relations of traditional AI areas with computer games). Then, some approaches to the problem of creating an AI middleware are presented, followed by a taxonomy regarding design methods and componentization, and related research. Finally, we discuss the impact of such middleware, open issues to be addressed and future directions.

DEFINITION

In other areas related to computer games development, such as computer graphics, networking support and physics modelling, it is already a common approach to use (or at least to know about) pieces of software that help the digital games developer in the creation of the games, allowing them to focus on the creative side of the game. This is only now staring to be seen in the artificial intelligence segment of game development.

An Artificial Intelligence middleware is basically a layer of software that provides services for the game engines for performing the AI functions in a computer game. Usually also called an Artificial Intelligence Engine, AI middleware handles the process of producing the desired behavior or decision-making of intelligent agents present in the game world.

APPROACHES

Traditionally, computer games developers use a small set of techniques over and over again in the implementation of artificial intelligence functionalities in games, specially Finite State Machines (FSMs) and Fuzzy Finite Sate Machines (FuFSMs) [24] that are basically a set of states and transitions between these states, used to represent some kinds of behaviours; the A* algorithm [24] used for calculating paths; and some Artificial Life (A-Life) techniques such as steering behaviours [23] that can give a bigger realism to movements. But even with this small set of techniques, it is possible to achieve pretty good results. Also some games make use of Decision Trees and Production Rules when some sort of reasoning is necessary.

The CPU time available for calculations related to the characters behaviours is usually very short, and so, the tasks related to it can not be so CPU intense. That's why lots of computer game titles utilize simple approaches as FSMs (where the transitions between states usually reflect some game world events) or rule-based systems (RBSs), which are more flexible than a purely stimuli-response approach (that according to [16] was the standard procedure for implementing behaviour until very recently) for they allow objects to incorporate a internal state, making it possible to achieve longer term goals and FSMs and simple RBSs are as fast as the pure if-then structures used when implementing the stimuli-response approaches.

However, the limitations of the finite state machines approach in the design of intelligent agents are well known, FSMs are limited specially by combinatorial explosions (as the environment complexity grows, so grows de FSM states and transitions sets because the FSMs have to predict all possible cases and situations on the environment) and by the potential repetitive behaviours, because the FSMs have a fixed set of states and transitions, if the same situation happens twice, the behaviour will be the same both times. This limitations were "attacked" with the creation of hierarchic finite state machines (HFSMs), a extension to traditional FSMs that allows the creation of composite states which contain, inside themselves, other states and transitions resulting in the creation of a hierarchy of FSMs; and the creation of Fuzzy FSMs (FuFSM) that add characteristics of fuzzy logic to FSMs in order to lower the previsibility of actions. All of this being supported by the creation of visual editors that help the creation and maintenance of theirs state diagrams.

An example of tools that uses this kind of approach is the SOFTIMAGE/BEHAVIOUR [28] toolkit, a toolkit for computerized animation that makes use of HFSMs in order to allow better representation and control of big real-time systems complexity, as complex crowds for example, allowing the animator to give intelligence to this crowd and to individual characters inside it. It also has its own visual tool for visual creation and editing of the HFSMs and supports a script language that enhances the tool flexibility.

With relation to the RBSs, they presents some advantages, as: they correspond to the way people usually think about knowledge, are very expressive and allow the modeling of complex behaviors, model the knowledge in a modular way, are easy to write (and debug, when compared to decision trees for example) and are much more concise than the finite stare machines. However, RBSs may have a large memory footprint, require a lot of processing power and even in some situations become extremely difficult to debug.

SimBionic [25] presents another approach to the design of AI support software that matches somewhere in between FSMs and RBSs, by providing a framework for defining the objects that display behavior within the game world. This framework is very state-oriented, supporting the creation of complex hierarchical state systems.

These state systems have several components that can be classified as descriptors and declarations. Descriptors are identifiers used to represent objects and behaviors that exist in the game world. SimBionic also describes attributes of objects as descriptors, for example, a Weapon descriptor could have two son descriptors Revolver and Rifle, Rifle in turn could have a son AR-15, and so on; and the gun attributes as weight, ammunition, would also be represented as descriptors. Declarations are symbolic associations used by the SimBionic project. These associations consist of actions, predicates (built-in functions that provide access and evaluation services), behaviors and variables. A behaviour implies the selection of either another behaviour or an action, and is responsible for the “decision making” in SimBionic.

As an example of the usage of decision trees we have the AI-implant [2] tool, an animation control engine that was designed to introduce AI to the computer game and video media character development process. Essentially, AI.implant provides autonomous character control for the game engine or animation engine offering pre-defined behaviors (like "Avoid Obstacles" for example) are assigned to the “agent” to implement a desired set of actions (preventing a guard from walking into a wall). Sensors are created for the “agent”, letting it perceive events in the game world, and based on those events use binary decision trees (BDT) to choose a course of action. The BDT can be used to create complex decisions of arbitrary depth. It is even possible to construct a finite state machine (FSM) using the BDT appropriately. As AI.implant work closely with 3D modeling software, it also provides functionalities for the placement of waypoints in the terrain to help the characters path-finding.

These techniques are very powerful, but sometimes they are also too simple. Alternatives motivated by academic research produced a rich variety of approaches for the creation of interesting and plausible intelligent agents. But unfortunately, this approaches tend to be very “heavy”, requiring “state evaluations” that are very processor intensive for each frame. As, for instance, deliberative systems, that have goals, a world model e can plan several steps necessary to achieve the goals, but are heavy and slow; or hybrid systems that calculate the planning beforehand and make use of rules in the lower levels (during runtime), but because of this pre-calculation of the plan, do not present good responses in highly dynamic worlds (as is the case with most computer game worlds).

Digital games developers need a solution in between the simple finite state machines and heavyweight complex cognitive models. First of all, it is necessary to avoid too frequent updates, and to rely in an event based model. In second place, the new solution needs to avoid presenting too much repetitive behaviours, exhibiting a variety of actions and reactions but still being verisimilar/acceptable ones. Third, the solution must provide a simple framework on which it may be possible to create highly customized new solutions; this framework shall also allow for scalable development. A possible approach would be to make use of “anytime planners”, planners that can improve the plan after each iteration, if there is little CPU time available, the system still manages to be reactive, and the plan gets to be refined if there is a bigger slice of available processor time, and also where the plan can be adapted/updated in the case of changes occurring in the game world, as for instance [8]. There are several options and approaches related to planning (as the presented one, belief-desire-intention (BDI)[21], HAP[14] and [15] this one exposing representational planning techniques to be used as a layer above the artificial intelligence mechanisms already in place, that explicitly handle the world representations in order to be able to choose the best behaviours for a given situation) but unfortunately, this approaches to planning are seldom known/used in computer games development.

Another separation of approaches could be “Functionality libraries” vs. “Agent based”, where the functionality library is a set of functions and algorithms, simple but efficient; and the agent based approach facilitates distribution and flexibility. The library approach is usually considered the best one because from the set of functions available, one can create an agent infra-structure, but it is not possible to turn a agent-based approach into a set of functions to be used as the developer wants; this without taking into account the fact that choosing an approach that imposes a certain methodology or architecture to digital games development may not be viable for some products, platforms or game genres. And also, there is a great interest in creating a wrapper to make use of a set of simple functions for the creation of intelligent agents [17].

Some approaches use inference engines to “conclude” what is the best course of action, one such example of inference engine, initially used in military simulations, is the Soar [27] architecture which combines the reactivity of stimuli-response systems with the context sensitive behaviours found in systems that use FSMs or scripting. In Soar, the knowledge is represented as a hierarchy of operators. Each level in this hierarchy represents a more specific decision level in the intelligent agent behaviour. The top level operators represent the agent goals and behaviour modes. The second level operators represent high level tactics used to achieve the goals specified in the higher level.

The lower level operators represent the steps (or sub-steps) necessary for the agent to implement and execute the tactics. The Soar architecture allows the persistence of the chosen operators as well as the persistence of the agents' internal memory, what allows the intelligent agents to react taking into account the environment context.

Yet another distinction can be made between the described functional approaches and the biologic/evolutionary approaches [7]. An example of such approach is the one in DirectIA [6] that provides tools and support for the creation of agents, with behaviours ranging from basic reactions to deep world state analysis. DirectIA is an agent-centric tool, which makes significant considerations of how and why a character makes a decision, with inspiration in the modeling of human and animal behaviours. Using a motivational model, i.e., an action selection mechanism that mimics the mechanism in animals. The DirectIA mechanism handles stimuli, emotion, states, motivations, behaviours and actions.

Such systems may be seen as a set of motivations that compete until it is decide which one must be applied to the situation, given the internal and external states, resulting in an emergent behaviour. The intelligent agents can weight tradeoffs, learn from their own experience and present behaviours nor specifically programmed by the game developers.

One can not forget the approaches that make use of machine learning techniques integrated into the artificial intelligence engine in order to obtain new (possibly emergent) behaviours [5].

DESIGN AND COMPONENTIZATION

Several issues are of great importance and must be examined when trying to create an artificial intelligence middleware. For example, in strategy games, it is necessary to apply AI in a strategy level applied to groups of unities, but it is also necessary to apply AI to an individual unity in order to have a certain autonomy without the player direct control.

Low level details are extremely important [19][20] to guarantee a good performance of the system implementation as well as free the game developer from the burden of having to implement this solutions every time, as for example: using event handling instead of polling(the act of continually asking for a resource until it is available), try to centralize the cooperation of game managers and execute the artificial intelligence procedures with the lower possible frequency in order to reduce the processing load, use AI with level-of-detail (LOD, when an object is in direct view of the player, it has to behave perfectly, but when it is far from it, the object can have a lower precision), make use of as much pre-processing as possible (pre-processing allows for the separation of decision support data from the level design itself, and also allows a better efficiency in the decision making process. And when combined with a dynamic action selection model, one gets a more robust and generalized solution), give preference to emergent behaviours instead of using scripted solutions. Always paying attention to a clear separation of the AI subsystem from the rest of the computer game implementation, because we are dealing with artificial intelligence middleware creation and not with a solution specific to a single problem.

The AI engine must present an execution cycle like the one presented bellow [11]:

1. Sense: Access the game world sensorial info

2. Think: Process the acquired knowledge (sensorial info), taking into account the current game world state and choose an action

3. Act: Execute the actions

In order to achieve this goal, the interface with the game world is a crucial point, and must follow two simple principles: be as close as possible to the one available to the human being (or the one that would be available to the entity being modeled) and have direct access to data structures, avoiding the unnecessary complexity associated with "simulating" the sense (in order to accomplish this and still have the AI functionality separated from the game itself, it will be necessary to write "glue code", code that will get the game native structure and convert it to a format that the AI module can understand). For example, in order to a monster sense the presence of the player character, the monster must see the player, the interface here is the "see" sense, but it should not be necessary to have a model of the screen and by applying image processing techniques try to recognize the player, it should suffice to access the relevant data that shows if the player is in front of the monster or not and convert it to a structure recognizable by the artificial intelligence middleware.

The game world is the master simulator; it manages the game state and also provides services related to it. This services are exactly the sensors and actuators, the sensors being able to receive events or to perform polling (try to get the information) if necessary, and the actuators represent the actions (short durations actions, such as jump or fire a gun) and effectors (actions with a longer duration and that are subject to game world changes). Another important aspect is the way in which the game world is modeled, because depending on the game world structure different interfaces (or glue code) may be necessary, as well as the cooperation with other game modules, as for example, the physics modeling engine (which some people consider part of the AI engine) in order to translate steering behaviours in real movement.

Another option related to game world interfacing would be to establish standard interfaces for a well defined functionality set (path-finding, influence maps, etc.), what also faces problems as how to correctly characterize concepts such as path-finding for example (as cited in [17]) or on what level to define this interfaces, in a basic level as managing a node list in a A* algorithm implementation or complete interfaces for problems such as path-planning (that use path-finding)?

The componentization (separation in components) is very similar to the presented approaches. Most artificial intelligence engines can be categorized as packs of similar functionalities, layered approaches or semi-independent components. Some of these packages could be: planners, path-finding, decision making, actions, sensing, infra-structure, learning, communication, emotions, motivations, coordination and tactical analysis.

One example of artificial intelligence middleware that is organized as sets of functionalities is Pensor [18] that is composed of a set of algorithms/techniques/technologies re-combinable to each individual project. Pensor has six planners, three of them being optimized path-finders (path-planners) that take into account terrain analysis information when calculating the paths; a decision module that implements finite state machines and fuzzy FSMs, rules and a shell; an actuator module that implements basic physics modeling and several steering behaviours; a perception module that provides several kinds of sensors and an infra-structure module that provides support for task priorities and resource managing for each task, as well as a scripting language interpreter.

DirectIA, that was already presented above, provides different components tightly grouped, a motivational engine to model the emotions and needs of the intelligent agents, a behavioral engine to model the agent's decision processes, a communication engine that supports communication between the agents, a perception engine, an action engine, and a knowledge engine to store each agent's representation of the game world. But it may also be considered as a layered approach, as the high level functionalities provide a agent-based approach, and the lower level functionalities provide support for commonly used techniques as path-finding and steering.

Yet another commercial artificial intelligence middleware is RenderWare AI [22] that presents a more purely layered design focused on the developers, providing a architectural layer, a services layer, an agents layer (actually it is a behaviour layer) and a decision layer. The agents layer being a set of behaviours that can be instantiated by a game character (behaviours such as flee, renged attacj, etc.), Renderware AI offers a set of C++ classes that the developers can extend and tweak to create their own agents and also offers FSMs and neural networks for the characters decisions processes.

Almost every middleware available splits the game objects into two categories, thinking entities and passive entities to facilitate their representation inside the game world model.

Other possible components: [11] says clearly that the knowledge base containing the goals, tactics and independent behaviours of a computer game is its most important feature. Without that, the second part, the inference engine (the component responsible for taking the actual decisions), would not be useful at all. And [13] states that it is extremely valuable to construct a good set of sensors as a priceless asset in order to improve the player experience interacting with the game world allowing the creation of much richer environments by the game developers and designers. There is also Spark! [26] that is a fuzzy logic editor created to help the use of this technique in games.

An important design issue is who the target user for the tool is. Tools like SimBionic and AIi.implant for instance are targeted towards visual animators and game designers, presenting easy to use visual tools often integrated into 3D modeling software packages and using approaches that do not require programming language expertise. On the other hand, tools like DirectIA and Renderware AI are clearly focused on the game developers (specially Renderware AI) requiring the developers to have a good understanding of the lower level implementations of the game and often offering source code to be modified by the developers (not the case for DirectIA though).

CONCLUSION

Many AI middleware solutions are flourishing, both in commercial environments and academic ones, but little is known about the real possibility of creating a set of interface standards to ease the creation of artificial intelligence middleware and how these standards will relate to implementation issues. Other groups (aside from the game development community and the AI research community), as the military, are also interested in the establishment of these standards. Cooperation among the computer games industry and the academy is crucial in this endeavour for the creation of such standards even if the goals for each community are not the same.

Another issue that should be investigated is the analysis of the possibilities of creating some sort of artificial intelligence accelerator hardware (as happened with 3D graphics cards). It is believed that this could turn AI into a mainstream area in games and entertainment products, especially in the games console market.

Much more thought will have to be dedicated (and currently is being dedicated) to the question of standardizing artificial intelligence interfaces in computer games (shall they be in source code form? In form of an ontology? Following the AUML [4]?). Because of these issues, the International Game Developers Association (IGDA) has set up the AI Interface Standards Committee to develop such interface standards, the initiative being a joint effort of game AI developers, middleware representatives and academics [17].

Even with these issues still open, artificial intelligence in games will have a growing priority in the digital game development process for some time, because it is still and area little explored but that already showed that can bring great advances in game design and gameplay.

However, the "not invented here" syndrome (that is very common in the digital games industry), the fear of not having complete control over the game and a possible performance hit that may occur due to the AI middleware "engine" or library are some of the obstacles for the massive adoption of AI middleware by game development companies. Also, attention must be paid so that the learning curve for implementing and integrating the AI middleware into a digital game title might be too high.

Some other areas that deserve special attention and should present a huge growth in the near future are automatic learning (as the KnoMic-Knowledge Mimic[11] initiative), a higher interaction with the game characters [29] and initiatives in intelligent narratives [3] [30] to improve gameplay.

REFERENCES

1. Adobbati, R., Marshall, A.N., Scholer, A., Tejada, S., Kaminka, G.A., Schaffer, S., Sollitto, C. GameBots: A 3D virtual world test bed for multiagent research. In Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS, Montreal, Canada, 2001.
2. AI.implant - Advanced AI for Games, Animation and Simulation. URL: http://www.ai-implant.com/ (15/07/2003).
3. Amant, R., and Young, M. Interface agents in model world environments, AI Magazine. 22(4): 95-107. Winter, 2001.
4. AUML – Agent UML Web Site, URL: http://www.auml.org/ (05/05/2003).
5. Ding, Z. Designing AI Engines with Built-in Machine Learning Capabilities. In Proceedings of the Game Developers Conference 1999, San Jose, USA, 1999.
6. Direct IA. URL: http://www.directia.com (10/06/2003).
7. Donnart, J., Jakobi, N., Kodjabachian, J., Meyer, C., Meyer, A., Trullier, O. Industrial Applications of Biomimetic Adaptative Systems. Proceedings of HCP'99 - Human Centered Processes. ENST Bretagne Pub, Brest, France, September 1999.
8. Excalibur Project, URL: http://www.ai-center/projects/excalibur (05/05/2003).

9. Kaminka, G., Veloso, M., Schaffer, S., Sollitto, C., Adobbati, R., Marshall, A., Scholer, A. and Tejada, S. Gamebots: A Flexible Test Bed for Multi-agent Team Research, Communications of the ACM, 45(1), 43-45, 2002.

10. Laird, J. E., van Lent, M., Human-level AI's Killer Application: Interactive Computer Games, Proceedings of AAAI 2000, pp. 1171-1178, Austin, USA, 2000.

11. Laird, J., van Lent, M., Developing an Artificial Intelligence Engine. In Proceedings of the Game Developers Conference 1999, San Jose, USA, 1999.

12. van Lent, M., Laird, J. E., Buckman, J., Hartford, J., Houchard, S., Steinkraus, K. and Tedrake, R., Intelligent Agents in Computer Games, Proceedings of the National Conference on Artificial Intelligence, pp. 929-930, Orlando, USA, 1999.

13. Leonard, T. Building AI Sensory Systems. In Proceedings of the Game Developers Conference 2003, San Jose, USA, 2003.

14. Loyall, A. and Bates, J. Hap: A Reactive, Adaptive Architecture for Agents. Pittsburgh, PA: Carnegie Mellon University technical report CMU-CS-91-147, USA, 1991.

15. Martin, C., Representational AI Planning Techniques, In Proceedings of the Game Developers Conference 2003, San Jose, USA, 2003.

16. Nareyek, A., Intelligent Agents for Computer Games, In Proceedings of the Second International Conference on Computers and Games, pp 414-422, Hamamatsu, Japan, October, 2000.

17. Nareyek, A., Knafla, B., Fu, D., Long, D., Reed, C., El Rhalibi, A. and Stephens, N. (eds). The 2003 Report of the IGDA's Artificial Intelligence Interface Standards Committee. International Game Developers Association, June 2003. URL: http://www.igda.org/ai/report-2003/report-2003.html (21/07/2003).

18. Pensor. URL: http://www.pensor.net (10/06/2003).

19. Rabin, S. Strategies for Optimizing AI. In Game Programming Gems 2, Charles River Media, 2001.

20. Rabin, S. Designing a General Robust AI Engine. In Game Programming Gems, Charles River Media, 2000.

21. Rao, A. and Georgeff, M. Modeling Rational Agents within a BDI-Architecture. Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91), USA, 1991.

22. Renderware AI. URL: http://www.renderware.com/renderwareai.htm (10/06/2003).

23. Reynolds, C. W., Flocks, Herds and Schools: A Distributed Behavioural Model, Computer Graphics, 21(4), pp. 25 –34, July 1987.

24. Russell, S. and Norvig, P., Artificial Intelligence: A Modern Approach, Prentice Hall, 1995.

25. SimBionic. URL: http://www.shai.com/products/ (10/06/2003).

26. Spark!. URL: http://www.louderthanabomb.com/ (10/06/2003).

27. Soar. Soar Homepage. URL: http://ai.eecs.umich.edu/soar (20/05/2003).

28. Softimage Behaviour. URL: http://www.softimage.com/products/behavior/ (21/07/2003).

29. Stern, A., AI Beyond Computer Games, Presented at AAAI AI and Interactive Entertainment Spring Symposium 2000, Palo Alto, USA, March 2000.

30. Young, M. An Overview of the Mimesis Architecture: Integrating Intelligent Narrative Control into an Existing Gaming Environment, Working Notes of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment, AAAI Press (2001).