AI middleware as means for improving gameplay *

Börje Karlsson
ICAD/IGames/VisionLab - PUC-Rio
Rua Marquês de São Vicente, 225
Rio de Janeiro, Brazil
borje@icad.puc-rio.br

Bruno Feijó
ICAD/IGames/VisionLab - PUC-Rio
Rua Marquês de São Vicente, 225
Rio de Janeiro, Brazil
bruno@icad.puc-rio.br

ABSTRACT
Current commercial AI middleware are still far from being a generic and flexible tool for developing computer games. Also the literature lacks proposals in this field. In this work we present some of our current research on developing a new proposal for a flexible architecture that can be used in several types of games. This AI engine is designed to provide support for the implementation of AI functionalities in computer games, streamlining this implementation and allowing the developers to focus their attention on the creative side of the game, but is also focused on introducing new techniques that could allow for an improved gameplay experience or even new gameplay styles. In order to fulfill this goal, this research focuses on the design issues of such a system and its integration into games, using more powerful techniques from academic AI and strongly relying on software engineering principles.

Categories and Subject Descriptors
I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems - Games; D.2.11 [**Software Engineering**]: Software Architectures - Domain-specific architectures, Patterns

General Terms
Design, Experimentation, Standardization, Documentation

Keywords
Artificial Intelligence, Middleware, Computer Games, Software Architecture, Game AI, Design Patterns

1. INTRODUCTION
As entertainment applications (especially digital games) have being steadily growing both in size and complexity, the challenges faced when developing such software have also increased dramatically; their users now present higher and higher expectations, requiring quality and believability in the game environment and in character behaviours.

In the current stage of computer graphics, computer games can present astonishing scenes and animations but that alone does not guarantee a good game. The usage of Artificial Intelligence (AI) techniques has proved of great help to create a better user experience, even using only simple techniques. And an ongoing trend is to use more sophisticated AI to achieve the goal of creating a game environment as believable and fun as possible. With this ever growing necessity of AI functionalities and the fact that some techniques are already in use in game development for some time, supporting tools were created in order to help with these tasks but these first tools presented little flexibility. In other areas related to digital games (such as computer graphics, networking support and physics modeling) it is already a common approach the usage (or, at least, the evaluation) of software packages that help the developer in creating a game (this pieces of software are usually called game engines), and that is only recently starting to be seen in the game AI segment. Middleware technologies are taking an increasing importance and it is believed that the next big step in the quality of AI game techniques depends on the creation of this AI middleware, to alleviate developers and allow them to concentrate on creative tasks related to AI.

In the following sections this work presents what is an AI middleware, some approaches to the creation of such software, our proposed architecture and some comments and future directions on our research effort.

2. AI MIDDLEWARE
An artificial intelligence middleware is basically a software layer (or a set of components) that provides services to the game engine for performing the AI functionalities in a computer game. Usually also called an AI engine, the middleware is responsible for handling the process of producing the desired behaviours or decision-making of intelligent agents present in the game world.

The term middleware is usually associated with distributed system environments but is used here in the sense found in some citations from the 70s that define "middleware" as computer software which has been tailored to the particular needs of an installation. The comparatively new term at the time was introduced because, as some systems had become more complex, there was a need for systems enhancements

*(Produces the permission block, copyright information and page numbering). For use with ACM_PROC_ARTICLE-SP.CLS V2.6SP. Supported by ACM.

or modifications for particular needs; the programs that effected this were called 'middleware' because they came between the operating system and the application programs. Thus, middleware is software that mediates software interacting (in our context) with the main game engine. According to the Software Engineering Institute (SEI), middleware is defined as the software layer that lies between the operating system and the applications, providing uniform, standard, high-level interfaces to the application developers so that applications can be easily composed and reused. The role of middleware is then to make application development easier, by providing common programming abstractions, by masking the heterogeneity of the underlying system, and by hiding low-level programming details; which fits our needs just fine. Therefore architectural issues play a central role in middleware design. Architecture is concerned with the organization, overall structure, and communication patterns, both for applications and for middleware itself.

The development of this kind of software in the context of game AI presents several design issues and challenges [6] and although there are already some commercial and academic offerings [7] the state of the art still has lots of room for improvement. Not even the exact definition of what is an AI middleware is a consensus. In order to help dealing with some of these issues and create interface standards between middleware solutions and game engines, the IGDA's (International Game Developers Association) AIISC (AI Interface Standards Committee) [14] was started, a joint effort of AI developers, representatives from middleware companies and from academia.

2.1 Usual approaches and current solutions

Traditionally, computer games developers only use a small set of techniques over and over again in the implementation of artificial intelligence functionalities in games, specially Finite State Machines (FSMs) that are basically a set of states and transitions between these states, used to represent some kinds of behaviours, and the A* algorithm used for calculating paths. But even with the small set of techniques used, current games achieve pretty good results.

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 make use of simple approaches as FSMs (where the transitions between states usually reflect some game world events), decision trees (as is the case of the AI.implant [1] engine) or simple rule-based systems (RBSs), which are more flexible than a purely stimuli-response approach (that according to [12] 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 FSM approach in the design of intelligent agents are well known, they are limited specially by combinatorial explosions 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. These limitations were "attacked" with the creation of hierarchic FSMs (HFSMs), a extension to traditional FSMs that allows the creation of composite states which contain, inside themselves, other states and transitions; and the creation of Fuzzy FSMs (FuFSM) that add characteristics of fuzzy logic to FSM transitions. RBSs, on the other hand, present 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 are much more concise than FSMs. However, RBSs may have a large memory footprint, require a lot of processing power and even in some situations become extremely difficult to debug (as a statement about RBS usefulness, its use is increasing in game, specially on the server side [14]). SimBionic [22] presents an approach that fits somewhere in between FSMs and RBSs, by providing a framework for defining objects that display behavior within the game world. This framework is very state-oriented, supporting the creation of complex hierarchical state systems.

Games developers need a solution in between simple FSMs and complex cognitive models. First of all, it is necessary to avoid too frequent updates, and to rely in an event based model. Second , the new solution needs to avoid presenting repetitive behaviours, exhibiting a variety of reactions but still being verisimilar. 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.

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 [9] 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. And the lower level operators represent the steps (or sub-steps) necessary for the agent to implement and execute the tactics. Another approach is the biologic/evolutionary one, as DirectIA [4] that provides support for the creation of agents, with behaviours ranging from basic reactions to deep world state analysis. DirectIA is an agent-centric tool, which "mimics" how and why a character makes a decision. Using a motivational model, its mechanism handles stimuli, emotion, states, motivations, behaviours and actions. In this system, 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 and present behaviours not specifically programmed by the game developers. Some other issues must be taken into account when designing AI engines. Laird and van Lent [10] state clearly that a knowledge base containing goals, tactics and behaviours independent of a specific game is its most important feature. And Leonard [11] makes his case that it is extremely valuable to construct a good set of sensors in order to improve the player experience. Unfortunately, approaches to planning are seldom used in computer games development with some very recent exceptions (e.g. [15]).

Also, one can not forget the use of machine learning techniques integrated into the AI engine in order to obtain new behaviours [3].

2.2 Some issues

As Nareyek points in [13], central to the AI computation is not only how actions are determined, but also which information about the environment is available and how this can be accessed. Another issue is how the AI middleware will integrate with other existing engines/middleware without causing conflicts or the like. AI middleware software needs not find optimal solutions, but must find good solutions under many constraints [21] and still offer real-time challenging AI. Some of the faced constraints are: large and non-stationary environments; partially observable states; possibly expensive actions and inaccurate world models. These issues coupled to the lack of literature on the subject make the problem of developing AI framework much harder; one needs not only to deal with the AI design but also with a high load of software engineering concerns.

3. ARCHITECTURE IMPLEMENTATION

The initial approach to the development of the first AI middleware prototype was a bottom-up approach where by implementing lower level functions and combining them the final solution could emerge. This approach also allowed us to have a version running soon and thus facilitated testing.

The main idea behind our current research in designing the architecture is to borrow as much as possible from academic AI and from software engineering principles to build it in the most flexible way possible. One can see this kind of software (game AI middleware) as a framework to be instantiated by the game developer; starting from this point of view and with a prototype implementation running, we went in search of what techniques were most used in game AI engines. Following the framework evolution process proposed by Roberts and Johnson [20], we then proceeded to the construction of a "component library", implementing the most common/useful used game AI techniques and then analyzing how they'd interact among themselves and with the prototyped framework. In the implementation of the component library, we tried to identify design patterns that could be applied to game development and thus help avoid common mistakes, contribute to formalize language in the field (through a design pattern catalog) by bringing game AI and software engineer together, and also provide a easy to build design that future developers could more easily understand.

Some of the design decisions for the first prototype were inspired by RenderWare AI [19], an AI middleware that also presents a layered design (architectural, services, agents and decision layers); the agents layer being a set of behaviours that can be instantiated by a game character. Another influence was Pensor, which was composed of a set of re-combinable algorithms and techniques; planners, path-finders, a decision module using FSMs and rules, a perception module and an infra-structure module that provides support for resource managing.

The previous prototype [5] was organized in three macro layers, Service, Behaviour and Cognitive. The Service layer, handled low-level implementation details [18][17] to guarantee a good system performance as well as freeing the game developer from this burden (as for example: using event handling instead of polling; centralizing the cooperation of game managers). The Behaviour Layer used FSMs for modeling simple behaviours (a tool familiar to game developers) and a simple set of "motivational rules" that allowed the selection of the behaviour to be executed next. And the Cognitive layer was not implemented.

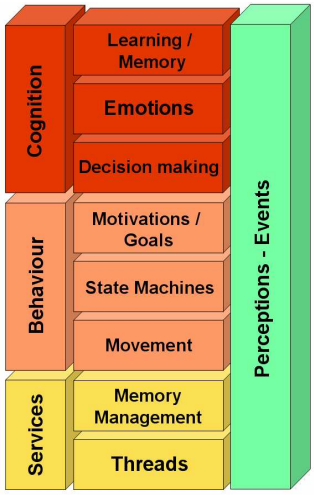

Figure 1: Middleware architectural layers

In its current version our middleware has implemented: FSMs, steering behaviours, some path-finding (navigation mesh - network of convex polygons), a simple perception system that communicates with every layer, and a simple goal arbitration system. Both the FSM and the goal system can be hierarchic to increase flexibility. Due to the nature of our framework, we also implemented a scripting system in Lua that also allows for scripted FSMs and scripted goals. We decided not to implement decision trees at this moment as this technique is not so well received [8][14].

Our new layer structure was inspired by the prototype architecture, but also on 3 other approaches: a) a brain-like organizational model were functions build on each other; b) the idea of multiple layered decision architectures learned from robotics efforts [2]; c) the concept of Distributed AI detailed by Schwab [21] (with the following layers: Perception/Event, Behaviour, Animation, Motion, Short term decision, Long term decision and location based information); and is shown in Figure 1. The learning module is not yet implemented and only very basic emotional modeling is used.

The current middleware architecture uses an event-driven system (created with basis on past experience in the embedded world) and thus avoids frequent polling of world items state and provides a more flexible and modular code organization; the event system is used both to represent sensorial data and as communication means between the different architecture parts. Also, it is currently being tested against a home-built engine for ease of debugging purposes and is going through one last round of refactoring.

4. CONCLUSION

Our middleware implementation provided many insights into the creation of such a tool for generic games. And as the higher layers are implemented/improved many more issues can be better understood and will be studied in depth. In its current state, the architecture allows for the use the most common/useful game AI techniques in a quite general way and also serves as foundation for developing higher level behaviours and cognitive systems. Much more thought will have to be dedicated (and currently is being dedicated) to the question of interacting with the game world and with other development support frameworks (graphics, network,) and we are keeping an eye on (and trying to help if possible) the IGDA's AIISC efforts in this direction.

The development of such kind of software still faces many open issues but is essential to provide advances in gameplay; as shown in the literature and in commercial games, somewhat simple initiates can allow for pretty good results from the point of view of user experience. From a software engineering point of view, we'll keep trying to increasing flexibility of our design by identify hot-spots in our framework that can then be generalized. Another contribution from software engineering is the collection of patterns on how to implement different AI techniques and the interaction between various architectural modules.

Another research direction we are pursuing is to create new forms of gameplay or entertainment, by integrating new techniques in the framework or building new tools on top of it. Areas such as character communication, personality modeling, goal oriented planning and neural networks are being studied. And also less character oriented (but equally AI intensive) areas like auto generation of quests for RPGs, integration with a effort in our group in interactive plots generation [16] are also being sought and can result in new entertainment/gaming applications. What about for example using games as spectator sports and having teams of AI characters compete? There is a lot of room for improvement in the field and we think our efforts are just at the beginning.

5. ACKNOWLEDGMENTS

This work is funded in part by a CAPES grant. Thanks also to FINEP for its support via the VisionLab project.

6. REFERENCES

[1] AI.implant - Advanced AI for games, animation and simulation. URL: http://www.ai-implant.com.

[2] R. Brooks. Intelligence without representation. *Artificial Intelligence*, (47):139–160, 1991.

[3] Z. Ding. Designing AI engines with built-in machine learning capabilities. In *Proceedings of the Game Developers Conference 1999*, San Jose, USA, 1999.

[4] Direct IA. URL: http://www.directia.com.

[5] B. Karlsson. Prototyping a simple layered artificial intelligence engine for computer games. In *Proceedings of the 4th International Conference on Intelligent Games and Simulation (GAME-ON'03)*, UK, 2003.

[6] B. Karlsson. Issues and approaches in artificial intelligence middleware development for digital games and entertainment products. In *Proceedings of the International Digital Games Research Conference 2003 (LEVEL-UP)*, Utrecht, The Netherlands, 2004.

[7] B. Karlsson and B. Feijó. State of the art in commercial AI middleware. In *Proceedings of the Brazilian Symposium on Computer Games and Digital Entertainment (SBGAMES'04)*, Brazil, 2004.

[8] P. Kruszewski. The challenges and follies of building a generic AI engine. In *AAAI-04 Workshop on Challenges in Game AI*, USA, 2004.

[9] J. Laird, E. Newell, and P. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[10] J. Laird and M. van Lent. Developing an artificial intelligence engine. In *Proceedings of the Game Developers Conference 1999*, San Jose, USA, 1999.

[11] T. Leonard. Building AI sensory systems. In *Proceedings of the Game Developers Conference 2003*, San Jose, USA, 2003.

[12] A. Nareyek. Intelligent agents for computer games. In *Proceedings of the 2nd International Conference on Computers and Games*, pages 414–422, Japan, 2000.

[13] A. Nareyek. Artificial intelligence in computer games - state of the art and future directions. *ACM Queue*, 1(10):58–65, 2004.

[14] A. Nareyek, B. Karlsson, I. Wilson, M. Chady, S. Mesdaghi, R. Axelrod, N. Porcino, N. Combs, A. El Rhalibi, B. Wetzel, and J. Orkin, editors. *The 2004 Report of the IGDA's Artificial Intelligence Interface Standards Committee*. AI-SIG, International Game Developers Association (IGDA), June 2004.

[15] J. Orkin. Symbolic representation of game world state: Toward real-time planning in games. In *AAAI-04 Workshop on Challenges in Game AI*, San Jose, USA, 2004.

[16] C. T. Pozzer. *Um Sistema para Geração, Interação e Visualização de Histórias para TV Interativa*. PhD thesis, Informatics Departament, PUC-Rio, Rio de Janeiro, 2005.

[17] S. Rabin. Designing a general robust AI engine. In M. DeLoura, editor, *Game Programming Gems*. Charles River Media, 2000.

[18] S. Rabin. Strategies for optimizing AI. In M. DeLoura, editor, *Game Programming Gems 2*. Charles River Media, 2001.

[19] Renderware AI. URL: http://www.renderware .com/renderwareai.htm.

[20] D. Roberts and R. Johnson. Evolving frameworks: A pattern-language for developing object-oriented frameworks. In R. Martin, D. Riehle, and F. Buschmann, editors, *Pattern Languages of Program Design 3*. Addison-Wesley, 1997.

[21] B. Schwab. *AI Game Engine Programming*. Charles River Media, September 2004.

[22] SimBionic. URL: http://www.shai.com/products.