An Architecture for Portable Serious Games

Ahmed BinSubaih[1], Steve Maddock[1], and Daniela Romano[1]

[1]Department of Computer Science
University of Sheffield
Regent Court, 211 Portobello Street
Sheffield, S1 4DP, UK
{A.BinSubaih, S.Maddock, D.Romano}@dcs.shef.ac.uk

Abstract. The current approaches employed for developing games using game engines tend to restrict the ability to port the game to different engines. Not addressing this issue causes the game to be surrendered to the engine's own proprietary format which causes a tight architectural coupling to exist between the two. This work shows how the development could follow a different route that leads to an architecture where the emphasis is placed on decoupling the game — thus making it portable. This approach has been put into practice and an architecture demonstrating how this is feasible has been developed. The architecture validity was then evaluated in three ways. First, a sample training game was developed and linked it to two different game engines without having to modify it. Second we have used an architectural evaluation method (ATAM) to verify that the architectural main goal was achieved and more importantly to find out which architectural decisions have contributed to its success and which ones undermined it. Finally, we have developed a serious game on the architecture to train traffic accident investigators how to attend a virtual traffic accident and an empirical study was conducted for 56 police officers to measure its effectiveness.

Keywords: portability, software architecture, game engines, serious games.

1 Introduction

Game engines are finding increasing use in academic research projects. The advantages for researchers include the outsourcing of unrelated work such as physics, networking, rendering of complex objects and character animation, which allows them to focus on the core of their work, and the reduction in software development costs. As a relatively new movement it seems like a winning formula for both researchers and the game industry. But, are there any hidden costs and unforeseen issues that could prove costly later on in the game lifetime? The issue we examine in this work is the complexity involved in moving a game between engines. We contribute this to the way the engines draw researchers or developers to surrender the game to the engine's specific format (using the game engine's object model and proprietary language). Examples of this vary across domains. From the military training domain there are: America's Army[1] which is built on top of the Unreal engine, DARWARS Ambush! [1] built on top of Operation Flash Point, and Tactical Iraqi [2] built on top of the Unreal engine. For first responders there are: Hazmat: Hotzone [3] developed on top of Unreal engine, a virtual terrorist attack on the computer science building built on top of Half-Life [4], and UnrealTriage [5] built on top of Unreal. In the field of search and rescue a game was developed in less than three months using the Unreal engine [6]. Unfortunately the issue of a proprietary format is not unique to developers using game engines only but also to developers using Virtual Environment (VE) engines such as VR Juggler[2], AVOCADO [7], and DIVE [8].

Considering that the game logic is the core of the game and where much time is spent during the development lifecycle it is unfortunate that the engines encourage the use of their proprietary format. What would be more beneficial is to have the logic separate from the rest of the engine. The benefits gained are threefold. First, it could encourage more researchers to make use of engines, since a particular engine's future capability (or potential discontinuation) would not be a worry as a different engine could easily be substituted. Second, it would reduce the development time by making the decision to chose an engine not so critical, as the core of the system (i.e. the game logic) is not affected by this choice. Third, it would increase logic reusability between projects using the same architecture and additionally it would increase interoperability between projects using game engines.

Besides the projects that use engines the work here should also be of benefit for providing an alternative development route for projects that have been developed from scratch (e.g. ACTIVE [9] and CACTUS [10]) by raising their awareness to the risks associated with the tight coupling of logic to the application.

[1] http://www.americasarmy.com/
[2] http://www.vrjuggler.org/

1.1 Problem Statement

Building a game from scratch and using commercial off-the-shelf (COTS) products are both vulnerable to surrendering the logic to the engine. The logic is the 'brain' of the game and the engine is the medium through which the game is experienced. The way the two are linked is the main focus of this research which hypothesis that the current approaches employed undermine an important architectural quality attribute, namely portability. Portability is the ability to migrate the game to another engine without having to redevelop or modify the game specifically to suit that engine. There is a need for a mechanism that reduces the tight coupling followed when using engines.

1.2 Current State of the Art Approaches

There are some existing and emerging initiatives and projects that can aid game portability. These can be divided into the following four groups: AI architectures, interfaces, standards and file formats, and frameworks or protocols.

The AI architectures group uses an AI component. This component could be custom made or off-the-shelf such as the AI Middleware (e.g. SOAR [11], RenderWare AI[3], AI.Implant[4], etc). The need for this emerged as the AI complexities increased and the processing time allocated for them also increased; it became more difficult to reinvent the AI wheel every time a game is developed. From a software engineering perspective this practice is encouraged as it promotes above all reusability. Having the logic in the AI middleware format is not what we eventually want since this merely moves it from one proprietary format (game engines) to another (AI middleware). Nevertheless it is a step in the right direction of moving the logic away from the game engine's format. The architectures we have found so far that promote portability more than others are those that allow complete removal of the logic from the game engine such as TIELT [12]. Others that only partially remove the logic are obviously less portable such as Mimesis [13] and MissionEngine [14]. Similarly to the game engines practices this group promotes the use of their own proprietary format. Furthermore suggesting a monolithic architecture as a complete entity is not what is need. Instead initiatives must examine the causes of the problem and provide practical solutions that can be employed even if their architecture or middleware is not chosen.

The second group is the interfaces group which could further be subdivided into two sections; ones that provide common interfaces and others that provide an interface per game engine. An example of the common interfaces is the initiative by the working group on world interfacing in the International Game Developers Association[5] (IGDA) which aims to provide a standard interface with the game's virtual world. In the group's 2005 annual report [15] three basic elements for interaction with the game's world have been mandated: sensing, acting, and data format used for communication. If the initiative gets acceptance by the game industry it would aid our portability cause by having one common interface through which we can access all the game engines rather than one per engine, as we currently have to do. Another project that looks at standardising game interfaces is OASIS [16]. Examples of interfaces targeting single game engines are Gamebots [17] which is interfaced with Unreal game engine and an interface to the Urban Terror First-Person Shooter Game by [18]. This group is facing an uphill struggle against a fast evolving industry that is nowhere near its maturity if only to be judged by the minimal use of software engineering practices that have been around for decades. This suggests that an agreement on the interface will not occur any time soon. Therefore, as the number of the projects using game engines increases there is a need for a more imminent solution that can provide alternative approaches.

The third group is the standards and file-based formats such as VRML/X3D[6] and COLLADA[7]. There are two reasons why for these kinds of approaches it is not easy to get a buy-in from the industry. First, they still lack the maturity needed for games. Second, standardization might not be the best practice to push down an industry known to have emerged from a 'cottage' industry.

The fourth group is the frameworks or protocols that aid interoperability between different simulations like the High Level Architecture (HLA) [19] and Java Adaptive Dynamic Environment (JADE) [20]. Despite the fact that this category focuses more on the interoperability between simulations and less on how the logic is linked to the simulation it is mentioned here to illustrate that portability exists at different levels. HLA for instance promotes it at the simulation and object level and JADE promotes interoperability at the functionality level.

1.3 Contributions

The main contributions of this thesis are:

- Identifying practices used to develop games using game engines and examining how they affect portability. This analysis should result in a diagnosis of the causes and a set of recommended practices to assist portability.

[3] http://www.renderware.com/ai.asp
[4] http://www.biographictech.com/
[5] http://www.igda.org/ai
[6] http://www.web3d.org/x3d/
[7] http://www.collada.org

- A new architecture for allowing games to be portable between game engines is developed using the practices identified above. A sample training game was then built using the new architecture to demonstrate how that is possible.
- A new architecture that shows how portability, modifiability, and performance quality attributes could be feasible on the same architecture without rendering it unusable. This is then evaluated using a structured architecture evaluation method developed by SEI called Architecture Tradeoff Analysis Method (ATAM) [21].
- To further evaluate the scalability of the architecture a serious game was developed on top of it to train traffic police investigators on how to attend virtual accidents. This was then followed by an empirical study to assess its usability.

2 New Architecture

This section starts by identifying the practices that cause tight coupling of the game to the engine. Then it provides a solution to the problem and demonstrates how it was used to develop the new architecture.

2.1 Tight coupling diagnosis

We have identified three dependencies that cause tight coupling to exist between the game and the engine. These are referred to as the unwanted dependencies in figure 1. The dotted line around the game space means that it does not exist physically on its own, but instead it lives inside the game engine. We put it on its own to demonstrate how it is linked within the game engine.

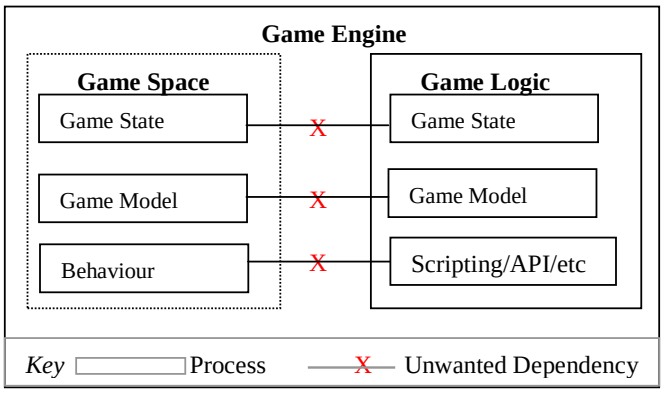

Figure 1: The three unwanted dependencies

The first unwanted dependency arises when a game is being developed and the engine used asks for the game state to be put in its own game state format. Instead what developers should aim to do is to have the game state living outside the game engine and find a way to communicate between the two states.

The second warning should be flagged when the game engine requires objects to be represented in its own game or object model representation. The goal should be not to have a game model that is only accessible through the game engine's interface but have a game model that can exist and be accessed outside the game engine. This could then be used by the behaviour engine to control the game by modifying the game state. The consequence of using the game engine's game model would mean that the manipulation of the game state would always be dependent on the game engine's interface which would correspondingly mean that it would have to be carried along with the game when moving to another game engine. The final flag should be raised when the game engine requires the behaviour to be specified in the game engine's own language.

2.2 Solution

Figure 2 shows the new architecture which consists of three subsystems: the game space, the adapters, and the game engines. The game space consists of game state, game model, and behaviour engine. The game space has components (not shown in the figure) like Application Programming Interface (API), scripting interpreter, sockets, and persistent storage. These are used to manage the game and communicate it to the game engines. In the View part of the diagram it shows the game engine components which follow the decomposition scheme suggested by [22].

The architectural decisions made to counter the unwanted dependencies are:

- *Dependency 1*: the direct link between the game space's game state and the game engine's game state should be broken. The architectural decisions made to achieve this were: model-view-controller (MVC) pattern [23], asynchronous messaging, mid-game scripting [24].

Dependency 2: the direct link between the game space's game model and the game engine's game model must be broken. To achieve this the following architectural decisions were made: ontologies [25], API, and mid-game scripting.

- *Dependency 3*: the game behaviour should not be formatted in the game engine's proprietary format. To achieve this the architectural decisions made were: API, mid-game scripting, scripts mapping table, and objects mapping table.

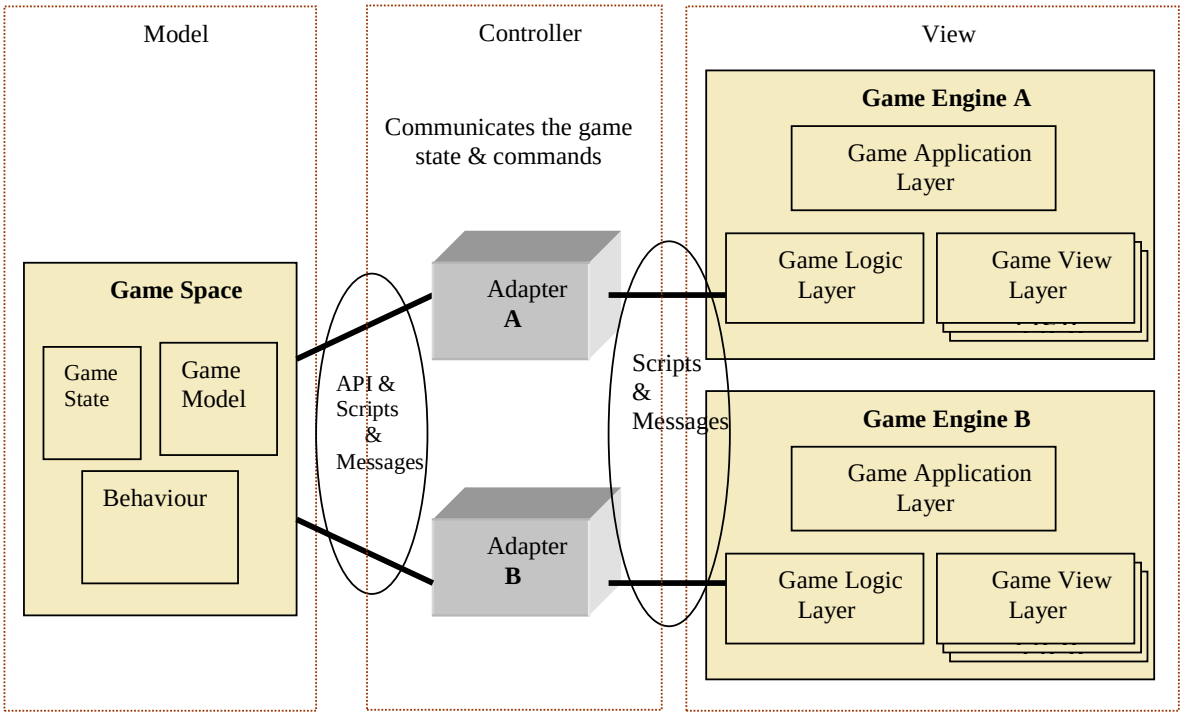

Figure 2: Game-based architecture using the MVC pattern

3 Evaluation

The main objective of any architecture evaluation is to address the concerns and anxieties of the different stakeholders involved in the architecture. From the author's perspective it is important to verify that the architecture works as described. Additionally it is also important to understand the weaknesses and strengths of the architecture before evolving it any further. From a user's point of view this evaluation provides him with a proof of the architecture usability and a thorough evaluation should allow him to make an informed decision about its adoption.

The evaluation started by throwing random challenges at the architecture. The first challenge was the ability to run the same training game on two different engines. The second challenge addressed the performance of the architecture. Both of these challenges are described in [27]. Obviously this approach does not follow any structured method and we call it the ad hoc approach. Despite this the approach managed to show us the capability of the architecture even if it was only indicative. We have some reservations about this approach that we have detailed in [28]. The major reservations are: this approach does not guarantee that all the main architecture's components are going to be exercised during the evaluation, and worst, some of the challenges might be redundant as they probe the same component.

Because of the above reservations we wanted a more thorough examination of our architecture. We wanted an examination that would reveal its weaknesses and strengths and could establish correlation between the architectural decisions we have made to solve the portability issue and how they have contributed or undermined our goal. We researched a number of architectural methods such as: ATAM, SAAM [29], ARID [30], ABAS, PASA, CBAM [31], etc. ATAM stood out as the most suitable one to address our concerns. Our case study demonstrating the use of ATAM and the results generated is described in [28].

Finally, being in the field of games where very often a game engine success is judged by the success of the games that have been developed on it we saw the need for another type of evaluation, an evaluation which shows a complete game development. Moreover, this should further address the usability and the scalability concerns (considering the small samples developed earlier during the ad hoc evaluation process). The game developed is a serious game to train traffic police investigators on attending virtual traffic accidents. An empirical study was conducted and the usability of the architecture was measured by the effectiveness of the game. We recently completed this study and we are in the process of analyzing and writing up the results.

4 Conclusions

The thesis work has investigated the issue of portability in game engines. It started by examining the way games are developed using game engines and identified the causes of the tight coupling that exists which restrict the ability to migrate the game to different engines. A solution was proposed to address the causes identified and an architecture was developed. The architecture managed to allow the same game to be ported to different engines

without having to modify the game. This architecture was then evaluated using an ad hoc evaluation approach and a more structured approach. These managed to reveal the architecture's strengths and weaknesses. On top of identifying the causes and developing a new architecture to show how the problem can be solved, the thesis should serve to raise awareness of the hidden dangers when surrendering the logic to the engine's format and provide an alternative development route that could be put into practice straightaway.

References

1. Diller, D., Roberts, B.: DARWARS Ambush! A Case Study in the Adoption and Evolution of a Game-based Convoy Trainer by the U.S. Army. Simulation Interoperability Standards Organization, 18-23 September (2005).
2. Elizabeth, L.: In Country with Tactical Iraqi: Trust, Identity, and Language Learning in a Military Video Game. Proceedings of the 6th Digital Arts and Culture Conference. Copenhagen, Denmark: University of Copenhagen, (2005): 69-78.
3. Carless, S.: Postcard From SGS 2005: Hazmat: Hotzone - First-Person First Responder Gaming http://www.gamasutra.com/features/20051102/carless_01b.shtml, November 2, (2005).
4. Hall, R.H., Wilfred, L.M., Hilgers, M.G., Leu, M.C., Walker, C.P., Hortenstine, J.M.: Virtual Terrorist Attack on the Computer Science Building: A Research Methodology. Presence-Connect, 4(4), http://www.presence-connect.com/ (2004)
5. McGrath, D., Hill, D.: UnrealTriage: A Game-Based Simulation for Emergency Response. The Huntsville Simulation Conference, October 2004. Sponsored by The Society for Modeling and Simulation International (2004).
6. Wang J., Lewis M., Gennari J.: Emerging areas: urban operations and UCAVs: a game engine based simulation of the NIST urban search and rescue arenas. 35th Winter Simulation Conference, New Orleans, Louisiana, 1039-1045, (2003)
7. Tamberend, H.: Avocado: A Distributed Virtual Environment Framework". Ph.D.thesis, University of Bielefeld, (2003).
8. Frécon E., Stenius, M.: DIVE: A Scaleable network architecture for distributed virtual environments. Distributed Systems Engineering Journal (special issue on Distributed Virtual Environments), 5(3), Sept. (1998), 91-100.
9. Romano, D.: Features that Enhance the Learning of Collaborative Decision Making Skills under Stress in Virtual Dynamic Environments. Ph.D.thesis, Computer Based Learning, University of Leeds, UK, August (2001).
10. Williams, R, J.: A Simulation Environment to Support Training for Large Scale Command and Control Tasks". Ph.D. thesis, School of Computer Studies, University of Leeds, UK, December (1995).
11. Laird, J.E., Assanie, M., Bachelor, B., Benninghoff, N., Enam, S., Jones, B., Kerfoot, A., Lauver, C., Magerko, B., Sheiman, J. Stokes, D. Wallace, S.: A Testbed for Developing Intelligent Synthetic Characters. In Artificial Intelligence and Interactive Entertainment: Papers from the 2002 AAAI Spring Symposium, Menlo Park, CA (2002).
12. Aha, D.W., Molineaux, M.: Integrating learning in interactive gaming simulators. Challenges of Game AI: Proceedings of the AAAI'04 Workshop (Technical Report WS-04-04). San Jose, CA: AAAI Press, (2004).
13. Young, R.M., Riedl, M.O., Branly, M., Jhala, A., Martin, R.J., Saretto, C.J.: An architecture for integrating plan-based behavior generation with interactive game environments. Journal of Game Development , 1(1), 51-70, (2004).
14. Vilhjalmsson, H., Samtani, P.: MissionEngine: Multi-system integration using Python in the Tactical Language Project, PyCon 2005, March 23-25, Washington, D.C. (2005).
15. Nareyek, A., Combs, N., Karlsson, B., Mesdaghi, S., Wilson, I.: The 2005 Report of the IGDA's Artificial Intelligence Interface Standards Committee. http://www.igda.org/ai/report-2005/report-2005.html, (2005).
16. Berndt, C., Watson, I., Guesgen, H.: OASIS: An Open AI Standard Interface Specification to Support Reasoning, Representation and Learning in Computer Games. IJCAI-05 Workshop on Reasoning, Representation, and Learning in Computer Games. 31-July (2005), Edinburgh, 19-24.
17. Adobbati, R., Marshall, A.N., Scholer, A., Tejada, S., Kaminka, G., Schaffer, S., Sollitto, C.: Gamebots: A 3D Virtual World Test-Bed For Multi-Agent Research. Proceedings of the International Conference for Autonomous Agents, Workshop on Infrastructure for Agents, MAS, and Scalable MAS, Montreal, Canada, (2001).
18. Kondeti, B., Nallacharu, M., Youngblood, M., Holder, L.: Interfacing the D'Artagnan Cognitive Architecture to the Urban Terror First-Person Shooter Game. IJCAI-05 Workshop on Reasoning, Representation, and Learning in Computer Games. 31-July (2005). 55-60.
19. Smith, R.: Essential techniques for military modeling and simulation. Proceedings of the 30th conference on winter simulation, (1998), 805 – 812, ISBN:0-7803-5134-7
20. Oliveira, M., Crowcroft, J., Slater, M.: An innovative design approach to build virtual environment systems. Proceedings of the workshop on Virtual environments 2003, ACM International Conference Proceeding Series; Vol. 39, Zurich, Switzerland Pages: 143 – 151, (2003) ISBN:1-58113-686-2.
21. Clements, P., Kazman, R., Klein, M.: Evaluating Software Architectures: Methods and Case Studies, Addison Wesley, (2001), ISBN: 020170482X.
22. McShaffrey, M.: Game Coding: Complete, Paraglyph Press, ISBN: 1932111913, (2005).
23. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Patter-Oriented Software Architecture: A System of Patterns. Volume 1, (1996), John Wiley and Sons, ISBN: 0471958697
24. Ousterhout, J.: Scripting: Higher-Level Programming for the 21st Century, IEEE Computer, 31(3), (1998), pp. 23–30.
25. Chandrasekaran, B., Josephson, J., Benjamins, V.: What are ontologies and why do we need them? IEEE Intelligent Systems, Jan/Feb (1999), 14(1), pp. 20-26.
26. Friedman-Hill E.: JESS in Action, Manning Publications Co, (2003), ISBN 1930110898.
27. BinSubaih A., Maddock S., Romano, D.M.: Game Logic Portability. ACM SIGCHI International Conference on Advances in Computer Entertainment Technology ACE 2005, Computer Games Technology session, June 15-17th, Valencia, Spain.
28. BinSubaih A.: Using ATAM to Evaluate a Game-based Architecture. Workshop on Architecture-Centric Evolution (ACE 2006). Hosted at the 20th European Conference on Object-Oriented Programming ECOOP 2006 July 3-7, 2006, Nantes, France.(accepted)
29. Kazman, R., Bass, L., Abowd, G., Webb, M.: SAAM: A Method for Analyzing the Properties Software Architectures. Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy, May (1994), pp. 81-90.
30. Clements, P.: Active Reviews for Intermediate Designs. (CMU/SEI-2000-TN-009), August, (2000).
31. Bahsoon, R., Emmerich, W.: Evaluating software architectures: development, stability and evolution. In the Proceedings of ACS/IEEE International Conference on Computer Systems and Applications, Tunis, Tunisia, July 14-18, (2003).