Game Logic Portability

Ahmed BinSubaih, Steve Maddock, Daniela Romano
Department of Computer Science
University of Sheffield
Regent Court, 211 Portobello Street, Sheffield, U.K.
+44(0) 114 2221800
{a.binsubaih, s.maddock, d.romano}@dcs.shef.ac.uk

ABSTRACT

Many game engines integrate the game logic with the graphics engine. In this paper we separate the two, thus making the logic portable between game engines. In our architecture the logic is represented as an ontology and a set of rules for a particular application domain. A mediator with an embedded rules-engine links the logic to a suitable game engine.

We demonstrate our architecture in two ways. First, we show a traffic accident scenario running on two different game engines, with a separate mediator for each engine. The logic type is smart-terrain logic, with participants triggering events based on interaction and proximity tests. In the second demonstration (a simple first-person shooting game) we show the extensibility and performance of the architecture to control non-player characters quickly manoeuvring using proximity tests and waypoints.

Categories and Subject Descriptors

I.3.6 [**Computer Graphics**]: Three-Dimensional Graphics and Realism.

General Terms

Design, Experimentation.

Keywords

Logic, Ontologies, Rules.

1. INTRODUCTION

Game engines have been used widely in supporting academic research. For example, [9] developed a search and rescue project in less than three months using the Unreal engine, [4] used the Quake 3 Arena game engine for real-time geo-spatial data visualisation, and other projects have focussed on AI [6], architecture (the VRND Notre Dame project [3]) and on military applications [10]. There are also examples of projects doing initial tests with game engines. For example, Romano used the first versions of Quake to test some of the hypotheses that were adopted in the development of the ACTIVE system [7].

One issue with many of these game engines (e.g. Unreal, Quake, Never Winter Nights) is that they require the logic to be formatted in their proprietary format (usually some form of script language). This is unfortunate considering that the logic is the core of the game and where much time is spent during the development lifecycle. It would be more practical if the logic was separate from the rest of the system and could be easily migrated to another system. The benefits of this are:

- It could encourage more researchers to make use of game engines, since a particular game engine's future capability (or potential discontinuation) would not be a worry as a different game engine could easily be substituted.
- It would increase logic reusability amongst projects, as a person could migrate it to a familiar engine and thus avoid the time required learning a new engine.
- It would increase the scalability possibilities for the logic, depending on the future development of game engine capabilities.
- The logic format could be standardised (or the translators for different logic formats could be standardised).

The main contribution of our work is to demonstrate the feasibility of separating out the logic by representing it using ontologies and rules, and by introducing middleware (an events space) between the logic and the game engine.

Section 2 gives an overview of our architecture and describes the types of logic to be used in the system, the events space components, and the simulation engines used. Section 3 presents the results of using our architecture for two different domains, each showing a different type of logic being serviced, and discusses issues of portability, extensibility, and system performance. Section 4 presents conclusions.

2. ARCHITECTURE

Our architecture is designed to allow both logic and reasoning to be separate from the game engine. Figure 1 gives a conceptual overview of the architecture. It shows how the logic is separated from the engines by using an events space mediator which has a rules-engine, an adapter, and a loader.

2.1 Game Logic

In ontological engineering [2] entities or objects with similar attributes are grouped together by a 'concept', where attributes form 'slots' to be filled in. Concepts can be further structured in a hierarchical format similar to classes in object-oriented design. In the following two subsections, we will describe the two kinds of logic that we have tested so far in our architecture. Each is

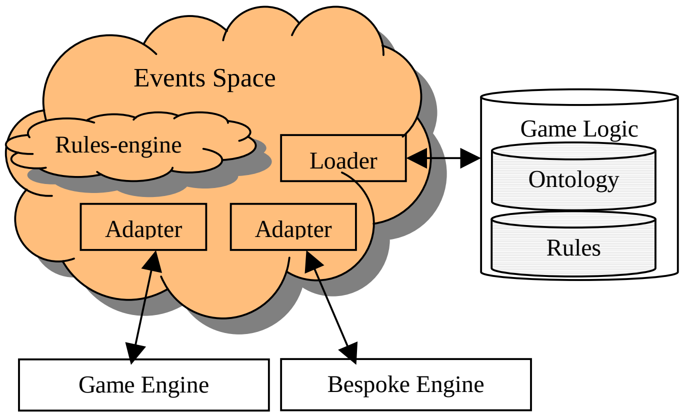


Figure 1: Conceptual overview of the architecture

employed in a different domain for which a separate ontology is outlined.

2.1.1 *Smart Terrain Logic*

We use smart terrain logic [8] in a traffic accident domain, which is used to train police officers in how to attend and investigate a virtual traffic accident (running in a collaborative virtual environment). Our earlier work on this [1] used manual observation and input to service logic. In the current paper, the events space automatically makes use of the logic. We use an ontology to store the following information:

1. The virtual environment content such as the scene layout which includes the position of objects.
2. Details about each person involved in the accident such as name, age, injury type, etc.
3. Answers to questions put to virtual actors.
4. Hints on tasks a trainee needs to perform on each object.
5. Specific zones in the virtual environment zones such as the complete accident scene zone and danger zones.

Example entities from our domain include: drivers, passengers, witnesses, investigators, vehicles, skid-marks, broken glass, road, etc. The rules for the smart terrain logic store two types of behaviours: reactive and time-based. Reactive behaviour is triggered by the trainee's actions in the environment, or could also be triggered by a trainer who has direct access to the working memory of the simulation and the rules-engine and can modify certain properties to guide the trainee down a desired path. An example of reactive behaviour is when the trainee enters a zone placed around danger sources in the environment, e.g. a burning car. In contrast, time-based behaviour is triggered at pre-set points in time. Examples of such behaviour include a vehicle catching fire or an injured person starting to scream or yell.

2.1.2 *NPC Movements Using Waypoints*

The second demonstration of our architecture is a simple first-person shooting game. The aim of this demonstration is to show a more complex logic process that needs to be used in a fast-changing environment. Essentially, the NPC must evade a human predator by navigating amongst known waypoints.

The ontology here includes player, human, NPC, waypoint, and movement. The human and NPC entities inherit from the player attributes and include human in sight, human in range, NPC in sight, NPC in range, reached destination, name, and id. The waypoint entity also shares these same attributes plus last waypoint visited. The movement entity holds player, destination, id and name attributes.

The rules are used to manoeuvre any NPC who is in danger from a human player. Being in the line of sight and range of a human player indicates danger. Figure 2 shows the rules governing this behaviour.

Rule 1: IF NPC in human sight AND NPC in human range AND waypoint NOT in human sight AND waypoint NOT in human range AND waypoint in NPC sight AND NPC last destination not this waypoint THEN Move to that waypoint.

Rule 2: IF NPC in human sight AND NPC in human range AND NPC last destination not this waypoint THEN Move to that waypoint.

Figure 2: Rules governing NPC behaviour

2.2 Events space (middleware)

The events space is composed of three main components: rules-engine, adapter, and loader. The rules-engine controls the game behaviour and the adapter synchronises the game status between the game engine (or bespoke simulation engine) and the rules-engine. The loader initialises the rules-engine with templates to describe object attributes, rules to govern behaviours and facts to represent objects in the game such as player (humans or NPCs) and waypoints for the movement logic of the NPCs (see section 2.1.2). The following sections describe these components in more detail.

2.2.1 *Rules-engine (JESS)*

The advantages of using rules are well documented. The IGDA working group on rule-based systems has discussed the importance for games in its 2004 report[1]. The main two reasons why we choose rules to store our knowledge are: portability and domain-independence.

The portability reason enables logic migration between different game engines. This should remove the restriction imposed by many of the current approaches used for formatting the logic to a specific game engine. We combine a rules-engine with an adapter (see section 2.2.2) to achieve this. The second reason is to achieve a domain-independent engine where the logic is separate from the game engine thus supporting the deployment of different games by changing the logic in the rules-engine without having to reprogram the game engine.

The role of the rules-engine is to reason about behaviour. It achieves that by storing facts in its working memory that represent the game world objects used in the game engine. The objects chosen for representation and replication are the ones that have some rules governing their behaviours. For instance, for controlling an NPC's movement in a game there is a need to store the NPC in the working memory of the rules-engine and when the game engine reports that the NPC player is in the line of sight of a human player the events space then updates the rules-engine and listens for any instructions on how to react.

[1] http://www.igda.org/ai/report-2004/rbs.html

2.2.2 Adapter
The adapter plays the role of the mediator which knows how to communicate with all parties (game engine, bespoke engine and rules-engine). This means that the adapter should be able to speak the language understood by each engine. The rules-engine understands JessScript, Torque speaks in TorqueScript and our own bespoke simulation engine speaks in Python. The adapter is also responsible for communicating game status between them. We have so far only run one game engine at a time but there is no reason why two game engines cannot be run concurrently.

The adapter's communication task is achieved by holding a translation or mapping protocol which maps between JessScript and the other two languages. It works by mapping the game and bespoke engine data structures to the ontology data structure stored as templates in the rules-engine. This mapping protocol is the mechanism that permits the logic portability. To satisfy logic portability the logic should stay unchanged when linking the events space with another game engine. The only modifications allowed are at the mapping protocol level which should result in a unique mapping protocol for each game engine to be used. The creation of the mapping protocol for each engine is a one-time process.

The mapping protocol relies on the ability of the engine to have a scripting language through which the translated script is communicated. It is also important that the engine permits on-the-fly scripting rather than pre-compiled scripts that are changed at run-time. Few game engines currently satisfy this constraint. Torque is one.

The translation between the languages is achieved by storing scripts (or sentences) with placeholders that are replaced at run-time by the appropriate values. Figure 3 shows an example of the scripts and their placeholders used in communication between Jess and Torque (i.e. between JessScript and TorqueScript): The top script communicates a decision made in the rules-engine to instruct a specific NPC to move to a specific waypoint, whereas the bottom script updates the rules-engine with the current situation in the game engine with regards to the NPC status. The placeholders are marked by variables between two '@' characters. These are replaced at run-time by the appropriate values. For instance, @NPCID@ in the top script is replaced by the NPC id who is the subject of the move instruction issued.

2.2.3 Logic Loader Interface
This module loads the ontology and converts it into 'deftemplates', which is representation in Jess. It also allows a user to enter the facts corresponding to the objects in the game world. Each instance created would represent an object in the targeted game world. The way each instance knows which object it represents is through the value set in its 'ID' slot by whoever created the instance e.g. the designer. The rules-engine gives each instance a unique id (different than the slot ID the designer manually specifies) when loaded into memory. These two ids allow access to an object in working memory at each end (rules-engine and adapter) and messages sent for replication between them make use of these ids to manipulate the corresponding objects.

2.3 Game Engine
Either the Torque game engine or our bespoke simulation engine can be plugged in the architecture. The Torque game engine is a commercial multiplayer game engine developed by Garage Games (garagegames.com). It is written in C++ and has a C-like scripting language (TorqueScript). The bespoke engine is a multiplayer engine developed by the authors and coded in C++ on top of DirectX. The scripting language used for this engine is Python.

3. RESULTS AND DISCUSSION
We now present the results of two separate demonstrations showing two separate AI techniques in action on our architecture. The first demonstration is an accident scenario which utilises smart terrain logic, and the second is a simple first-person shooting game which includes evasive movements by an NPC using waypoints.

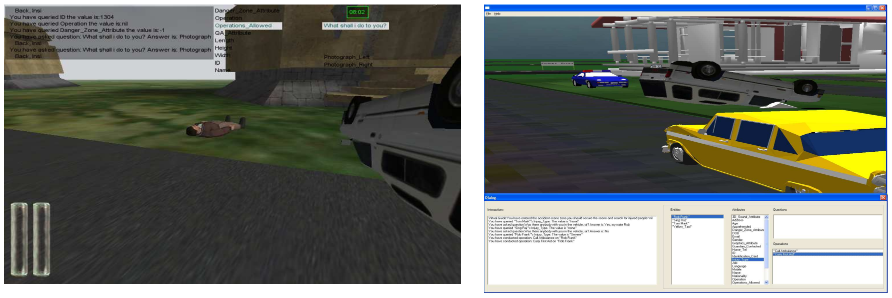
In this section the results of running the two AI techniques (smart terrain and evasive movements by an NPC) are described. The smart terrain logic supports a scenario used to train new traffic recruits on accident investigation. This demonstrates the ability to run the same set of logic on more than one engine. The second demonstration showing evasive movements by an NPC shows that the architecture is extensible and caters for different game techniques.

The aim of the accident scenario is to train police officers how to deal with traffic incidents. The particular scenario we have used concerns two drivers involved in an accident that results in one severely injured passenger and no injuries to the drivers. In addition, both vehicles have leaked hazardous material at the scene. A trainee enters the virtual environment and must decide how to carry out the necessary investigation. The smart terrain logic guides the trainee through the training session by giving him hints of what he should do next if he gets stuck. This is achieved by the trainee querying the 'smart' objects in the scene. These objects hold the information necessary to provide appropriate guidance. For example, if the trainee clicks on one of the vehicles involved in the accident it informs him of the operations he can

```
for($c=0;$c<MissionCleanup.getCount();$c++){
  $NPC = MissionCleanup.getObject($c);
  if ($NPC.getId() == @NPCID@)
  { break; }
}
for($w=0;$w<WaypointGroup.getCount();$w++){
  $waypoint = WaypointGroup.getObject($w);
  if ($waypoint.getId() == @waypointID@)
  { break; }
}
$NPC.setMoveDestination($waypoint.getTransform()
```

```
(modify (fact-id @factID@)
    (HumansInRange @HumansInRange@)
    (HumansInSight @HumansInSight@)
    (NPCsInRange @NPCsInRange@)
    (NPCsInSight @NPCsInSight@)
)
```

Figure 3: Top: TorqueScript; Bottom: JessScript



Back, Insi
You have queried ID the value is:1304
You have queried Operation the value is:nil
You have queried Danger_Zone_Attribute the value is:-1
You have asked question: What shall i do to you? Answer is: Photograph
Back, Insi
You have asked question: What shall i do to you? Answer is: Photograph
Back, Insi

Virtual Guide:You have entered the accident scene zone you should secure the scene and search for injured people^nil
You have queried "Tom Mark"'s Injury_Type. The value is "none"
You have asked question:Was there anybody with you in the vehicle, sir? Answer is: Yes, my mate Rob
You have queried "Sing Raj"'s Injury_Type. The value is "none"
You have asked question:Was there anybody with you in the vehicle, sir? Answer is: No
You have queried "Rob Frank"'s Injury_Type. The value is "Severe"
You have conducted operation: Call Ambulance on "Rob Frank"
You have conducted operation: Carry First Aid on "Rob Frank"

Entities	Attributes	Operations
"Rob Frank"	3D_Sound_Attribute	"Call Ambulance"
"Sing Raj"	Address	"Carry First Aid"
"Tom Mark"	Age	
"Yellow_Taxi"	Apprehended	
	Danger_Zone_Attribute	
	DOB	
	Email	
	Gender	
	Graphics_Attribute	
	Guardian_Contacted	
	Home_Tel	
	ID	
	Identification_Card	
	Injury_Type	
	Job	
	Language	
	Mobile	
	Name	
	Nationality	
	Operation	
	Operations_Allowed	

Figure 4: The accident scenario running on (a) the Torque engine; (b) our bespoke simulation engine

Welcome to the Torque demo app ahmed
You picked up a health kit

Figure 5: Torque engine running a scenario in which an NPC must run for cover.

carry out on it, i.e. photographing it, measuring the distance between its rest position and the accident point, etc. Figures 4a and b show illustrations of running this scenario on the Torque engine and our bespoke engine respectively.

For this scenario the logic was created as part of an extensive knowledge gathering exercise with police personnel. This logic was formatted for our architecture and then the same logic was serviced to each of the game engines, thus demonstrating logic portability. The two elements of the architecture that make this possible are the rules-based approach and mapping protocol. The rules-based approach helps to separate the implementation from the behaviour. The mapping protocol is then written for the particular game engine. A new (capable) game engine just needs a new mapping protocol to be supplied, and the logic remains the same.

In the second demonstration of the architecture an NPC must evade a predator (controlled by the user) who is chasing it around a landscape. Figure 5 illustrates an NPC character running for cover after a human has approached him (i.e. in sight and range). This is achieved by logic that is based on manoeuvring the NPC using waypoints. This scenario demonstrates both the extensibility and performance of our architecture.

Extensibility is illustrated in two ways. First the knowledge mechanism (ontology and rules) is easily configured to hold the scenario information. Essentially, the ontology-based approach is similar to object-oriented programming. Second, the mapping protocol which uses scripts mapping with placeholders can be extended to map any part of the data structure as long as it is exposed through a scripting language.

The shooting game also demonstrates the performance of our architecture. Since the game is fast-paced and the NPC is continually manoeuvring and moving in and out of range of the predator and the waypoints, a lot of traffic is generated back and forth between the game engine and the rules-engine. This traffic had no appreciable effect on the workings of our architecture or the game play. Although this cannot be considered conclusive and further more tests need to be conducted, it serves as a positive indicator for the practicality of the architecture.

4. CONCLUSIONS

We have presented an architecture that enables game logic to be portable. The logic is separated from the game engine by providing middleware that translates game engine status to the rules-engine and services the game engine with the appropriate behaviour. We deployed two common AI techniques on the architecture. A traffic accident scenario employing the smart terrain technique showed that the same logic can be run on two different engines (bespoke and commercial). The second AI technique was to control an NPC in a game. This demonstrated the extensibility and flexibility of the architecture. Future work will continue to extend the range of AI techniques supported, and carry out more investigation into loading and performance issues.

5. REFERENCES

[1] BinSubaih A, Maddock S and Romano D (2004) A Domain-Independent Multiplayer Architecture for Training. *International Workshop in Computer Game Design and Technology* (Nov. 1994), Liverpool, UK, 144-151.

[2] Chandrasekaran B, Josephson J R and Benjamins V R. What are ontologies and why do we need them?, *IEEE Intelligent Systems*, *14,1* (Jan/Feb 1999), 20-26.

[3] DeLeon, V and Berry R. Bringing VR to the desktop: are you game? *Multimedia, IEEE* 7,2 (April/June 2000), 68–72.

[4] Fritsch D and Kada M. Visualisation Using Game Engines. *ISPRS commission 5*, (Jul. 2004) Istanbul, Turkey, 621-625.

[5] Friedman-Hill E. *JESS in Action*, Manning Publications Co, 2003, ISBN 1930110898.

[6] Laird, J. Using a Computer Game to Develop Advanced AI, *Computer*, *34 ,7* (Jul. 2001), 70-75.

[7] 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.

[8] Rabin, S. *Promising Game AI Techniques*. AI Game Programming Wisdom 2, Charles River Media, 2004.

[9] Wang J, Lewis M and 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*, (2003), New Orleans, Louisiana, 1039-1045.

[10] Zyda, M, Mayberry A, Wardynski C, Shilling R and Davis, M. The MOVES Institute's America's Army Operations Game. *Proceedings of the ACM SIGGRAPH 2003 Symposium on Interactive 3D Graphics*, (Apr. 2003), 217-218.