SEAL – A SIMPLE ENTITY ANNOTATION LANGUAGE

Eike F Anderson
The National Centre for Computer Animation
Bournemouth University, Talbot Campus
Fern Barrow, Poole, Dorset BH12 5BB, UK
E-mail: eanderson@bournemouth.ac.uk

Keywords

Behaviour definition language, annotated environments, affordance theory.

Abstract

In recent years a number of novel techniques for creating intelligent behaviour have been applied to computer games (*see [Rabin 2004]*). One such method is based on the use of annotated environments. [Doyle 1999] describes a virtual game world made up from smart objects that are "annotated" and therefore holds all the information necessary for NPCs (*Non-Player Characters*) to interact with it. This method for behaviour definition was used to great effect in the game "The Sims" where it was dubbed "Smart Terrain". While annotated environment present a very promising behaviour definition paradigm, so far it has not been implemented in many games. It is our belief that a system that simplifies the application of annotated environments to computer games would greatly increase the acceptance and uptake of this new technology. To this end we introduce the simple entity annotation language SEAL, a fully working subset of the AvDL behaviour definition language introduced by [Anderson 2005]. SEAL presents a simple scripting solution for embedding behaviour definitions in objects, effectively annotating them and allowing NPCs to exchange information with this annotated environment.

1 Introduction

The specification of SEAL is the result of our research in game AI techniques and directly related to the development of AvDL, a behaviour definition language for the creation of NPCs. The design of SEAL specifically addresses the provision of domain knowledge that is required by NPCs to allow them to function in their game world. NPCs need to be able to clearly map - or anchor - their environment to their understanding of that environment. This problem is

discussed in the context of autonomous robotics in real-world environments by [Coradeschi and Saffiotti 1999]. They especially stress uncertainty as being the main difficulty in matching real-life sensor data to the symbolic representation of knowledge by the AI. While in virtual environments this "anchoring" problem of matching sensor information to stored knowledge becomes trivial as the incoming sensor data can be controlled to a much higher degree than real-life sensor data, the process of providing this knowledge in the first place however still remains quite complex. Annotated environments provide a possible solution for this problem.

1.1 Affordance Theory

The idea of annotated environments is based on the theory of affordance (*or affordance theory*) developed in the fields of psychology and visual perception. Affordance theory states that the makeup and shape of objects contains suggestions about their usage. A real world example would be a mug whose handle "affords" to be gripped to pick up the mug. Transferred into the context of a computer game, this means that the objects in the virtual world contain all of the information that an NPC will need to be able to use them, effectively making the environment "smart". As a result, NPCs can be less complex which consequently – as [Cornwell et al 2003] note – allows for the rapid development of game scenarios. This greatly benefits the development process and also makes the NPC's AI much more extensible. An object will broadcast information about itself to NPCs that approach it, including all necessary instructions enabling meaningful interaction between the NPC and the object.

1.2 AvDL

SEAL is closely related to AvDL, an NPC AI specific extension programming language based on the specification of the C++ programming language introduced by [Stroustrup 1997]. AvDL is more than just a game AI scripting language but rather a form of behaviour definition language for game AI entities which is easy to use and offers powerful AI functions while maintaining the flexibility of traditional programming languages. AvDL can be used to define NPCs that display deterministic, as well as goal directed behaviour.

2 SEAL

The AvDL language provides the core of a generic, modular and easily extendable system for the definition of believable intelligent game character behaviour. SEAL is one of the modules of the AvDL system, made up from a subset of the AvDL scripting language (*like the*

programming language C is a subset of C++). While the bytecode of compiled SEAL programs (*generated by the compiler*) will not be binary compatible with the AvDL system, SEAL programs are source code compatible with AvDL and should compile on an AvDL compiler. Just like AvDL programs, SEAL programs are executed in a virtual machine which interfaces with a game engine. Similar to the mechanism in the game "The Sims" described by [Forbus and Wright 2001], all entities in the game world – NPCs as well as inanimate objects that can be interacted with – are defined as scripts (*in this case SEAL programs*) to be interpreted by a virtual machine. The system contains a JIT (*just-in-time*) compiler, allowing programs to be compiled just before they are loaded into the virtual machine. The SEAL virtual machine has at its core a parallel stack machine. Separate programs run on separate stacks. Some functions which can be exported for use by other entities reside within segregated areas of their parent entities' process (*separate sections within these stacks*). Using the JIT compiler, this allows functions in the code to be replaced interactively during run-time.

2.1 SEAL in AvDL

The SEAL subset of AvDL is restricted to the syntactic features of AvDL that are considered necessary for the creation of annotated environments. SEAL functions are very much identical to those found in AvDL with the exception that SEAL functions cannot be defined through forward declaration, i.e. there are no function prototypes in SEAL. In addition to that SEAL only supports a fraction of the datatypes found in AvDL. The only primitive datatype in SEAL is the scalar type which encodes any (*binary*) logical or numerical value. SEAL does not support the traditional aggregate datatypes found in AvDL, i.e. arrays, structures or classes and therefore does not support object orientation. SEAL uses AvDL's "action" type to enable programs to directly call functions that are defined within the host application. Variables of the action type provide function bindings that map the SEAL actions to functions in the host application. Other types in SEAL are the "state" structure for the creation of finite state machines (*SEAL does not support fuzzy state machines*) and the "event" type. The latter is used for the definition of event handlers within the virtual machine whose execution can be triggered by events in the host application.

3 Future Work

The obvious next step is the completion of the SEAL prototype implementation and a demonstration of the capabilities of the SEAL system in a working example. This will be

followed by the implementation of the AvDL system for further testing of SEAL programs within the AvDL environment. While we believe that this could be used as proof of concept, ultimately a field trial may need to be conducted when the first full AvDL prototype is ready for deployment to verify the suitability of SEAL, as well as AvDL for computer game development.

References

[Anderson 2005] Anderson, E.F. (2005). Scripting Behaviour – Towards a New Language for Making NPCs Act Intelligently. To appear in Proceedings of zfxCON05 2nd Conference on Game Development

[Coradeschi and Saffiotti 1999] Coradeschi, S. and Saffiotti, A. (1999). Anchoring Symbolic Object Descriptions to Sensor Data. Problem Statement. Linköping Electronic Articles in Computer and Information Science, Vol. 4(1999): nr 9

[Cornwell et al 2003] Cornwell, J.B., O'Brien, K., Silverman, B.G. and Toth, J.A. (2003). Affordance Theory for Improving the Rapid Generation, Composability, and Reusability of Synthetic Agents and Objects. Submitted to Coference on Behaviour Representation in Modelling and Simulation (BRIMS)-2003

[Doyle 1999] Doyle, P. (1999), Virtual Intelligence from Artificial Reality: Building Stupid Agents in Smart Environments, AAAI '99 Spring Symposium on Artificial Intelligence and Computer Games.

[Forbus and Wright 2001] Forbus, K.D. and Wright, W. (2001), Some notes on programming objects in The Sims™, Class Notes from Northwestern's Computer Game Design Course, retrieved from http://qrg.northwestern.edu/papers/papers.htm

[Rabin 2004] Rabin, S. (2004). Promising Game AI Techniques. AI Game Programming Wisdom 2, Charles River Media, pages 15-27

[Stroustrup 1997] Stroustrup, B. (1997). The C++ Programming Language, 3rd Edition. Addison Wesley