OASIS: An Open AI Standard Interface Specification to Support Reasoning, Representation and Learning in Computer Games

Clemens N. Berndt, Ian Watson & Hans Guesgen
University of Auckland
Dept. of Computer Science
New Zealand
clemens.berndt@gmail.com, {ian, hans}@cs.auckland.ac.nz

Abstract

Representing knowledge in computer games in such a way that reasoning about the knowledge and learning new knowledge, whilst integrating easily with the game is a complex task. Once the task is achieved for one game, it has to be tackled again from scratch for another game, since there are no standards for interfacing an AI engine with a computer game. In this paper, we propose an Open AI Standard Interface Specification (OASIS) that is aimed at helping the integration of AI and computer games.

1. Introduction

Simulations with larger numbers of human participants have been shown to be useful in studying and creating human-level AI for complex and dynamic environments [Jones, et al. 1999]. The premises of interactive computer games as a comparable platform for AI research have been discussed and explored in several papers [Laird, & Duchi, 2000; Laird & van Lent, 2001]. A specific example would be *Flight Gears*, a game similar to Microsoft's Flight Simulator, that has been used for research into agents for piloting autonomous aircraft [Summers, et al. 2002].

If interactive computer games represent a great research opportunity why is it that we still see so comparatively little research being conducted with commercial grade games? Why is most AI research confined to games of the FPS genre and the mostly less complex open-source games? We believe that the single most important reason for this phenomenon is the absence of an open standard interface specification for AI in interactive computer games.

2. Standard Interfaces

Standard interfaces allow a piece of software to expose functionality through a common communication model that is shared amongst different implementations with equivalent or related functionality. The advantage of a common communication model is that other software may request similar services from different programs without being aware of a vendor specific implementation. The usefulness of standard interfaces has been widely acknowledged and found widespread application in many computing disciplines and especially within the software engineering community.

Successful examples of open standard interfaces in the industry are plentiful. They include TCP/IP, DOTNET CLI, XML Web Services, CORBA, SQL, ODBC, OpenGL, DirectX, the Java VM specifications and many others. We suggest that applying the same principle to AI in computer games would significantly reduce the effort involved in interfacing AI tools with different games. In the absence of a common communication model for interacting with the virtual worlds of computer games, AI researchers have to concern themselves with implementation specifics of every game they would like to interface with. This usually entails a significant amount of work especially with closed source commercial games that do not expose a proprietary mod interface.

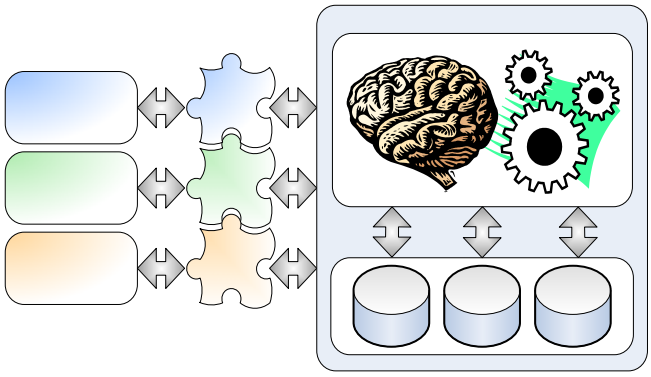

Fig. 1 Non-Standard Game AI Interfaces

Games in the FPS segment have been a leader in implementing proprietary mod interfaces to encourage third parties to develop mods (i.e. modification or extensions) to their games. These interfaces significantly reduce the effort required to build custom extensions to the original game engine. As a result of this FPS games have been used as a platform for AI research [Laird, 2000; Khoo & Zubek, 2002; Gordon & Logan, 2004]. Application of similar interfaces to real time strategy games has been suggested by some researchers [van Lent, et al. 2004]. Others have suggested game engine interfaces for supporting particular

areas of AI research such as machine learning [Aha & Molineaux, 2004].

However, proprietary mod interfaces, whilst having the potential to significantly reduce effort when working with a particular game, do not provide AI developers with the benefits associated with an open framework of standard interfaces. An AI mod created for one game will still have to be fitted with an additional interface module to be able to support another game (Fig. 1). This makes it difficult for AI researchers to validate their work across different computer games.

Rather than implementing non-standard interfaces for each and every computer game in the market, we believe it would be useful to create a set of open standard interface specification that are applicable to computer games of all genres. In addition an open standard interface specification for game AI would also have the potential of commercial success as it could provide a means of both reducing AI development costs by acting a guideline and boosting game popularity through third party add-ons while allowing intellectual property to be protected.

In brief, we are pursuing the following goals:

Simplicity: The interface specification should be simple, yet powerful and flexible enough to cover the various aspects of AI associated with computer games.

Extensibility: Modularity should be a core requirement, so that further functionality can easily be added to the framework as necessary.

Encapsulation: The interface specification should consist of layers that provide access to the game engine with an increasing degree of abstraction.

Cohesion: There should be a clear separation of function and logic. Each component of the framework should either play a functional (e.g. symbol mapping) or a logical role (e.g. plan generation), but not both.

3. Related Work

Researchers active in different areas of AI have long realised the importance of developing and employing standards that alleviate some of the difficulties of interfacing their research work with its area of application. Even though work in this area has led to advances in providing standardised tools for AI researchers and developers little work has been done in the area of standard interfaces. One of the main reasons for this is probably the heterogeneous nature of AI research. Computer games, however, represent a comparatively homogenous area of application and thus may see a more profound impact from standard interface specifications.

Past standardisation efforts in the area of AI can be roughly grouped into two categories:

1. work on standard communication formats and,
2. the development of standard AI architectures.

The development of standard communication formats is occupied primarily with the standardisation of expressive representational formats that enable AI systems and tools to flexibly interchange information. Work in this category encompasses standards such as KIF [Genesereth & Fikes, 1992] and PPDL [McDermott, et al., 1998]. We will refer to efforts in this category as *information centric standards*. Although information centric standards play a crucial part in the communication model of a standard interface specification they by themselves are not a replacement for such a framework. In addition, most of the information centric standardisation work in the past, whilst being well suited for most areas of AI, does not meet the performance requirements of computer games.

The second category of work comprises the creation of *architecture standards* for AI tools. Successful examples of such standard architectures are SOAR [Tambe, et al., 1995] and more recently TIELT, the Testbed for Integrating and Evaluating Learning Techniques [Aha, & Molineaux, 2004]. Architecture standards are similar to standard interface specifications in the sense that they involve similar issues and principles and both represent service centric standards. As such architectures like TIELT signify a cornerstone in reducing the burden on AI researchers to evaluate their AI methods against multiple areas of application through the interface provided by the TIELT architecture.

However, standard architectures cannot achieve the same degree of flexibility and interoperability as an open standard interface specification. The ultimate difference between something like TIELT and a standard interface specification is that a standard architecture functions as middle-ware. As such it is not directly part of the application providing the actual service, but acts as translation layer. Therefore it in turn must interface with each and every game engine that it is capable of supporting, just like an AI engine had to previously be interfaced with every game that it should be used with. This solution in its very nature only pushes the responsibilities of creating the actual interface to a different component – the underlying issue, however, remains unsolved.

The need for both researchers and developers to address the discussed issues with the present state of game AI and their current solutions has been indicated by the recent emergence of the game developer community's own efforts. These efforts, organised by the IDGA AI Special Interest Group through the game developer conference round table attempt to make some progress on AI interface standards for computer games. The IDGA AI SIG has established a committee with members from both industry and academia to accelerate this process. However, at the time of writing these efforts were still at a conceptual level and had not yet resulted in any experimental results.

4. OASIS Architecture Design

4.1 OASIS Concepts and Overview

Standard interfaces become powerful only when they are widely implemented by industry and other non-commercial projects. OpenGL and DirectX would be conceptually interesting, but fairly useless standard interface frameworks, if video card developers had not implemented them in practically all 3D acceleration hardware on the market. The OSI networking model on the other hand is an example of an academic conceptual pipe-dream. Viewed purely from an interface point of view, the OSI networking model is an incredibly flexible design that was in its original specification already capable of delivering much of the functionality that is nowadays being patched on to the TCP/IP networking model. However, the OSI networking model remains a teaching tool because it is too complicated to be practicable. Firstly, the process of arriving at some consensus was overly time-consuming because it involved a very large international task force with members from both academia and industry and attempted to address too many issues at once. Secondly, when the standard was finally released, it was prohibitively expensive for hardware manufacturers to build OSI compliant devices, especially in the low budget market segments. In comparison the TCP/IP model was developed by a much smaller group of people, is much simpler, and although failing to address several problems, it is the most dominant networking standard today [Forouzan, 2000].

Despite the shortcomings of TCP/IP, there is an easy explanation for its success; TCP/IP is simple, yet modular and extensible. The focus of TCP/IP is on necessity and efficiency rather than abundance of features. This is what makes it a successful standard interface. Thus we believe a standard interface for AI in games should be modular, extensible and simple while still fulfilling all core requirements. An Open AI Standard Interface Specification (OASIS) should feature a layered model that offers different levels of encapsulation at various layers, allowing interfacing AI modules to choose a mix between performance and ease of implementation adequate for the task at hand. Thus the lower layers of OASIS should provide access to the raw information exposed by the game engine, leaving the onus of processing to the AI module, while higher layers should offer knowledge level [Newell, 1982] services and information, freeing the AI developer from re-implementing common AI engine functionality.

We suggest there be a small number of layers in the OASIS framework. Each layer ought to be highly cohesive; that is, every layer, by itself, should have as few responsibilities as possible besides its core functionality and be completely independent of layers above it, thus allowing layer based compliance with the OASIS framework. This permits game developers to implement the OASIS specifications only up to a certain layer.

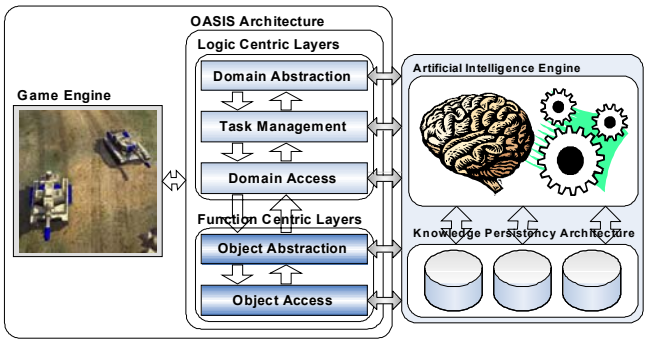

Fig. 2 OSASIS Architecture

This is especially useful when some higher layer functionality is either unnecessary due to simplicity of the game or because resource constraints imposed by computationally intensive games would not permit the use of layers with greater performance penalties without seriously impacting playability. Such implementation flexibility reduces both financial and time pressure on developers to comply with all specifications of the OASIS framework. Since modularity is a core feature of OASIS, compliance can be developed incrementally. This is not only good software engineering practice, but would allow game developers to provide patches after a game's release to add further OASIS compliance.

As a prototype design for the OASIS framework we have conceived a simple five layer architecture comprising:

1. an *object access* layer for direct manipulation of the game engine,
2. an *object abstraction* layer to hide runtime details from higher layers,
3. a *domain access* layer to expose the game domain in a form accessible to reasoning tools,
4. a *task management* layer providing goal arbitration and planning services, and
5. a *domain abstraction* layer that hides the complexity of the underlying game engine domain from more generic AI tools.

The bottom two layers of the architecture (i.e., 1 & 2) are function centric; that is, they are concerned mainly with the runtime specifics of single objects implemented in the game engine. In contrast the top three layers of the OASIS architecture (i.e., 3, 4 & 5) would be knowledge centric and hence would be concerned with manipulation of the domain at the knowledge level and are not directly interacting with single run-time objects. Note, that different from middle-ware architectures such as TIELT or SOAR, the OASIS framework is actually a set of specifications rather then a piece of software. The actual implementation details of the OASIS architecture should not matter as long as the interface specifications are complied with. This design makes the AI engine of a game a separate and readily interchangeable component.

The following sections discuss the suggested

functionality and responsibilities for each of the OASIS layers and their respective components depicted in Figure 2. All of this represents our initial ideas on how the OASIS architecture design could be structured and what features it might need to posses and should be considered neither final nor complete.

4.2 Object Access Layer

The access layer directly exposes objects defined in the game engine that may be manipulated by an interfacing AI engine. Objects exposed by the object access layer include everything from the tangible parts of the game environment such as an infantry unit to more abstract components such as the game state. For every object, the access layer specifies properties, operations and events that may be used to interact with the corresponding object in the game engine.

At the object access layer speed should be the main concern. Here the metadata should define information not observable from the signature of the object's operations and events such as preconditions, post conditions, extended effects and duration in terms of low-level descriptors. While this is computationally efficient processing is required before the information provided at this layer can be used to establish the semantics of the objects. In order to not impair performance each object would be a lightweight wrapper around its counterpart in the game engine, simply passing on the received messages with little or no intermediate processing (Fig. 2).

4.3 Object Abstraction Layer

The object abstraction layer provides framing of the resources provided by the object access layer into more readily usable structures. The function of the object abstraction layer is three fold, it manages all aspects of object assemblies, it orchestrates objects and assemblies to perform tasks and it compiles metadata from the data access layer into object semantics that define the logical relations between both objects in the game world exposed by the object access layer and object assemblies derived from those objects.

Object assemblies are essentially groupings of game objects with additional properties and functions that allow viewing and manipulating the underlying objects as a single unit. These groupings should be allowed to be very flexible for example it should be possible for an interfacing AI engine to define all objects of a specific type as an object assembly. Object assemblies themselves should in turn permit aggregation thus providing for recursive hierarchies of object assemblies. After creating a new assembly, the interfacing AI engine might then specify additional properties and operations that are not defined by the underlying objects thus making the game engine programmable without requiring access to the source code, which often represents a problem with commercial games. Since the behaviour and execution steps of user created properties and operations need to be explicitly specified some kind of high-level programming language must be part of this layer's protocol suite.

Another function of this layer is compiling the object metadata retrieved from the lower layer into logical relations between objects that are directly usable for example by an execution monitor to verify the progress of a plan and recognise its failure or success. These object semantics should also cover any object assemblies created by the user. This might necessitate the specification of metadata for object assemblies by the user if the metadata of the assemblies' components is insufficient to automatically derive the semantics of the assembly.

Lastly, the object abstraction layer is responsible for object orchestration. This means that it verifies the validity of execution of operations for both objects and assemblies and informs higher layers of invalid requests. It also deals with any runtime concurrency issues and processes events received from the object abstraction layer into a semantic format that may be used by higher layers for reasoning. This should effectively insulate the function centric from the logic centric layers of the OASIS framework.

The protocol suit required for communication with this layer would probably need to be more diverse than that of the object access layer. There are two main issues that need to be addressed. Firstly, fast access to the functions of the object access layer to allow for manipulating objects and assemblies. Secondly, capabilities for creating and programming of object assemblies. Although the focus of protocols at this layer should be to provide more abstraction, speed and lightweight remain a core requirement.

4.4 Domain Access Layer

The domain access layer provides a high-level abstraction of the game engine. This includes task execution management and domain description services. Task execution management is concerned with the execution of the logical steps of a plan specified in some expressive standard high level format. The task execution manager functions much like an execution monitor for planners. It translates the high level logical steps of a plan into an instruction format understood by the object abstraction layer, negotiates conflicts, monitors the execution results and informs higher layers of irresolvable conflicts and illegal instructions. The steps it executes may either manipulate objects within the domain (e.g. move tank X behind group of trees Y) or the domain description itself by creating or manipulating object assemblies in the object abstraction layer (e.g. add average unit life time property to infantry type assembly). Concurrency issues between competing plans executed in parallel need to be also managed at this layer. In order to reduce overhead this should occur as transparent as possible only making the AI engine aware of conflicts that are irresolvable.

The domain description component of this layer addresses two separate issues. First, it describes the semantics and mechanics of the domain created by the game engine in a standard high level knowledge representation. Second, it is directly usable by planners and other AI reasoning tools. The domain description provided should

include both native game objects and user created object assemblies. The other task of the domain description is to communicate to any interfacing AI engine the current state of the objects defined in the game world and any changes thereof.

The protocols used to communicate with this layer are fairly high level in terms of the information content they portrait. Optimally, the domain access layer should be able to support different formats for specifying plans to the task execution manager, so that AI engines using different types of AI tools may directly interface with this layer. In terms of protocols the domain description component is probably the most complex to address in this layer since it should allow a variety of AI tools to be able to directly interface with it. The domain description needs to be probably communicated in a variety of standards such as the planning domain description language developed for the 1998/2000 international planning competitions [McDermott, et al. 1998]. One of the major challenges posed by the protocol suite at this layer is to minimize the number of standards that have to be supported by default without limiting the nature of the AI tools interfacing to this layer. This could potentially be achieved by providing support for certain popular standards, while making the protocol suit pluggable and allowing third parties to create their own plug-ins to communicate with this layer. However, the feasibility of such an approach would need to be studied.

4.4 Task Management Layer

The domain abstraction layer, unlike all of the other layers, would not primarily serve the purpose of hiding the complexity of the lower layers from the layers above, but rather the provision of services that form an extension to the functionality of the domain access layer. Therefore some functions of the domain abstraction layer will not require the services provided at this layer. Thus in some cases this layer would be transparent to the top layer and simply pass through requests to the domain access layer without any further processing. Overall this layer should provide planning related services such as, plan generation, heuristic definition and goal management.

The plan generation capability of this layer is probably the single most important service offered here. It provides planning capabilities to the top layer as well as AI engines that do not posses the required planning capabilities to interact directly with the domain access layer. The plan generation component of the task management layer outputs a plan that is optimised using any heuristics given by the user and achieves the specified goals. This output plan is fed to the task execution management component in the layer below for processing and execution. The plan generation should be implemented very modular allowing third parties to create pluggable extensions to this functionality to adjoin different planning architectures to the OASIS framework that might not have been part of it originally. This would have two effects. First, this would enable AI researchers to verify, test and benchmark new planning architectures using OASIS. Second, it would provide an easy way to complement the set of the OASIS planners should there be shortcomings for certain kind of domains without needing to release a new version of the OASIS specifications.

Heuristic definition and goal management complement this planning capability. They allow AI engines to specify goals to be achieved and heuristics to be honoured by the planning component. The AI engine should be able to specify these in terms of symbols from the domain description provided by the domain access layer. The user should be permitted to prioritise goals and mark them as either hard goals that must be attained or soft goals that may be compromised. A planner in this layer should be allowed to re-shuffle the order of soft goals as long it does not increase the overall risk of failure. Any heuristics supplied by the AI engine are then applied to create a plan that will satisfy all hard goals and as many soft goals as possible.

Communication at this layer should use high level protocols describing both heuristics and goals in terms of the symbols found in the domain description at the layer below so that the planner does not need to support any additional mapping capabilities and may operate pretty much on the raw input provided. Excluding mapping and transformation capabilities from the task management layer will most definitely have a positive impact on performance.

4.5 Domain Abstraction Layer

The domain abstraction layer represents the top of the OASIS layer hierarchy and hence provides the greatest degree of abstraction from the implementation details of the game engine and the lower layers of the OASIS framework. High level functions such as domain model adaptation services, the domain ontology and task management services will be rooted at this layer. The main aim of this layer is to provide a knowledge level access point for AI reasoning tools that are either very limited in their low level capabilities or highly generic in their application. The interfaces provided by the domain abstraction layer and its components are not primarily geared towards speed, but much more towards interoperability and high level problem representation and resolution.

The domain model adaptation service provided here plays an important role in bridging the gap to generic reasoning tools and agents that are designed to handle certain tasks within a particular problem domain such as choosing what unit to add next to a production queue. Such problem description is very generic and will occur in slightly different variants in many games, especially in the real time strategy genre. Domain model adaptation will allow symbols of the domain defined by the game engine to be mapped to semantically equivalent symbols of the agent's domain model. In this way the agent can continue to reason in the confines of his own generic view of the world and is at the same time able to communicate with the game engine using expressions built from its own set of domain symbols. In order to facilitate this translation the domain model adaptation module would have rely on the ontology services provided by this layer and might in certain cases

require the interfacing AI engine to explicitly specify certain mappings. The domain model adaptation component is probably going to be by far the most complex and least understood component in the entire OASIS architecture. This is because domain model adaptation is still mainly a research topic although there are a few successful practical applications [Guarino et al. 1997].

The purpose of the ontology component of this layer is to provide a semantically correct and complete ontology of the symbols found in the domain description of the underlying game. Although fairly straight forward this could prove time intensive for developers to implement because it almost certainly requires human input to create a useful and comprehensive ontology for the game being built. Creating a standardised ontology for similar games and genres will be a key to successful implementation of this feature.

The second major service is task management. This involves facilitating the specification of very high-level tasks in terms of the elements contained in the ontology exposed at this layer and their completion using the functions provided by lower layers. Such task might in terms of logic resemble assertions like *"(capture red flag OR kill all enemy units) AND minimize casualties)*". The task management component would have to take such a task description and transform it into a set of goals and heuristics that may then be passed on to the task management layer. In order to extract goals, heuristics, priorities, etc. from the high-level task description, the interfacing AI engine would be required to flag the description's components. The task management component should also be responsible for tracking task progress and inform the AI engine of completion or failure. Concurrency issues of any kind and nature arising from competing tasks being executed in parallel should be handled by the lower layers.

5. Conclusion

Obviously there is still much uncertainty about the exact details of the OASIS framework and there are many issues that this paper has left unsolved. In the future it would probably be valuable to survey, document and analyse in detail the requirements of both game developers and AI researchers to form the basis of a formal requirements analysis. This would provide a better understanding of the problems being addressed and support a better set of design specifications for the OASIS framework. In the immediate future we will take advantage of the modularity and extensibility requirement of OASIS and implement a vertical prototype as a proof of concept. During this process we will also explore the usefulness and feasibility of some of the proposals made by the Artificial Intelligence Interface Standards Committee (AIISC) of the IDGA AI SIG. Potentially, a small number of diversified vertical prototypes might help us gain a more accurate understanding of the requirements for the framework that could form the stepping stone for further work in this area. We would also seek input and comments from other researchers and developers working in this area.

References

Aha, D. and Molineaux, M. 2004, Integrating Learning in Interactive Gaming Simulators, Challenges in Game Artificial Intelligence – Papers from the AAAI Workshop Technical Report WS-04-04

Forouzan, B. 2000, Data Communications and Networking 2nd Edition, McGraw-Hill

Gordon, E. and Logan, B. 2004, Game Over: You have been beaten by a GRUE, Challenges in Game Artificial Intelligence – Papers from the AAAI Workshop Technical Report WS-04-04

Guarino N., et al. 1997, Logical Modelling of Product Knowledge: Towards a Well-Founded Semantics for STEP, In Proceedings of European Conference on Product Data Technology

International Game Developers Association – Special Interest Group on AI (IDGA – AI SIG), http://www.igda.org/ai/

Jones, R., et al. 1999, Automated Intelligent Pilots for Combat Flight Simulation, AI Magazine

Khoo, A. and Zubek R. 2002, Applying Inexpensive AI Techniques to Computer Games, In Proceedings of IEEE Intelligent Systems

Laird J. 2000, It knows what you're going to do: Adding anticipation to a quakebot, Artificial Intelligence and Interactive Entertainment – Papers from the AAAI Workshop Technical Report SS-00-02

Laird, J. and Duchi, J. 2000, Creating Human-like Synthetic Characters with Multiple Skill Levels: A Case Study using the Soar Quakebot, In Proceedings of AAAI Fall Symposium: Simulating Human Agents

Laird, J. and van Lent, M. 2001, Human Level AI's Killer Application: Interactive Computer Games, AI Magazine Volume 2 – MIT Press

McDermott, D., et al. 1998, PDDL - The Planning Domain Definition Language, Yale Center for Computational Vision and Control - Technical Report CVC TR-98-003/DCS TR-1165

Newell, A. 1982, The Knowledge Level. Artificial Intelligence, 18 (1)

Summers, P., et al. 2002, Determination of Planetary Meteorology from Aerobot Flight Sensors, In Proceedings of 7th ESA Workshop on Advanced Space Technologies for Robotics and Automation

Tambe, M., et al. 1995, Intelligent Agents for Interactive Simulation Environments, AI Magazine

van Lent, M., et al. 2004, A Tactical and Strategic AI Interface for Real-Time Strategy Games, Challenges in Game Artificial Intelligence – Papers from the AAAI Workshop Technical Report WS-04-04

Genesereth, M. and Fikes, R. 1992**,** Knowledge Interchange Format, Version 3.0 Reference Manual*,* Technical Report Logic-92-1 – Computer Science Department Stanford University