HIGH LEVEL ARCHITECURE FOR DISTRIBUTED AGENT SIMULATION IN COMPUTER GAMES

Pawan S Kumar, Qasim Mehdi and Norman Gough
Research Institute in Advanced Technologies (RIATec)
University of Wolverhampton
Wolverhampton, UK WV1 1SB
{pawan.kumar, q.h.mehdi, n.gough}@wlv.ac.uk

KEYWORDS
High-level architecture, HLA, Unreal technology, game engines, Beowulf cluster, Gamebots, multi agent system, MAS, parallel and distributed simulation

ABSTRACT

Multi Agent System (MAS) have been in existence for quite a long time and have been a focused area of research in different paradigms over the years. As a result, several test beds and simulation tools exist for their deployment in software engineering problems. However, these test beds are suited for specific agents types and environments and lack support for interoperating with other agent architectures and environments. Further, they do not exploit the power of modern distributed and parallel computing environments. Here at RIATec, there is a powerful e-science Beowulf cluster that provides ample resources for distributed simulation of MAS. Such simulation has several possible target applications such as in computer games and training virtual environments and therefore it is important that such simulation is reused. Moreover, good visualization, rendering and interactive tools are needed for evaluating, testing and interacting with the system. In this paper, a proposal for a reusable architecture and visualization test bed based on High-level architecture (HLA) and Unreal technology is provided for a high performance MAS simulator.

INTRODUCTION

High-level architecture (HLA) is a general-purpose architecture (IEEE 1516 standard) envisioned by U.S. Defence Modelling and Simulation Office (DMSO) to support reuse and interoperability of large numbers of different types of simulations developed by the U.S. Department of Defence (DoD) (DMSO, 2004). It is a generic and language independent specification that allows for integrating separate simulation models on same or different platforms at geographically different places in a robust way over a network. It also provides specifications for advanced time and data management services for these simulations. This also allows for developing separate simulation models that can be reused in different applications.

Parallel and distributed simulation systems are mainly concerned with distributing the execution of the simulation across multiple machines or processors (Fujimoto R, 2003; Fujimoto R, 2001; Fujimoto R, 1999). *Parallel* simulation is concerned with execution of simulation on tightly coupled computer systems such as super computers or shared memory multi-processors. Here the main reason for distribution is to reduce the execution time of the simulations. Further, this distribution enables a large simulation program limited by memory requirements of a single machine to be executed by utilizing memory of many processors. On the other hand, *distributed* simulation is mainly concerned with the execution of simulations on loosely coupled systems where interactions take place between geographically distributed computers over local area network or wide area network. This type of distributed simulation is a typical of games and virtual environments that allows for creation of virtual worlds with multiple participants at physically different sites. Further, this type of distribution allows for separate simulation models to be integrated into a single simulation environment. However, with the advent of cluster and grid computing, this distinction between the parallel and distributed simulations has become less clear and are therefore all are referred to as forms of distributed simulation systems (Fujimoto R, 2003). HLA supports several existing parallel and distributed simulation algorithms. These algorithms are briefly discussed in this paper.

The high performance Beowulf parallel processing cluster located at RIATec will be used as a test bed for a generic distributed MAS simulator. Such simulation has several potential target applications such as emergency evacuation, military training, war games, multi-player online games, etc. Apart from reusing the simulator in different applications, the scalability of agents and complexity of environment could play an important part that may require adding more nodes to the cluster. Both of these issues can be handled by making the simulator HLA compliant.

In addition, a test bed is needed to visualize and interact with the agents. For this, Unreal engine provides a powerful and efficient 3D game simulation and visualization environment (Lewis M and Jacobson J, 2002). It not only incorporates highly optimised 3D graphics rendering capability but also has level editing tools that can expedite the modelling of agent's test environments. In fact Gamebots (Kaminka G A et al, 2001) based on Unreal client-server model has become a test bed for multi agent research. The overall system here would entail an HLA compliant Unreal in order to visualize agents' behaviours from the cluster.

The rest of the paper is organised as follows: A brief overview of HLA is provided first. Then an overview of parallel and distributed simulation algorithms is provided along with HLAs' support for these algorithms. This is followed by a review of how game engines like Unreal can be used for agent based research. Further, a brief review of MAS is provided which is followed by the proposed high level system design. Finally, there is conclusion and future work.

HIGH LEVEL ARCHITECTURE (HLA) OVERVIEW

HLA comprises of three main components namely the Federation framework and rules (IEEE 1516, 2000), the Run-Time Infrastructure (RTI) Interface specifications (IEEE 1516.1, 2000) and the Object Model Template (OMT) (IEEE 1516.2, 2000). Under HLA nomenclature a *federation* is an overall distributed simulation where the members participating in a federation are called *federates.* A *federate* can be an individual simulator application, a manned virtual training platform, a support utility such as data collector or viewer or even an interface to a live player (Dahmann J S, 1997). The RTI component is the software that implements the HLA interface specification and runs the federation. In effect, it is like a distributed operating system for federation that provides a set of services to support federate to federate interactions and federation management support functions. These services are shown in the Figure 1 along with the functional view of HLA architecture.

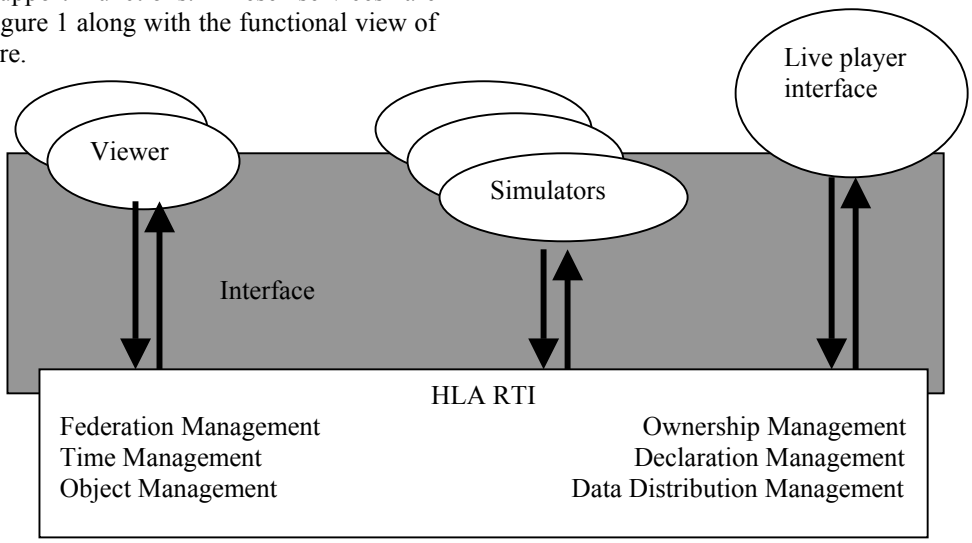

Figure 1: Functional view of HLA federation. Adapted from (Dahmann J S, 1997)

The HLA RTI interface specification describes six types of runtime services between federates and the RTI. The *Federation Management* service provides functionality that allows for creation of federations, joining and resigning of federates and the overall management and synchronization of federations. The *Declaration Management* service provides for an efficient data exchange between federates. Each federate use this service to define which data it will provide to and require from the federation. Through the *Ownership Management* service, RTI controls ownership of the HLA objects and their attributes. Only a federate owning the attribute can update the attribute of an HLA object in a federation. However, RTI can dynamically transfer the ownership of an attribute/object during a federation. The *Object Management* service facilitates the creation, deletion, identification and other services at the object level. The *Time Management* service is responsible for synchronisation of runtime data exchange. Through this service, RTI controls when federates update and receive the events. Finally, the *Data Distribution Management* services allows for efficient routing of data among federates. A thorough description of these services can be found in (IEEE 1516.1, 2000).

The HLA OMT is a standard template that defines form, type and structure of data shared within the federation. The OMT defines two formats for describing this information. Further, HLA specifies two types of object models namely *Federation Object Model* (FOM) and *Simulation Object Model* (SOM) and requires that these models be documented in accordance with the OMT. The FOM describes the set of objects, attributes and interactions that are shared across a federation. The SOM describes the simulation (federates) in terms of the types of objects, attributes and interactions it can offer to future simulations (Dahmann J S, 1997). A federation FOM is composed of parts of the SOM of all of its participating federates.

Finally, HLA specifies set of rules that apply to HLA federation and federates and is divided into two groups i.e. federation and federate rules. Some of these have been discussed above and more can be found in (IEEE 1516, 2000). These rules are described in the standard mainly to allow for interoperability of different simulation systems.

PARALLEL AND DISTRIBUTED SIMULATION ALGORITHMS

Taxonomy of parallel and distributed simulation algorithms is presented in Figure 2. These are categorized in two different classes namely *synchronous* and *asynchronous* algorithms. *Synchronous* algorithms uses global clock to synchronize the progress of each process in the distributed system. Each process advances its local clock to the time indicated by the global clock. The global clock in turn advances to either next time step or smallest time of next event. *Time driven* or discrete time algorithms advances time in fixed intervals. *Event driven* algorithms advances time based on the time stamp of the next unprocessed event. Only processes whose next event takes place at that time are allowed to execute a transition.

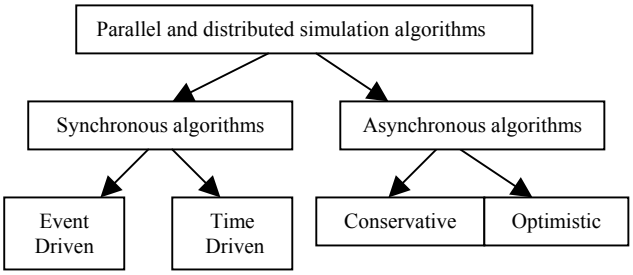

Figure 2: Taxonomy of parallel and distributed simulation algorithms

Asynchronous algorithms allows event processing beyond the global time of next event. Local clocks are synchronised whenever process interactions occur. Further, these algorithms could take *conservative* approaches that allow process to execute internal events so long as it receives any external input. Their main goal is to avoid any causality violations and they achieve this by using a lookahead value that predicts next output time of a process before the output generating state is reached. *Optimistic* approaches allow the processing of events well in advance of the global time of the next event. This may result in causality violations and when such violations are detected, a roll back mechanism is used to restore the state of the offending process to some safe checked point state. Then the process proceeds ahead once again.

For a complete description on these algorithms, reader is directed towards Fujimoto R (2001; 1999; 1990). In addition, HLAs' support for all these algorithms can be found in Fujimoto R (2003).

UNREAL TECHNOLOGY AND GAMEBOTS

Game engines have made their presence in scientific research (Lewis M and Jacobson J, 2002) because of their reusable and modular simulation code. They use flexible and robust technology for fast generation of virtual environments for research. In addition to advance rendering facility, they provide modules to deal with basic input output, sound, networking and general physics and dynamics. These features have made them as an ideal test bed for research in human level AI (Laird J E, 2002). In fact, Unreal Tournament (UT), released by epic games, has been used in several research projects. UT is a multi-platform multi-player network based game built on top of the Unreal engine. Further, it is inexpensive and provides a powerful and efficient 3D game simulation and visualization environment. The game is designed in such a way that it can be easily modified using Unreal's high-level language *UnrealScript* (Gamebots, 2004). With *UnrealScript*, user can modify most part of the game including levels and players by using accompanied tool *UnrealEd*-Unreal editor. This allows for custom building and loading of levels and models respectively. Moreover, Unreal engine includes *Karma* physics engine and a skeletal animation system that simplifies the modelling of physical processes.

UTs' networking architecture is based on a client-server model in which, server controls the interactions among clients and the authoritative state of the simulation (Wang J et al, 2003). The server is responsible for updating the states of the clients and sends them this information. Clients too, send data to the server about their actions and locally maintain a subset of the simulation state. This allows for a client predicting its next simulation state thus lowering the amount of data exchanges between a client and the server. The client gets the simulation state from the server through *replication* that deals with how information is exchanged between a client and the server. This powerful networking architecture of UT has lead to the development of a new research infrastructure namely *Gamebots* (Kaminka G A et al, 2001) developed jointly by University of Southern California and Carnegie Mellon University. Gamebots provides a powerful test bed for testing AI and multi-agent systems by utilising UTs' client-server approach. Typically, UT has two entities: a human player and a computer-controlled player or *bot.* Gamebots makes modification to the existing UT game by allowing the *bots* to be controlled by an external program through a TCP/IP socket connection as shown in Figure 3.

The Gamebots server (Figure 3) consists of two components i.e. Gamebots module and UT game server. The Gamebots server provides sensory information to, and receives information from, both Gamebot clients and Unreal clients using sockets (Figure 3). The Unreal client is the human controlled character/ viewer whereas the Gamebot client is an external AI controlled bots. The Gamebots module accepts information from Gamebot clients and passes this information to the Unreal server that will update the state of agents represented by the Gamebot clients on Unreal clients. This architecture allows for human players to interact with the agents thus providing a test bed to study human-agents collaborative behaviours. More on Gamebots can be found on their published website (Gamebots, 2004).

Silverman B et al (2005) used another variant of Gamebots engineering where instead of utilising UT client-server architecture as above, they used Microsoft Component Object Model (COM) interchange standard

in a client-server approach. Their agent program acted as a COM server that exposes itself to any other COM aware application. On the Unreal side, DLLs were designed to work with Unreal Script that turned Unreal into a COM client. This approach offers more flexibility as it does not impose limits imposed by UT game server to certain number of connections. However as it uses COM, it is restricted to only Windows based platforms.

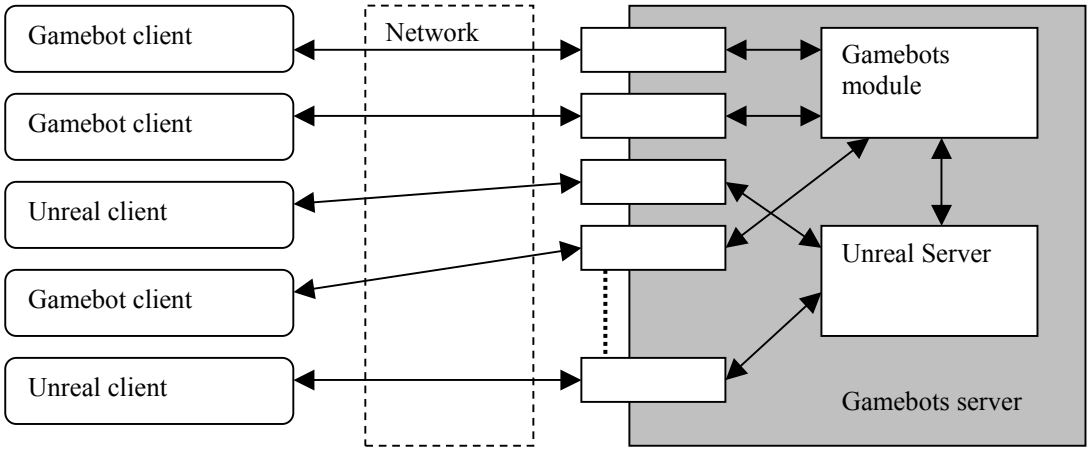

Figure 3: Gamebots architecture. Adapted from (Kaminka G A et al, 2001)

Further, as HLA is being used in this work, this would require an HLA compliant Unreal Tournament equivalent of Gamebots. For this, MAK technologies offers a product namely HLA GAME-LINK (MAK, 2005) that allows for an HLA compliant UT and thus would alleviate any additional overheads required for making an HLA compliant UT.

MULTI AGENT SYSTEMS (MAS)

Multi Agent System (MAS) have been in existence for quite a long time and have been a focused area of research in different paradigms over the years. From computer games perspective, these are non-player characters that are controlled by the computer. Typically these characters are computational entities that perceive and act and decide their actions in accordance with their task and goals. Although there is no pure definition that completely defines an intelligent agent that comprises of MAS, there exist certain facts about agents to which all agrees. From the implementation point of view, there exists three different agent architectures that are classified as *reactive*, *deliberative* or *hybrid* (reactive + deliberative). *Reactive* agents are simple behaviour based agents that operate in a hard-wired, stimulus-response manner (Mac Namee B, 2004). These are implemented in a complete deterministic manner and agents do not preserve any internal state of the world and are also not capable of long term planning and reasoning. On the other hand, *deliberative* agents build internal models of their world and are capable of formulating plans to achieve goals. This leads to added complexity in this type of system that may not live up to the real-time requirements as needed in several application areas like games (Mac Namee B, 2004). *Hybrid* agents combine the best of both (reactive and deliberative) agent types. In addition, agents within the MAS can further be classified as *homogeneous* or *heterogeneous*, *competing* or *benevolent*, *stable* or *evolving*, *communicating* or *non-communicating*, etc based upon their abilities, environment and goals, etc. A comprehensive literature on these agent types can be found in (Stone P and Veloso M, 2000; Kaminka G A, 2004).

Apart from agent types, MAS can be organised in three levels as proposed by Gurvitch (Ferber J, 1999). At *micro-social* level, there are agents interaction with two or small number of agents. At *groups* level, the main interest is in the intermediate structures that intervene in the composition of a more complete organisation. At this level, study mainly involves at the differentiations of roles and activities of the agents, the emergence of organisational structures between agents and the general problem of the aggregation of agents during the constitution of organisation (Ferber J, 1999). Finally, at level of *global societies* (populations), interest is mainly on the dynamics of a large number of agents together with the general structure of the system and its evolution. This work, mainly focus at the groups and societies level.

Agents and MAS are generally characterised by their architectures, environment and behaviours. Over the years, lot of research has been carried out in MAS and as a result several test beds and simulation tools exist to support the design and analysis of agent architectures and systems. Some of these can be found in these (Durfee E H and Montgomery T A, 1989; Collier N, 2001; Gasser L et al, 1992; MULTIAGENT, 2005). However, no one of them can be applied to all agent architectures and environments. Further, as the scalability of agents and complexity of environments increases, the computational requirements of MAS exceeds far beyond the capabilities of a single computer. Moreover, a good visualization, rendering and interactive tools are needed to analyse, test and interact with the system. All these issues are considered in the proposed design.

PROPOSED SYSTEM DESIGN

Here we combine HLA and Unreal to provide an overall system design for a high performance distributed agent simulator. Figure 4 shows the design of the system. The system can be subdivided into three components namely the Beowulf cluster, HLA RTI and the UT GAMELINK.

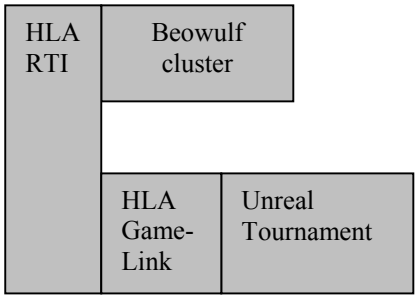

Figure 4: Proposed high level system design.

The e-science Beowulf cluster is a 9-node cluster. Each node has a dual processor of Pentium 2.8 GHz machines with 4Gb of shared ram and 40Gb of shared hard disk. In addition, there is 400Gb of shared disk available to all the processors. The cluster is part of RIATec e-science grid that allows it to be accessed and used by network of high performance machines utilising the grid infrastructure. This cluster forms an open platform development environment for a MAS simulation. This has significant performance benefits over loosely coupled network of computers and further, the MAS can be made to scale by adding more nodes to the cluster.

HLA RTI, which is the software implementation of HLA specifications, is responsible for running and maintaining the overall distributed simulation. All nodes within the cluster use services from the RTI and also interact directly with each other. The message passage among the agents can be made through the RTI and/or directly among the nodes using Message Passing Interface library (MPI) (Pacheco, 1997). Further, with RTI up and running, this simulation can be made to run with any other HLA compliant simulator.

Finally, the third component that consists of UT HLA GAMELINK would allow for visualizing, testing, analysing and interacting with the distributed agent simulation from the cluster.

CONCLUSION AND FUTURE WORK

In this study, a brief overview of tools and commercial of the shelf (COTS) game engine uses in scientific research in the area of multi agent simulation has been presented. A short description of parallel and distributed simulation algorithms has been presented along with an overview of HLA and MAS. HLA provides for interoperability and reusability of a simulation model in different environments and it also supports for several parallel and distributed simulation algorithms. Unreal provides powerful 3D visualization and simulation environment along with user-friendly tools for rapid prototyping of agents' environments. Thus, based on HLA and Unreal, a high-level system design has been proposed in this paper.

A first step to future work would require a thorough investigation on discrete event systems specification (DEVS) (Zeigler B et al, 2000) that provides a sound framework for modelling and simulation of a generic system. A typical agent uses a perceive-decide-action cycle that is modelled based on discrete event cycle. In addition, further research is needed in the area of mixed continuous-discrete systems. Continuous systems are needed to handle the dynamics or physics along with discrete systems for modelling and simulation.

REFERENCES

Collier N, 2001. "Repast. The REcursive Porous Agent Simulation Toolkit" [Online], Available: http://repast.sourceforge.net/ [Accessed 09 February 2005]

Dahmann J S, 1997. "High Level Architecture for Simulation", In Proceedings of the 1st International Workshop on Distributed Interactive Simulation and Real-Time Applications, Eilat, Israel, January 1997.

DMSO, 2004. "High Level Architecture" [Online], Available: https://www.dmso.mil/public/transition/hla/ [Accessed 16 December 2004]

Durfee E H and Montgomery T A, 1989. "MICE: A Flexible Testbed for Intelligent Coordination Experiments", in proceedings of 9th Distributed Artificial Intelligence Workshop. Rosario, Washington, 1989. pp 25-40

Ferber J, 1999. "*Multi-Agent Systems An Introduction to Distributed Artificial Intelligence*" Addison Wesley, Harlow, England, 1999

Fujimoto R, 1990. "Parallel Discrete Event Simulation" Communications of the ACM, Volume 33, Issue 10, October 1990, pp 30-53

Fujimoto R, 1999. "Parallel and Distributed Simulation" in proceedings of 31st conference on Winter simulation: Simulation---a bridge to the future, Volume 1, Phoenix, Arizona, United States, 1999

Fujimoto R, 2001. "Parallel and Distributed Simulation Systems" in proceedings of 33rd conference on Winter simulation, Arlington, Virginia, United States, 2001, pp 147-157

Fujimoto R, 2003. "Distributed Simulation Systems" in proceedings of 35th conference on Winter simulation: driving innovation, New Orleans, Louisiana, United States, 2003, pp 124-134

Gamebots, 2004. [Online], Available: http://www.planetunreal.com/gamebots/ [Accessed 16 December 2004]

Gasser L, Braganza C, Herman N, 1992. "MACE: A Flexible Testbed for Distributed AI Research", In Hunhs M N, ed. *Distributed Artificial Intelligence*, Pitman Publishers, 1987. pp 119-152.

IEEE 1516-2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Framework and Rules, September 2000

IEEE 1516.1-2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Federate Interface Specification, September 2000

IEEE 1516.2-2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Object Model Template (OMT) Specification, September 2000

Kaminka G A, Schaffer S, Sollitto C, Adobbati R, Marshall A N, Scholer A, Tejada S, 2001. "GameBots: A 3D Virtual World Test-Bed for Multi Agent Research", in proceedings of the 2nd International Workshop on Infrastructure for Agents, MAS, and Scalable MAS, Montreal, Canada, 2001.

Kaminka G A, 2004. "Multi-Agent Systems" in the *Encyclopaedia of Human Computer Interaction,* Berkshire Publishing, 2004 and [Online], Available: http://www.cs.biu.ac.il/~galk/ [Accessed 15 May 2005]

Laird J E, 2002. "Research in Human-Level AI Using Computer Games", Communications of the ACM, January 2002.

Lewis M and Jacobson J, 2002. "Game Engines in Scientific Research", Communications of the ACM, January 2002.

Mac Namee B, 2004. "Proactive Persistent Agents: Using Situational Intelligence to Create Support Characters in Character-Centric Computer Games", PhD Thesis, University of Dublin, Trinity College, 2004

MAK, 2005. MAK HLA GAMELINK [Online], Available: http://www.mak.com [Accessed 15 May 2005]

MULTIAGENT, 2005. [Online], Available: www.multiagent.com [Accessed 15 May 2005]

Pacheco, 1997. *Parallel Programming with MPI.* Morgan Kaufmann Publishers, San Francisco, California, 1997

Silverman B, Bharathy G, O'Brien K, Cornwell J, 2005. "Human Behaviour Models for Agents in Simulators and Games" [Online], Department of Electrical and Systems Engineering, University of Pennsylvania, Available: http://www.seas.upenn.edu/%7Ebarryg/PRESENCEpt2.doc [Accessed 15 May 2005]

Stone P and Veloso M, 2000. "Multiagent Systems: A Survey from a Machine Learning Perspective", in Autonomous Robots, Volume 8, Issue 3, June 2000, pp 345-383

Wang J, Lewis M, Gennari J, 2003. "A Game Engine Based Simulation of The Nist Urban Search And Rescue Arenas " in proceedings of 35th conference on Winter simulation: driving innovation, New Orleans, Louisiana, United States, 2003, pp 1039-1045

Zeigler B, Praehofer H, Kim T G, 2000. "*Theory of Modeling and Simulation, 2nd edition*", Academic Press, New York, 2000