An Architecture for Rapidly Reconfigurable MOUT Simulations

Shang-Ping Ting
Defence Science & Technology Agency
Singapore
tshangpi@dsta.gov.sg

Suiping Zhou
Nanyang Technological University
Singapore
asspzhou@ntu.edu.sg

Abstract

Entertainment technology, such as game engines, can provide low cost operational realism for Military Operations on Urbanized Terrain (MOUT) simulations. However, apart from achieving low cost and high fidelity virtual environments, MOUT simulation systems should also be adaptable with the frequent shifts in operational needs. We found that existing system architectures may not be optimized for reconfiguration with a game engine based approach. To this end, we describe our work on building agile and reconfigurable systems that can construct MOUT virtual environments rapidly with game engines. We propose the Architecture for Rapid Configuration (ARC) to support model interoperability and reusability. Along with ARC, we will explain how two other components, namely Game-based Object Standard Interfaces (GOSI) specifications and the ARC-based Objects Repository (ARCOR), work in tandem with the ARC. In our previous work, we constructed Twilight City, a virtual training environment for MOUT. Our experiences show that the proposed architecture has transformed Twilight City into a rapidly reconfigurable simulation system.

1. Introduction

According to a United Nation's report [1], the world population in urban areas will grow from 2.86 billion in 2000 to 4.98 billion by 2030. The rapid urbanization infers that Military Operations on Urbanized Terrain (MOUT) are getting increasingly important. However, live MOUT trainings are costly and time consuming. To avoid the high costs and lengthy turnaround time, building virtual MOUT environments seems to be a logical and powerful alternative.

However, building high resolution virtual environments at a low cost is insufficient to fulfill the requirements of a modern simulation system. New generation simulation systems should be agile and rapidly reconfigurable due to a growing emphasis to reduce the development time. Such a system is possible if development efforts can be minimized by composing existing models from a model repository into new virtual environments. To achieve reusability and composability, the simulation systems must have interoperable models. Presently, model interoperability is a huge challenge due to the application specific nature of existing models. These models are built with a stovepipe development approach. As a result, replicates of existing models are continually being rebuilt for different applications. This needless reinvention causes costly and time consuming development iterations. To address these issues, an elegant architecture is needed to quickly develop virtual environments and support reuse from a common repository of interoperable and extensible models.

Many experts are involved with ongoing work on model interoperability and reusability. The Simulation Interoperability Standards Organization (SISO) holds many conferences to promote model interoperability, reusability and composability [2]. In [3], Ray J. Paul discussed the impact of model reuse in the quality of simulations. Bob Lutz explored and did some groundbreaking work on interoperability concepts. Paul Gustavson worked on the piece-part concepts of Bob Lutz and created the Base Object Models (BOMs) to enable interoperable simulation components [4], [8], [9].

Although there is ongoing research in the area of interoperability, it is observed that more work can be expanded in the area of interoperability between models supported by game engines. Unlike common models, game engine based models are developed at a high level as they can utilize the application

programming interfaces (API) of game engines to perform low level tasks. Therefore, any architecture or standard enabling interoperability between the high level models need to consider the interactions of high level models with their low-level APIs.

In this paper, we propose the Architecture for Rapid Configuration (ARC) to support model interoperability and reusability for MOUT simulations built on top of game engines. Along with ARC, we will explain how two other components, the Game-based Object Standard Interfaces (GOSI) specifications and the ARC-based Objects Repository (ARCOR), works in tandem with ARC to provide an agile and easily re-configurable system. With ARC, we will be able to rapidly configure MOUT virtual environments with the appropriate scenario context.

The remainder of this paper is organized as follows. In Section 2, we give an overview of the *Twilight City*, a virtual training environment for MOUT. The ARC, GOSI and ARCOR are elaborated in Section 3. Some examples of using the proposed architecture in *Twilight City* are discussed in Section 4. Conclusions are drawn in Section 5.

2. Overview of Twilight City

The objective of building the *Twilight City* [6] is to create a high fidelity training and simulation system for MOUT. *Twilight City* aims to act as a testbed for various MOUT operations such as hostage rescue or raid operations. A typical scenario in *Twilight City* is the special squad operation to save hostage held in a building by terrorists. The squad may consist of several human-controlled characters and some AI (artificial intelligence)-driven Non-player characters (also known as *bots*). The terrorists may also be some human-controlled characters and bots. Various tactics can be implemented in *Twilight City*, and the users (trainees) can get familiar with the real operation environments, assess the effect of different tactics, and choose the best tactics to be used in the real operations. Figure 1 shows some screen shots of *Twilight City*.

Twilight City is built using a commercial game engine, the *Unreal Tournament* (UT2004) engine. The motivation of using a game engine lies in the rapid growth of commercial game engines that are affordable and also provide excellent support for high fidelity 3D modeling. With the support of game engines, developers may focus on human behavior models and tactics rather than on the 3D graphics. The native support from game engine APIs enables rich and high level programming mapping game-specific concepts onto the language definition. Therefore, the construction of high quality virtual environments and models are greatly simplified.

(a) Squad operation

(b) Crowd

Figure 1. Twilight City

Twilight City is designed to work on top of the UT engine with various modifications. These modifications adapt various UT game features to cater for MOUT simulations. With these modifications, *Twilight City* can be used as a MOUT simulation platform for various scenarios. *Twilight City* uses client-server architecture and allows for multi-player setup. In particular, an AI framework is implemented to generate realistic tactical behaviors for AI bots, a speech module is introduced for human voice recognition, customized animations and special effects are used to enhance the fidelity of environment and enrich a human player's experiences in *Twilight City*.

Various tactical scenarios are used to evaluate the performance of *Twilight City*. Case studies show that *Twilight City* has high fidelity on visual effects and the AI bots also demonstrate some human-like tactical behavior [6], [7].

Although the adaptation of game engines for simulation is gaining popularity, there are no existing architectures enabling interoperability and composability in this area. Our experience in constructing *Twilight City* shows that it is still very time consuming to build the simulation environment from scratch due the large number of models involved.

We have incorporated qualitative physics for modelling movable objects in *Twilight City*. Qualitative methods enhance the realism of *Twilight City* by adding more movable objects without significantly increasing computational overheads. In this paper, we will show how the proposed *ARC* helps to speed up the modelling of the qualitative physics objects.

3. Achieving Rapidly Re-configurable MOUT Simulations

In this section we will describe the *ARC* and its major components that drive the evolution of *Twilight City* into a flexible tool that can rapidly configure itself to meet different MOUT simulations requirements.

At the conceptual level, the *ARC* is designed to facilitate task abstraction and modularity in design. The *ARC* provides a framework that supports composability and interoperability. The *GOSI* specifications control the interactions between modules and acts as the linkages that hold the objects together within the framework of ARC. The ARC-based objects in the rich repository of ARCOR can be activated to compose simulation entities rapidly.

3.1 Overview

This paper assumes a basic understanding of game engine specific object oriented concepts such as the object graph, serialization, object lifetime, and polymorphism. We adopt an object-oriented modelling approach to creating reusable and composable objects in MOUT simulations. With this approach, complex simulation models are broken down into some basic building blocks. For the purpose of this paper, anything that exists individually within the virtual environment is described as an *entity*. Examples of *entities* are bots, movable objects, vehicles, etc. *Entities* contain components described as *objects*. *Objects* are responsible for the behaviors of the *entities*. In our test bed, *objects* are the superset from which *actors* and *models* are extended from via separate branches from the abstract base class, *Object*. *Actors* only store the current state of the simulated *entity*. *Models* are further segmented into *interaction models* and *logic models*. *Interaction models* are the data exchange layers between *logic models*, *actors* and the virtual environment. *Logic models* should only exchange data via *interaction models*. With the data, they will perform reasoning and decision making. Thus, an *entity* within *Twilight City* will contain an *actor* attached *with logic models* and *interaction models*.

The class hierarchy in Figure 2 shows how the *object* models are extended from the base class. *Weapons* and *human* actors are extended from the base *actor* class. Models such as *Path Finding* and *brain* are also extended from the base *model* classes. Characters such as *terrorists* and *soldiers* are sub classes of the *human actor* class. Similarly, different weapons such as *bombs* for the *terrorists* and *rifles* for the *soldiers* can be produced from extending the *weapon* class. After the required *objects* are formed, the resulting *entity* is built by putting the o*bjects* together. For example, the *soldier entity* is built up by the *rifle*, *human motion* and *soldier brain* objects. Likewise, the *terrorist entity* is an aggregation of the *bomb*, *human motion* and *terrorist brain* objects.

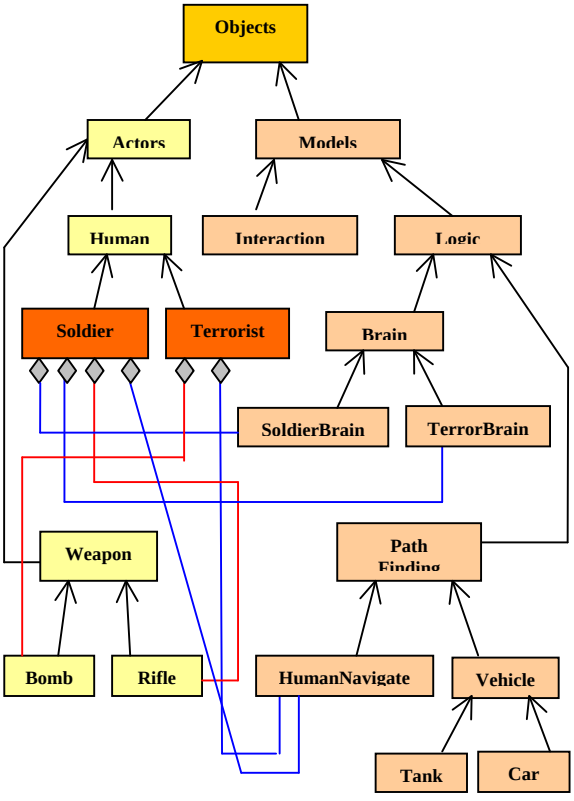


Figure 2. Class Hierarchy of ARC based objects

3.2 Architecture for Rapid Configuration

Actors, *logic models* and *interaction models* are the building blocks of the *ARC*. *Entities* houses *actors*, *logic models* and *interaction models in* relationships clearly defined by *ARC*. The explicit definition of tasks allows model substitution and ensures modularity. Being modular, specific components of an *ARC-based entity* can be modified without affecting the rest of the components. In figure 3, we illustrate ARC with a simple diagram.

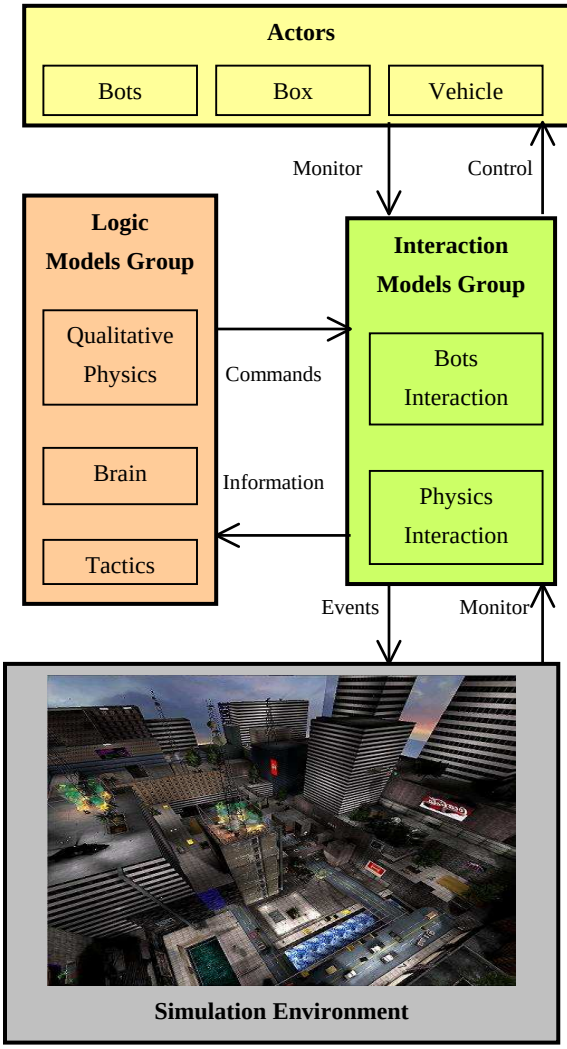


Figure 3. Architecture for Rapid Configuration

Being able to access game engine APIs, *actors* enable *entity* to move, interact with other *entities* and perform useful actions. The high-level artificial intelligence such as physics reasoning is built in *model* classes. *Models* are attached to *actor*s in a plug-and-play manner to provide *entity* behaviors. The function of each *model* should be distinct to ensure modularity. With *ARC*, simple *models* are composed together to perform complex behaviors. *Models* are separated into the *interaction models* and the *logic models*. *ARC* allows coordination between *logic models* and *interaction models*. The *interaction models* retrieve information from the *actors* and the environment for the *logic models* to perform reasoning. The *logic models* affect the *actors* and the environment via the *interaction models.*

Being a modular framework, *ARC* allows the migration and swapping of *models* in an *entity* while minimizing the propagation effect sustained by the remaining *models*. The ARC-supported model swapping provide flexibility for the simulation system to switch *entity* behaviors rapidly. With a common architecture, it will also be more convenient to construct interoperable models for sharing and reuse.

3.3 Game-based Objects Standard Interface

To be a rapidly re-configurable system, the frequent model swapping is expected. If there is no standardization of model data exchange methods, much time and efforts will be incurred when a model is migrated. Therefore, we propose the *Game-based Objects Standard Interfaces (GOSI)* to provide standard interfaces specifying data exchange methods. The *objects* will adopt interfaces according to their individual functions. Figure 4 shows an example of *GOSI* interfaces for the *actors*, *interaction* and *logic models* of movable boxes.

```
Actor MovableBox{
// Execute Game Engine API
void Rotate (PsiVelocity, PhiVelocity, ThetaVelocity);
void MoveTo(x, y, z);
//Internal functions
void BeginPlay();
void PostBeginPlay();
void Tick();
//Information functions
void GetMovableBoxWeight();
void GetMovableBoxDensity();
void GetMovableBoxForce();
//Attributes
int x, y, z;
float density, weight;
```

```
Interaction Model BoxInteract{
// Executing functions
void BoxRotate (PsiVelocity, PhiVelocity, ThetaVelocity);
void BoxMove(Displacement);
//Information functions
void GetBoxWeight();
void GetBoxForce();
//Attributes
float weight, density, force;
}
```

```
Logic Model BoxPhysics{
// Physics reasoning functions
Strength AngularEffect (Force, Weight);
Displacement DisplacementEffect(Force, Weight);
```

Figure 4. Game-Based Object Standard Interfaces

The *BoxPhysics* model can be decomposed into *AngularEffect* and *DisplacementEffect logic models.* The two separate components perform distinct functions and will again follow a set of *GOSI* interfaces. The two *logic models* can be composed to form the *BoxPhysics model*.

Defining *GOSI* specifications to standardize data exchange can achieve interoperability, reusability and composability. Presently, *GOSI* is largely working within the MOUT simulation domain. *GOSI* relies on the fact that general MOUT operational concepts are similar across projects [16]. For example, most MOUT simulation projects will need human behaviors, weapons, sensors and missions. By standardizing *object* interfaces according to their domain, scope and function, information exchanges between *models* can be controlled. To share *models* across different game engines, wrappers shall be used to convert the game-specific code to suit another game engine. However, as this is beyond the scope of this paper, we shall not go into greater details on this.

3.4 ARC-Based Objects Repository

It is important to build a repository to store all the *ARC-Based Objects* for future re-use. From figure 2, it is seen that the *ARC-Based objects* are held within a class hierarchy. To facilitate the rapid retrieval of objects within *ARCOR*, *objects* are kept in the ARC-based class hierarchy. The capabilities and limitations of the *objects* should be documented to give a working knowledge on the *objects*. For validation purposes, *objects* should record the projects that they participated before. Note that confidence level increases with the frequency in usage.

The development flow with an *ARC-based* approach for MOUT simulations is shown in Figure 5. The simulation is made up of many *entities*. Each *entity* exhibits distinct behaviors as controlled by its *objects*. We compose *entities* by assembling the *objects* in an *ARC* structure. From Figure 5, we can see that after identifying the purpose of an *object*, the conceptual model for the *object* will be defined. Then the *ARCOR* is scanned for an existing *object* similar to the conceptual model. If an appropriate *object* is found, the *object* will be used to assemble the framework of the *ARC-based entity*. Otherwise, a new *object* needs to be built. This new *object* shall adopt an appropriate interface based on the *GOSI* specifications. Subsequently the *object* will be further broken down into smaller components which can be composed together to form the original *object.* As shown by the dotted line in Figure 5, a multi-layer search for the composable components will be performed. The development process will be repeated recursively until the *entity* obtains all required *objects*. If a conceptual model cannot be further decomposed, the *object* shall be developed from scratch. Thus, all *objects* are constructed either by composing existing *objects* or from scratch. All newly built *objects* are added to *ARCOR* for future re-use.

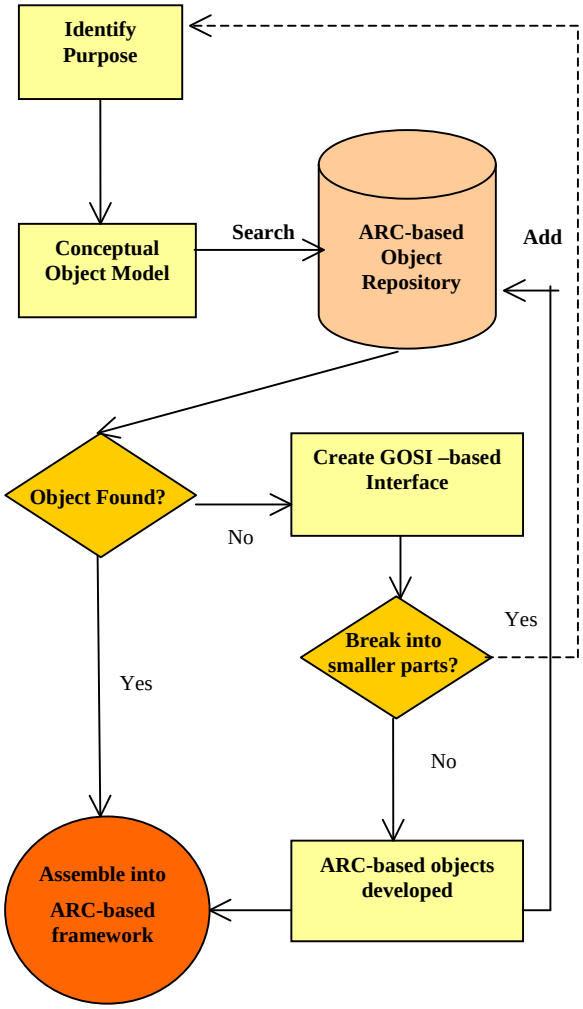


Figure 5. ARC-based Objects Development

4. Example results

In this section, we describe how the proposed framework helps to extend our previous work on *Twilight City*. As in most First Person Shooter (FPS) games, most objects in *Twilight City* are static objects, i.e., they cannot move. The reason for this is that modeling and implementing movable objects is a challenging task in terms of computational costs and mathematical complexities. The responsiveness of

the system may be greatly affected by simulating the behaviors of the movable objects. However, as movable objects such as bottles, boxes and chairs, etc. are quite common in real-life, it is important to incorporate these objects in the virtual environments to enrich a human player's experiences. Figure 6 shows some of the movable objects that we implemented within Twilight City.

(a) Falling boxes

(b) Falling barrels

Figure 6. Movable Objects in Twilight City

In *Twilight City*, we adopt an approach of Qualitative Physics (QP) to model the behaviours of movable objects. This approach will help to reduce the computational cost of simulating behaviours of movable objects and increase the fidelity level. We implemented a *QP* engine to simulate the behaviour of various movable objects. Figure 7 shows how the *QP* simulation on movable objects is achieved. This part of work had been presented in the 39th Annual Simulation Symposium [5].

Using *ARC*, we streamlined the development process of *QP* entities within *Twilight City*. We rebuilt the *QP* components in accordance to the *ARC* framework. The interactions between components in the dotted circle in Figure 7 were modified with the influence of *ARC*. Instead of both the *event handlers* and *object behavior* accessing the *Qualitative Physics Engine* separately, the *object behavior* should only interact with the *QP engine* via the *event handlers*.

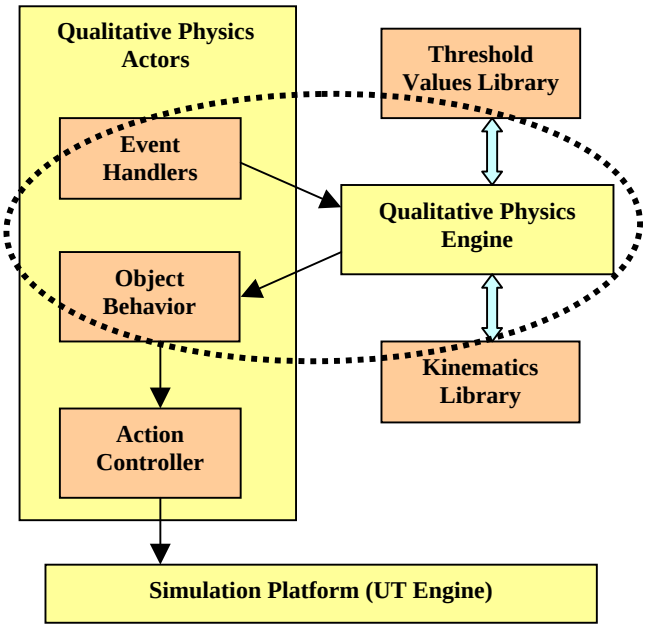


Figure 7. Qualitative Physics Framework

The *Event Handlers* in Figure 8 are built using the *interaction models* from the previous project within the *ARCOR*. These *interaction models* are constructed based on the *GOSI* to ensure composability and interoperability. Similarly, the *Object Behavior* module is built up by composing the *logic models* stored in within the *ARCOR*. *Qualitative Physics Actors* are extended from the existing *actors* as well.

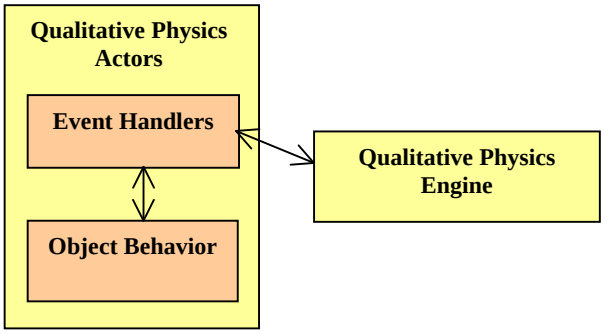
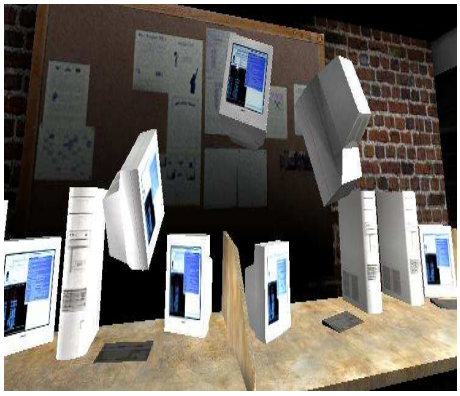


Figure 8. ARC- based Qualitative Physics

With *ARC*, the development time for creating various movable objects in *Twilight City* was

shortened from a projected 8 months to 2 months. Figure 9 shows some examples of movable objects that we implemented in *Twilight City*.



(a) Movable Computers

(b) Tumbling Tires

(b) Falling chairs

Figure 9. ARC- based QP Movable Objects

There are many other benefits of using the *ARC* framework, e.g., the rapid generation of terrorist behaviors in *Twilight City*. The terrorist behaviors are extended from the generic human *entity* and are combined with existing models such as the shooting actions of the soldier models. When there is a need to change the behavioral patterns of terrorists, we can conveniently adapt to this change by extending the existing models in the *ARCOR*. At the current stage of our project, we have implemented more than 50 *ARC-based* models. We expect that the current model repository will greatly shorten the development time of various MOUT simulation setup and fulfil the goal of building a rapidly configurable simulation system.

5. Conclusions and future work

In this paper, we propose the *Architecture for Rapid Configuration (ARC)* to build rapidly reconfigurable game engine based simulation systems. At the same time, *ARC* will enable interoperability, reusability and composability among various *models* across different MOUT simulations. The *GOSI* specifications decide the data exchange interfaces within the *ARC*. The model repository, *ARCOR*, is developed to store various commonly used *object* classes. With *ARCOR*, new models can be easily built by reusing the existing models. This reduces the cost and time to build new simulation environments. Thus, the developer can focus on refining the system rather than wasting time on needless reinvention of existing components.

We will continue to work on the proposed *ARC*, improve the *GOSI* specifications, and enrich the *object* classes in the ARCOR. In particular, we shall focus on models for representing human behaviors and movable objects.

6. References

[1] Source: Findings of the 2004/05 UN-HABITAT report '*State of the world's cities*'

[2] Simulation Interoperability Standards Organization, www.sisostds.org

[3] Ray J. Paul and Simon J.E. Taylor, "What Use Is Model Reuse: Is There A Crook at the End of the Rainbow?", in *Proceedings of the Winter Simulation Conference*, San Diego, CA, December 8-11, 2002

[4] Paul Gustavson, "Building and Using Base Object Models (BOMs) for Modeling and Simulation (M&S) focused Joint Training", in *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference* (I/ITSEC), Orlando, FL, November 28-December 1, 2005

[5] Shang Ping Ting, S. Zhou, “Qualitative Physics for Movable Objects in MOUT”, in *Proceedings of the 39th Annual Simulation Symposium*, Huntsville, AL, April 2-6, 2006

[6] S. Zhou, S.P. Ting, Z. Shen, and L. Luo, “Twilight City – A Virtual Environment for MOUT”, submitted to *International Journal of Computers and applications*

[7] Z. Shen., S. Zhou, C. Y. Chin, and L. Luo, “Design of an AI Framework for MOUTbots”, in *Proceedings of the 14th Conference on Behavior Representation In Modeling and Simulation* (BRIMS 2005), Universal City, CA, May, 2005

[8] Paul Gustavson and Tram Chase, “Using XML and BOMS to Rapidly Compose Simulations and Simulation Environments”, in *Proceedings of the 2004 Winter Simulation Conference*, Washington, D.C., December 5-8**,** 2004

[9] Paul Gustavson, Phil Zimmerman, and Chris Turrell, 2003, “Capturing Intent-of- Use Meta Data for the Conceptual Model - A Key to Component Reuse”, in *Proceedings of the 2003 Fall Simulation Interoperability Workshop*, Orlando, FL, September, 2003

[10] R. Adobbati and et al., “GameBots: A 3D Virtual World Test Bed for Multiagent Research”, in *Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS*, Montreal, Canada, May 28 – June 1, 2001

[11] Lewis M. and Jacobson J., “Game Engines in Scientific Research”, *Communications of the ACM*, 45(1):27–31, January 2002

[12] IEEE 1516.3, IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP), March 2003

[13] SISO, “Base Object Model (BOM) Template Specification”, available at http://boms.info

[14] SISO, “Guide for Base Object Model (BOM) Use and Implementation”, available at http://boms.info

[15] Dennis McGrath, Mark Ryan and Doug Hill, “Simulation Interoperability with a Commercial Game Engine”, *European Simulation Interoperability Workshop*, Toulouse, France, June 27-30, 2005

[16] Roger D. Smith, “Essential Techniques for Military Modeling and Simulation", in *Proceedings of the 1998 Winter Simulation Conference*, Washington DC, USA , December 13-16, 1998,