

# MRL 3DDevelopment Team Description Paper

Farid Haji Zeinalabedin, Hamed Rafi, Ebrahim Yazdi

Mechatronics Research Laboratory, Qazvin Azad University, Qazvin, Iran  
{farid.zeinalabedin, hamed.rafi, abraham.yazdi}@gmail.com

**Abstract.** This paper describes MRL3D Development team's modified version of 3d soccer simulation engine, jssserver3d. Based on Java's cross-platform technology, the engine simulates a multi agent football match, with capability of adapting new humanoid robot designs. It supports over-network IP-based connections for agents and monitors, regardless of the programming language and platform architecture. A coach agent can also be connected to the simulation engine for each team. And the system is setup for a referee agent to connect and judge the game. The simulator uses selectable PhysX and ODE as the physics engines, but new engines could be easily added. Besides an external 3d monitor is also implemented using the Irrlicht engine. A user friendly GUI is provided with the engine to configure the simulation circumstances, with respect to Java technology.

## 1 Introduction

Based on last year's effort in creating a soccer simulator, we continued to work on the project and modified some parts and added lots of new features. The need for secure and precise tracking of agent processes and synchronizing actions, led to modification and improvement of JSynchronizer. Extendibility, reusability and adaptation which are presented by software design enabled us to use different physics engines and connecting different agents e.g. player and coach. Built over Model-View-Controller (MVC) design pattern, the server is an event based environment that takes advantage of user interactions for runtime and offline reconfigurations.

Considering the goal of 2050, changing to humanoid simulation is inevitable. Yet switching from a sphere agent simulator directly into a fully structured humanoid robot one requires enough effort. For a simulator, fair play not only involves process management etc. also requires a talented referee that is able to detect humanoid types of faults. Besides, physical characteristics of the robot should be well designed. So because of adaptation, new robot models and new physic engines can be added or implemented as easy as possible.

The project consists of three main subprojects, a process synchronizer that brings up a multi-process environment and keeps track of agent activities, and a soccer simulation engine and controller that uses the user selected physic engine. The last module is a cross platform C++ implemented monitor using Irrlicht engine which connects to the server via UDP protocol on multicast port. Here comes a summary of what we did last year. Then the improvements and modifications for the china 2008 competitions are introduced.

## **1.1 Java Technology**

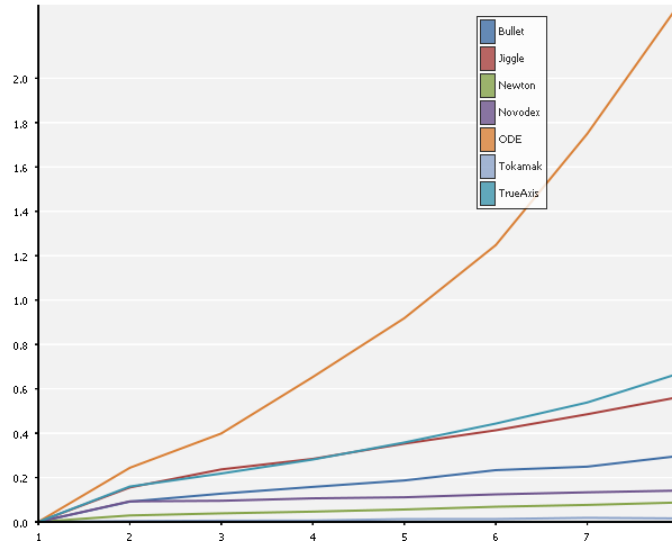
Taking advantage of Java's cross platform technology makes Jssserver3d capable of being run in all POSIX, Win32 and OSX platforms, with obviously no different results. In fact the simulator can be run over a network consisting of different platforms. A process' CPU usage is monitored by means of Java native interface (JNI). All required libraries are written in C/C++ language regarding the host operating system's related functions. Unlike an ordinary java program, using JNI, jssserver3d runs almost as fast as a C/C++ written application, because it removes all java byte code overheads except for function jumps. All client/server implementations are done through synchronized Java threads that bring the most possible concurrency. Java brings the possibility to combine two or more Java Technology-based applications to create highly customized services. This is the main feature that integrates the simulation engine and soccer controller in a client / server architecture.

## **1.2 Client/Server Technology**

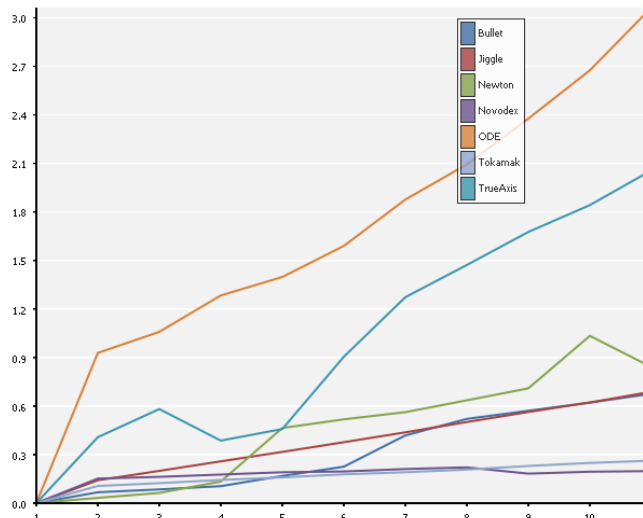
Jssserver3d is designed in client/server architecture that runs both locally and over network. The simulation engine is divided into two parts, a server and some clients. The server is attached to the soccer controller and remains in the server computer, while the client part is run on the client computer over the network. It is always the clients' responsibility to run the agents and monitor their activity. By using IP-based protocol, regardless of the programming language or platform architecture of the agent application, the actions from agents are sent to the server/soccer controller and the senses are sent to the agents from the world model inside the soccer controller. Several monitors can be connected to the soccer controller simultaneously through IP. There is always the possibility for new monitors to be connected in the middle of a running game.

## **2 New Physic Engine**

Besides ODE we have worked on creating a java wrapper for Ageia PhysX engine and importing it into Jssserver3d. The result of the efforts is a Linux and Windows compatible version of JPhysX, the ready to use wrapper for PhysX. Because of great performance of the engine we have tested up to 22 agents, without any crash or miscalculation in contrast to the crashes of ODE and some lack of precision. However, although we still support ODE, but using JPhysX is highly recommended. The engine steps accurately for 0.02 seconds each cycle. Following are two figures demonstrating benchmarks between seven physic engines. As can be seen, in both cases ODE has the most computation time and error. On the other hand Novodex PhysX is one of the two best engine, accompanying Tokamak.



**Fig. 1.** The computation time required to update the physics engine for the corresponding number of stack objects



**Fig. 2.** The constraint error measured from the accumulated difference in the distance between two links minus relative to the initial case

The only fact that should be taken care of is that because of different accuracy in the engines, agent may need to use different constants, when selecting different engines.

### 3 Architecture Improvements

As well as adaptation, reusability is another important factor that guarantees ease of development in the future. So we tried to implement independent modules that could be instantiated anywhere within the code. Also saving the runtime configuration is done through serialized classes in java that brings an end to the file modification before each run.

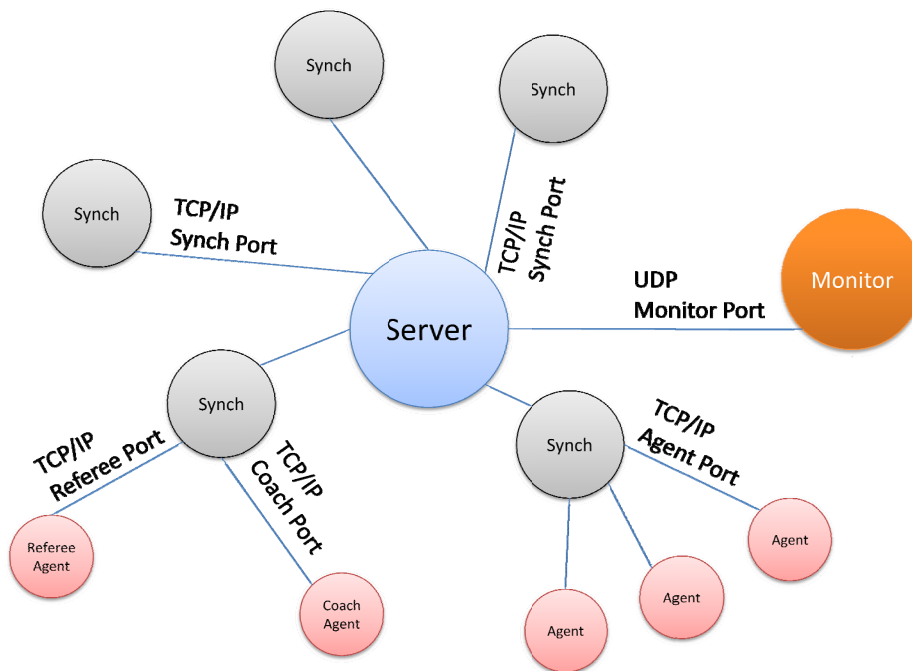


Fig. 3. Improved Client/Server Architecture

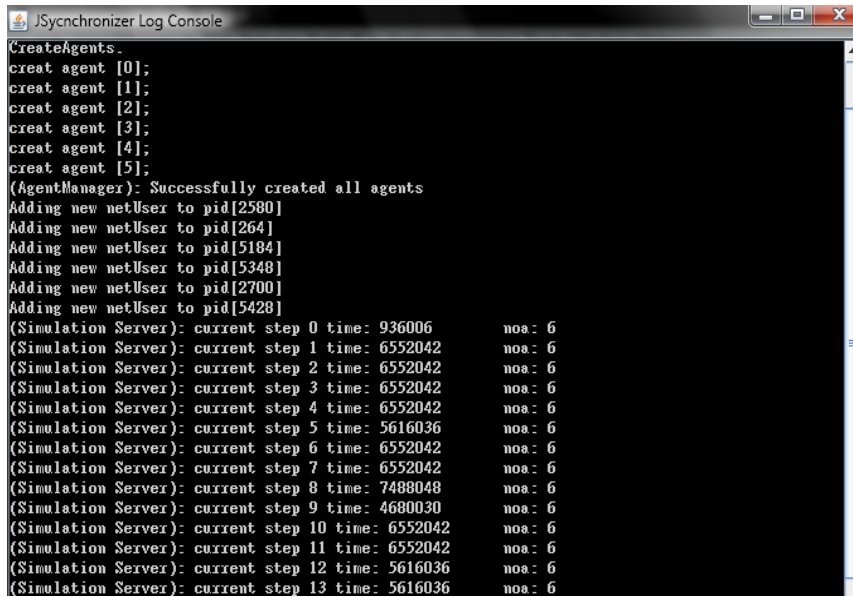
And in case of security because each network connection is assigned to the PID of the connecting process which may be one of players, coach or referee, there is no chance of cheating. Meaning that no player agent can connect to e.g. coach port, and if one does so, the process will be kept sleep.

### 4 JSynchronizer

Process management and tracking agents' CPU usage is an OS dependent job which should be handled with special care. Synchronizing agents in a sense-think-act manner is as vital and important for a precise simulation as a good physic engine.

JSynchronizer wraps Operating System dependent functions for getting a process's PID and CPU usage in user or kernel times into java programming language and gives Jssserver3d the opportunity to simulate a fair soccer match. In near future, when

agents have implemented time consuming skills for controlling humanoid robots, the need for high performance computing is inevitable. But using the architecture described above, Jssserver3d can be easily distributed among several computers to achieve the best performance. Each computer hosts some agents of different types e.g. coach, referee and player, using JSynchronizer. After each step the one and only one instance of JSynchronizer besides Jssserver3d, collects all the information from other host computers and passes the data to the simulation engine. Then the world model is refreshed through the selected physic engine and finally the new senses are sent back to the agents through the JSynchronizer network.



```

JSynchronizer Log Console
CreateAgents .
creat agent [0];
creat agent [1];
creat agent [2];
creat agent [3];
creat agent [4];
creat agent [5];
(AgentManager): Successfully created all agents
Adding new netUser to pid[2580]
Adding new netUser to pid[264]
Adding new netUser to pid[5184]
Adding new netUser to pid[5348]
Adding new netUser to pid[2700]
Adding new netUser to pid[5428]
(Simulation Server): current step 0 time: 936006      noa: 6
(Simulation Server): current step 1 time: 6552042   noa: 6
(Simulation Server): current step 2 time: 6552042   noa: 6
(Simulation Server): current step 3 time: 6552042   noa: 6
(Simulation Server): current step 4 time: 6552042   noa: 6
(Simulation Server): current step 5 time: 5616036   noa: 6
(Simulation Server): current step 6 time: 6552042   noa: 6
(Simulation Server): current step 7 time: 6552042   noa: 6
(Simulation Server): current step 8 time: 7488048   noa: 6
(Simulation Server): current step 9 time: 4690030   noa: 6
(Simulation Server): current step 10 time: 6552042  noa: 6
(Simulation Server): current step 11 time: 6552042  noa: 6
(Simulation Server): current step 12 time: 5616036  noa: 6
(Simulation Server): current step 13 time: 5616036  noa: 6

```

Fig. 4. JSynchronizer Log Console

In each cycle if any of the agents cannot finish its job during the predefined legal CPU usage, that agent is kept out of the active agent list and will not be given sense in the next step. Then using following formula the exact number of cycles the agent should be kept slept as the punishment is calculated.

$$PUNISHMENT_{CYCLES} = TOTAL\_CPU_{USAGE} \% LEGAL\_CPU_{USAGE}$$

After the  $PUNISHMENT_{CYCLES}$  the punished agent is again put in the active list and will be given the sensation according to the current game state. Also if an agent does not respond in less than the timeout, that agent would automatically be killed.

## 5 Cssmonitor3D

Visualization is another important part that is taken care of in Jssserver3d. Thanks to high performance and good quality of Irrlicht engine, we have implemented Cssmonitor3D as an external monitor. Using UDP broadcast technique; monitor information is sent over network and can be used by any client without platform and architecture considerations. Although written in C++, Cssmonitor3D is available in both Linux and Windows operating systems.

Considering performance and fps, the UDP client part of the application is threaded and *listener* design pattern is used to update current data model. This leads to current fps speed of about 100 for more than 10 agents on the scene. Cssmonitor3D is completely independent from the type of simulation and the objects used, meaning that if the robot model is changed, the monitor will automatically draw the new model.

The protocol is easy to use and contains type of object, and the transformation data and the unique name of that object, and makes it very easy for programmers to change the color or texture of any specific object. Creating and importing 3d models such as stadiums, has never been easier thanks to IrrEdit, the free and open-source scene graph editor. Cssmonitor3D is also easily compatible with rcssserver3d using S expression and RSG parser.

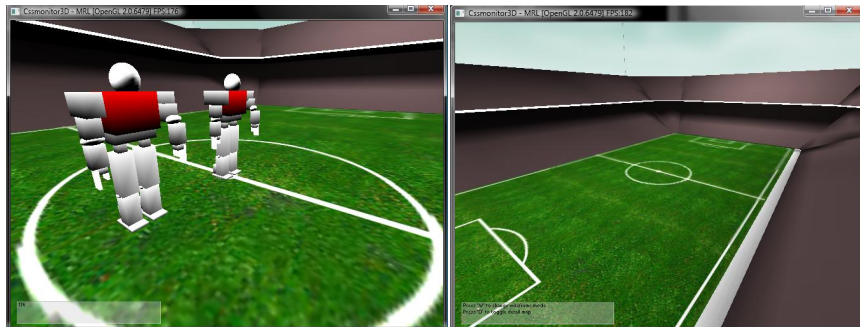


Fig. 5. Two screenshots of cssmonitor3d

Showing debug information is another advanced feature for cssmonitor3d. Running the monitor in debug mode, will help the user to visualize some numerical information such as agent speed vectors, positions and rotations in Cartesian or polar coordinates.

## 6 Special Features

Jssserver3d incorporates design rules and patterns to deliver the best possible help for the agent developers in case of debugging, training, new models for robots etc. The following is a list and description for some of the features in jssserver3d.

## **6.1 Rcssserver3d Compatibility**

Sense and action strings sent to or received from agents are just compatible to the rcssserver3d. So, current agent developers do not need to modify their source code for jssserver3d adaptation. The information available via the sense messages contains the opponent positions in polar coordinates relative to the torso of the agent looking at the scene, following current game time and state. The action commands contain the new angle for each joint motor. And most importantly, the robot model and specification is just the same as rcssserver3d.

## **6.2 Trainer**

Trainer has always been a very good help in 3d soccer simulation league. To obey this rule, jssserver3d also runs in trainer mode in which the agents can connect to the predefined port for training environment. This mode, gives all agents the opportunity to get the whole noise free worldmodel or any part of it in any step they want so. Such information is very useful for learning and prediction algorithms. It helps agent developers to determine whether their prediction algorithms work fine in comparison to the real world events or not.

## **6.3 Robot Model Importer**

Importing new robot models is possible using XML file, defining the structure and joint types of the robot. We have a robot model designer tool in progress which will be available for the 2008 competition. The tool gives the user an advanced graphical user interface through which structured meshes such as box, sphere, and capsule can be added to the model. Depending on the target physic engine which may be one of ODE or PhysX, the supported types of motors and joints are available. The design will be saved as an XML file and can be imported in jssserver3d.

## **References**

1. IEEE, Guide to the Software Engineering Body of Knowledge, 2004
2. Patrick Riley, SPADES System for Parallel Agent Discrete Event Simulation User's Guide and Reference Manual, 2003
3. [www.java.com](http://www.java.com)
4. [irrlight.sourceforge.net](http://irrlight.sourceforge.net)