MRL 3D Development Team Description Paper

Hamed rafi, Farid Haji Zeinalabedin, Shaghayegh Tabatabaee, Ebrahim Yazdi, Ashkan Ferdowsi

Mechatronics Research Laboratory, Qazvin Azad University, Qazvin, Iran
{hamed.rafi, farid.zeinalabedin, sh.tabatabaee, abraham.yazdi}@gmail.com , a.ferdowsi@mrl.ir

Abstract. This paper describes MRL3D Development team's brand new 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 local and over-network IP-based connections for agents and monitors, regardless of the programming language and platform architecture. It is run in three different modes, simulator, trainer, and a set of debugging tools. The simulator uses Xith3D as the graphics and ODE as the physics engine. For each team a coach agent can be connected to the simulation engine, which brings the design to new level of foreseeing. A user friendly GUI is provided with the engine to configure the simulation circumstances, with respect to Java technology.

1 Introduction

Creating a fair soccer simulation environment, involves secure and precise tracking of agent processes and synchronizing actions. On the other hand, software design presents extendibility, reusability and adaptation. Jssserver3d is the place were these two important facts, meet. Considering software design patterns, Model-View-Controller (MVC) introduces 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. It is obvious that none of these goals are achieved in one year development tournament, but creating a simulator requires very good foreseeing of what is going to happen. So that new features and components can easily be added to the system.

As a conclusion from above statements, we decided to found a project that is capable of being developed throughout the years, supporting all aspects mentioned above. We opened the project in a completely extendible way that newer programmers and designers can simply develop the simulator towards the goal of 2050.

The project consists of two main subprojects, a simulation engine that brings up a multi-agent environment and keeps track of agent activities, and a soccer controller that handles football rule checking in conjunction with a physics engine.Jssserver3d introduces a humanoid simulator capable of importing robot models from external editors using ODE's xml parser. The robot can use predefined or user defined textures.

2 JAVA Technology

Jssserver3d takes advantage of Java's cross platform technology which makes it 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 as fast as a C/C++ written application, because it removes all java byte code overheads. 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.

3 Clients / Server Architecture

The main design idea in development of jssserver3d is to reduce the coupling and increase the cohesion characteristics of the application. Using MVC pattern, it responds to the client requests or user actions as soon as possible and sends all listening view components e.g. monitors, an event indicating that the state is changed.

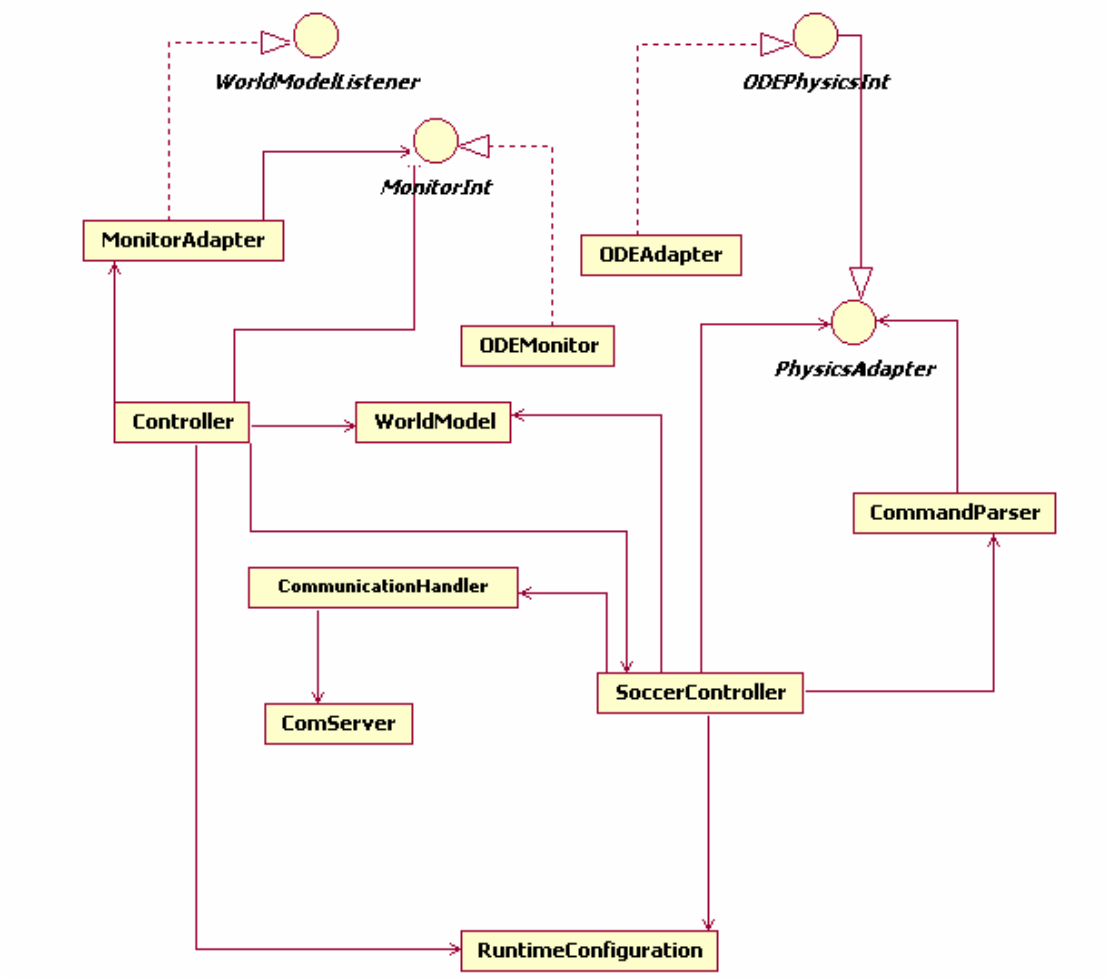

Fig. 1. Software Architecture

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.

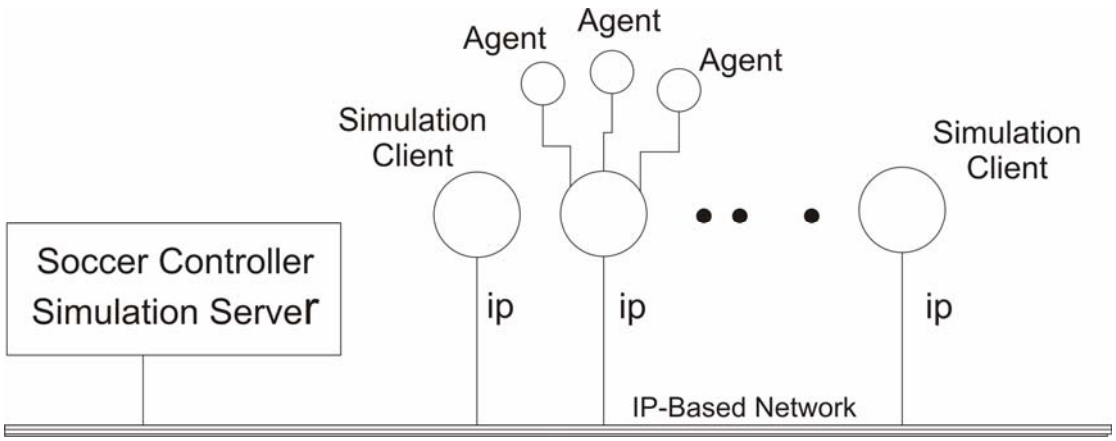

Fig. 2. Client Server architecture

4 Graphics and Physics Engine

Jssserver3d incorporates Open Dynamic Engine (ODE) as the physics and Xith3D as the graphics engine. The two mentioned engines are in close relation to each other so that any defined object in ODE is dynamically and automatically added to the graphics scene graph. This results in real time refreshment of the monitor. Using JNI technology Xith3D is an excellent wrapper for OpenGL that brings both quality and performance. And so is ODE based on JNI technology and encapsulates the C++ library within java environment.

5 Running jssserver3d

Jssserver3d is a reconfigurable application that can be set to run in three different modes. These configurations can be saved for future use and reloaded into the application. By configuring the running mode the application is divided into three parts:

- Simulator
- Trainer
- A set of debugging tools

5.1 Simulator

In normal simulation mode, the simulator starts into initialization mode, runs the agents and waits for them to send their initial positions. From now on the simulation gets into running state, in which the main loop executes. As shown in figure 3, at the beginning of each step, soccer controller passes out the senses to the agents, waits for them to send their action commands. Then the commands are passed to the physics engine through the parser attached to it. The results of the physics update world model. Then soccer rule checker examines the data to see if the game state has changed. At this stage the whole process is started over.

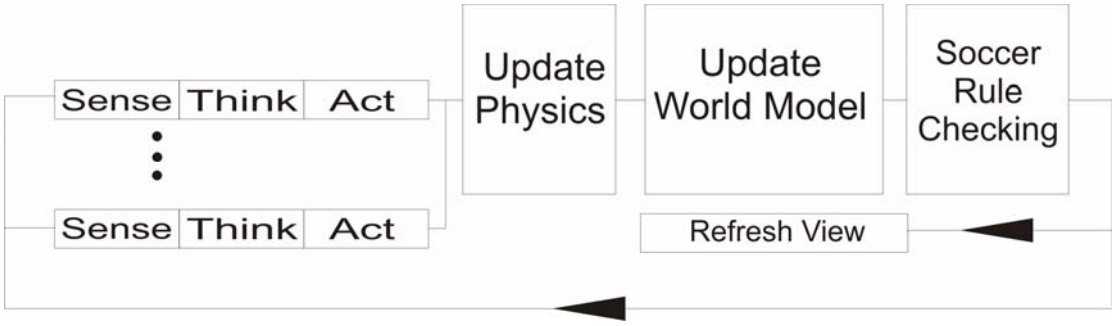

Fig. 3. Simulator Engine main loop

Some noise relative to the distance of the agent form the object sensed is applied to the data passed out to simulate a more realistic situation.

5.2 Trainer

In trainer mode it is possible for agents to connect to the server and practice new situations. Commands can be sent to the server using scripts and the environment is updated in the next simulation step. A scenario can be written from these commands. The trainer will rerun the scenario once it has been finished and the data is saved for future use.

5.3 Debugger

In debugging mode jssserver3d gives you opportunity to debug your agents. Suppose a situation in which the agent encounters a problem and suddenly crashes. Because the simulation environment is none deterministic, the same situation may never happen again. So the developer might have lost his chance to trace the problem. But this is not the end, jssserver3d stores the data sent to agents and is able to put the agent in the same circumstances that the problem occurred. By running the same binary with these specific senses the developer can simply trace the problem.

References

1. IEEE,.Guide.to.the.Software.Engineering.Body.of.Knowlewdge.2004.Version.SWEBOK.(2004).pdf
2. IEEE SOFTWARE March-April 2004 - Practical Requirements Engineering Solutions.pdf
3. Patrick Riley, SPADES System for Parallel Agent Discrete Event Simulation User's Guide and Reference Manual, 2003
4. www.java.com