SimSpark – Concepts and Application in the RoboCup 3D Soccer Simulation League

Joschka Boedecker[1] and Minoru Asada[1,2]

[1]Dept. of Adaptive Machine Systems,
[2]HANDAI Frontier Research Center,
Graduate School of Engineering, Osaka University, Osaka, Japan
{joschka.boedecker,asada}@ams.eng.osaka-u.ac.jp

Abstract. Simulators have a long tradition within RoboCup. In the Soccer Simulation League, one of the earliest leagues within RoboCup, traditionally single-purpose robot simulators with high levels of abstraction have been used. These systems proved valuable as tools for multiagent research, but were difficult to extend. Furthermore, there were concerns that research results would not be easily transferable to the real world due to the extensive simplifications. In order to alleviate these problems, a new, more generic, and more realistic robot simulation system was proposed in the year 2003 and first used in the league's competitions in 2004. This paper presents the architecture and concepts of this system, and its application in the RoboCup 3D Soccer Simulation League. Moreover, it presents ongoing and future development plans.

1 Introduction

Ever since the early days of RoboCup, simulators have played in important part in it, with the RoboCup Soccer Simulation League being one of the earliest competitions in its history [1]. The *Soccer server* [2] has been used as the simulation platform in this league in order to address research questions in high-level decision making and team coordination amongst others. The level of abstraction of the Soccer server has been quite high: the games are simulated in a 2D environment with simplified physics rules and agent representations. Furthermore, the Soccer server has been built as a special purpose simulator, solely capable of soccer simulations and difficult to extend. This is why a new simulator was proposed in 2003 [3] which was intended to be more realistic and usable for many kinds of physical multiagent simulations.

In this paper we describe *SimSpark*, a multi-robot simulator based on the generic components of the Spark [4] physical multiagent simulation system that was built in response to the proposal mention above. Simulators built with Spark components have been used in the RoboCup Soccer Simulation League since 2004. The simulations have changed significantly over the years, going from rather abstract agent representations to more and more realistic humanoid robot games; but due to the flexibility of the Spark system, these transitions were achieved with little changes to the simulator's core architecture.

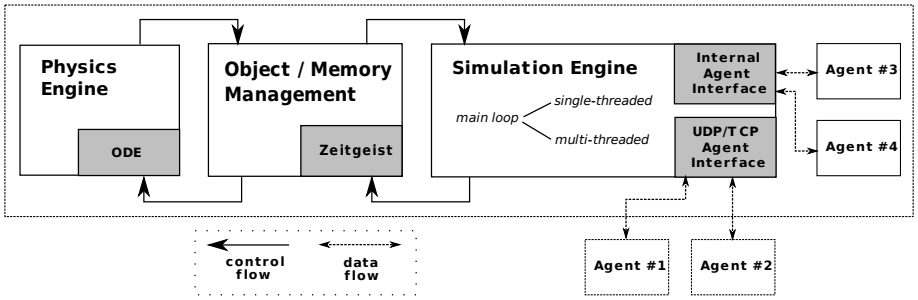

Fig. 1. Control flow and data flow between the main components of the simulator. (adapted from [4]). Note that the graphics component is not shown since it is an optional part of the system.

We will give an overview of the main parts of the SimSpark system in section 2, and take a closer look at robot model creation and representation in section 3. After that, we will give an account of the application of Spark based simulators within the RoboCup 3D Soccer Simulation League (from hereon abbreviated as 3DSSL, (section 4) and briefly mention related work on multi-robot simulation software. Finally, we introduce some ongoing work to extend SimSpark, and outline future development plans (section 5).

2 System Overview

The main components of the SimSpark simulator are the simulation engine, the object and memory management system, and the physics engine. The creation and destruction of class objects is handled by an application framework called *Zeitgeist*. It provides an extensive plugin system and an interface for scripting languages. Rigid-body physics calculations are handled by an external library which is integrated into the overall architecture of the system. The simulation engine binds all the other components together, advances simulation time in a main run loop, and calls various plugins to handle things like network communication with external agent or visualization programs. The overall architecture of the system including control and data flows is shown in figure 1. Below, the features of the system are described in more detail.

2.1 Zeitgeist Concepts

The flexibility of Zeitgeist is mainly due to two key concepts. The first key concept is the implementation of a variant of the reflective factory pattern [5]. It allows for factory based instantiation of objects at runtime while storing information about the creating factory in the object. This can be used to access class names and supported interfaces once the class objects are created. It the basis for

the scripting language interface of the simulator which could support different scripting languages, but is currently only implemented for Ruby.

The second key concept is the organization of object factories and created objects in a virtual file system. Every node in this tree-like structure stores its own path name and a reference to its parent and child nodes. This way, services and objects can easily be located during runtime simply by providing a path expression. The object factories are stored at well-defined locations in this virtual file system which makes it possible to instantiate objects of classes that are unknown at compile time, e.g., through the scripting language interface. This also facilitates to add new sensors, actuators, robot model formats, and other things as plugins to the simulator. More details about the Zeitgeist application framework can be found in [4].

2.2 Physical simulation

The SimSpark simulator in its current implementation uses the Open Dynamics Engine (ODE) [6] for physically realistic dynamics simulations. ODE offers fast rigid body simulations, collision detection, and supports the use of articulated body structures. Furthermore, it has integrated joint motors that help to stabilize the simulation. The most important ODE concepts are available im SimSpark, encapsulated in classes to fit into the overall object-oriented design of the simulator.

2.3 Simulation Engine

The core of the simulation binds together processes like timing, event management, and communication with external processes. It contains the main runloop of the simulator. From the beginning the design allowed for a customization of the runloop with replaceable components. Two runloop modes were built in: a simple, straightforward loop that would realize actions sent by the agents as soon as they arrive, and a more elaborate one using the SPADES middleware [7], designed for maximum reproducability of distributed simulations.

It turned out that the SPADES based runloop was too complex and system setup not mature enough to be used in competition situations like in those in the 3DSSL. The simple runloop model was, however, too indeterministic for accurate control, exhibiting large variations in execution time based on the load of the server. Therefore, a new timer was implemented that runs the control loop at 50 Hz, the rendering at 25 Hz, and allows for different delays for different sensor and actuator types. Furthermore, it can run in single-threaded or multi-threaded modes. In the multi-threaded mode, stepping the physical simulation and processing agent communication for the next simulation cycle are executed in parallel.

In order to customize this generic simulation core for specific simulations, the game or application logic can be implemented as plugins. In each simulation step, the plugins are triggered with certain event types to which they can choose to respond.

2.4 Network Support and Integrated Agents

Agents as well as external monitors for visualization can be connected to the simulator via network. TCP and UDP connections are supported, although only TCP is used at the moment. Furthermore, there is an interface for a so-called *trainer* application. A trainer can connect to the server and make use of a special protocol in order to move objects in the simulation, set game states, etc. This is especially useful for debugging and machine learning applications.

Having the agents as external processes connected to the server via network is the default situation for the RoboCup competitions to ensure fairness. This setup, however, causes some network overhead and makes it difficult to debug an agent application. To address these problems, a possibility was added recently to run agent behaviors directly as plugins in the server, allowing to stop the server and agent behavior together while debugging, and avoiding any network overhead. It is possible to connect internal and external agents at the same time.

2.5 3D Visualization

SimSpark includes a 3D visualization application based on OpenGL. It supports internal rendering of the simulated scenes, as well as streaming of scene description data over the network for visualization in external monitor applications. In addition, it supports recording of log files for later view-independent playback and analysis.

2.6 Cross-Platform Support and GUI

SimSpark implementations exist for Linux, Windows, and MacOS X operating systems. RPM packages are available for Fedora and OpenSUSE Linux, and Debian packages for Ubuntu Linux are provided. A an installer is available for Windows platforms, and stand-alone applications for Mac OS X have recently been released for OS versions 10.4 and 10.5.

The current release of SimSpark also contains an experimental graphical user interface (GUI) which is expected to replace the simple internal monitor of SimSpark in future versions. Details on this will be presented in a future publication.

3 Robot Model Creation and Representation

The creation of new robot models for a given simulation is generally a time-consuming process involving lots of fine tuning of parameters. If a robot model is constructed purely in C++ code, recompilation will be necessary many times. This can be avoided by using the Ruby interface of SimSpark. Since Ruby is an interpreted language, recompilation is not necessary after the model has been changed. However, with this approach, it is still cumbersome to create hierarchies of objects, having to specify the path where each new node should be linked into

the hierarchy each time. This is why SimSpark offers a scene description language called *RubySceneGraph*. The language is based on Lisp-like S-Expressions, and its hierarchical parenthesis structure is mapped directly into an object hierarchy of the scene tree. A powerful feature is that Ruby and RSG descriptions can be mixed, enabling procedural and descriptive scene constructions at the same time. For a detailed specification, we refer the reader to [4].

In order to use highly detailed 3D meshes for visualization of robot parts, it is possible to load triangle meshes in the Wavefront Object (OBJ) format from within an RSG description. The OBJ format is rather simple, describing vertices, faces, and normals of a 3D mesh, as well as providing material definitions. Exporters for this format exist for all major 3D modeling packages.In this case, the different parts of a robot are exported individually (in order to facilitate assignment of collision primitives), and RSG acts as a "glue" to assemble all the parts together. An advantage of this approach is also that the description of the visual and physical parameters are kept separate so that changes in one do not affect the other. Support for other 3D file formats can easily implemented as plugins to the simulator (e.g. for the RoSiML [8] for which an experimental importer plugin is available).

The next section takes a look at the different simulators that have been built using Spark over the years in the 3DSSL.

4 Application In The RoboCup 3D Soccer Simulation League

The first simulator for the 3DSSL based on the components of Spark was introduced at RoboCup 2004 in Lisbon, Portugal. It was called *rcssserver3D* and it did not make use of all the available Spark features (e.g. RSG or the rendering library *Kerosin*). At that time, the soccer simulation featured a soccer field in standard FIFA size, and allowed a game of 11 vs. 11 agents. The agents were modeled as spheres with a mass of 75 kg equipped with an omni-directional drive effector for movement on the field, a kick effector for kicking the ball, and an omni-direction vision perceptor (an abstract camera-like device). Later, in 2006, agent communication was introduced, and the field of view of the vision perceptor restricted. A screen shot of this simulation is shown in figure 2 (Left). For more information on the perceptors and effectors in that simulation, please refer to [9].

In late 2005, the development of the first articulated robot models for SimSpark had started. The first humanoid robot for SimSpark was modeled after the commercial HOAP-2 robot from Fujitsu (see Fig. 2 (Right) for a screen shot).

This robot served as a basis for the "Soccerbot" model used in the 3D competition (2 vs. 2 games) at RoboCup 2007 in Atlanta, USA, which marked the transition point from the more abstract sphere shaped agents to more realistic humanoid types (see Fig. 3 (Left)). The robot has 20 DOF, an abstract omni-vision sensor (similar to the one used in the spheres simulation), joint motors

Fig. 2. (Left) The first simulation: 11 vs 11 spheres with omni-drive and kick effector. (Right) The first humanoid model based on the HOAP-2 robot from Fujitsu.

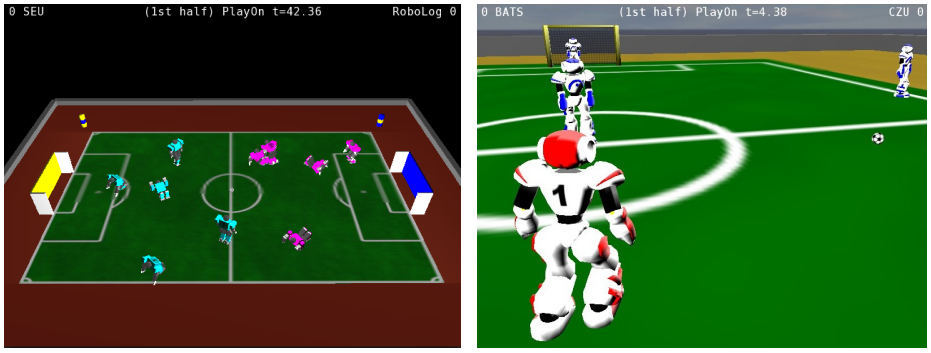

Fig. 3. (Left) The Soccerbot humanoid soccer simulation as used in Atlanta 2007, showing a 5 vs 5 game. (Right) The Nao simulation as used in Suzhou 2008.

and angle sensors, a gyro sensor, and force sensors in the feet (cf. [9] for details on the robot).

At RoboCup 2008 in Suzhou, China, a new robot model based on the Aldebaran Nao robot [10] was introduced (see Fig 3 (Right)). The robot model has 22 DOF, its height is about 57cm, and its mass about 4.5kg. In the current simulation, it is equipped with an omni-directional vision sensor (the same as in the Soccerbot), motors and angle sensors for each joint, a gyro sensor, a device for agent communication, and a force sensor in each foot. In the 2008 competition, 3 vs. 3 games were played on a 12m by 8m field. Details on the robot model can be found again in [9].

Even though the more recent humanoid robot soccer simulations are quite different from the early sphere based games, it turned out that a large number of plugins for the simulation (e.g. the rules plugins for automatic rule enforcement) could be reused with only minor changes. New effector and perceptor models could easily be added as new plugins, and new robot descriptions were supplied in the RSG language described above. No major update to the core simulation

engine was necessary for these changes. This highlights the flexibility of the overall architecture of the system.

Within RoboCup, a large number of other robot simulators is currently used. The most widely used ones include the commercial Webots simulator [11], Microsoft Robotics Studio [12], the open-source simulators Gazebo [13] and USARSim [14], as well as SimRobot [15]. Due to space limitations, we cannot discuss the advantages and disadvantages of each of the presented systems in comparison with SimSpark here. There are many similarities among all of the systems, like full rigid-body dynamics, advanced 3D graphics, robot model formats, and network support (except for SimRobot). Some important distinctive points about SimSpark are that it is completely open-source, has multi-platform support (with packages/installers provided), it is fully scriptable via Ruby, and it has built-in support for automatically enforced rules (like e.g. soccer rules) via plugin modules.

5 Ongoing Work and Future Plans

SimSpark has undergone continuous development since its initial version, and while it has been used successfully in several RoboCup competitions, the ever-changing requirements of the 3DSSL drive further development every year.

Currently, work on a more realistic camera perceptor has begun which will deliver images rendered via OpenGL hardware accelerated offscreen buffers. Besides that, a new effector has reached its first usable version which implements the Harmonic Motion Description Protocol (HMDP) as presented in [16] (both as part of the *3D2real* project [17]).

In the near future, several significant changes are planned to parts of the SimSpark architecture; in order to take advantage of the rapid improvement of several freely available physics engines, it is planned to integrate support for the Physics Abstraction Layer [18] library, providing support for ODE, PhysX [19], and others; the internal rendering library *Kerosin* is planned to be replaced by support for state-of-the-art open-source solutions like OGRE 3D [20] and the GUI will be extended; the network protocols will be optimized for faster information transmission.

6 Acknowledgements

We wish to thank the original authors of SimSpark and rcssserver3D, and their components: M. Kögler, M. Rollmann, and O. Obst. Furthermore, thanks go to the numerous volunteers in the Maintenance Committee of the RoboCup Soccer Simulation League, as well as many 3DSSL community members for their help. Thanks also go to Tim Laue for providing robot models in the RoSiML format. Finally, one of the authors (J. B.) acknowledges support by a JSPS Fellowship for Young Researchers.

References

1. Kitano, H., Tambe, M., Stone, P., Veloso, M., Coradeschi, S., Osaka, E., Matsubara, H., Noda, I., Asada, M.: The Robocup synthetic agent challenge 97. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence, Morgan Kaufmann (1997) 24–19
2. Noda, I., Matsubara, H., Hiraki, K., Frank, I.: Soccer server: A tool for research on multiagent systems. Applied Artificial Intelligence **12** (1998) 233–250
3. Kögler, M., Obst, O.: Simulation League: The next generation. In Polani, D., Bonarini, A., Browning, B., Yoshida, K., eds.: RoboCup 2003: Robot Soccer World Cup VII. Lecture Notes in Artificial Intelligence, Springer (2004)
4. Obst, O., Rollmann, M.: SPARK – A Generic Simulator for Physical Multiagent Simulations. Computer Systems Science and Engineering **20**(5) (2005)
5. Hargrove, C.: Reflective Factory. `http://www.gamedev.net/reference/articles/article1415.asp` (2000)
6. Smith, R.: Open Dynamics Engine webpage. `http://www.ode.org/` (2007)
7. Riley, P.: SPADES: A System for Parallel-Agent, Discrete-Event Simulation. AI Magazine **24**(2) (2003) 41–42
8. Ghazi-Zahedi, K., Röfer, T.L.T., Schöll, P., Spiess, K., Twickel, A., Wischmann, S.: RoSiML – Robot Simulation Markup Language. `http://www.informatik.uni-bremen.de/spprobocup/RoSiML.html`
9. Boedecker, J., Dorer, K., Rollmann, M., Xu, Y., Xue, F.: SimSpark User's Manual. (June 2008)
10. Aldebaran Robotics: Aldebaran Robotics webpage. `http://www.aldebaran-robotics.com/eng/` (2007)
11. Michel, O.: Cyberbotics Ltd. - Webots(tm): Professional Mobile Robot Simulation. Journal of Advanced Robotic Systems **1**(1) (2004) 39–42
12. Microsoft Corporation: Microsoft Robotics Studio webpage. `http://msdn.microsoft.com/robotics/` (2008)
13. Koenig, N., Howard, A.: Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In: Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems. (2004) 2149–2154
14. Carpin, S., Lewis, M., Wang, J., Balakirsky, S., Scrapper, C.: USARSim: a robot simulator for research and education. In: Proceedings of the 2007 IEEE Conference on Robotics and Automation. (2007)
15. Laue, T., Spiess, K., Röefer, T.: SimRobot - a general physical robot simulator and its application in RoboCup. In: RoboCup 2005: Robot Soccer World Cup IX. LNAI, Springer (2006)
16. Mayer, N.M., Boedecker, J., Masui, K., Ogino, M., Asada, M.: HMDP: A New Protocol for Motion Pattern Generation Towards Behavior Abstraction. In: RoboCup 2007: Robot Soccer World Cup XI. LNAI, Springer (2008)
17. Mayer, N.M., Boedecker, J., da Silva Guerra, R., Obst, O., Asada, M.: 3D2Real: Simulation League Finals in Real Robots. In: RoboCup 2006: Robot Soccer World Cup X. LNAI, Springer (2007)
18. Boeing, A.: Physics Abstraction Layer webpage. `http://adrianboeing.com/pal/` (2008)
19. NVIDIA Corporation: NVIDIA PhysX webpage. `http://www.nvidia.com/object/nvidia_physx.html` (2008)
20. Torus Knot Software Ltd.: OGRE 3D webpage. `http://www.ogre3d.org/` (2007)