OSA: an Integration Platform for Component-Based Simulation

[Poster Abstract]

Judicael Ribault - Fabrice Peix - Julian Monteiro - Olivier Dalle
Join team MASCOTTE
INRIA, I3S, CNRS, Univ. Nice Sophia, France.
B.P. 93, F-06902 Sophia Antipolis Cedex, FRANCE.
{firstname.lastname}@sophia.inria.fr

ABSTRACT

Many discrete-event simulators are developed concurrently, but with identical or similar purpose. This poster presents the Open Simulation Architecture (OSA), a discrete-event component-based simulation platform whose goal is to favor the reuse and integration of simulation software components and models. To favor reuse, OSA uses a layered approach to combine the modeling, simulation, and related concerns, such as instrumentation or deployment. OSA is both a testbed for experimenting new simulation techniques and a tool for real case studies. The ability of OSA to support challenging studies is illustrated by a Peer-to-peer system case study involving millions of components.

Categories and Subject Descriptors

I.6.7 [**Simulation and Modeling**]: Simulation Support Systems—*environments*; D.2.11 [**Software Engineering**]: Software Architectures—*domain-specific architectures*; D.2.13 [**Software Engineering**]: reusable software

General Terms

Design, Experimentation, Languages, Measurement

Keywords

Separation of Concerns, component-based software, modeling and simulation

1. INTRODUCTION

OSA (Open Simulation Architecture) is an open component-based architecture for discrete-event simulation. OSA is born from the observation that many discrete-event simulators are developed concurrently, but with identical or similar purpose. In order to ease reuse of existing or new simulation software and models, OSA exploit and apply everywhere it is possible the latest relevant Software Engineering (SE) techniques such as Component-Base Software Engineering (CBSE) and Aspect-Oriented Programming (AOP).

Following the model of Eclipse, the OSA software components are considered optional and replaceable independently from each other. The separation of concerns principle is extensively applied to ease the reuse of existing or new simulation software and models. Hence, OSA is both a testbed for new simulation techniques and a tool for real case studies.

Initially, the OSA development was motivated by the study of challenging real applications for which new simulation techniques need to be investigated, such as the study of Peer-to-peer systems involving up to millions of components. In order to solve the resulting issues, multiple third party components and contributions are experimented and assembled in a seamless manner. These components provide support for Graphical User Interface, programming, Deployment of large distributed execution, data sampling, on-the-fly data processing, and so on.

2. THE OSA ARCHITECTURE

OSA is designed to support the simulation end-users in a wide number of their activities such as modeling, experimentation, instrumentation or deployment. It relies on the ObjectWeb's Fractal component model [2] and its Architecture Description Language (ADL). Fractal is a hierarchical component model. Each Fractal component has a versatile structure that separates the functionnal code of the component from its non-functionnal part thanks to a *membrane*. This membrane contains a set of *controllers*, that are in charge of these non-fonctional concerns.

The key elements that have been specifically developed for the OSA architecture are a generic modeling API and a prototype implementation of simulation engine. The modeling API is plugged into the membrane of the Fractal component, in addition to the default elements of a Fractal component. The simulation engine itself is implemented using Fractal components. Thus, it is easy to replace it by another engine, which makes the OSA platform a flexible testbed for testing new simulation techniques.

Since the modeling API itself may be replaced, OSA is also able to emulate other existing discrete-event simulators. Furthermore, given that the specification of the component membrane may be provided on a per component basis in the

SIMUTools 2009, Rome, Italy

Fractal ADL, this means that OSA theoretically allows the interoperability of model components designed for different simulators.

Fractal ADL allows the separation of the modeling effort amongst several experts each having their own area of system expertise. Thanks to Fractal, Fractal ADL and AOP, we separate the non-system expertise effort such as scenarisation, instrumentation or deployment planning. This separation results in a set relatively independent layers, each of which being responsible for a specific concern.

To separate these layers, we use the inheritance and overloading ability of the Fractal ADL, which allows to extend or overload components definitions found in multiple files. We also use aspect-oriented programming to transparenty introduce the required instructions into the code of the models in order to build plug instrumentation probes on the components or to build advanced scenarios.

3. A CHALLENGING APPLICATION

In the context of the SPREADS project (Safe P2P Reliable E Architecture for Data Storage) we study issues related to storage and data backup in peer-to-peer environments. We use simulations to explore the system parameters and find the best configuration trade-off (bandwith usage vs. durability of storage) in realistic, large scale scenarios. This results in a large amount of events to schedule and huge volumes of data to manage during the simulation. Since the OSA architecture is natively distributed (the simulator engine is plugged directly withing the membrane of each component) we naturally chose a massively parallel approach, targetting execution on Grid Computing facilities.

Hereafter, we describes the layered architecture.

- The **model** layer uses Fractal components to implement a hierarchical model of the P2P system. The control part of the system is abstracted by means of a unique, centralized server component. The peers components each contain a shared network component (a unique feature of the Fractal component model) that allows the peers and server to communicate with each other.
- The **scenario** layer is based on the previous model and uses the inheritance concept of FractaADL. The scenario also embeds its own set of components, that can be linked in aseamless manner with model components to stimulate the system (eg. to model peer failures).
- The **simulation** layer is an API located in the membrane of each Fractal component. This simulation API is connected to a simulation engine which is itself a Fractal component. This structure is also described thanks to the FractalADL.
- The **instrumentation** layer consists in collecting data samples during the simulation run. The data are collected using probe controllers located in the membrane of model components. The exact behavior of each probe is generated using AOP. The instrumentation can also implement aggregation policies for samples collected by several probes or compute statistics on-the-fly during execution. This on-line processing is delegated to a third-party framework named COSMOS[3]. Since COSMOS is also based on Fractal components, its configuration is described using FractalADL.
- The **deployment** layer describes the configuration of a distributed execution of the simulation by associating, thanks to FractalADL, sets of components to virtual execution nodes. Virtual nodes are then mapped to real machines thanks to the FractalRMI directory service. To execute the deployment tasks, we use a third-party tool such as Fractal Deployment Framework (FDF) [5].

4. RELATED WORKS

While some of the underlying ideas found in OSA are unique (eg. the use of AOP and Fractal components for Modeling & Simulation), the idea of using Eclipse itself as an integration platform to provide a rich user interface has already been adopted by others, like DESMO-J[4] or Omnet++[1]. A few other environments, like CD++[8] or Omnet++ provide an ADL to describe their model architecture and experiment settings. In [6], Gianni et al. achieve an interesting layered decomposition of a distributed discrete-event simulation using a Model Driven Architecture approach. Extending the layered approach to non-functionnal concerns was also recently adopted by Himmelspach et al. in the JAMES II environment[7].

5. REFERENCES

[1] Omnet++ community site. Available from `<www.omnetpp.org>`. [Accessed December 20, 2008].

[2] E. Bruneton, T. Coupaye, and J. Stefani. The fractal component model specification. Available from `<fractal.objectweb.org/specification>`, February 2004. Draft version 2.0-3. [Accessed December 20, 2008].

[3] D. Conan, R. Rouvoy, and L. Seinturier. Scalable processing of context information with cosmos. In *Proc. of the 7th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'07)*, volume 4531 of *Lecture Notes in Computer Science*, pages 210–224. Springer, June 2007.

[4] R. Czogalla, N. Knaak, and B. Page. Simulating the Eclipse way. In *Proc. of the 20th European Conference on Modelling and Simulation (ECMS 2006)*, Bonn, May 28-31 2006.

[5] A. Flissi and P. Merle. A generic deployment framework for grid computing and distributed applications. In *Proceedings of the 2nd International OTM Symposium on Grid computing, high-performAnce and Distributed Applications (GADA'06)*, volume 4279 of *Lecture Notes in Computer Science*, pages 1402–1411. Springer, Nov. 2006.

[6] D. Gianni, A. D'Ambrogio, and G. Iazeolla. A layered architecture for the model-driven development of distributed simulators. In *Proc. SIMUTools, International Conference on Simulation Tools and Techniques for Communication, Networks and Systems*, Marseille, France, March 3-7 2008.

[7] J. Himmelspach, R. Ewald, and A. M. Uhrmacher. A flexible and scalable experimentation layer. In *Proc. of the Winter Simulation Conference (WSC08)*, pages 827–835, Miami, FL, Dec. 2008.

[8] G. Wainer. CD++: a toolkit to develop DEVS models. *Softw., Pract. Exper*, 32(13):1261–1306, Nov. 2002.