SIMULATING THE ECLIPSE WAY: A GENERIC EXPERIMENTATION ENVIRONMENT BASED ON THE ECLIPSE PLATFORM

Rainer Czogalla, Nicolas Knaak and Bernd Page
Faculty of Informatics
University of Hamburg
Vogt-Kölln-Straße 30, 22527 Hamburg, Germany
E-mail: {czogalla, knaak, page}@informatik.uni-hamburg.de

KEYWORDS
Discrete Event Simulation, Eclipse Platform, Experimentation Environment

ABSTRACT

Though repeatedly pointed out in the literature, simulation technology still suffers from a lack of software tools that support the planning, execution, and analysis of large-scale experiments. The conceptual separation of models and experiments suggests the development of generic experimentation environments whose functionality can be reused by a large number of modeling and simulation tools. The Eclipse platform provides a contemporary architectural framework for the development of such systems. In this contribution, we present a generic experimentation environment that integrates the functionality of experimentation tools previously developed in our group with prototypical Eclipse plug-ins for interactive experimentation support.

INTRODUCTION

Though repeatedly pointed out in the literature, today's simulation technology still suffers from a lack of software that supports the planning, execution and analysis of large-scale experiments (see e.g. Kelton 2001). Many simulation tools focus on modeling and implementation issues, while only facilitating the execution of single simulation runs and some basic data collection. Fewer tools provide built-in experimentation and analysis environments; sometimes handling experiments of multiple simulation runs or aiding in complex tasks such as simulation-based optimization.

However, regardless of Zeigler's well-known claim to separate models from experiments (see e.g. Zeigler et al. 2000), most of these tools are closely coupled to a certain modeling environment. Experimentation support, though in principal independent of specific modeling styles, is therefore re-implemented over and over again. Some promising steps towards generic and extensible experimentation environments have been taken recently (see e.g. Schöllhammer 2001) but the respective systems are often prototypical and lack adequate usability.

During the last years, the Java-based open-source project Eclipse has evolved from an IDE (integrated development environment) to a general tool platform. Eclipse's plug-in-mechanism allows to integrate arbitrary extensions into a consistent application frame. Thus, the platform also provides an appropriate and contemporary architectural framework for the development of simulation tools. In this paper, we present the concept and a prototypical implementation of an interactive experimentation environment based on Eclipse. It integrates the functionality of generic experimentation tools previously developed in our group into experimentation support plug-ins.

The paper is organized as follows: First, we briefly introduce the terminology of our experimentation framework. Next, we describe the Eclipse platform emphasizing its simulation-specific benefits. We then review experimentation tools from our simulation group and related work from the literature. From this summary, we derive requirements on a generic experimentation environment. As the main part of this contribution, we elaborate on our object-oriented framework and present a concept for an Eclipse-based experimentation system with prototypical plug-in implementations. Finally, we conclude and provide an outlook to our further work.

SIMULATION EXPERIMENTS

Experimentation is a fundamental method of scientific research. Wittmann (1993, p. 47) stresses the analogy between experimentation in the empirical sciences and computer simulation. From this, he develops a terminology that we adopt with modifications for our framework.

An *experiment* is "a number of [simulation] runs that we execute with different models in order to answer a certain question" (literal interpretation of

Proceedings 20th European Conference on Modelling and Simulation
Wolfgang Borutzky, Alessandra Orsoni, Richard Zobel © ECMS, 2006
ISBN 0-9553018-0-7 / ISBN 0-9553018-1-5 (CD)

Wittmann 1993, p. 57). This definition mirrors the separation of models and experiments: Different experiments can be conducted with the same model if the attended questions lie within the model's validity range, and an experiment can include different models (e.g. for the purpose of model comparison).

Wittmann refines the notion of models by distinguishing a *model* from a *model class*. As in object-orientation, a model class is a template that is defined by "a set of model elements [i.e. constants, parameters, state variables, and derived elements] and a description of their dynamics" (Wittmann 1993, p. 55). A model is an instance of a model class with concrete values assigned to these elements.

To allow for generic experimentation support, it seems reasonable to neglect internal model structure and behavior and consider a model class as a *black-box* with a well-defined input-output interface. Following Bachmann (2003, pp. 77), we define a model class by a set of *access points*, i.e. typed *model and experiment parameters* as inputs and observable *results* and *runtime variables* as output. Thus, an experiment might be reused with any model realizing the same model class.

An experiment is described by means of an *experiment specification* and an *experimental setup* that jointly constitute a kind of *experimental frame* (see e.g. Zeigler et al. 2000). The experiment specification states which model classes to use and how to vary their parameter values. Parameter variations are either specified in terms of *iterations* (e.g. similar to "for/to/next" loops) or through higher-level specifications of simulation-based optimization problems. We refer to the former as *manual experimental design* and call the latter *automated experimental design*.

Following the often-cited idea of a "virtual laboratory", the experimental setup describes the control and observation apparatus applied in an experiment. This includes settings of experiment control parameters (e.g. simulation duration) on the one hand, and the setup of observers and (online and offline) analyses on the other hand. The execution of an experiment specification within an experimental setup leads to a series of *simulation runs* (see also Wittmann 1993, p. 56).

THE ECLIPSE PLATFORM

Originally designed as a Java IDE, Eclipse evolved into an extensible tools platform maintained by a broad open source community (`www.eclipse.org`). An underlying plug-in runtime environment based on the OSGi standard (Open Services Gateway Initiative, `www.osgi.org`) provides a dynamic extension mechanism, enabling the addition of functionality to existing components.

Eclipse's core features are concentrated in the *Rich Client Platform* (RCP) which is a framework that facilitates the development of arbitrary applications. As an integral part of RCP, a *generic workbench* supports user interaction. The *Java Development Tools* (JDT) extend the Eclipse platform with a set of plug-ins that add Java IDE functionality, including team support, debugging, and refactoring facilities. Plug-ins can be created using the *Plug-in Development Environment* (PDE).

Many sub-projects aim at developing Eclipse extensions for various purposes such as IDEs for other languages, editors for specific document types, or modeling tools. A broad range of proven and tested plug-ins are available. Following the concept of software reuse, many of them can be adapted and integrated into an experimentation environment to support the tasks that arise in the context of a simulation study. Below, we briefly outline some existing plug-ins or projects that could facilitate simulation:

- The *Eclipse Modeling Framework* (EMF) supports the specification of structured data models and offers code generation facilities. Within an experimentation environment, it can be used to model data structures such as experiment specifications or parameters and results with arbitrary complex data types. Additionally, EMF incorporates XML serialization for persisting simulation projects.
- The *Graphical Editing Framework* (GEF) allows to create graphical editors that are suitable for the construction of conceptual models. The related *Graphical Modeling Framework* (GMF) provides an infrastructure for developing visual editors for EMF models.
- The *Business Intelligence and Reporting Tools* (BIRT) project contributes a reporting system for Java applications. It could serve as a basis for the implementation of simulation result presentation and model documentation tools.
- Furthermore, there are several extensions of the Eclipse platform available that support Java software development in the modeling phase of a simulation study such as testing, code management or documentation tools.

RELATED WORK

Numerous custom simulation engines are used in fields such as logistics, robotics or physics. These tools are often rigidly coupled to a dedicated modeling and simulation environment confined to a specific application domain.

In contrast, universal simulation systems are suitable for a much broader range of application. As they typically offer an integrated solution, aiming to support all phases of a simulation study (Page and

Kreutzer 2005, p. 248), they include infrastructure for experimentation and result analysis. However, these components cannot normally be detached from the simulator. *Simplex3* (Schmidt 2001) is a universal simulation system with academic background. It consists of a specific simulator and a user environment that supports modeling, experimentation and result analysis. Most commercial products, such as *Extend* (Krahl 2002), fall into this category as well.

An advantage of simulator-specific experimentation environments is that they can account for special characteristics of the associated simulation engine. However, it seems desirable to reuse experimentation functionality, following principles of modern software development. Only few such generic experimentation environments have been proposed in the past.

The distributed simulation-based optimization framework *DISMO* (Gehlsen and Page 2001) has been developed at the University of Hamburg. It allows to integrate different Java-based simulators by using wrapper classes. DISMO provides a simple experimentation environment that focuses mainly on automated experimental design and parallel execution of independent simulation runs distributed over a computer network.

Another generic simulation tool developed in our group is the component-based infrastructure *CoSim* (Bachmann 2003) which follows a different approach. It provides cross-platform simulation in distributed environments, ensuring CORBA-based interoperability by specific model interface descriptions. CoSim includes an experimentation environment *ExpU* (Schöllhammer 2001) that permits experiments to be defined and managed independently from specific model implementations (e.g. by using experiment variables). Recent literature contains some further approaches to universal experimentation environments, including the work of Lampe (2004) and the *SimEnv* system (Flechsig et al. 2005) which, however, is confined to the field of climate research and emphasizes quality assurance matters.

Over the last few years, the Eclipse platform's qualities were increasingly noticed by the simulation community. Several Eclipse-based simulation tools have been proposed so far, including *Etomica* (Bucher et al. 2005). The Etomica development environment allows to construct, run and save molecular simulations. In addition, it facilitates dynamic model modification and real-time visualization during experiment execution. The *UrbanSim* (Freeman-Benson and Borning 2003) system for modeling and simulation of urban development contained an interactive Eclipse-based user interface. Recently, this user interface has been removed and is not likely to be developed any further.

The examples show that existing Eclipse-based simulation tools are confined to specific simulators and application domains. Furthermore, most tools rather concentrate on the modeling phase while experimentation is hardly incorporated. To our knowledge, no universal experimentation environment based on the Eclipse platform has been proposed so far.

REQUIREMENTS

Considering the previous summary of related work, we identified requirements that we regard as particularly important for generic simulation and experimentation environments. This section is based on discussions by Page and Kreutzer (2005, pp. 240–243) and Wittmann (1993, Ch. 4).

General requirements that are demanded of modern software systems include:

- a modular architecture consisting of discrete functional units instead of a monolithic approach,
- the reuse of existing software,
- an open, extensible design facilitating customization for specific applications, and
- usability (e.g. supported by a consistent UI concept).

Conformance to these requirements is strongly supported by using the Eclipse platform as a technical basis.

One of the most relevant *simulation-specific requirements* is genericness which can be achieved by separating simulation and experimentation functionality. This leads to a broader applicability of an experimentation environment. Though our approach currently focuses on experimentation, an integrated simulation system should facilitate model design and implementation as well. Ideally, this incorporates specific support for different kinds of model representations such as UML diagrams or Java source code.

Requirements concerning the *experimentation phase* include:

- the detachment of experiments from specific model implementations,
- support of experiment specification and subsequent automatic experiment execution, performing systematic parameter variations,
- an integration of both manual and automated experimental design,
- user interaction facilities such as simulator control and information on experiment execution progress,
- parallel and distributed execution of simulation runs, and
- monitoring of selected simulations that may run on a remote server, including logging, filtering, and condition checking mechanisms.

Result analysis demands integration of data analysis capabilities facilitating both statistical aggregations and interactive exploration of simulation data. An experimentation environment should also allow for report generation including visualizations. Parallel to other phases, a simulation study involves several *auxiliary processes* that need to be supported equally. First of all, validation (including e.g. calibration) and verification are of considerable relevance. Therefore, an experimentation environment requires functionality for interactive debugging and tracing of model behavior. In addition, automated operational validation could be supported (e.g. using data mining methods). Other important requirements concern simulation project management; including an appropriate simulation data persistence mechanism, the preservation of data consistency, the reconstructability of results, and a project-related model and experiment documentation system.

A GENERIC EXPERIMENTATION FRAMEWORK

The foundation for our experimentation environment is an object-oriented experimentation framework that allows to integrate arbitrary simulators and models. As the only precondition, these must provide a *Java* interface to control the initialization and execution of simulation runs. This restriction seems appropriate since numerous models and simulators are either written in Java or can be adapted via the *Java Native Interface* (JNI).

The framework architecture builds upon the DISMO system and the CoSim experimentation environment. Models and simulators are adapted to the framework by implementing a *simulation wrapper* interface that provides standardized access to the model's parameters, results, and runtime variables and to the simulator control (e.g. start, stop, and step functionality). The simulation wrapper encapsulates a model and an associated simulator as a black-box; thereby implementing a model class. At the current stage, the wrapped model's internal consistency must be taken for granted, since it cannot be checked by the experimentation framework.

The framework's components reflect the main tasks occuring in experimentation (see Figure 1). The central component is an *experiment manager* that arranges the experimental setup and controls the experiment's execution. In particular, it initializes an *experiment planner* responsible for the systematic variation of parameter settings; and an *execution manager* that encapsulates the (potentially distributed) execution of simulation runs. The experimental setup is completed by registering *observers* and *analyses* that monitor the experiment execution and process available runtime values or simulation results.

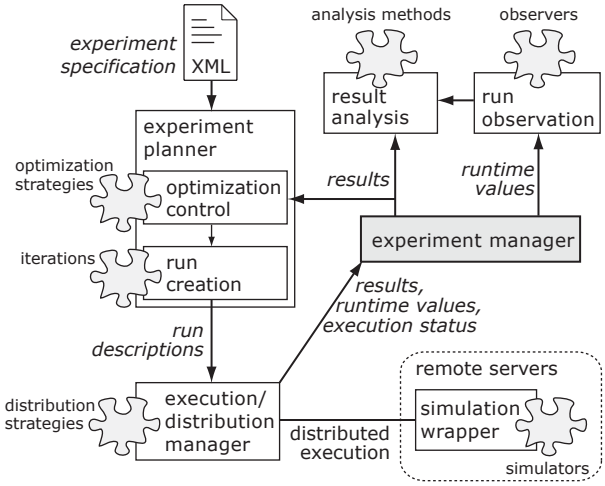

Figure 1: Architecture of a Generic Experimentation Framework

The experiment planner incorporates two subcomponents supporting manual and automated experimental design. The manual experiment planner reads experiment specifications stated in a newly developed XML-based Experiment Specification Language (XESL). This language provides several iteration types such as value ranges, sequences and cartesian products to describe systematic variations of model and control parameters.

The automated experiment planner's input are descriptions of simulation-based optimization problems. These are stated in terms of an optimization algorithm and an objective function implemented as Java classes. Both are referenced and parametrized in the XESL scripts and instantiated through *Java reflection*. From this input the automated experiment planner generates a sequence of *scenarios* (i.e. parameter settings) that are evaluated through simulation and rating of results by the objective function. Since scenarios need to be repeated with variations to ensure statistical validity, descriptions of parameter iterations might be added.

From an experiment specification, the experiment planner creates *simulation run descriptions* that are passed to the execution manager. These descriptions include the name of a simulation wrapper and a list of parameter settings. The execution manager instantiates and executes these runs; where the execution is carried out either sequentially on a single machine or in parallel and distributed on remote servers in a network. The execution manager employs a *distribution strategy* that encapsulates a middleware and a strategy for load distribution. During and after the simulation, runtime variables and results are observed, filtered, and passed to several analysis components.

The framework provides a number of extension points based on Eclipse's plug-in mechanism. Additional simulators, analysis techniques, runtime ob-

servers, filters, experimental design and distribution strategies can be integrated. The prototype contains basic implementations such as a simple distribution strategy based on the *JADE agent platform* (`jade.tilab.com`) where load distribution is achieved through a ranking of remote servers according to their prior performance. As a simulator, our discrete event simulation framework DESMO-J (Page and Kreutzer 2005, Ch. 10) has been integrated.

EXTENDING ECLIPSE FOR EXPERIMENTATION SUPPORT

One of Eclipse's main advantages is the fact that a large number of useful components already exist that can be reused for conducting simulation studies (e.g. a view to navigate within a project structure or team support). Additional functionality can easily be embedded using extension points. In the following, we outline some experimentation-specific Eclipse extensions that we have developed or plan to provide.

In particular, we defined a *simulation project nature* (i.e. an Eclipse project type) that establishes an association between our experimentation tools and projects within Eclipse's workspace. A graphical wizard facilitates the creation and configuration of new simulation projects (e.g. by generating a specific folder structure). Existing simulation projects can be browsed using the standard navigator view.

We provide an *experiment wizard* that creates pre-configured experiment specification files (containing XESL scripts). A tree-based graphical editor supports the actual experimental design. It is intended to be extensible by field editors for user-defined complex data types and by support for new iteration types or optimization methods.

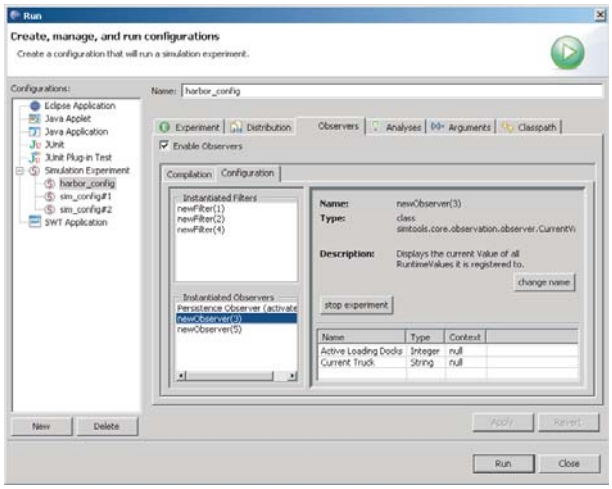

Figure 2: Launch Configuration Dialog with Observer and Filter Setup (User Interface Prototype)

Since the manual creation of simulator-specific model adapters is a complex and error-prone task, a *model wrapper wizard* guides the user through this process of model integration. After reading a model's parameters, results, and runtime variables using Java reflection, the wizard enables a user to make a specific set of these access points available for experimentation. Finally, the wizard automatically generates the model wrapper source code.

We have developed a special *launch configuration type* that appears in Eclipse's launch dialog and facilitates the creation of experimental setups. This involves the registration of observers and filters (see Figure 2), the choice and configuration of a distribution strategy, and the selection of required data analyses.

In addition, the current prototype contains an open *analysis tool*, which provides an extension point for user-defined analyses. Some exemplary standard statistics have already been included as a plug-in. A wizard supports the configuration and execution of offline analyses.

All simulation-specific Eclipse extensions are accessible within a *simulation perspective* (defining the initial setup of views and editors) to provide the user with a consistent experimentation interface. Figure 3 gives a first impression of this perspective.

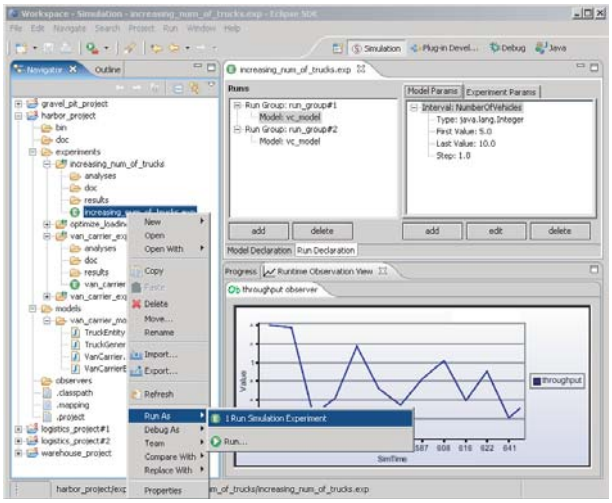

Figure 3: Eclipse-based Experimentation Environment (User Interface Prototype)

In the future, we plan to integrate functionality such as, among others, *simulation-specific action sets* (i.e. a collection of buttons or context menus), *simulation project builders* that aid in automatic generation of results or analyses, and specific views showing, for example, the status of registered remote simulation servers.

CONCLUSIONS

On our way to an Eclipse-based generic experimentation environment, we have developed an object-

oriented experimentation framework that allows to integrate arbitrary models and simulators via a simulation wrapper interface. The framework integrates the functionality of experimentation tools developed earlier in our simulation group; including manual experimental design with an XML-based language (XESL), simulation-based optimization, distributed execution, observation, and analysis of experiments with multiple runs. The discrete event simulation framework DESMO-J has been integrated as an example simulator.

Based on this framework, we have designed several experimentation support plug-ins for the Eclipse platform. In our opinion, Eclipse is an appropriate foundation for an interactive simulation environment: On the one hand it provides a large number of plug-ins that can be reused for conducting simulation studies. On the other hand we could straightforwardly realize simulation-specific extensions based on built-in concepts. We have started to implement these plug-ins as part of a student project.

In our future work, we will evaluate the utility of our experimentation system in large-scale simulation studies. To prove its generality, we will integrate other simulators based on different world-views; such as the Petri net simulator *Renew* (Kummer 2002). We also plan to combine the experimentation plug-ins with graphical modeling tools; thereby realizing an Eclipse-based simulation system.

Another main focus of our work is the development and integration of tools and techniques that aid in the understanding and validation of complex (e.g. agent-based) simulations. This includes better model documentation techniques and the application of advanced analysis methods from pattern recognition and data mining. An example are automated operational validation suites similar to the well-known *unit tests* from software engineering.

ACKNOWLEDGEMENTS

The authors would like to thank the participants of the SimTools teaching project at the University of Hamburg for their support in developing the presented concepts and prototypical tools.

REFERENCES

Bachmann, R.: 2003, *Ein flexibler, CORBA-basierter Ansatz für die verteilte, komponentenbasierte Simulation*, PhD thesis, Faculty of Informatics, University of Hamburg.

Bucher, H., Schultz, A. and Kofke, D. A.: 2005, An eclipse-based environment for molecular simulation. eclipse Technology eXchange (eTX) Workshop at OOPSLA 2005. ACM Digital Library.

Flechsig, M., Böhm, U., Nocke, T. and Rachimow, C.: 2005, Techniques for quality assurance of models in a multi-run simulation environment, *in* K. M. Hanson and F. M. Hemez (eds), *Proceedings of the 4th International Conference on Sensitivity Analysis of Model Output (SAMO 2004)*, Los Alamos National Laboratory, Los Alamos, pp. 297–306.

Freeman-Benson, B. N. and Borning, A.: 2003, Experience in developing the UrbanSim system: tools and processes, *in* R. Crocker and G. L. Steele (eds), *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, ACM, Anaheim, pp. 332–333.

Gehlsen, B. and Page, B.: 2001, A framework for distributed simulation optimization, *in* B. A. Peters et al. (eds), *Proceedings of the 33rd conference on Winter simulation*, IEEE Computer Society, Washington, pp. 508–514.

Kelton, W. D.: 2001, Some modest proposals for simulation software: Design and analysis of experiments, *Proceedings of the 34th Annual Simulation Symposium*, IEEE Computer Society, Seattle, pp. 3–9.

Krahl, D.: 2002, Extend: the extend simulation environment., *in* J. L. Snowdon and J. M. Charnes (eds), *Proceedings of the 34th Winter Simulation Conference*, ACM, pp. 205–213.

Kummer, O.: 2002, *Referenznetze*, Logos-Verlag, Berlin.

Lampe, T.: 2004, Parallelisierung von Simulationsexperimenten, *in* T. Schulze, S. Schlechtweg and V. Hinz (eds), *Simulation und Visualisierung 2004 (SimVis 2004)*, SCS Publishing House, pp. 141–152.

Page, B. and Kreutzer, W.: 2005, *The Java Simulation Handbook – Simulating Discrete Event Systems with UML 2 and Java*, Shaker, Aachen.

Schmidt, B.: 2001, *The Art of Modelling and Simulation: Introduction to the Simulation System Simplex3*, SCS Publishing House, Ghent.

Schöllhammer, T.: 2001, *Eine offene Experimentierumgebung für verteilte Simulationsobjekte*, Master's thesis, Faculty of Informatics, University of Hamburg.

Wittmann, J.: 1993, *Eine Benutzerschnittstelle für die Durchführung von Simulationsexperimenten: Entwurf und Implementierung der Experimentierumgebung für das Simulationssystem SIMPLEX II*, PhD thesis, Universität Erlangen-Nürnberg.

Zeigler, B. P., Praehofer, H. and Kim, T. G.: 2000, *Theory of Modeling and Simulation*, Academic Press, San Diego (CA).

AUTHOR BIOGRAPHIES

BERND PAGE holds degrees in Applied Computer Science from the Technical University of Berlin, Germany, and from Stanford University, USA. As professor for Applied Computer Science at the University of Hamburg he researches and teaches in the field of Computer Simulation as well as in Environmental Informatics.

NICOLAS KNAAK obtained his diploma degree in Computer Science from the University of Hamburg, Germany, in 2002. Since then he has been working as a scientific assistant and PhD student in the simulation group led by Prof. Page.

RAINER CZOGALLA graduated with a diploma degree in Computer Science from the University of Hamburg, Germany, in 2003. He currently holds a PhD scholarship from the federal state of Hamburg.