USING JAVABEANS TO TEACH SIMULATION AND USING SIMULATION TO TEACH JAVABEANS

Herbert Praehofer
Department for Systems Theory and Information Engineering

Johannes Sametinger, Alois Stritzinger
Department for Software Engineering
Johannes Kepler University
A-4040 Linz / Austria
hp@cast.uni-linz.ac.at, sametinger@swe.uni-linz.ac.at, stritzinger@swe.uni-linz.ac.at

KEYWORDS

Simulation education, discrete event simulation, Java, Java Beans, general systems theory

ABSTRACT

This work reports on two courses for computer scientists. The first is a graduate course on modeling and simulation and the second is a course on component based software engineering. For both of these courses we principally use the same basis, that is, the Java Beans component technology and the DEVS system theory formalism for discrete event simulation. However, in the two coursed, we pursue quite different objectives. While in the first course the Java Beans technology is used to teach discrete event simulation, in the second course we use simulation as an example to show how a component library can be realized using Java Beans.

MODELLING AND SIMULATION COURSE

Introduction

The first course is a classical modeling and simulation course for computer science students. Emphasis is more on simulation modeling and simulation programming than on system analysis and experimental data evaluation. The course takes into account the background and also the interests of the students. They have a strong background in software engineering and object oriented programming but also a good knowledge of statistical methods. However, many students lack knowledge and experience in classical technical fields and mathematics, e.g. differential equations. The course is an attempt to teach simulation from a systems theoretic perspective and using modern software engineering approaches. Much efforts have been put into the pedagogical preparation of the topic.

So the emphasis is to give them an overview and an general understanding of simulation and programming of simulation systems. After the course students should

- have a general understanding what simulation is and how simulation works
- have an understanding of the different simulation approaches
- have an overview of the manifold application areas of simulation
- know when and where the different simulation approaches can be applied
- be able to see potential applications and limitations of simulation approaches
- be able to evaluate the advantages and limitations of various simulation languages and tools
- know in detail how simulation systems are built up and how they are implemented
- have a broader view of programming, especially, be able to work with state space descriptions and state transition diagrams, event communication, and hierarchical object composition (see below).

We try to achieve these manifold objectives pursuing the following points:

- We base on a general systems theoretic background and a state space description for continuous models, discrete time models as well as for discrete event models.
- We work out that system simulation in general deals with the complexity emerging from components‘ dynamic behavior and components‘ interaction.
- We work out the essential of the modeling approaches without considering any particular application, however, with referring to examples in different application domains.
- We work out more the commonalties of the modeling approaches and application domains than stressing their differences and specialties.
- Instead of taking a commercial simulation language or system students work with a general purpose programming language (Java) and a library of simulation components which, however, does not enforce any particular modeling approach.

Course outline

The students are introduced step by step into the concepts of modeling and simulation:

- By starting with finite state automata, they should get a first flavor of modeling and simulation. They should get a feeling what dynamic systems are. Simulation is explained as the generation of behavior over time.

- By cellular automata it is shown how complex behavior emerges from elementary components and communication between components.
- An event based simulation of cellular automata is introduced to show the efficiency advantage of an event approach to simulation.
- The DEVS formalism as a system theoretic formalism is used to introduce event behavior and event scheduling.
- Discrete process modeling and simulation is discussed as involving three fundamental elements:
  - resources which are needed to do the processing of some items
  - items which need and occupy resources and are distributed between the resources
  - control which decides which items get which resources.
- A component library is introduced which provides elementary building blocks for the three elements.
- It is shown how this general schemes manifests itself in various application domains and on different system levels.
- Continuous simulation is discussed from a systems theoretic perspective.
- By discussing elementary continuous systems it is shown that feedback is the reason for complex behavior.
- Combined discrete/continuous modeling is presented as a combination of DEVS modeling and continuous modeling.
- The course is finished by presenting the automatic highway simulation (AHS) architecture [Eskafi and Göllü 98] as an advanced example showing elements of combined simulation as well as process simulation.

In the accompanying exercises small groups of students realize some basic simulation programs. In line with the course the students realize a cellular automata simulator first, then use the component library to realize several discrete event simulation examples and finally use a block oriented simulation system to get some experience with continuous simulation. Emphasis however strongly is on the discrete event part and on using the JavaBeans implemented component library for simulation. We, therefore, describe in detail the background, objectives, design, and the use of this component library in the following.

JAVABEANS COMPONENT LIBRARY FOR DISCRETE EVENT SIMULATION

The basis for the component library for discrete event simulation is (1) the DEVS formalism as a system theoretic formalism for modular hierarchical simulation modeling, (2) the JavaBeans component model. As we will see these two complement each other. We will shortly review these two topics.

DEVS Formalism as a Formal Basis for Modular Hierarchical Simulation

Modular, hierarchical system modeling [Zeigler 84, Zeigler 90, Pichler, Schwärtzel 92] is an approach to complex dynamic system modeling where modular building blocks, i.e., system components with a well defined interface in the form of input and output ports, are coupled in a hierarchical manner to form complex systems. In system modeling, one distinguishes between atomic or coupled models. While an atomic model specifies its internal structure in terms of its set of states and state transition functions, a coupled model's internal structure is specified by its components and coupling scheme, i.e., how ports are connected. System modeling means building complex systems by interface based object composition. Modularity allows for setting up bases of reusable building blocks which can be plugged into a system through their well defined input and output interface.

Modular hierarchical system modeling concepts have been applied in several domains, most notably hardware design and communication engineering. Zeigler [Zeigler 76, Zeigler 90] introduced modular hierarchical modeling for discrete event simulation. The DEVS-formalism is an application and computer implementation independent formalism for discrete event modeling and parallels the differential equation specified and finite state automaton formalism.

Java/JavaBeans as a Simulation Language

Java is a programming language with interesting features for simulation. It is purely object oriented, it is widely available, already well-known and appreciated, and easy to use. Its run-time environment providing garbage collection frees the programmer from storage management duties. Due to its popularity, many libraries are available.

JavaBeans is the component model for Java. JavaBeans are ordinary Java classes that adhere to certain property and event interface conventions. JavaBean components are portable and platform-independent. They enable developers to write reusable components that may run anywhere. Beans are manipulated and composed together into applications in visual builder tools. JavaBeans defines a convention for events and event communication which can advantageously be used for discrete event simulation. We adopted the JavaBeans event model for the realization of event couplig and communication in discrete event modeling. We review the JavaBeans event model in this sense.

Events: Events are used for event communication among bean components. With events, a bean can notify other beans that something interesting has happened. For that purpose event listeners are registered to event sources. An event source has to provide two methods that allow the registration of listeners, viz., `add<`*`Event`*`>Listener` and `remove<`*`Event`*`>-Listener` to register components that are interested in receiving `<event>` events (where `<event>` is the event name).

Properties: Properties are named attributes that can be read and written by calling appropriate methods. Naming conventions have to be obeyed for definition of methods. Two methods, 'get<*Property*>' and 'set<*Propert*>' have to be provided for reading and writing property values. So-called *bound properties* allow *PropertyChangeListeners* to watch properties for value changes. PropertyChangeListeners are added to properties by method *addPropertyChangeListener*. PropertyChangeListeners and are intended to be informed of changes in the property by a call of their *propertyChange* method.

We use the event and bound property change concepts to realize event coupling and communication in discrete event models. Models which rely on states of other models are registered as PropertyChangeListeners of the other model's state property and are so informed whenever a change in state happens. For example, a processor needing a particular tool registers itself as a listener of the *available* property of the tool pool. As soon as a tool gets available, it is informed and can access it. In similar way, events can be used to distribute objects between model components in an eventistic way.

Synergism between DEVS Formalism and JavaBeans Component Technology

Zeigler [Zeigler 90] compares modular hierarchical system modeling and the object oriented paradigm. While both share various concepts they also show major differences. System models as well as objects use the concept of internal states. However, objects are not dynamic systems in the sense of system theory which show dynamic, concurrent behavior specified over a time base. Nor are objects modular units in the above sense. In particular, objects lack an explicit output interface. The output interface is only given implicitly by the method calls to other objects. Therefore, an object has to have knowledge of the other objects with whom it communicates. In contrast to that, a system model only defines an output port, the communication with other components is defined independently on the higher coupled system level. Object oriented systems also are not hierarchical in the above sense.

The component based software engineering represents a step towards a modular system modeling approach. Components define an explicit input and output interface in the form of methods calls they accept and events they generate. Component based programming should primarily mean interface based component composition. Components can be constructed hierarchically using smaller components.

JavaBeans Simulation Framework

In the Java Beans simulation framework, the realization of simulation systems is based on the following ideas and concepts:

- A set of elementary, most general, yet powerful building blocks for simulation modeling and simulation output, statistical evaluation, and visualization is provided.
- A library of utility and support objects are provided. By them, model components can be customized in their functionality to meet particular requirements.
- Model interfaces are defined which specify where and how model components can be used. The interface specifications are put into a classification hierarchy to define compatibility between model components [Thomas 94].
- Interfaces and interface-based classification of model components are used to define generic templates for coupled models which define the components' interfaces and coupling structure but not the components itself. At design time, these generic components can be configured by instantiating model components which obey to the interfaces. For example, a *ParallelProcessor* coupled model will specify that it has several parallel components and all have to obey to the *Processor* interface. At design time we can choose from different processor realizations.
- Simulation systems are primarily built bottom-up by hierarchical composition and coupling of model components.
- The component library can easily be extended to meet special needs.
- Employing the elementary component library, special purpose simulation environments for very particular application domains can be realized.

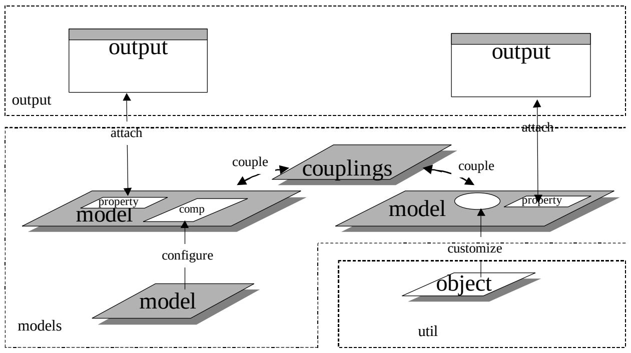

Figure 1 Putting together simulation systems from components

Figure 1 illustrates how simulation systems are put together from components. According the ideas above, we distinguish the following ways for component composition:

- *customize*: Means to customize simulation components to meet different requirements by using utility objects, e.g. random distribution functions, control strategies etc.
- *configure*: Choose and instantiate concrete models for component in a coupled model for which only the interfaces are specified in a top-down manner.

- *couple*: Couple together models in a hierarchical bottom-up way.
- *attach*: Attach beans for simulation output, statistical computation, and visualization and animation to state variables (properties) of models.

Simulation Beans for Discrete Process Simulation

For discrete process simulation a library of elementary beans for simulation modeling is used which is designed according to the view of simulation as encompassing the three elements, that is, resources, items flowing through the system, and control of the item flow. It is supposed to give a perspective of simulation which abstracts from classical process interaction and transaction oriented world views.

Interfaces

Elementary to the whole package are two very general interface definitions for the components in the resource layer. These are the `SItemReceiver` for any component which may receive items and `SItemProvider` for any component which may provide items. Simulation then works by distributing items between providers and receivers.

Interface `SItemReceiver` specifies the following features:

- a method `receive(SItem)` for handing over an item to the component
- a bound property `free` to signal that the component is free to receive an item (can also be interpreted that the component *needs* an item)
- a predicate `test(SItem)` used to see if an item is appropriate to be received by the component
- an event `sItemReceived` which is triggered at the time when the item is actually received.

Analogously, the interface `SItemProvider` specifies the following features:

- a method `SItem provide()` for allowing other components to access available items
- a bound property `available` to signal that an item is available in the component
- an access method `SItem inspect()` to get the next item for inspection
- an event `sItemProvided` which is triggered at the time when the item is actually retrieved.

Model beans

Based on these two very general interfaces a set of elementary building blocks are realized which implement the interfaces in different ways. Building blocks include Generator, Sink, Processor, Queue, Place, and Delay (Figure 2).

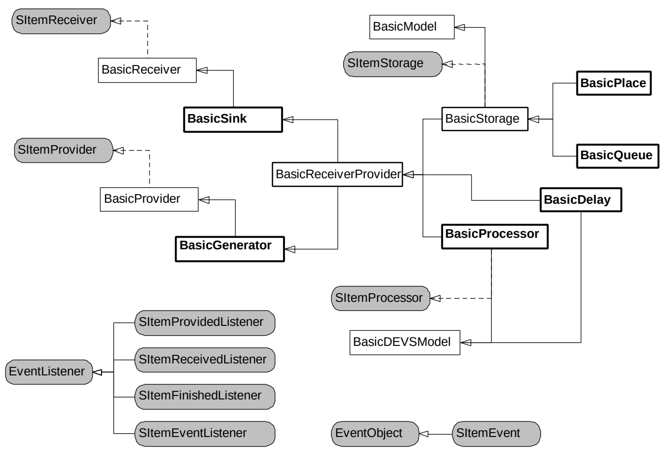

Figure 2 Class hierarchy for elementary model beans: interfaces are gray, model beans are bold.

Coupling components

More complex components and simulation (sub-)systems are built up from these basic components. The general idea of coupling components is that the containing model listens to the property changes (`free` and `available`) of its subordinates and reacts to those by distributing the items between its subordinate providers and receivers based on its individual control scheme (Figure 2).

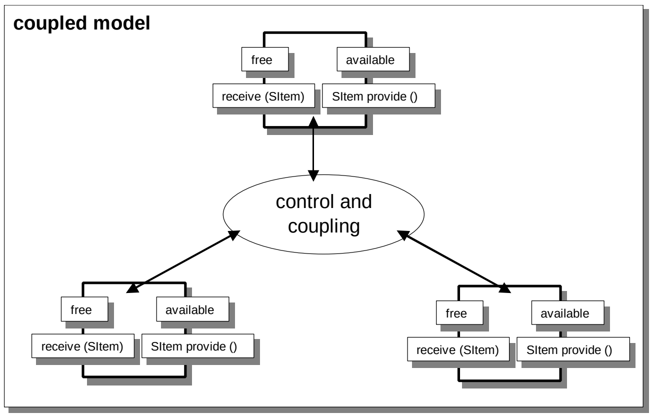

Figure 3 Hierarchical coupling of provider and receiver components: The coupled model listens to changes in the property values of its subordinate and reacts according to its control scheme by distributing the items from providers to receivers.

Usage of the beans library in the simulation course

This general approach then is used to demonstrate various modeling approaches and different applications domains.

Modeling of typical Flow Shop systems is demonstrated by introducing components to realize the linear couplings typical for those systems. A `Forwarder` is a component which directly forwards an item from a provider to a receiver as soon as an item is available at the provider and the receiver is free to take it. A `ReceiverDecision-`

`Point` is a component to couple one provider with several receivers. The selection of the receiver for an available item can be customized by different control strategies, like random, according to a percentage, least recently received, longest free or user defined strategy. The component `ProviderDecisionPoint` is analogous for the selection from several providers.

The next step in the course is to show to the students how JobShop models can be realized. The same elementary components are used which, however, now are coupled in more complex way by individual coupling and control schemes.

Finally by considering manufacturing and transport systems it is shown how the same elementary components are used for quite different purposes at different system levels, e.g., once in the flow of items and then also in the transport system.

COMPONENT PROGRAMMING COURSE

The objective of the component programming course is to introduce this new programming paradigm to software engineering students. Those students already have a strong background in object oriented programming and also Java. They are supposed to learn component programming, theoretically and also practically.

The component approach to programming has emerged from graphical user interfaces and has not widely been used in other areas yet. Simulation certainly is an area where component-based programming can be applied. The following points were relevant for our decision to used simulation as an example:

- the basics of discrete event simulation are easy to comprehend and can be introduced in a few lessons
- components can easily be identified
- through its event-driven nature, discrete event simulation is most appropriate to teach event-driven programming.

Especially the last point makes simulation an most interesting area to study the problems of component-based, event-driven programming. For example, during the development of the simulation beans, we identified several severe problems in the event handling of the standard JavaBeans model which are of general nature and of which the software engineering community seems not to be aware (see [Stritzinger, Sametinger, Praehofer 98] for details). It is our opinion that simulation can and should play a similar important role for the development of component-based programming as it played for the development of object-oriented programming.

SUMMARY AND OUTLOOK

In this work we have reported on two courses which combine two areas which complement in an optimal way. Using JavaBeans as a basis for discrete event simulation makes this new powerful programming paradigm available for simulation programming. On the other side, discrete event simulation with its complex event handling can give interesting insights into component based programming in general. The positive feedback from the students strongly confirm this. Also after having done similar modeling and simulation courses using other languages, namely Scheme, object-oriented Pascal, and C++, in former years, we very much appreciate Java/JavaBeans and regard it a milestone in computer science.

Students were able to realize quite nice simulation system in very short time. However, the first steps were not that as easy as supposed. It took them some time to get an understanding and overview of the component library and how to work with it. The general message from them was that they began to comprehend and appreciate this type of programming not before they had finished the programming examples. So the plans for the future are to spend an even greater time on the discrete event part and also to provide a continuing student project where they should deal with a more realistic simulation application.

REFERENCES

F. Eskafi and A. Göllü, Simulation Framework Requirements and Methodologies in Automated Highway Planning. Transactions of the SCS, 14(4), 1997, pp. 167 - 180.

Java Beans: *A Component Architecture for Java.* Sun Microsystems, WhitePaper, Dec. 1996.

M. D. McIlroy, Mass Produced Software Components. In J.M. Buxton, P. Naur, and B. Randell, eds., *Software Engineering Concepts and Techniques*, pp. 88-98, 1968 NATO Conference on Software Engineering, 1976.

Anthony Wasserman and S. Gutz, The Future of Programming, *CACM*, 25 (3), 1982.

Oscar Nierstrasz and Laurent Dami, Component-Oriented Software Technology. In Oscar Nierstrasz and Dennis Tsichritzis, *Object-Oriented Software Composition*, Prentice Hall, pp 3-28, 1995.

Oscar Nierstrasz and Dennis Tsichritzis, *Object-Oriented Software Composition,* Prentice Hall, 1995.

F. Pichler and H. Schwärtzel (eds.), *CAST Methods in Modeling*, Springer, 1992.

A. Stritzinger, J. Sametinger, and H. Praehofer, *Perspectives of component-based programming with Java Beans.* Johannes Kepler University, Linz, Austria, 1998.

Clemens Szyperski, Component-Oriented Programming:A Refined Variation on Object-Oriented Programming. *The Oberon Tribune*,1(2):1, pp. 4-6, 1995.

C. Thomas, Interface-based classification of simulation models, *WSC 94*, Orlando, FL, 1994.

B.P. Zeigler, *Theory of Modeling and Simulation.* Wiley, 1976.

B.P. Zeigler, Multifacetted Modelling and Discrete Event Simulation. Academic Press, 1984.

B.P. Zeigler, *Object Oriented Simulation with Modular, Hierarchical Models.* Academic Press, 1990.