# Object-Oriented Modeling with Rand Model Designer

Yu. B. Kolesov [1] Yu. B. Senichenkov[2]

[1]Mv Soft, Russia, ybk2@mail.ru

[2]Institute of Computer Science and Technology, Peter the Great St. Petersburg Polytechnic University, Russia, senyb@dcn.icc.spbstu.ru

## Abstract

Rand Model Designer is a modern tool for modeling and simulation hierarchical multicomponent event-driven dynamical systems. It utilizes UML-based object-oriented Model Vision Language for designing dynamical and hybrid systems using modification of State Machines, and large-scale multicomponent systems: control systems with "inputs-outputs", "physical" systems with "contacts-flows", and novel variable structure component systems, particularly "agent" systems. This article provides a brief overview of the *«Object-Oriented Modeling with Rand Model Designer 7»* book contents (Kolesov et al., 2016), highlighting the differences between RMD and similar environments.

*Keywords: object-oriented modeling, visual environment for modeling and simulation of event-driven complex dynamical systems, dynamical and hybrid systems, component "physical", "causal", and "agent"-based models, behavior of event-driven complex dynamical systems*

## 1 Introduction

Object-oriented Modeling (OOM) is a modern technology of computer simulation (also referred to as computer modeling) widely used in scientific research, design of technical systems and analysis of business processes. OOM helps creating robust computer models in a faster and more cost efficient manner.

Computer modeling practically emerged at the same time with an appearance of first computers. At that point, computer models were created manually using programming languages like Fortran or even Assembly. For instance, one of the authors happened to develop his first computer model – imitator of the hardware (yet to be developed) on the test stand – with machine commands of BESM-4. Conversely, the level of abstraction of any designed system is significantly higher than the level of abstraction of any programming language. Thus, at the time, developer had to keep conformity between the model and its program implementation in his own mind, which, evidently, had led to numerous errors. Oftentimes, programmers used own program implementations of numeral methods to reproduce behavior of continuous systems, though this approach led to questionable results. Partially this issue had been addressed by the use of professional libraries (1980-1983): LINPACK, EISPACK, LAPACK (computational methods in linear algebra), ODEPACK (numerical methods for solving ordinary differential equations).

From 1950s, new high-level programming languages emerged, together with supporting software tools that created the executable computer models automatically. General Purpose Simulation System (GPSS) was one of the first modeling languages (Schriber, 1980). It was meant to be used as a simulator for queueing systems. This language is notable as it relied on object-oriented approach; however, this terminology was not used at that time. Transactions in the simulated system input were created as copies of the standard "Transaction" class, with different values of their attributes, and were destroyed after the execution.

Simulation programming language Simula-67 was released in late 1960s (Dahl et al., 1969). Simula introduced classes, objects, inheritance and polymorphism. Regrettably, this revolutionary project did not become widespread in the field of simulation practice, mainly due to limited computing power and high OOM use overheads of those days. At that time, emphasis was put on single-component isolated mathematic models, transcribed as large and unique equation systems, while the need of increasing speed and accuracy of computation dominated over other problems, such as structuring and modification of models, for instance. These circumstances led to the significant delay, measured in decades, in full-scale application of object-oriented approach in modeling. Conversely, Simula-type objects are re-implemented in a range of object-oriented programming languages, such as C++, Java and Object Pascal that are still in use today.

## 2 Matlab

MATLAB suite, probably one of the best-known universal simulation environments, was developed in late 1970s. It provides an opportunity to describe an isolated single-component dynamic system in a vector-matrix form, to call built-in solver for ordinary differential equations, as well as other solvers from professional systematic collections, and to visualize solutions in form of a time and phase diagrams. Despite the fact that the models in MATLAB suite are described in algorithmic language, similar to Fortran, not in mathematical language, the suite development was a big step forward in the field, thus it justifiably gained the merited recognition.

Although MATLAB suite was primarily oriented toward creation of traditional mathematical models, an additional package added the functionality of component-based modeling. In 1992, this package acquired its present name - Simulink (Dyakonov, 2013). Graphical language of Simulink package has been used already at the time of analog machines and is familiar to developers of control systems – it allows building model of standard blocks, which have real physical equivalents in technical devices. Equations – often disliked by engineers – give way when graphical language of blocks becomes the main tool of model description. Simulink subsystem uses block language to describe models and exempts engineers from the time-consuming process of compiling system of equations corresponding with the whole model. The system of equations is then passed on to MATLAB suite solvers and the result of numeric solution is passed to different "visualizer".

Visual simulation technology of the Simulink suite comes down to an assembly of models from readily available "blocks" - elements of standard libraries with adjustable parameters. In essence, we are talking about OOM, as those "blocks" are principally objects of library classes. Well-designed library of basic classes allows creation of more complex classes by using available proven designs as elements. Structural complexity of new classes is disguised under multi-level hierarchical structure, thus complex classes on higher levels appear as basic elements. However, MATLAB's internal algorithmic language has to be used for any development or modification or of the abovementioned set of basic classes.

## 3 Unified Modeling Language

In late 1990s OOM, essentially, has had a major breakthrough in modeling practice, due to the introduction of the Unified Modeling Language (UML) (Booch et al., 2006) and the language for physical systems modeling Modelica (Modelica Association, 2012) reinforced with corresponding modeling software tools.

UML language is used to design specifications of complex software and hardware systems at the early stages of development. Its importance for discrete-event simulation lays foremost in canonization of concepts of object-oriented approach in software development, as it became de-facto standard of object-oriented approach. Moreover, these concepts were combined with exceptionally convenient and descriptive state diagrams (machines), invented in 1980's by D. Harel. UML language is easily extendable to simulate continuous-discrete (hybrid) systems. One indication of UML popularity could be seen in introduction of the State Flow subsystem, which supports state diagrams, to the Matlab suite.

## 4 Modelica language

Modelica language solved the problem of component simulation automation in the components with non-directional connections use. External variables of Simulink model components are referred to as "entrance" and "exit", which underlines the principal implication of the information transmission direction. However, in many application areas, there is no direction of external variables, for instance, the direction of currents and voltage symbols in electric circuits are rather conventional. Connection of external variables of different blocks in electrical circuits simply means the equality of the voltages and the equality of the sum of currents to zero. This fact fundamentally changes the method of constructing the final equations, which, in turn, leads to substantial problems. Modelica authors overcame these difficulties through introduction of components with non-directional connections, which in effect, broadened the range of application areas for computer simulation. Previously complex models were built using a limited set of basic components that could not be created in input simulating language, however today Modelica allows creating component libraries for electrical, hydraulic, mechanical, and other physical systems utilizing language's own capabilities. Nowadays, user could build any complex structure, where final equations for the structure are automatically generated, similarly to block models of Simulink. Moreover, Modelica classes could be inherited, defined and redefined.

## 5 Tools classification

Currently, there is a number of modeling tools, which support full OOM technology without narrowing focus to any of the application areas. These tools are divided in two groups:

1) tools focused on UML and its essential part - state machine formalism (hybrid automation);

2) tools focused on Modelica language (mechanisms supporting technology of "physical modeling").

First group includes environments like Ptolemy (University of California, Berkeley, USA) (Ptolemaeus, 2014), AnyLogic (The AnyLogic Company, Saint Petersburg) (Karpov, 2005) and Rand Model Designer (MVSTUDIUM Group, MvSoft and Peter the Great St. Petersburg Polytechnic University) (Kolesov et al., 2006; Kolesov et al., 2013). Additionally this group should also include the ever-developing Matlab that is now a complex system of components (MATLAB + Simulink + StateFlow + ToolBoxes). Matlab could be categorized as an object-oriented only with certain provisions – OOM technology is fully supported only by the StateFlow subsystem within Matlab. However, this does not prevent Matlab from holding the leading role in modern applied simulation.

Second group includes environments developed within European project "Modelica", such as Dymola, Open Modelica, and MathModelica etc. For the purpose of this work, we consider OOM technology-based environments to be the most potential and thus we shall focus this discussion only on those.

**Table 1.** Object-Oriented approach.

| Object-Oriented approach | |
|---|---|
| Simulink+Tollboxes | no |
| Modelica-based tools | yes |
| Ptolemy II | yes |
| RMD | yes |

Modern simulating environment that claims to be universal has to allow development of the following models, in frameworks of the OOM technology:

- single-component continuous models
- single-component discrete-event models
- single-component hybrid models
- multi-component models with continuous, discrete or hybrid components and oriented connections (block models)
- multi-component models with continuous, discrete or hybrid components and non-oriented connections (physical models)
- multi-component models with variable composition of components and variable connection structure

**Table 2.** Universality. Definition.

| Universality | |
|---|---|
| Dynamical and hybrid systems («isolated») | Models with «input/output» components («causal») |
| Models with «contact/flows» components («physical») | Models with variable structure («agent-based») |

**Table 3.** Universality. Usage.

| Universality | |
|---|---|
| Tollboxes (Simulink) | yes |
| Modelica-based tools | yes |
| Ptolemy II | no |
| RMD | yes |

Undeniably, if the model has only two components and each one has own state machine with only to states – combined state machine will have for states, and every combined state will have an own corresponding system of equations to solve. Whereas, for components with non-directional connections, construction of aggregate system of equations cannot be reduced to a simple mechanical unification of component equations, typical for components with directed connections, and in general, requires a very complex analysis and transformation of equations. Matlab suite allows working with physical models, though only within frameworks of basic component sets.

**Table 4.** Compliance with UML.

| Compliance with UML | |
|---|---|
| Simulink+Tollboxes | no |
| Modelica-based tools | partial |
| Ptolemy II | yes |
| RMD | yes |

Modelica language supports "physical modeling", however it uses own concept of objects that does not match the one of UML language; it also uses special constructions to describe discrete events in hybrid models. Rather complex analysis of aggregate system of equations is conducted at the stage of model compilation; however, it could only be applied for a limited class of hybrid models, where the number of unknown variables and solved equations does not change with switching.

Modeling practice indicates that state machine of UML language is more convenient and graphical when it comes to the description of discrete-event and hybrid models than the description provided by the authors of Modelica language. Furthermore, numerous practically meaningful hybrid models are difficult to handle when using Modelica language. Besides, within the limits of Modelica's approach it is rather problematical to model systems with variable composition naturally.

## 6 Rand Model Designer

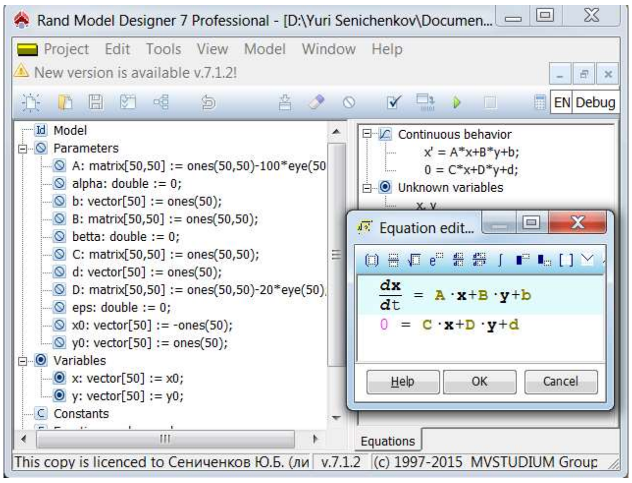
Rand Model Designer - high-performance visual environment for the modeling and simulation of multi-domain component models
PC MAGAZINE
BEST 2015 SOFT

The Rand Model Designer tool attempts to combine the strengths of both directions: supporting the "physical modeling" suggested in Modelica language, while using object paradigm and states machine of UML language. Although, this design comes with an atonement - the need to implement part of the analysis of aggregate system of equations at the stage of model execution with every switch. It occurs that this analysis could be conducted by dint of "linear complexity" algorithms, and thus RMD – created industrial models, based on components with non-directed connections, successfully work in real time. Differences of the input language of RMD environment – Manipulative Visual Language (MVL) language – and Modelica language are analyzed in (Martin-Villalba et al., 2014; Kolesov et al., 2014).

RMD environment features are briefly summarized below:

a. Elementary continuous behavior models

Continuous behavior may be represented in the form of implicit time dependencies, nonlinear algebraic equations (NAE), ordinary differential equations (ODE), and differential-algebraic equations (DAE). Equations are inputted by the user though the equation editor, similar to one in "mathematical" suites (Figure 1).



```
Rand Model Designer 7 Professional - [D:\Yuri Senichenkov\Documen...
Project  Edit  Tools  View  Model  Window  Help
New version is available v.7.1.2!
EN Debug

Model
Parameters
    A: matrix[50,50] := ones(50,50)-100*eye(50
    alpha: double := 0;
    b: vector[50] := ones(50);
    B: matrix[50,50] := ones(50,50);
    betta: double := 0;
    C: matrix[50,50] := ones(50,50);
    d: vector[50] := ones(50);
    D: matrix[50,50] := ones(50,50)-20*eye(50)
    eps: double := 0;
    x0: vector[50] := -ones(50);
    y0: vector[50] := ones(50);
Variables
    x: vector[50] := x0;
    y: vector[50] := y0;
Constants

Continuous behavior
    x' = A*x+B*y+b;
    0 = C*x+D*y+d;
Unknown variables
    x, y

Equation edit...
```

$$\frac{dx}{dt} = A \cdot x + B \cdot y + b$$

$$0 = C \cdot x + D \cdot y + d$$

```
Help  OK  Cancel
Equations
This copy is licenced to Сениченков Ю.Б. (ли | v.7.1.2 | (c) 1997-2015 MVSTUDIUM Group
```

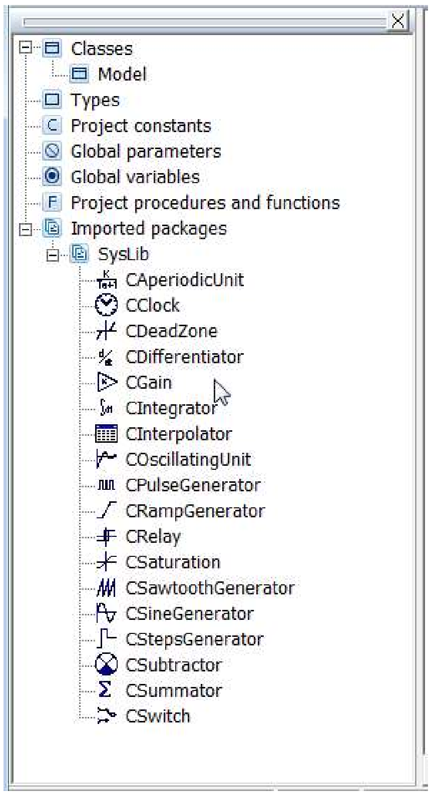
Figure 1. Example of continuous behavior.

Usage of both scalar and vector-matrix forms is possible. Alternatively, equations may be inputted in a block form, as it is done in the Simulink environment. SysLib database (Figure 2), which contains Simulink blocks, helps those users who are accustomed to a graphic description the continuous behavior.

b. Discrete and hybrid behavior models

Models with discrete and hybrid behavior are usually described using hybrid automata. Hybrid machines within RMD are UML state machines without parallel (orthogonal) activities (Figure 3).

Behavior charts feature "orthogonal" time. Many models with hybrid behavior require basing the selection of the next state on fairly complex calculations within the time gap. In particular, this may require modeling of additional dynamic systems, auxiliary to the core model. Thus, this simulation should be carried out in its own hybrid time, which is "orthogonal" to the principal time and is infinitely prompt – "instant". Modeling of additional systems may be carried out in parallel, should the hardware capabilities suffice. Furthermore, RMD includes an option for keeping the specifications of any complex model as a class, and consequently consider abovementioned class's behavior as "elementary" piecewise continuous state machine activity (Figure 4).



```
Classes
    Model
Types
Project constants
Global parameters
Global variables
Project procedures and functions
Imported packages
    SysLib
        CAperiodicUnit
        CClock
        CDeadZone
        CDifferentiator
        CGain
        CIntegrator
        CInterpolator
        COscillatingUnit
        CPulseGenerator
        CRampGenerator
        CRelay
        CSaturation
        CSawtoothGenerator
        CSineGenerator
        CStepsGenerator
        CSubtractor
        CSummator
        CSwitch
```

Figure 2. Library of "a la Simulink" blocks.

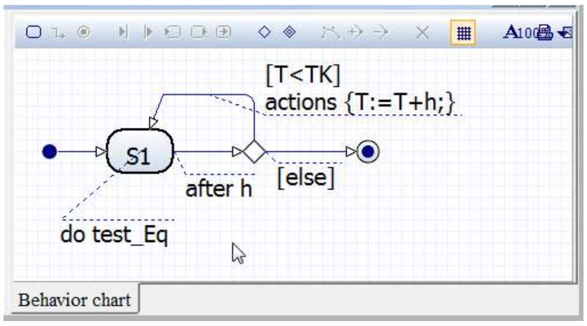


Figure 3. Behavior chart with UML notation.

c. Multi-component models with oriented components

Hierarchical multi-component models with components with the "input-output" and internal hybrid automata (with attributed continuous activities in form of equations or class instance activities, corresponding to complex subsystems models) demonstrate fairly complex behavior, which is best described by a component hybrid machine composition, with an extremely large number of states. It is not considered necessary to create this composition in an explicit form, as an appropriate state behavior could swiftly be created for the implementation of a specific event,

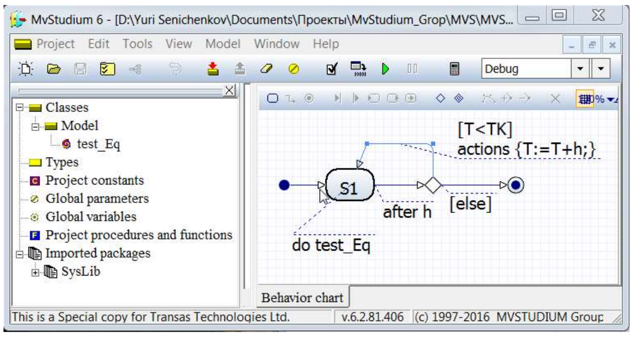

MvStudium 6 - [D:\Yuri Senichenkov\Documents\Проекты\MvStudium_Grop\MVS\MVS...
Project Edit Tools View Model Window Help
Debug
Classes
Model
test_Eq
Types
Project constants
Global parameters
Global variables
Project procedures and functions
Imported packages
SysLib
[T<TK]
actions {T:=T+h;}
S1
after h
[else]
do test_Eq
Behavior chart
This is a Special copy for Transas Technologies Ltd.
v.6.2.81.406
(c) 1997-2016 MVSTUDIUM Group

**Figure 4.** State activity as behavior of model converted into class.

d. Multi-component models with non-oriented components

Inheriting Modelica's technology for creating the multi-component model with oriented components (external variables such as "contact-flow") RMD introduces two significant additions:

- component hybrid automata may have continuous behavior, expressed as equation systems of any size and any type (NAE, ODE, DAE);
- equations, corresponding to component hybrid automata are created "on the fly" during the implementation, rather than at the compilation stage.

e. Multi-component models with variable structure

Construction of the variable structure models or models with "dynamic" components became possible owing to the introduced ability to create the equation for the whole model composition of component hybrid automata. Thus it is possible to create "agent-based models", as the component type (oriented or non-oriented) becomes irrelevant (Figures 5 and 6).

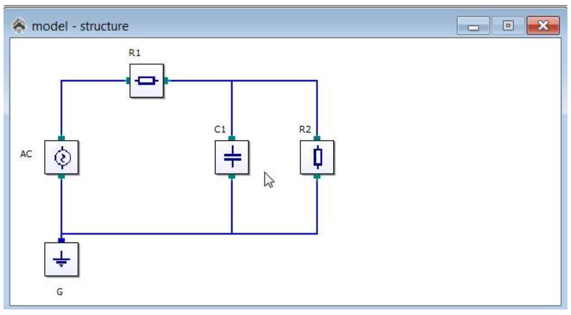


**Figure 5.** Initial structure of a component model.

Rand Model Designer version 7 allows creation of all the above mentioned types of models, based on the OOM technology. Trial version of RMD 7 is available at www.mvstudium.com.

Information about new book (Figure 7) you may find at http://www.labirint.ru/books/539673/ .

Tools of the first group – those that are based on UML – support all the aforementioned model types except for "physical models". This is due to the exponential growth of number of states in a state machine that corresponds to the whole model.

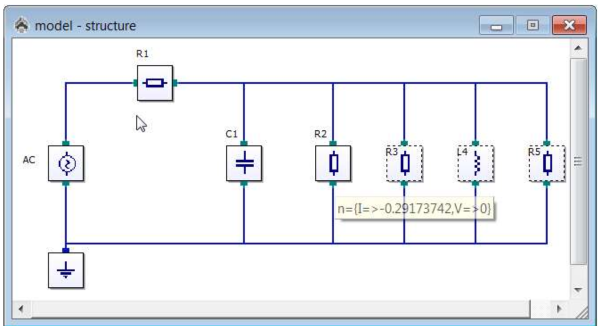


**Figure 6.** Structure of the model after the fifth event.

Ю. Б. КОЛЕСОВ, Ю. Б. СЕНИЧЕНКОВ
ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ МОДЕЛИРОВАНИЕ
В СРЕДЕ RAND MODEL DESIGNER 7
УЧЕБНО-ПРАКТИЧЕСКОЕ ПОСОБИЕ

**Figure 7.** New book «Object-Oriented Modeling with Rand Model Designer 7».

## References

G. Booch, I. Jacobson, and J. Rumbaugh. *UML 2.0*, Saint Petersburgh, Piter, 2006.

Claudius Ptolemaeus, Editor. *System Design, Modeling, and Simulation using Ptolemy II*, Ptolemy.org, 2014. – http://ptolemy.org/books/Systems.

O.J. Dahl, B. Murhaug, and K. Nigard. *Simula-67. Common Base Language.* - M: Mir, 1969.

V.Dyakonov. *Simulink: manual for self-tuition*. – M: DMK-Press, 2013.

Yu. Karpov. *Introduction in modeling and simulation with AnyLogic 5* – Saint Petersburgh: BHV-Petersburg 2005.

C. Martin-Villalba, A. Urquia, Y. Senichenkov, and Y. Kolesov. Two approaches to facilitate virtual lab implementation. *Computing in Science and Engineering*, 16(1): 79-86, 2014.

Modelica Association, Modelica® - *An Unified Object-Oriented Language for Systems Modeling*, Language Specification version 3.3. Modelica Association, 2012. - https://www.modelica.org (accessed in 2013).

Yu. Kolesov, Yu. Senichenkov, A. Urquia, and C. Martin-Villalba. Hybrid systems in Modelica and MvStudium. *Humanities and Science University Journal*, 8: 102-111, 2014.

Yu. Kolesov and Yu. Senichenkov. *Modeling of systems. Dynamical and hybrid systems*. Saint Petersburg, BHV, 2006

Yu. Kolesov and Yu. Senichenkov. *Mathematical modeling of hybrid dynamical systems*, Saint Petersburgh: Publishing of Polytechnic University, 2014.

Yu. Kolesov and Yu. Senichenkov. *Object-Oriented Modeling with Rand Model Designer 7.* Saint Petersburg, Prospect, 2016.

T. J. Schriber. *Modeling and simulation in GPSS.* — M: Mashinostroenie, 1980.