STUDIA INFORMATICA
Nr 1/2(7) Systemy i technologie informacyjne 2006

Basic solutions for a program system for dynamic systems simulation

Timofeev Alexander O.
Institute of Computer Science,
University of Podlasie, Siedlce,
Poland, professor,
a.timofiejew@imm.org.pl

Abstract. The article presents basic solutions of the *Amethyst* program system. The system generates interactive applications for simulating complex dynamic systems. The method of presenting the information about the state of the simulated objects and method of calculating the component state are proposed. Method of decentralized processing the signal delays and filtering is discussed. Method of analog objects simulating is proposed.

The article presents examples of projects produced for *C++ Builder*, *Visual C++* and *Visual Basic 2005* environment projects. The executable times of simulation are compared for the above mentioned environment applications.

Key words: dynamic system simulation, creating interactive applications, the *Amethyst* program system, calculating the component state, decentralized processing the signal delays and filtering, analog objects simulating.

1 Introduction

This article deals with some of the problems, arising during the computer generation of interactive applications for simulating complex dynamic systems, and solutions to them. Autonomously working interactive applications that simulate complex dynamic systems are a good way to teach by the game experiment with a system model.

The advantages of interactive teaching programs are well known, but their development is very time-consuming. There is an urgent need for a program quickly generating autonomously working interactive programs to be developed.

Creating projects of the interactive applications with the help of a computer to simulate the complex dynamic systems helps to verify the dynamic system models. As the whole program text is created, each part of the algorithm can be tested and improved. For example, one can add to any program block instructions to measure the execution time, in order to search the weak points and replace or modify the corresponding blocks.

An algorithm for generating interactive application projects modeling the complex dynamic systems was implemented in *Amethyst program system*, designed for Windows. Amethyst generates application projects for the following environments: *Borland C++ Builder v. 3, 5* or *6, Microsoft Visual C++ or Basic 2005*.

Standard *Amethyst* libraries are designed to simulate electrical devices.

2 Basic solutions of *Amethyst* program system

Most of the known systems simulating technical objects are designed for gathering the information in the form of the data about the state of the nodes, that connect the structural parts of the object. This presentation causes trouble during complex objects simulation, for example in the case of microprocessor simulation a functional scheme is required.

The *Amethyst* system involves a new method to present the object state information. The state of the component is shown in a special format (template). Besides the data fields, this format can hold pieces of text and graphics. The data format fields contain the initial values of the variables or the parameter values. That is why the data format looks like an image. The author of the model can customize this format to his taste, the hypertext data format allows all the 256 extended ASCII characters (or the “Terminal” font).

An autonomously working interactive application designed to simulate complex dynamic systems must have a simulation monitor. To reduce the size of the monitoring application, a new method was created: it suggests decentralized processing of time delays and filtering immediately in the basic element models. This method is discussed later.

In order to eliminate the influence of the order of the component calculation on the simulation result, the existing systems support refreshing of the node values in the adding phase. To reduce the time losses on the operation in case, *Amethyst* offers a new, two-phase method of calculation of the component state: in the first phase the output values are calculated on the basis of the of internal variables, in the second phase the values of the internal variables are calculated on the basis of the input variables.

The Amethyst system contains a new method of analog objects simulation by means of interaction of the analog component autonomous models. The advantages and the details of implementing this method are discussed below.

3 Method of decentralized processing of signal delays and filtering

Most of the known simulation systems process the signal delays and short signal filtering in the central monitoring application. The monitor holds event lists and selects the component state to be calculated. The monitor's complexity cuts down the complexity of the models and impedes the production of autonomous simulating applications.

Amethyst uses a decentralized method of processing the signal delays and filtering immediately in the basic elements.

The decentralization helps save the resources of the simulating system monitor. It allows to building models of highest complexity (for example, microprocessor models), counting upon the fact that the arrangement normally has only one component of this complexity, and the total complexity of the component and the monitor does not exceed the limit.

The system in question introduces into the basic element models time variables t_z, to hold the time of the next event in the element model or its structural component.

The value of the time variables t_z is changed by operator of the type $t_z = T + d$, where T stands for system simulation time, d - time parameter, usually delay.

The simulation monitor of application, uses four global variables: present time T; starting time $T_{b,}$ highest possible time $T_{m,}$ and next event time T_e.

The Object simulation algorithm, realized in the application monitor, is shown on fig. 1

In the beginning of the algorithm the next event time T_e is ascribed the value of the simulation start time T_b. In the simulation cycle the present time T is first set as the next event time T_e, - that is equal to the "step in the simulation time". Then the next event time T_e is ascribed the value of the highest possible time T_m.

The procedure of calculating new state of each component of simulated object is divided into two procedures Type_of_component.f1() and Type_of_component.f2(). The Type_of_component.f1() procedure calculates the values of the output variables on the basis of the internal variables. The Type_of_component.f2()procedure calculates the values of internal variables on the basis of the input variables.

The f1() procedure, shown on fig.1 calls all the procedures of Type_of_component.f1() type (this is phase 1), procedure f2() calls all the procedures of Type_of_component.f2() type (this is phase 2).

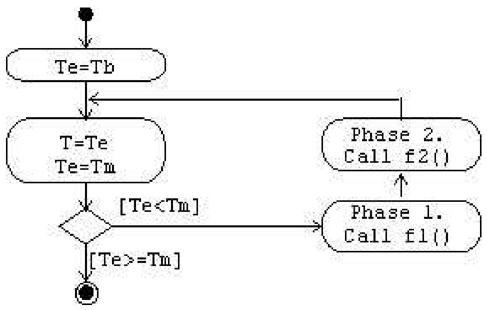

Figure 1. Object simulation monitor algorithm

If the component is a dynamic one, it has the t_z time of the next event in the component. The f1() or f2() procedure of each component compares the system time T_e of the next event with t_z, time of next event in the component. If T_e is more than t_z, operator $T_e = t_z$ is executed.

In the end of phase 2 the system time T_e is equal to the minimal of t_z in the components, i.e. is equal to the model next event time.

The other side of the considerable decrease in monitor complexity is the necessity to call the component models continuously. Similarly to working in Windows after the simulated application calls a procedure, none of the variables might be changed in this procedure, for example it can happen in phase 1, if the predicate ($T = t_z$) is not true.

The advantage of the algorithm proposed for object simulation is the possibility of setting zero delays. In this case the model sets its next event time and system next event time equal to the present time T value and makes the system repeat the calculation with the same present (model) time. This also happens during the analog model state calculation.

4 Analog elements simulation method

Most of the known analog object simulation systems create a general model of the object and process the overall system of differential equations. The model of each component sends its parameters to the monitor and awaits the result.

The Amethyst system uses an unusual method for analog objects simulation by means of active autonomous models of the analog components. The model of each component takes an active part in solving the implicit system of differential equations, describing the device. Each component model compares pairs of adjacent values of its output equivalent current sources and conductivities and in the case of unequal values asks for one more calculation of the component state at present time T.

The implicit system of differential equations is solved by Euler method using integration with variable step. The step is calculated in each component model on the basis of the current state and the forecast of such an output voltage change, that will not exceed the desired mistake.

The advantage of the autonomous model is that when the model participates in solving the system of equations, there arises a possibility of efficient correction of the model parameters, to increase the speed and precision of the calculation. The user can access the parameters of each element’s model, and change them even during simulation.

The model of an analog or complex scheme is made up from separate element models by simple indication of their being part of the global model and usual linking them to each other. Nothing prevents joining together the models of analog and digital components, since the behavior model of both analog and digital element has on inputs and outputs equivalent current sources, shunted by equivalent conductivities. Through these inputs / outputs, equivalent current sources and conductivities the model interacts with other models.

The advantage of the proposed method of simulating analog and complex units is a considerable simplification of simulation monitor; the monitor needs not keep the events list, since the event data is formed inside each component model, while the monitor gets ready-made value of the next event time.

This method of simulating analog elements is based on the node potential method. It uses a well-known equation – the voltage of a circuit node equals the total of the input currents divided by the total of their corresponding conductivities:

$$U_r = \Sigma J_i / \Sigma G_i \,,\ i=1, ..., K_r$$

where r - the node index; U_r - the node voltage; ΣJ_i - the sum of node currents; ΣG_i – the sum of node conductivities, K_r - number of joints.

Each model of the analog element is based on the presentation of this element in the form of N - polar unit (fig. 2), poles of which are connected to the scheme nodes. Each component is replaced by an aggregate of current sources J_{ek}, with conductivity G_{ek}, where k=*1,...,n*. That is a Norton equivalent circuit. The k - pole of the element is subject to the external equivalent current source J_{sk} with conductivity G_{sk} representing the total all node currents minus the J_{ek} current of the element in question. Likewise, the conductivity G_{sk} is the total of all the node conductivities minus G_{ek} conductivity of the element in question.

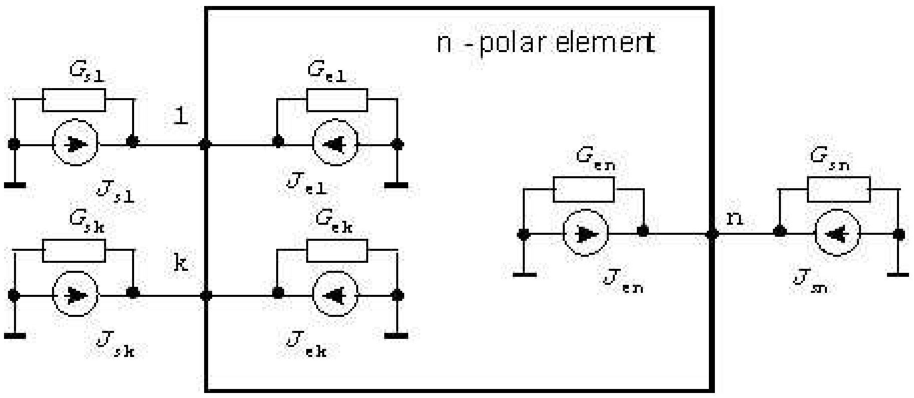

Figure 2. Equivalent scheme of a component with environment

The component model passes the external currents, connected to its poles and the respective conductivities to its own poles with due regard to its internal current sources and conductivities.

The calculation of currents and conductivities transmission is carried out according to the scheme of simulated component substitution. The substitution scheme is to contain simpler elements than the simulated component and reflect with desired precision the processes in the component.

The substitution scheme gives base for writing the formulas for the external currents and conductivities passed through this element. On this step it is most convenient to use the superposition method and Thevenin's theorem. The number of formula pairs for the passed currents and conductivities is equal to the number of external nodes.

The idea of model autonomy is based on the fact that after reiteration of equivalent current sources J_{ek} and conductivities G_{ek} calculation the model and the other ("adjacent") models come to balance.

The node potential method can be applied only linear circuits. A linear approximation is required to use substitution scheme with nonlinear constituents, such as analog and digital-analog elements.

For the substitution schemes with nonlinear elements the used method of scheme calculation - approaching to balance - can give a divergence of calculating process. The matter is that if the inclination coefficients of linearly approximated model parameter change from one calculating to another, it can lead to oscillatory process, i.e. a divergence process.

For the process convergence it is necessary, that the incline coefficients of linearly approximated parameter of the model do not change during balance reiterations. It means that the linear approximation of parameters must be done once only time before the reiteration at present time.

To solve the problem of iteration convergence we suggest:

- using section overlapping during piecewise approximation, in order to use the parameters of current section of approximation as long as possible;
- shunting of nonlinear elements by capacitor;
- limitation of iterations number.

As a result of the linear approximation of nonlinear elements the simulated component is replaced with an equivalent scheme with current sources, voltage sources, resistors, conductivities, capacitors, inductances.

Equivalent schemes of analog components frequently have capacitors, and there is a need to integrate the capacitor's charge as the simulation time advances.

Integration with variable step involves piecewise approximation of capacitor's change depending on time. In the beginning of model state calculation the capacitor tension is modified on the value of an accumulated increment, and during the calculation the voltage on the capacitor is considered invariable.

It is worthwhile to link the simulation step to the mistake of simulation set in the system. In case of linear approximation the change ΔU_c of capacitor's voltage is determined by its capacity C, current i_c through capacitor and time increment Δt:

$$\Delta U_c = (i_c / C) * \Delta t$$

If the determined mistake ΔU of simulation equals the change ΔU_c of capacitor's voltage, the formulas for "local" step of simulation go as follows:

$$\Delta t = \begin{cases} T_m - T & \text{if } |\, i_c \,| < (C * \Delta U\,) / (\,T_m - T\,), \\ (\,C * \Delta U\,) / |\, i_c \,| & \text{otherwise.} \end{cases}$$

The first formula is the case of such a small current through the capacitor, that the change of capacitor's voltage does not exceed the determined mistake till the end of simulation.

5 Creation of application

The project of application is created on the basis of a functional diagram (scheme) of the device. The minuteness of the scheme should correspond to the existing library of components, which means that the scheme component must be a library component.

Let us consider application projects in C++ Builder and Visual Basic 2005 environments.

The directory structures, produced by Amethyst in each environment, have a similar structure. On the top of the hierarchy there is a directory called „Project.APR", where „Project" is the name, selected by the user.

„Project APR" is a subdirectory of root Amethyst directory. „Project APR" directory contains main files of the project and subdirectory named „Project.MOD". This subdirectory contains files with standard description of the scheme, similar to „Project.MOD" subdirectories with component description. The „Project.MOD" contains subdirectories of the„Component.MOD" type for each kind of components. The „Component.MOD" subdirectories in directory „Project.MOD" are copies of „Component.MOD" subdirectories from the main Amethyst directory.

The main Amethyst directory is used to hold all library components. A simple library component – atom – occupies a "Component.MOD" directory, but a compound library component – scheme model – occupies "Component.MOD" directory inside the "Project.APR" directory. The Amethyst application shows directory hierarchy of library components in the tree form. The main Amethyst's directory contains "_amethyst" subdirectory with system libraries and files.

A standard description of a component includes the following elements:

- text file with parameters and a template of presentation window (hypertext form),
- program files in a high-level language, for example C++ or Visual Basic 2005, for simulating the component's function.

Component's functions are described in the form of a class. A class contains not only functions for the calculation of the component's new state, but also methods for showing in the corresponding windows the component state, model properties, setting parameters, values of the variables to be shown. The windows with information on the model state are called presentation windows. For showing the actual values mainly files with extension “.ame” are used.

An application project, produced in Borland C++ Builder environment, contains the following files and directories: Builder project file („Project.bpr”), resource file („Project.rc”), implementation file („Project.cpp”), header file („Project.h”), file with Amethyst’ settings („Project.pra”), implementation and header files of the unit, which joins the universal simulation monitor with the model („unitMod.cpp” and „unitMod.h”), and the directory with the scheme model („Project.MOD”).

The project uses also files and libraries from system the system library „_amesyst”. The units „_amemain” (application main windows), „_ameimit” (simulation thread), „amestat” (procedures to show component states) and „baseUnit” (base class for library components) are included in the uncompiled form for the program debugging. The project also uses seven libraries with system procedures (service, fonts, etc.).

The structure of the files and directories in Visual Basic 2005 environment is similar. The „Project.APR” directory contains: Visual Basic 2005’s project file („_Project.vbproj”), file with Amethyst’s settings („_Project.pra”), unit file, which joins universal simulation monitor with model („_unit.Mod.vb”), and the directory with the scheme model („Project.MOD”).

The following items are added from system directory „amesyst” to the project files: resource file („Form1.resx”), main formant file („Form1.Designer.vb”), main implementation file („Form1.vb”), system procedures („amesyst.vb”), simulation thread („_ameinit.vb”), procedures to show the component states („amestat.vb”), system fonts („amefont.vb”), base class for library components („base Unit.vb”).

The „_unitMod” unit plays a significant part in each environment. It connects universal simulation monitor with the produced model. The „_unitMod” unit adds a buffer component, which translates simulation commands from the main application window to the model procedures. The simulation monitor uses a constant name for this buffer component, and the latter uses name of the model, given by the user. So there is no need to produce a new simulation monitor for each project.

File fragment edition method is used for the production of project files. Each of the project file fragment is converted into a format, which is the base for writing in the actual component or scheme parameters. Files are divided into parts, because there is a limit on the buffer size in the formatting procedures.

The procedures for creating project files are gathered in DLL (Dynamic Linking Library) to increase Amethyst’s flexibility.

If there is a need to create a project for a new environment, the following method can be used. The first step is to create two – three projects with different level of complexity. The second - to analyze all the files produced by system on the “text” level and mark out the “variable” parts, i.e. parts, depending on the input data.

The applications, produced in Borland C++ Builder environment, work considerably slower than those, produced in Visual Basic 2005 environment. The test showed, that the factor of slowing in Borland C++ Builder environment was 4.5. The test process with 200 000 events took 93 s in Borland C++ Builder environment and 21 s in Visual Basic 2005 environment (PC with 2.53 GHz processor). However, the parallel simulation thread was not working in Visual Basic 2005, due to a simple reason – the *Thread* class required in this case has insufficient functional capabilities in this environment.

6 Conclusion

The computer program for quick production of the autonomously working interactive applications simulating complicated dynamic systems allows to building and testing models of different systems. The “Amethyst” program system, using the described above base solutions, has following advantages:

- wide range of component complexity (from capacitor to microprocessor),
- independence of models and, consequently, no limitations on the complexity and “intellectuality” of component models,
- absence of limitations on component type (digital or analog) and on the running order of digital and analog components,
- convenient form of the component state output, including contents of internal registers and memories,
- possibility to change registers and memories contents quickly,
- possibility to control the time parameters between any signals,
- possibility to set the time parameters quickly,
- possibility to simulate using real blocks, which allows half-location experiment.

References

[1] Timofeev A.O. (2004): Computer production of programs for simulation of dynamic systems. Proceedings of the 15th International Conference on Systems Design (7-10 September 2004, Wroclaw, Poland). Vol. 2. Wrocław, Oficyna wydawnicza Politechniki Wrocławskiej, 2004. pp. 91-95.

[2] Dropia R., Czeberkus P., Timofiejew A. (2005): Koncepcja programowego systemu do tworzenia interaktywnych symulacyjnych aplikacji „Studia informatica”, N1(5), 2005. pp. 49-57.