Distributed Development of Embedded Systems

Prof. Dr. Wolfgang P. Kowalk
Department for Computer Networks and Telecommunication
Carl-von-Ossietzky-University of Oldenburg
Escherweg 2, 26121 Oldenburg
OFFIS e.V. and IRT GmbH, Oldenburg
Germany
Email: kowalk@offis.uni-oldenburg.de

KEYWORDS

Rapid Prototyping. Synthetic Environments. Multi-disciplined Team-Working. Virtual Models and Products. High Level Modeling and Design.

ABSTRACT

In this report we present a method to develop embedded systems by assistance of a new simulation system, ***Lancelot***. This system has been designed to sustain graphical, animated simulation in a process oriented manner. It can be used for discrete, continuos or hybrid simulation, so that, besides others, embedded control systems can be implemented and analyzed by Lancelot.

1 INTRODUCTION

An embedded control system is a system that controls execution of real world systems like anti-blocking systems for car breaks, autopilots for airplanes, or heat control for technical processes or living rooms. These types of systems are to be controlled, for which today typically computerized control equipment is utilized.

To construct embedded control systems, knowledge from both, computational science as well as control engineering is required. Currently, computer science supplies the engineer with new methods to control systems, particularly fuzzy control. Here, control is no longer constructed by mathematical functions of control variables and time delays, but by some „syntactical“ descriptions of more or less discrete control values and corresponding changes of the state of a controlling effect. Also, the state of the controlled quantity is no longer measured by an accurate quantity, but by a „fuzzy“ value, which has not any more a unique outcome.

From these considerations follows, that several methods can be supplied, if control equipment is to be constructed. It is still no longer obvious, which technique is the best one to solve a given problem. Thus careful decisions have to be made, before expensive physical experiments are to be performed. We suggest to use simulation experiments to do this task.

Simulation can be considered to be a future oriented technique for modeling real word phenomena. To supply control engineers with efficient modeling paradigms, it is necessary to sustain graphical representation of the models, besides numerical highly precise simulation methods. Also, different modeling techniques should be available in the same system, to assist the work of engineers in different applications.

If control systems are to be constructed, the usual way is to model both, the controlled real world system as well as the controller. Both areas require in general dedicated expert knowledge, so that usually different engineers will be involved in these tasks. Of course, control circuits consists of quire more; at least a measurement instance is required. It is therefore necessary to combine several different models into a single one. This implies distributed development of control systems.

Lancelot has been created to fulfill such requirements. It runs on a graphical interface in X, uses C-syntax as programming language, and sustains efficient management of complex model libraries. Thus, while an engineer develops a physical system that is to be controlled, another one can construct the control unit. Both models can simply be combined via the model libraries and being analyzed by a third engineer. If the control system fails, the control engineer can change the controller, which can immediately be inserted into the composed model. Of course, the controller might also be changed or adjusted in the model, so that interactive development becomes possible.

Thus all requirements for distributed development of control systems are supplied. If the model basis is situated in a common file structure, accessible by all engineers from their workstation, they can exchange their models quickly and with no delay. Since ***Lancelot*** can be executed on a platform different from where its graphical interfaces are displayed, it is also possible to share the same sight of a model at different locations, which sustains distributed development as well.

This paper describes development of a (very simple) control unit for a simple oscillating system by example. This demonstration is used to display the features of ***Lancelot*** for developing of embedded control systems.

2 PROGRAMMING LOGIC OF C-SIM

The programming language ***Lancelot*** is based on a C-based simulation language, called C-Sim (this name was used by some other C- or C++-based simulation systems, which we will not consider here).

The logic of C-Sim can be described to be

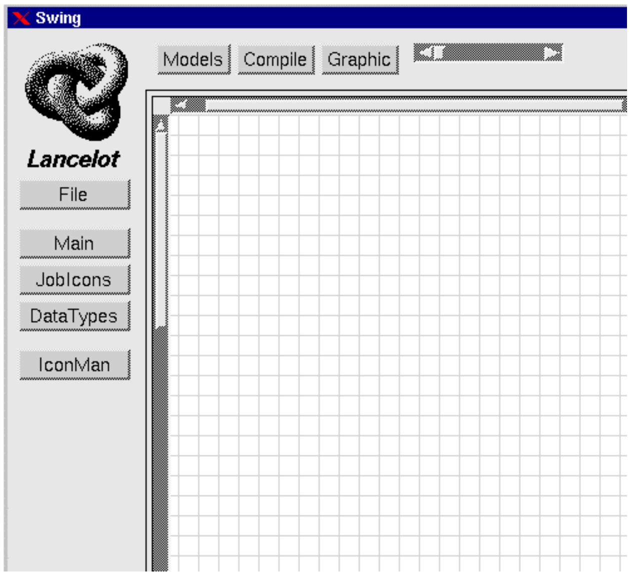
1. process-oriented (similar to Simula 67)
2. event-oriented (similar to SimScript)
3. asynchronous or synchronous message handling
4. complex model management system
5. structuring complex models by a hierarchical technique

Since we will explain some of these concepts in the next sections, we won't elaborate further on them. However, while C-Sim can be applied to state discrete systems as well as time discrete systems, our example will only discuss the second application.

3 SOLVING A CONTROL PROBLEM

We consider the following simple control problem to demonstrate the use of Lancelot as distributed development tool.

An oscillating system, like a spring, with a given mass M and Hook constant c, so that retaining force F(s) depends on the way s from the origin; s can become positive or negative. To model this „real world system“ with Lancelot, one has to start the program, so that the following window opens, in which we have to fill in some objects.



Picture 3-1: The interface of Lancelot

Now we have to open a „basic object“ from the TypeManager (see section 4.1), which looks like Picture 3-2. Just opened, this window is of course „empty“, while we have already inserted the program text.

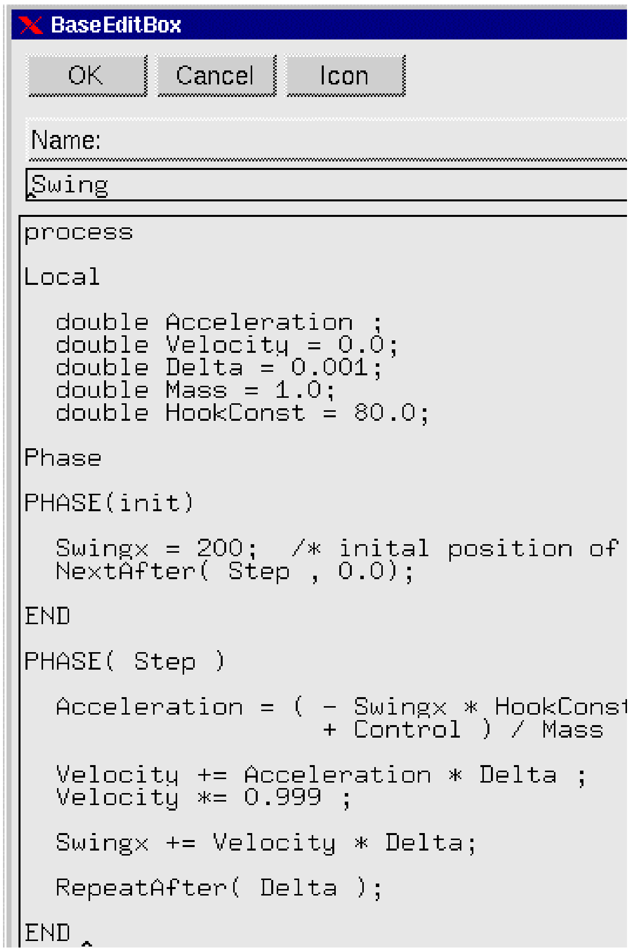
Programs are written in the C programming language. A basic object is usually a processes, which consists of a name (`Swing` in the example), a declaration part (introduced by the keyword `process`), and some sections (each called `PHASE`), where activities of the process are specified, starting with an initial phase, followed by further phases, each identified by a unique name.

The program written in this example solves the problem by numerical approximation, which yields a faithful equivalent to the real behavior of the system. However, if higher accuracy is required, some additional support is given by Lancelot, we will not discus here in detail.

If in the interface of ***Lancelot*** the `Main`-button is pressed, the central control program of this simulation study is opened. It simply consists of some statements to control simulation time and might look like

```
main()
double Swingx;
SimulationTime = 100;
```

It would be useful to observe the behavior of this system. This can be done by including an output process, which can for example be loaded from the library, explained in more detail in 4.1. This would open a further window, which shows the value of the quantity `Swingx`, depending on advance of time. Picture 3-3 shows this result, which helps to understand the behavior of that system much better.

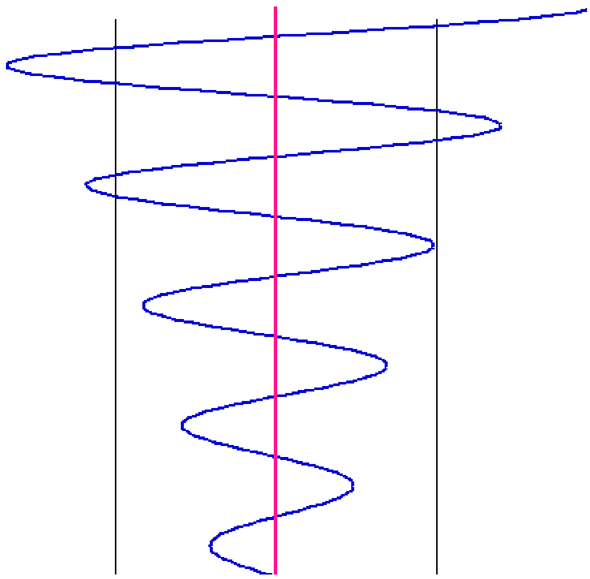


Picture 3-2: A basic object window to model processes

Now, it is the task of the control engineer to model a controller. This can be done in several ways. A very easy way is to use a linear equation of location `Swingx` and the velocity of the mass at the corresponding location. This might yield a formula, which looks like

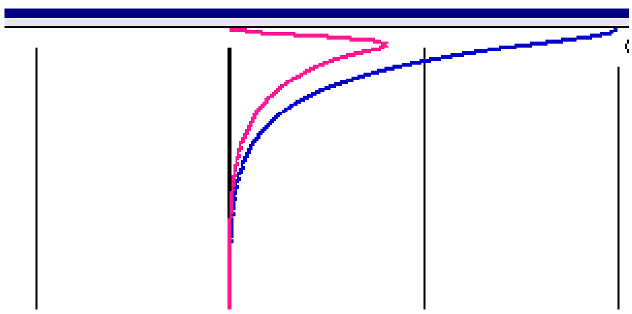
```
Control = Velocity*30 - Swingx*20;
```

which is adequately simple; since it is not our aim to improve control theory, we are supposed to be satisfied by this solution to the problem in question. Applying this to the variable `Control`, which influence the force on the spring, results in a final outcome of the behavior of the spring, that looks like



Picture 3-3: Output of a demonstration process

what is perhaps a sufficient solution for the problem in question (the lower curve displays the velocity of the body). Otherwise, parameters might be varied or another control method might be tested, until ample results are gained.



Picture 3-4: Controlled Spring

4 PROPERTIES OF *LANCELOT*

The programming language ***Lancelot*** is based on the C-based simulation language C-Sim. To sustain the task demonstrated above, some requirements of ***Lancelot*** can be derived.

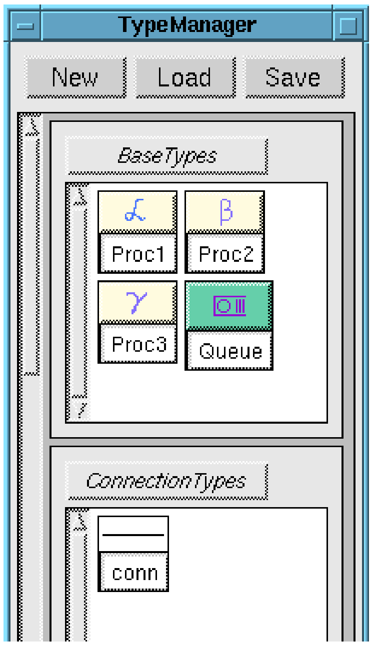
1. Processes should be structured, providing local environments for each process.
2. Global environments are used to model relationships between processes.
3. A hierarchical concept is introduced to structure several processes into one object.
4. Effective model management from the programmers interface sustains reusability of models.
5. Model checking is sustained by automatically animated state transitions.
6. Simulation results are presented under the same graphical environment as the simulation programs.
7. Additional features for presentation of continuos models are provided.

4.1 *Lancelot*'s file management

To sustain distributed development of models, Lancelot provides an efficient file management facility. It is controlled via the **TypeManager**, which resides in its own window.

The TypeManager holds any number of objects, which can be of very different kind. They can be basic objects, which we have just used to demonstrate construction of control units, but they also enclose connection objects, which can be used to build relationships between processes, as well as some type objects for different environments or message objects.

The TypeManager also can hold some complex objects, which is the most advanced feature of ***Lancelot*** to structuring complex models. Any collection of processes can become a complex object; complex objects can even enclose other complex objects, so that a hierarchy of processes can be created and managed in a very efficient way. This concept shows the real power of ***Lancelot***, since each model can become an instance as often as required. Thus complex objects can become a very smart concept for complex models to be developed, demonstrated and executed.



Picture 4-1: Management of simulation objects

5 OUTPUT ANALYSIS WITH *LANCELOT*

One of the main features of ***Lancelot*** is its output analysis. In many simulation systems, this has to be sustained by manually inserting function calls, which are only difficult to remove, if the simulation program is to run efficiently. Control of this has been automated by ***Lancelot*** by several methods.

If a program runs under control of ***Lancelot***, the current execution of a program can be observed with several methods. The first one is the coloring of state changes, highlighting of the process in execution, and animation of message transfer.

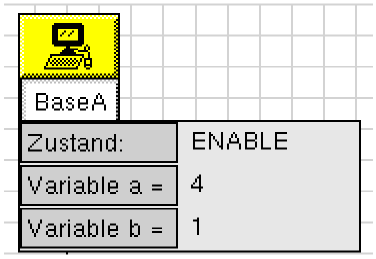
The second visualization method is to output sequences of data, called time series, to observe averages or variation of quantities like sojourn times, number of jobs in queues, etc. The third method to visualize output results is to sample data for distributions, plotting them in diagrams, where by some parameters the plotting rate as well as the type of displayed distribution can be controlled. Since these methods are central features to sustain efficient and structured modeling, they will be explained in more detail in the next sections.

5.1 Display of states and transitions

During simulation execution, ***Lancelot*** helps the programmer to observe the performed program by some special features.

If a process is executed, it can step through several phases. It is useful to know, which phase a process is just performing. This can be displayed by a special color the process' icon is changed to. Thus the programmer can proceed the sequence of phases the process executes.

Because of the quasi-parallel executing of the processes, exactly one process is active at any time. ***Lancelot*** highlights this process by a special color, selected in the icon formula. Thus, the sequence in which the processes of a model are activated can be observed by the programmer as well.



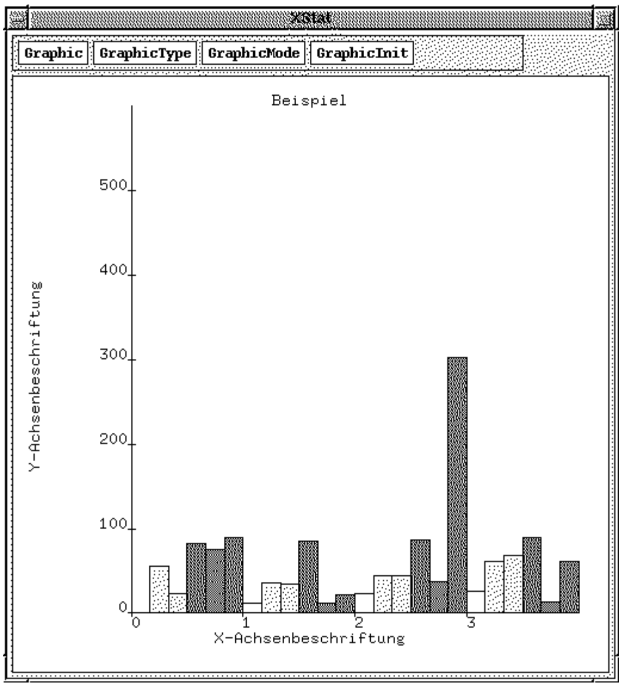
Picture 5-1: Display of process variables

When a process sends a job to another process, using the `Send`-statement, this activity is animated by a moving icon, pursuing the connection between these two processes. Thus the programmer can follow explicitly all paths a job is sent through the model. This helps to detect errors like mistaken process names or incorrect activation rates.

Further features can be implemented to control the state of a process. If some quantities or the state of a process are to be exhibited, a display function called „buffer“ can be used. The defined values are displayed only, if the process' icon is clicked by the right mouse button, see Picture 5-1.

5.2 Statistical output of time sequences

If some state variables are changing permanently, it is more useful to plot a graph over time of them than to display their actual value. ***Lancelot*** provides a time series feature that is comfortable to use and to adopt to different applications.



Picture 5-2: Plotting time sequences in ***Lancelot***

This Picture shows a possible output of a time series. Here, an array of quantities is sent to the corresponding system function, which plots this graph during simulation run. The programmer can follow this values, recognizing immediately utilization and other performance data of his simulation program. Besides histograms, also sequences of dots, linked dots, numbered histograms, pie diagrams etc. can be depicted. This can be dynamically, i.e. during output the presentation of the data can be altered in very many ways, assisting the analyst to plot its results in the best possible way. Also captions can be plotted, displaying the results in different colors and dots. Other features of the time sequence diagrams allow the programmer to change the number of displayed values, the displayed range, and explicit representation of the values of the x-axis (to show the corresponding time, for example).

5.3 Statistical analyzes of simulation results

Besides on-the-fly analyzes of performance information, analysts are interested in final results of a simulation run. For this several statistical function are provided by ***Lancelot***.

The simplest functionality computes moments of a distribution. Here, first and second moment, as well as mean, variance, squared coefficient of variation, and confidence intervals of quantities like series of values (e.g. service times) or functions over the time (e.g. number of jobs occupying a queue) are estimated. Some advanced statistical concepts like confidence interval estimation by Student's t-distribution are also provided.

Another simple functionality provides the station concept. Here, a station is defined, where entrance or departure are determined by some function calls. Thus, any collection of processes can be defined to be a „station", where data like throughput, utilization, averages of number of jobs etc. can be estimated in a simple and secure way.

To get more information than only moments or variances, one needs to determine distributions of some quantities. ***Lancelot*** sustains these requirements by an improved statistical method. Here, data of a quantity can be sampled with some statistic functions, where density, distribution, or other quantities are estimated and confidence intervals are generated automatically. These results are plotted graphically, and can be stored numerically in files and maintained for further analyses.

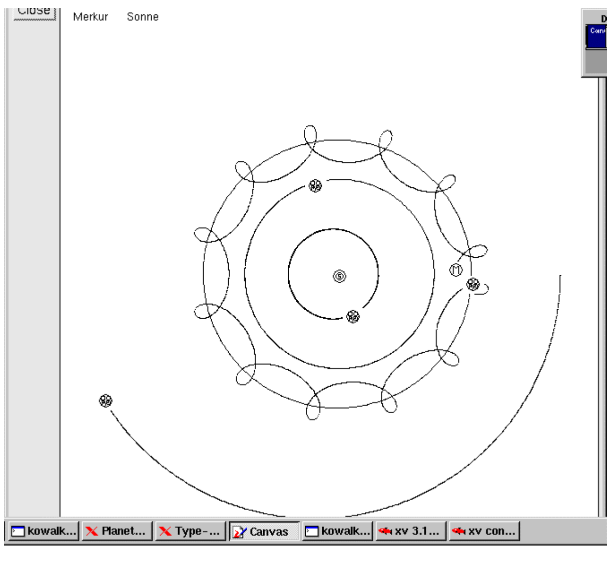
Besides this, also some additional functionality for data analyses can be implemented by the programmer or enclosed into the system in further releases of ***Lancelot***.

5.4 Animating continuos simulation models

C-Sim is based on an event oriented simulation concept, which means, that change of states occur never continuously in time. Even systems with usually continuos state changes can often be modeled in this way. However, if this is not possible, ***Lancelot*** provides some features to sustain also these requirements.

The simplest way to implement continuos models is to approximate continuos flow of time by discrete time steps of sufficiently small sizes. This method has some advantages (it is simple), but also some disadvantages, since it is not always very accurate. In spite of this, we use this method in ***Lancelot***, improving results by some refined highly precise approximation techniques, which we cannot present in detail at this place [Kowalk96]. Now we need to present the behavior of such continuos systems by some graphical technique. For this, an additional graphical interface can be opened (called a *canvas*), in which lines, circles, etc. can be plotted. Also the movement of objects can be displayed, providing the observer a realistic view of the behavior of some objects.

An application of this method has been used to implement the movement of planets around the sun. Each planet was modeled by Galilei's laws of movement, using Newton's gravitational law to estimate acceleration. The resulting picture is a demonstration of planet movement. Picture 5-3 shows us an intermediate state, where the Moon's distance to Earth is increased by a factor of fifty.



Picture 5-3: Movement of the inner planets (including Earth's moon) around the sun

Other applications of these methods are used to model embedded control systems for oscillating bodies or studying fuzzy control logic. The latter is a very well established technique to implement control systems by a discrete concept, which is much more adequate for computer controlled systems.

6 CONCLUSIONS

Lancelot has been developed to model real time systems of a wide range of application, from time discrete queuing models to time continuos systems like movement of planets and embedded systems, where both, the real world model as well as the control system can be implemented with the same interface, simplifying modeling and evaluating of real world systems. *Lancelot*'s uncomplicated graphical interface helps to model faithfully and implement quickly complex systems, demonstrate their behavior and analyze them efficiently.

Lancelot's features are useful for many further applications. They can also be considered to supply a process oriented programming paradigm, where programming logic is no longer provided by a central control programming (the main program's control flow) and a set of functions or (arbitrarily abstract) data structures, but by concurrently working processes, changing their states almost independently, interfering only by exchange of messages or change of values of common global address space.

7 ACKNOWLEDGMENTS

I am grateful acknowledged to the student, who have implemented the most of the interface (Torsten Sterzenbach) and the C-Sim compiler (Gerd Block). Most tests have been performed by Marcelo J. Drumond Lanza, who also gave many useful indications for improvements of usuability and efficiency.

Thanks also to OFFIS e.V. for funding some of the work on the compiler, and IRT GmbH for funding the work on the interface.

8 BIOGRAPHY

The author, Wolfgang P. Kowalk, born in 1950 in Hamburg, Germany, studied computer science at the university of Hamburg, Germany. After achieving his Ph.D. in 1982 and habilitation in 1986, he worked for some years at an European electronic company, researching in ATM communication systems. Since 1990 he is a professor at the university of Oldenburg, leading a department for computer networks and telecommunications. His main research interests have been: computer networks, network and system management, computer network security; modeling, performance evaluation, simulation; programming languages; temporal logic.

Mr. Kowalk is also an active member of the research institute OFFIS e.V. and he is leading a consulting company for computer networks and systems management, IRT GmbH, Oldenburg.

9 LITERATURE

J.Banks, J.S.Carson, and B.L.Nelson, 1996: „Discrete Event System Simulation." Prentice Hall, Englewood Cliffs, N.J.

B.S.Bennett, 1995: „Simulation Fundamentals." Prentice Hall, Englewood Cliffs, N.J.

O.Föllinger, 1987: „Nichtlineare Regelung." Oldenbourg Verlag, München.

P.G. Harrison, and N.M.Patel, 1993: „Performance Modelling of Communication Networks and Computer Architectures." Addison-Wesley, Reading, Massachusetts.

G.Jaanineh, and M.Maijohann, 1996: „Fuzzy-Logik und Fuzzy-Control." Vogel, Würzburg.

W.P.Kowalk, 1992: „Das Simulationssystem C-Sim." Internal report, University of Oldenburg, Oldenburg.

W.P.Kowalk, 1993: „Simulation in C-Sim. Principles of modeling, simulation techniques, and simulation tools." Internal report, University of Oldenburg, Oldenburg.

W.P.Kowalk: „High-Precision simulation technique" Internal report of the University of Oldenburg (1996).

A.M.Law, and W.D.Kelton, 1991: „Simulation, Modeling & Analysis." McGraw-Hill, New York, N.Y.

T. Sterzenbach, and G. Block, 1996: „Das Simulationssystem ***Lancelot.***" Internal report, University of Oldenburg, Oldenburg.

D.Traeger, 1993: „Einführung in die Fuzzy-Logik". Teubner, Stuttgart.