Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

SIMSOLUTION - AN OPEN SIMULATION ENVIRONMENT FOUNDED ON EXTREME MULTITASKING

Thomas Wiedemann

University of Applied Science Dresden
Friedrich List Platz 1
Dresden, 01069, GERMANY

ABSTRACT

There is no universal standard for discrete simulation. Models, created with leading simulation tools can not be exchanged between the systems. In result, there are very high investments and maintenance costs for simulation studies and some additional problems with portability and performance in large simulation studies. This paper discusses in detail, a special approach by using an assembler based, very fast multitasking routine combined with efficient discrete event scheduling algorithms. The basic system approach is realized with Standard C/C++ and Delphi-compilers and offers an unlimited flexibility and very good runtime performance. Language independent, XML-based code generators convert simulation models between different run-time platforms without manual changes.

1 INTRODUCTION

The main algorithms and mathematical foundations of simulation systems are well defined and efficient (Wiedewitsch and Heusmann 1995). Nevertheless, the real application of simulation systems is still difficult (Kuljis and Paul 2000). Not more than 10% of all industrial firms use simulation tools by a number of reasons:

- Unlike the continuous simulation marketplace there is no leading discrete simulation system. The market shares of the existing tools (AutoMOD, TAYLOR ED, Arena, SLX) are very different in the global regions and industrial branches. As a result, there is no universal standard for discrete simulation. Models can not be exchanged between the systems.
- As a result of the small market, the prices of the systems are very high. Typical prices of more than $50,000 are too high for medium-sized firms.
- Especially in the area of optimization with simulation models exists a performance problem. It seems like a paradox, that an older simulation language like GPSS is significantly faster than modern simulation systems.

These problems indicate the need of a new strategy for the development of simulation tools. Like in the database software domain, **we need powerful standards for modeling and simulation.** A first step will be the application of Open-Source-ideas, which was very effective and successful in the LINUX-development. The main advantage of Open-source software consists in free access to all parts of a software, which gives a high degree of flexibility of such a system. First Open-Source simulation systems with interesting concepts were presented, like DSO (Jacobs 2004) or SILK (Kilgore 1998).

The main ideas of Open Source and the advantages for simulation tools are discussed in detail by Kilgore in the original paper outlining the OpenSML-project during the Winter Simulation Conference 2001 (Kilgore 2001).

2 THE MAIN PROBLEMS IN SIMULATION

The actual problems of simulation are already 30 years old. They are based on the basic principles of simulation, which are explained in detail in other sessions of the conference (see former "How it works" sessions at the WSC, e.g. (Schriber 2003)) In short terms, modeling and simulation of the real world systems requires a parallel execution of a large number of processes in a specific order. This task is solved by all simulation systems. But the absolute speed of calculation often decreases with each new system. It is useful to discuss some details, although some simulation experts do not spent much attention to this question.

Process switching is the first task of parallel execution. The executing processor must switch from one process to an other process by preserving all states for the future re-switching. Often there are thousands of small processes with a high switching rate. Some operations are also conditionally. Switching inside basic functions is called co-routine switching. After a first realization in SIMULA such switching technologies were not integrated in C / C++ or similar languages. Other technologies, like pointer based functions calls and multi-threading are too slow and too complicated.

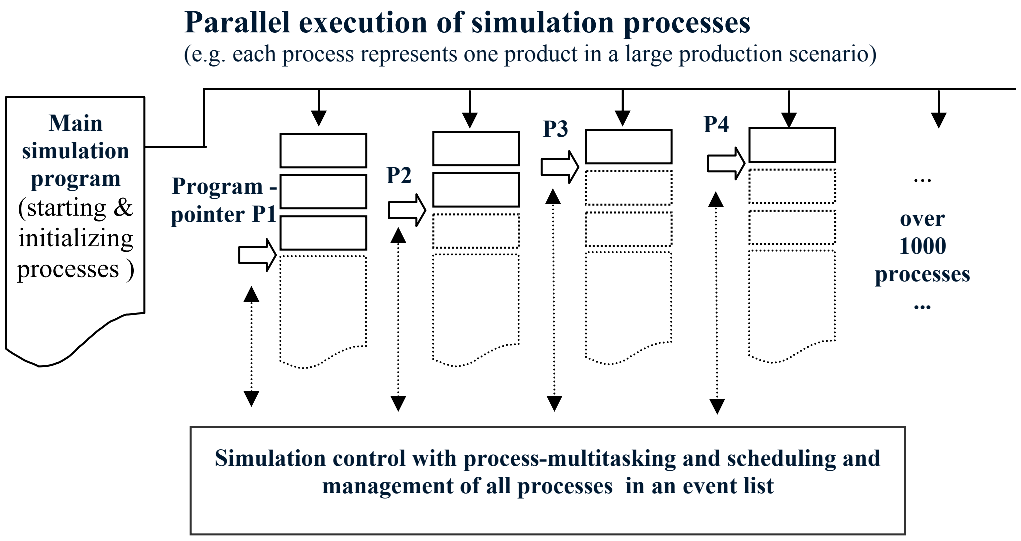
Parallel execution of simulation processes
(e.g. each process represents one product in a large production scenario)
Main simulation program (starting & initializing processes)
Program - pointer P1
P2
P3
P4
...
over 1000 processes
...
Simulation control with process-multitasking and scheduling and management of all processes in an event list

Figure 1 : Parallel Execution and Switching of Simulation Processes

Process scheduling is the second task. The sequence of the process switching must be determined by the simulation control unit ! This is uncritical, if the schedule is simply determined by time or priority. It is critical, if the scheduling order depends on conditions, like blocking states in sequential organized queues.

Performance problems with simulation systems are often based on bad or non adequate switching and scheduling algorithms:

- Using standard multitasking algorithms from C/C++ or Delphi libraries are critical, because they are designed for switching a small number of large processes. Often, the maximum number of threads is limited and the scheduling order can not be changed by the developer.
- The number of threads inside Java is limited and the performance of switching a large number of threads is critical (see Kilgore 2001).

Between switching and scheduling are main differences: process switching is a quite simple task and defines the main performance, process scheduling is quite complex, and less critical in performance. Although it seems possible to develop a very efficient switching implementation, it is nearly impossible to develop a optimal scheduling algorithm for all applications, because there are dozens of Scheduling algorithms on trees, sorted lists etc.

From this view, a main design decision was made: The **switching should be separated from scheduling** by using an open and flexible interface, which allows the simulation model builder a free choice of possible switching and scheduling modules.

Because of the fact, that nearly all existing computers are based on sequential (non-parallel) processors, the switching will always change from the actual to the next process. If the scheduler has determined the next process, the switching will need only the information of the actual and next process by using the following interface (see Figure 2).

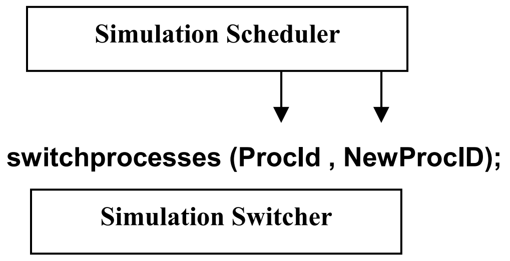

Figure 2 : Separation of Switching and Scheduling

This simple interface allows a wide spectrum of different switching and scheduling algorithms. The following pages will present some first implementations.

3 SWITCHING BY EXTREME MULITASKING

3.1 Options for Switching Processes

The switching algorithm must save all local variables and the state of the processor of the current process, then he should load the new program and stack pointer address and must restore the processors register and local variables of the new process. Traditionally, the saving and restoring of the local variables is done by copying all memory blocks to backup areas, which is very time consuming.

Because of the fact, that in standard programming languages like C++ or Delphi all local variables are located on the stack, it seems possible to **switch all local data and the return address for the new process by only changing the actual stack pointer address**. This simple change of the Stack pointer value reduces the time for process switching significantly and allows very high rates of process multitasking. Otherwise there are some critical points of this approach:

- The change of the Stack context is non trivial, because all local variables of all calling functions are switched off. In result, this method requires some special initialization of the stack during the start of each process.
- In general, the stack must provide memory space for an unknown number of functions calls. The size of stack space in standard implementations is between 16 Kbytes up to 64 Kbytes. The real used space is very different – efficient simulation functions need only some Hundred bytes of stack space, but Windows functions often require dozen Kilobytes of stack space. If any simulation process would use 64 Kilobytes of stack space, there would not be enough memory in the computer. For this reason the stack space is limited to 500 … 2000 Bytes per simulation process. If any simulation function calls an expansive Windows function, this call is mapped to a larger stack space.
- Changing the stack pointer address could be dangerous for complex programming environments. The approach must be tested with each compiler and new version for avoiding stability problems.

In conclusion, the switching of processes by only changing the stack pointer is simple and very fast., but it has also some smaller disadvantages. For this reason, the attribute "extreme multitasking" is used to inform potential users about this specific approach.

3.2 Implementation Results

The approach was tested by using DELPHI with the Object Pascal language. The stack pointer addresses are moved by assembler commands (see lines 7 – 10 of fig. 3) to and from a process address table. The push and pop commands save and restore the processor registers to the stack before switching. The number of POP/PUSH-operations depends on the specific processor and can change for other versions of compilers and languages.

```
procedure switchprocesses(OldProcId: integer;
NewProcID:integer );
begin asm  push eax  // save calling environment
           push ebx
           push ecx
           push edi
           mov stackold,esp; end; // store old STACKP
        stacknew := cal[NewProcID];
        cal[OldProcId]:= stackold;
        asm mov  esp,stacknew; // get new STACKP
           pop edi
           pop ecx
           pop ebx
           pop eax  // get old environment
        end;
end; //AT THIS POINT THE SWITCHING HAPPENS !
```

Figure 3 : The Code of the Process Switching Module

Because of the fact, that there was no secure information about the possibility of changing the whole stack context by such a direct way, the author was impressed by the fact, that this code is also Debugger-safe. So if any application developer uses this code, he can still see all steps in stepwise execution: The old process enters this code sequence and after ending the switching code with the *end;* - statement (which is in practice a RETURN-assembler statement), the high level code–pointer will continue with the new process.

The necessary memory for this approach is simply the size of the stack of each process multiplied by the maximum number of processes. With a stack size of 2 Kilobytes about 500 processes are possible per Mbyte memory. If there are 100 Mbytes free memory, it allows 50.000 processes, which is a good value also for large models. If this size is too small, the simulation user should spend 100$ for an extra 1 Gigabyte RAM Memory.

In conclusion, we **PAY PERFORMANCE WITH MEMORY**, which is actually a cheap option !

4 FLEXIBLE SCHEDULING

4.1 Options for Scheduling

As defined by the interface (see fig. 2), the scheduler must "only" select the next process for execution. This selection should be very fast for large numbers of processes and without long calculation times for inserting and deleting processes from the selection table. The kind of selection of course depends from the kind of simulation. In result, there will be different scheduling options for different simulation types.

4.1.1 Simple Sequential Scheduler

A simple sequential scheduler selects all processes one by one in the table and activates them. This kind of scheduler is only useful, if nearly all processes are executed in a strong periodic way. Related simulation models are used in traffic simulations, where all simulations objects (like cars or humans) are moving with small steps in every time step of simulation. The disadvantage of this scheduler is the bad performance in systems with very different activation rates. The implementation of such a sequential scheduler is simple (see Figure 4).

```
function scheduler_enumall( );
begin
   newsimobid := actsimobid +1;  // processes counter
   if   newsimobid> SimobCount then
            newsimobid :=1;
     // check for inactive process
    if sobs[newsimobid].State <> Active then exit;
    actsimobid := newsimobid; // get new process ID
    actsimob   :=sobs[actsimobid];
    // now switch from MAIN to next process
    switchprocesses(0,newsimobid);
end;
```

Figure 4 : The Code of the Simple Sequential Scheduler

Together with the switching module this scheduler allows a first test scenario for building up a simulation model. The resulting time for one whole cycle, measured over 1 Million switching / scheduling sequences was about 13 – 17 Nano-seconds on a 1,3 GHz Centrino PC and less than 10 Nano-seconds on a 2,5 GHz Desktop PC´s. In fact, that this time corresponds to about 30 basic assembler operations this cycle time seems to be the **lowest possible multitasking time cycle time**. Thread switching has cycle times from 500 ns up to some Mikro-seconds.

4.1.2 Future Event List Schedulers

For complex simulation models the sequential scheduler is not good enough. Better characteristics are possible with Future event list schedulers. They manage all processes in a sorted list. New processes are inserted by using their next activation time as the sort value. In result, the entry at the start of the list is always the next process for execution.

A simple list is critical for large amounts of processes, because the time for finding the place for insertion is linear growing with the number of processes. The actual implementation task consists in finding algorithms with a better performance characteristic.

One option is an array-based tree with only 4 levels. In this scenario the time value is represented as a 32 bit long integer value. Each byte of this time is used as an index in one of the four levels (see fig. 5). With this approach, the insert time does not increase with a growing number of processes. The disadvantage is the same as before with the switcher – a high memory consumption. A test implementation shows, that about 3 Mbytes of RAM is necessary for running a typical production scenario.

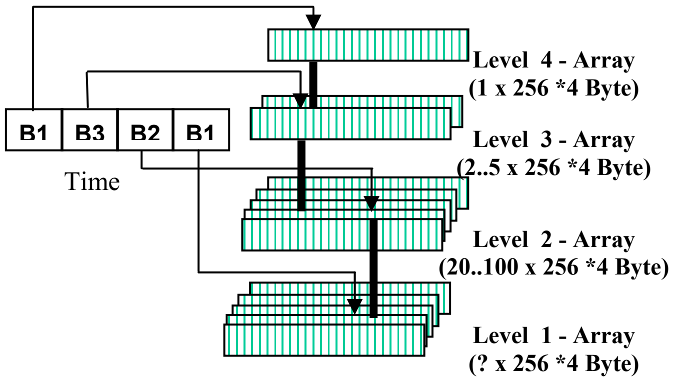

Figure 5 : A Improved Future Event Scheduler

The main difference to existing simulation systems is the freedom of choice in the area of schedulers. While switching is assembler based and not very comfortable for High-level programmers, the development of new and much more improved scheduling algorithms is quite simple for experienced simulation kernel developers. After an initial time of building up different schedulers, the simulation user can select one of already existing schedulers. It is also possible to use different schedulers for different areas of a simulation model.

5 THE SIMSOLUTION SYSTEM

All described basic routines will generate the kernel for a larger simulation environment, called “SIMSOLUTION”. The whole picture of the future “SIMSOLUTION”-simulation environment shown in Figure 6 and is based on former development of the author (Wiedemann 2000 , Wiedemann 2002). Above the Code-level are the GUI-interfaces or interfaces to other information systems. The large block in the center of the system controls all processes. It is also an interfacing layer between the specific tools at the tool level and the universal and standardized modules at the Model level.

The communication between all modules is based on file or network techniques. The communication protocol uses XML-coded information. In many cases the content of the XML-databases or XML-encoded simulation results is only wrapped by an additional XML-layer and transported over the network. Larger amount of data, for example simulation results, will be compressed by well-known compression algorithms for better transportation speed. For the end user this data conversions will be transparent.

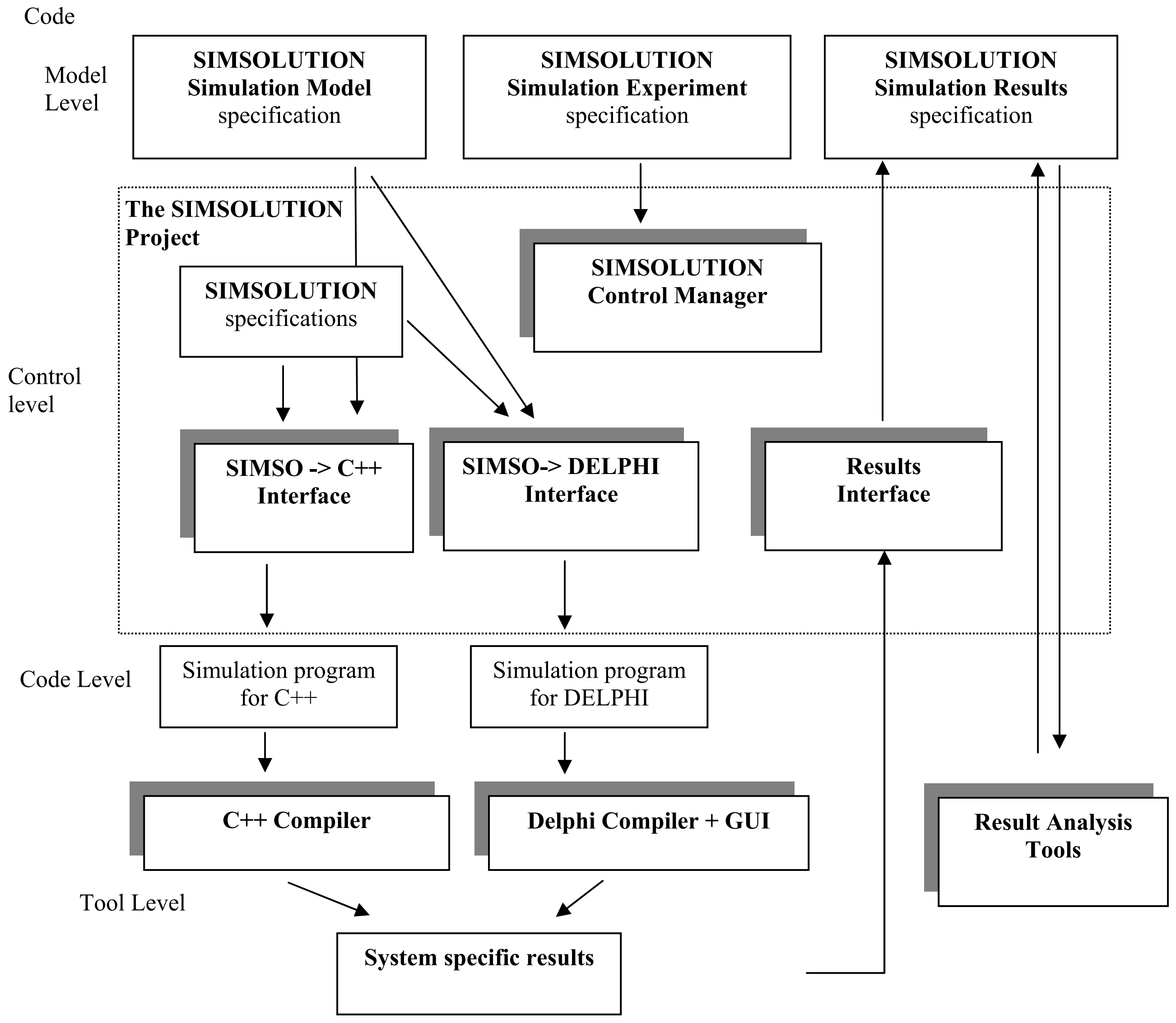
Code
Model Level
SIMSOLUTION Simulation Model specification
SIMSOLUTION Simulation Experiment specification
SIMSOLUTION Simulation Results specification
The SIMSOLUTION Project
SIMSOLUTION specifications
SIMSOLUTION Control Manager
Control level
SIMSO -> C++ Interface
SIMSO-> DELPHI Interface
Results Interface
Code Level
Simulation program for C++
Simulation program for DELPHI
C++ Compiler
Delphi Compiler + GUI
Result Analysis Tools
Tool Level
System specific results

Figure 6 : The Main Architecture of the SIMSOLUTION - System

6 SUMMARY

The new approach of dividing switching and scheduling of simulation processes could be a potentially beneficial evolution in the development of simulation software.

The first advantage is the larger flexibility from the use of different scheduling algorithms. Instead of having only one fixed kernel system the end-user can select the best solution depending on the needed interfaces and performance aspects.

The second advantage is a distribution with the Open Source Lesser/Library General Public License licensing model. This license model is a good mix of the Open source principles and the requirements of simulation customers.

The third advantage is the usage of an universal, language independent XML-description. Code parsers and generators convert SIMSOLUTION-models to programs in C++, Delphi or .NET-languages. With two sequential transformation processes a simulation model can be transferred between different platforms without manual changes.

The usage of some specific Assembler-routines for switching could be seen as some disadvantage. But the resulting simulation speed is very high and offers new solutions especially in the area of optimization and simulation. For that reason, the actual goal of development is to make the SIMSOLUTION-system the fastest simulation system, even if there are some disadvantages or missing functions compared to other simulation systems.

The actual state of the SIMSOLUTION-project is ongoing and further information is available at (SIMSOLUTION). Its future development will provide a universal and open simulation system. Any interested simulation expert or user is invited by the author for sharing his ideas, experience and cooperation inside the SIMSOLUTION-consortium.

REFERENCES

Kilgore, R. A., Healy, K. J. and Kleindorfer, G. B. 1998. The future of Java-based simulation. Proceedings of the 1998 Winter Simulation

Kilgore, R. A. 2001. Open source simulation modeling language (SML). In Proceedings of the 2001 Winter Simulation Conference, ed., B. Peters,J. Smith. Piscataway, NJ: 2001

Kuljis, Jasna and Ray J. Paul, 2000: A Review of web based simulation: whiter we wander?, *Proceedings of the 2000 Winter Simulation Conference*, Orlando Florida, page 1872-1881

Jacobs, Peter, 2004: The DSO Simulation System. *Proceedings of the European Simulation Symposium*, Budapest, Hungary, October 2004

Phillips, Lee Ann 2001. Special Edition using XML. Que Bestseller Edition, 2000

Schriber, Thomas J.; Brunner , Daniel T. : Inside Discrete-Event Simulation Software: How It Works and Why It Matters *Proceedings of the 2003 Winter Simulation Conference*, December 7-10, 2003, New Orleans, LA

SIMSOLUTION: Simulation Environment . Available online via http://www.simsolution.net [accessed April 1, 2005].

Wiedemann, T., 2000. VisualSLX – an open user shell for high-performance modeling and simulation, *Proceedings of the 2000 Winter Simulation Conference*, Orlando Florida, 1865-1871

Wiedemann, T., 2002. Next generation simulation environments founded on open source software and XML-based standard interfaces, *Proceedings of Proceedings of the 2002 Winter Simulation Conference*

Wiedewitsch J.; and Heusmann J. 1995. "Future Directions of Modeling and Simulation in the Department of Defense", Proceedings of the SCSC'95, Ottawa, Ontario, Canada, July 34-26, 1995

AUTHOR BIOGRAPHY

THOMAS WIEDEMANN is a professor at the Department of Computer Science at the University of Applied Science Dresden (HTWD). He has finished a study at the Technical University Sofia and a Ph.D. study at the Humboldt-University of Berlin. His research interests include simulation methodology, tools and environments in distributed simulation and manufacturing processes. His teaching areas include also intranet solutions and database applications. His e-mail address is wiedem@informatik.htw-dresden.de.