CERN / CN / 93 / 4

EUROPEAN LABORATORY FOR PARTICLE PHYSICS

CERN - CN Division

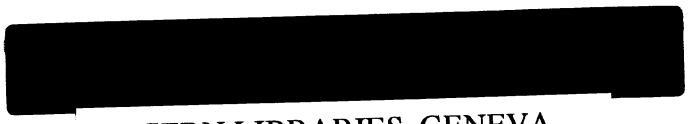
Executable System Specifications

L. Pregernig

ABSTRACT

With modern computer-aided-engineering tools it has become possible to model and to simulate specifications of time-dependent systems, regardless of their application domain. This allows one to analyze the dynamic behavior of systems and to evaluate alternatives, at the specification level, before committing any resources to detailed design. The paper describes the foundations for these methods and illustrates results of their application with an example.

Presented at the Schweizer Automatik Pool Forum 21
Zurich, Switzerland, March 9–10, 1993



1. Introduction

Looking at the evolution of cars is a good opportunity to see how systems are becoming more and more complex. For instance, the original Beetle had a reputation of a simple car which any local blacksmith could fix. Probably, one of its most complex systems was the ignition system. Today, engine-management systems replace traditional ignition systems and, we are likely to find more systems of similar complexity in a car, controlling features like cruising speed, (anti-lock) brakes, theft deterrents, etc.. For instance, in a typical American car, the programs which control all these systems count some 50,000 lines of code.

In the future, one will observe a further increase in complexity and there are strong reasons to deal with it as early as possible in the design process. Studies show that a company's gains are highest during the first few months of a successful product introduction. Yet, studies also show that it costs considerably (up to several order of magnitudes!) more to fix a problem in a finished product than to fix it at an earlier design stage.

This article attempts to address this issue. In particular, it will show how to build and how to simulate models of system specifications, with the aim to verify that the specifications fully meet the requirements. The following chapter will concentrate on system development in general. Then chapter 3 will give details about modeling and simulating system specifications. In chapter 4, an application example will illustrate this methodology. Finally, some conclusions will be drawn, based on the results of the example.

2. System Development

Talking about system development, it is important to distinguish between (i) development process, (ii) development methods, and (iii) notations.

First, as *development process* one understands all those organizational steps which are undertaken to develop a system. It means defining (i) stages for system development, (ii) an order to execute those stages, and (iii) criteria for proceeding from one stage to the next one.

Second, *development methods* are the ways to design, build, and test a system.

Third, as *notations* one understands the specifications for documenting a design; e.g., using circuit diagrams, flow charts, hardware description languages, etc..

Neglecting this differentiation may cause problems. For instance, if an organization just adopts a notation, but assumes that, in this way, it has adopted a methodology or even a process, expected "process improvements" are unlikely to occur. Also, if an organization adopts a standard methodology as their development process, frequent readjustments may become necessary for different system-development efforts. Models for system-development processes have been around for some time. The following describes some typical representatives and their implications for system development.

In the absence of a defined system-development process (*"ad-hoc development process"*), one can observe a series of shortcomings. Most likely, engineers will apply individualized methods which may differ from one project to the next. Furthermore, it will be difficult to judge development progress, because no milestones have been defined. And, on larger projects, significant integration problems may occur.

The classical model for system development is the *Waterfall Process* [1] (Fig. 1). It differentiates between system specification and system implementation. It will work, if the requirements are right. But, specifications may change during the process, for instance, as a consequence of certain implementation limitations which were not foreseen during the specification phase. The waterfall process puts the emphasis on completing a stage, before moving on to the next stage. All stages have equal weight. Therefore, it is difficult to take system-development risks into account.

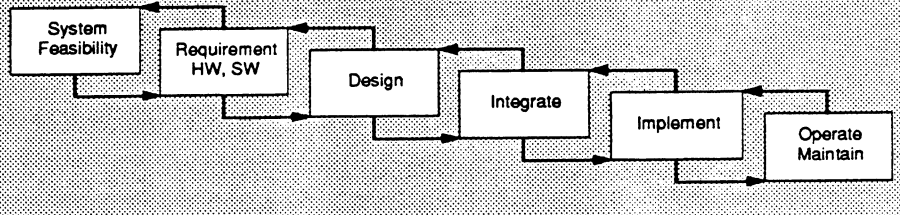


Figure 1: Waterfall Process (Principle)

In an attempt to address system-development risks, the *Spiral Development Process* [2] (Fig. 2) has been proposed. It focuses on risk analysis, and incremental planning and development. At the beginning of each cycle, one identifies the objectives of the relevant portion of the product, alternative means for implementation, and potential constraints. The subsequent phase is evaluating the alternatives, and identifying and resolving risks. This may include prototyping, simulation, etc.. Then, the next level is defined, taking remaining risks into account. Each cycle ends with a review.

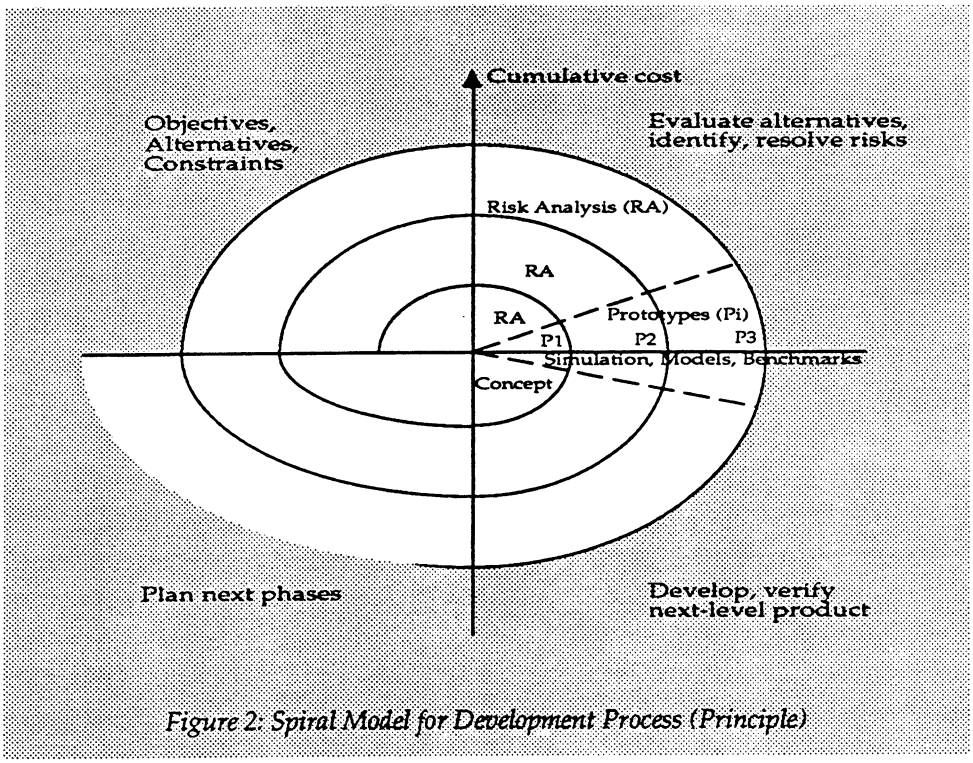


Figure 2: Spiral Model for Development Process (Principle)

After that brief discussion of some typical models for system-development processes, the following will focus on methods to describe systems in an implementation-independent way.

Research has been carried out, aiming at describing systems and analyzing their behavior in the problem domain (as opposed to the implementation domain). Results from that research, combined with progress in computer technology, made it possible to come up with *operational specifications*. Operational specifications are both implementation independent and executable. Therefore, they allow one to analyze and to validate a system early in the development process, before transforming the system specifications into the actual system implementation. The next chapter describes in more detail system modeling and simulation aspects.

3. System Modeling

A formal approach is necessary to build executable system models. In particular, one needs to describe a system with a precision such that a simulator can execute the description. At the origin of a class of methods and notations to do so is *Structured Analysis* [3] (DeMarco). Its main elements are processes, data-flows, data-stores, sinks, and sources. With these elements one can describe a system in a form which is known as data-flow diagram (Fig. 3). But, data-flow diagrams based on structured analysis do not contain timing and control information of a system. Therefore, their main use is for business data systems.

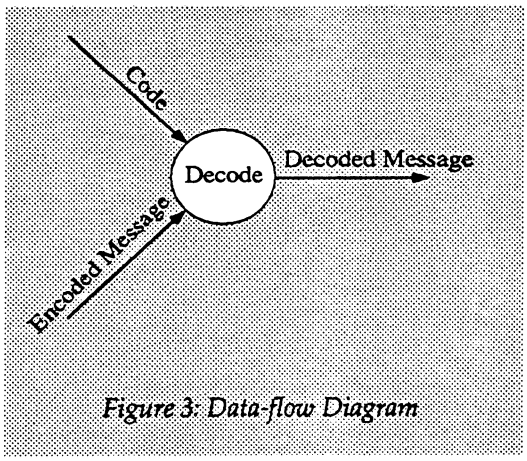


Figure 3: Data-flow Diagram

To describe time-dependent systems, further developments lead to *Real-Time Structured Analysis* (RTSA) [4, 5]. RTSA defines control or event flows, and control processes, in addition to the previously mentioned data-flow description elements. This allows one to describe the timing and control behavior of a system. In Figure 4, "Increment" and "Decrement" belong to the timing-and-control category, while the other elements are data-flow elements. A further evolution was to combine characteristics of these notations and methods into a graphical language called ESML (Extended Systems Modeling Language) [6]. Based on these methods, it became feasable to build executable system models.

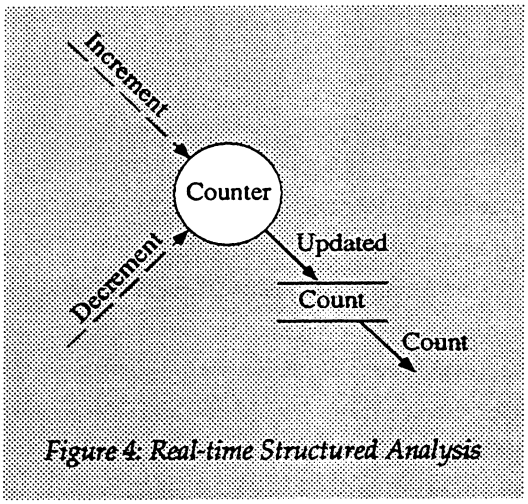


Figure 4: Real-time Structured Analysis

Executable system models include the precision necessary for simulation and, consequently, one can analyze a system's dynamic behavior early in the development process, and independent from the implementation. Therefore, they play an important role (i) in understanding the problem which one intends to solve, and (ii) in communicating this knowledge to others. Furthermore, one can refine the specifications, as the design process evolves.

The term system simulation covers a broad range of activities. One can split them into three main areas: (i) performance simulation, (ii) behavioral simulation, and (iii) functional simulation. Performance simulation mainly deals with timing and throughput analysis. Behavioral simulation allows one to analyze control and process sequencing aspects of a system, while functional simulation refers to the analysis of data and process transformations. After this discussion about the foundations for generating executable system specifications, the example below serves to illustrate the concepts.

4. Application Example

This example was built to analyze a queuing problem, as illustrated in Figure 5. Event data will arrive at a given rate at the input of a buffer (derandomizer). The event rate is constant over time, but the intervals between individual events are unequal, they follow an exponential distribution. The buffer has a fixed size and

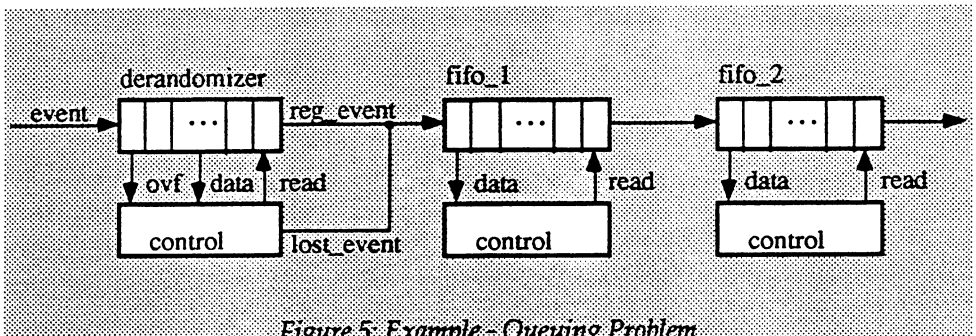


Figure 5: Example - Queuing Problem

can accept a limited amount of event data. Whenever the buffer contains data, they will be read. This takes a certain (for this example fixed) amount of time per registered event (readout delay). Events which occur while the buffer is full will be lost.

It is now the designer's task to find optimal values for the buffer size and the readout delay, in order to minimize event losses. Further restrictions are (i) that under realistic conditions the minimum readout time will be in the order of a few microseconds, and (ii) that the buffer has to fit into an integrated circuit.

This system has been successfully modeled and simulated with Foresight [7]. Foresight is a tool-set that consists of (i) *editors and libraries* to build a model, (ii) an *analyzer* to check the model and to create the simulation structure, and (iii) a *simulator* to exercise the model (Fig. 6). The model to analyze the queuing problem is a hierarchical model. It consists of a series of data-flow diagrams and state-transition diagrams. The graphical way of building and debugging the model, and features like animation during simulation, facilitate the designer's task. This helps to optimize the use of creative engineering time during the model building and validation phase. But, the price to pay for this interactivity is a reduction in simulation speed.

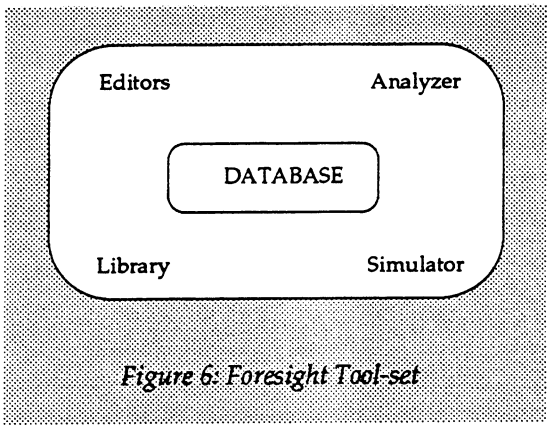


Figure 6: Foresight Tool-set

The analysis of the system model shows plateaus for both the buffer size and the readout delay (Fig. 7). Therefore, (i) increasing the derandomizer size to accommodate data from more than 7 events, and (ii) pushing the readout delay to values below 9 μs will only result in somewhat marginal improvements. In addition to these results, the simulation helped to detect a potential problem with regard to emptying the buffer. Under certain circumstances a race condition was observed. Event data could enter the buffer unregistered and "clog" it. Losses close to 100% were the consequence. But, with a properly designed buffer-control electronic, one can eliminate this hazard.

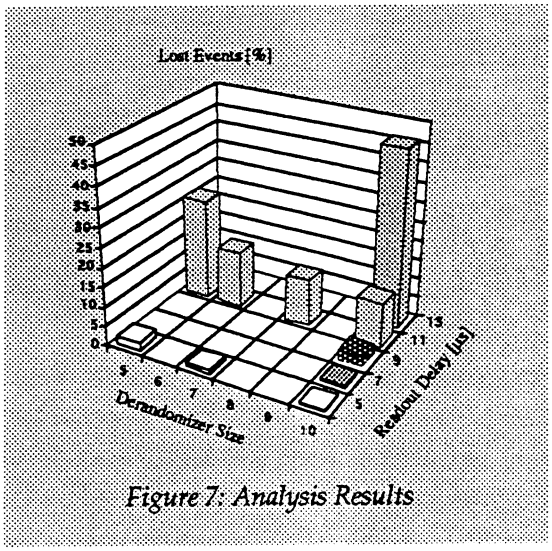


Figure 7: Analysis Results

5. Conclusions

The example above demonstrates that (using Foresight) it was possible to successfully model and simulate a system, independent of any implementation details. The results of the simulation show (i) how important system parameters influence the system behavior and (ii) which parameter values are acceptable choices. Furthermore, the simulation allowed to detect a potential design problem which may have been difficult to identify at a later design stage.

Work is under way to address issues, like improved simulation speed, or automating the transformation from the specification to the implementation phase. It is expected that enhancements which will automatically convert the system model into compilable code become available in the near future (early prototypes exist). As early test results have shown, this will improve simulation speed by a factor of 40 (for a test example, some 720 events per second were measured in interpreted mode, and about 30,000 events per second in compiled mode). Yet, at the time of this writing, no estimates can be made, if or when tools become available that convert a system specification into an implementation, at the push of a button.

Nevertheless, the conclusion of this presentation is that applying system-modeling and simulation techniques present considerable advantages.

Acknowledgements

The author would like to thank C. Eck and D. Jacobs from CERN for very actively supporting system-level modeling and simulation. A. Thys and N. Gómez have been very helpful in getting this new activity off the ground, and P. Farthouat helped with the example. Special thanks to B. Gaiser, B. Robinson, and M. Vertal from Nu Thena Systems. Their expertise and assistance were important factors in quickly reaching system-level-modeling proficiency. Furthermore, many useful discussion with them were fundamental to express the ideas which are presented in this paper. Thank you to B. Deutsch and G. Goetschmann from Intergraph for their efforts during the acquisition phase.

References

[1] Royce W. W.: *"Managing the Development of Large Software Systems: Concepts and Techniques,"* Proc. Wescon, August 1970.

[2] Boehm, B. W.: *"A Spiral Model of Software Development and Enhancement,"* ACM Sigsoft Software Engineering Notes, 1986, Vol 11, No 4, pp. 14–24.

[3] DeMarco, T.: *"Structured Analysis and System Specification,"* Prentice Hall, 1978.

[4] Mellor, S. and Ward, P.: *"Structured Development for Real-Time Systems,"* Vols. 1–3, Yourdon Press, 1985.

[5] Hatley, D. J. and Pirbhai, I.: *"Strategies for Real-Time System Specification,"* Dorset House, 1987.

[6] Bruyn, W., Jensen, R., Keskar, D., Ward, P.: *"ESML: An Extended Systems Modeling Language Based on the Data Flow Diagram,"* ACM Sigsoft, Software Engineering Notes, Vol 13 No 1, pp 58–67.

[7] Nu Thena Systems Inc.: *"Foresight,"* McLean, Virginia.