A CASE STUDY FOR PARTITIONED MODELLING OF A CONTROL SYSTEM

Bernd Däne, Wolfgang Fengler

Ilmenau Technical University
P.O. Box 100565
98684 Ilmenau
Germany
{bernd.daene|wolfgang.fengler}@tu-ilmenau.de

ABSTRACT

This paper describes a case study and additional considerations about modelling an embedded high-performance control system. Special consideration is given to a partitioning concept that supports distinction of partitions for the actual implementation, the environmental model and instrumentations for simulation only. This concept does not press the model into a certain structure but still provides hierarchical properties to the partitioning process. The model uses a multi-domain approach with both continuous-time and discrete-event views. It includes signal-flow components as well as statechart diagrams. Goals of the modelling process are functional validation of the solution and performance evaluation. In future the approach should contribute to a methodology for generating software from the model. The modelling environment in the study has been a multi-domain modelling tool. An experimental implementation of the partitioning algorithm helped investigating the method. Further improvements are discussed too.

KEY WORDS

Modelling, Computer Control, Real-Time Programming, Mixed Models.

1. Introduction

Modelling and simulation environments provide a useful way for the systematical design of embedded systems. This approach avoids programming errors and provides a number of ways to check the design by simulation and formal analysis. This work deals with graphical modelling methods where models consist of blocks, signals and hierarchical refinements.

In the modelling process some kind of partitioning occurs. This means that model elements will be mapped to certain partitions of the model. One common procedure is partitioning into hardware and software partitions during hardware-software-codesign. But other views to model partitioning are also of interest, as described below.

The model must not only include the embedded system being designed (e.g. the controller) but also some embedding environment (e.g. the controlled process). Only in this way the interaction of the controller with the controlled process can be investigated and useful information about the solution can be derived. So the model includes the embedded system, the embedding environment and the interface elements between them. The latter constitute the input and output ports of the embedded system.

Since the model partition that contains the embedded system is intended to implement a real system it will be referred to as **implementation** partition of the model. The partition that contains the embedding environment will be referred to as **environment** partition.

Other elements may have been added for simulation purposes only (sometimes formal analysis may also be supported by additional model elements). Such elements may generate stimuli or collect data, for instance. The elements can work interactively (user input and live display during simulation) or not. They may be supported by other operational elements that do some scaling, formatting and so on.

Such elements must be distinguished from the embedding environment because they do not establish interfaces to the embedded system. This special partition of the model here will be referred to as **instrumentation** partition.

Generally hierarchical model structures are used. Therefore there must be a concept how to propagate the mapping information downwards the hierarchical tree. This kind of partitioning a model mostly must be done manually because this information derives from user's intention. But a method is needed that supports the user in this process and provides flexibility while preserving the model's consistency.

2. Known Approaches

This section will outline how the mapping of the blocks of the model to the partitions mentioned above can be handled during the modeling process. Some shortcomings will be identified.

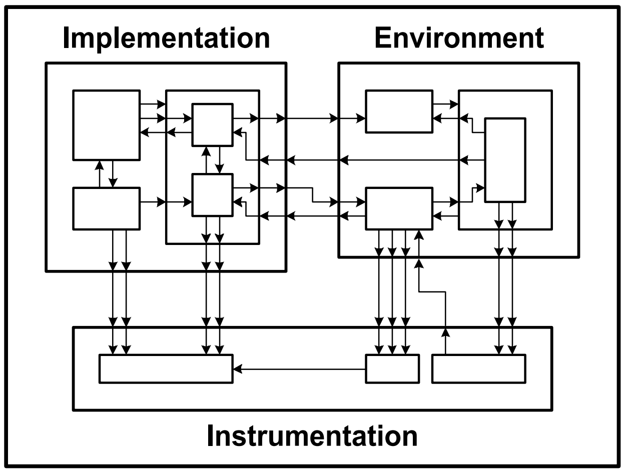

Figure 1: Structural view to a model partitioned by top level blocks.

Top Level Blocks

Some implementation frameworks require the whole implementation part to be in just one block at an upper (or the uppermost) hierarchical level. The ports of this block automatically constitute the embedded system's input/output interface.

As an example the *Polis* framework uses this approach, as described by Balarin et al. [1]. This framework allows generating code from a discrete event model. Here exactly one block must be identified that fully contains the generated system. Similar methods have been used by H. Rath [2] and by ourselves in former work [3].

Since there is no distinction between embedding environment and instrumentation, the latter parts must be removed from the model prior to generating code. Otherwise unwanted inputs and outputs would be generated.

The approach may be generalized by defining exactly one block for each of the three partitions (implementation, environment, instrumentation). A similar method would be defining one block for implementation and one block for environment. Everything outside these blocks belongs to instrumentation. The blocks mentioned above will most likely reside at top hierarchical level.

This generalized scheme provides full support for defining the actual inputs and outputs of the system generated. They can easily be examined and extracted since they all are located at the same hierarchical level.

But a closer look at some nontrivial models reveals the main shortcoming: All connections that cross the borders of the partitions at first must be routed to the top level block of this partition, possibly traveling through some hierarchical levels and requiring numerous additional ports at the borders of lower level blocks. This leads to a model structure where the sources and the sinks of cross-partition connections typically reside at different places and must be followed through some levels in order to find the counterpart.

While this approach clearly represents the organizational structure of the model it unfortunately hides its functional structure. As an example think about control loops that are deeply embedded in the model. The implementation partition contains the control algorithms, while the environment contains the process controlled. In this scheme the control loops are divided into two distant parts each, and their relationships are not clearly visible.

So this approach leads to poorly structured models that are overcrowded by connections and ports and that hide their functional structure instead of exposing it. Figure 1 schematically illustrates this problem.

Textual Reference

This overcrowding by connections and ports may be removed if the cross-partition connections are represented by textual reference. By doing so the graphical model effectively divides into two or three non-connected parts. The advantage is that these references may be placed in deep levels near the interfacing blocks. So there is no need for routing them through higher hierarchical levels. But the approach lacks any kind of graphical representation of the relationships, since text symbols must be searched for instead of looking for visible connections.

However, this approach has been used in a former work dealing with an object oriented modelling paradigm [4]. As expected the models had clean structures, but it was difficult to recognize functional relationships.

3. Concept: Individual Assignment of Blocks

To overcome these shortcomings, our approach deals with blocks that may be individually assigned to different partitions, at any hierarchical level. This seems straightforward and easy to implement. But some considerations are needed about handling such properties when refining the model into deeper levels and about visual representation. Furthermore the concept should contribute to validation procedures. The approach explained below deals with such questions. It has been practically examined in a case study [5].

Mapping Information

The method attaches the mapping information as a parameter to each of the blocks of the model. This applies to bottom level blocks (leaves) as well as to higher level blocks that may contain deeper levels of the model. This parameter is the primary representation of the mapping information, i.e. the membership of the block to one of the partitions.

Exactly one of the following values is to be assigned:

- **None.**
  The block is not yet assigned to one of the partitions.
- **Implementation.**
  The block is assigned to the partition of the model that constitutes the embedded system that will be implemented.
- **Environment.**
  The block is assigned to the partition of the model that constitutes the embedding environment.
- **Instrumentation.**
  The block is assigned to the partition of the model that contains elements for simulation only.
- **Miscellaneous.**
  The assignment of this block is to be defined automatically.

While the purposes of the 'Implementation', 'Environment' and 'Instrumentation' values already have been discussed, the others need further explanation.

'None' means that no assignment for this block has been made. This is the initial state of a block if no other assignment is done by rule. For a fully partitioned model, all bottom level blocks (leaves) must have a mapping differently to 'None'.

'Miscellaneous' is introduced as a kind of 'don't care' property. Typically it is used for elements that do not really process data, such as splits and joins. When the interface of the embedded system is to be defined they will automatically be assigned to one of the partitions. This decision optimizes for minimal cross-partition connections and consequently for minimal input/output effort of the embedded system. Since the value 'Miscellaneous' will be preserved in the model, this assignment may change if the generation process is repeated after modifying the model.

Initial Mapping Values

The initial mapping value of a newly created block instance is defined by the following rule:

- If the block resides inside another block (i.e. refines it) that has one of the values 'Implementation', 'Environment' or 'Instrumentation', it inherits this value.
- Otherwise, the block gets the mapping value 'None'.

By using these rules, the refinement elements of blocks that already belong to one of the model partitions by default will be assigned to just this partition. This does not apply to the refinement of 'Miscellaneous' blocks because this value does not represent a partition membership.

Changing Mapping Values

The mapping value of any block may be changed by user decision. In some cases this change will propagate into deeper levels of the model. The following rules apply:

- The user can assign any value to the block. This will override the former value, unless the next rule applies.
- Successful assignment of one value out of 'Implementation', 'Environment' and 'Instrumentation' to a block will propagate this value downwards to the next hierarchical level (i.e. to the refinement of this block), if available. All blocks with value 'None' in this level will be overridden by the value propagated. For each of these blocks this process iteratively continues downwards the hierarchy. Blocks with mapping values other then 'None' remain unchanged.

Note that these rules are intended to support the process of model creation rather than defining the model's final structure only. The partition membership of blocks can be defined prior to or after their creation because the assignment applies to previously undefined elements und to elements that will be created in future.

The mapping information is handled consistently to the model's hierarchical structure. So the partitioning process leads the user hierarchically through the model. While many blocks through some hierarchical levels can be assigned at once, intentional assignments previously made at deeper levels will be preserved.

Support for Implementing a Target System

To continue with the design process the implementation part must be extracted from the model. In this step the input/output interface must be defined. The following rules are to be used:

- Look for bottom level blocks ('leaves') with mapping value 'None'. If found, stop the process and prompt an error message.
- Look for bottom level blocks with mapping value 'Miscellaneous'. Assign to each of them one *temporary* mapping value out of 'Implementation', 'Environment' and 'Instrumentation'. This decision should minimize the number of cross-partition connections.
- Collect all bottom level blocks with mapping value 'Implementation' (including temporary values).
- Collect all ports of these blocks that are connected with at least one block with mapping value 'Environment'.

The collections produced by the last two rules constitute the embedded system to be generated and its input/output interface, respectively. Obviously only the mapping values of bottom level blocks contribute to the final mapping. This means that mapping values of higher level blocks are aimed at controlling the refinement process during model development.

4. The Case Study

As already mentioned this concept has been practically examined in a case study. In this study a general-purpose multi-domain modelling tool experimentally has been equipped with the capabilities needed. Using this setup, some complex models for a real project, a high-precision measurement system as explained in [6], have been developed and examined.

Modelling Tool

The case study uses the general-purpose modelling and design tool *MLDesigner* [7]. *MLDesigner* is a commercial tool that utilizes modelling principles from the well-known *Ptolemy* tool [8], also known as *Ptolemy Classic*, in a renewed implementation.

In contrast to *Ptolemy* the *MLDesigner* tool uses XML-files to store the models. This makes the model accessible for manipulation by external processes.

Implementing the Mapping Concept

For this experimental implementation the tool's parameter mechanism has been utilized to contain the mapping information of each block. The functionalities of value propagation and visual representation have been added by using an external program. This program filters the stored model representation, manipulating it in the desired manner. It propagates changed values into deeper levels by applying the rules mentioned above. Then it modifies a color property of the blocks to provide a visual representation. For a final solution the algorithm will be integrated into the modelling tool.

Example Models

Example models have been created for a real project that deals with high precision measuring machines [9]. Basically the machine works by moving a carrier with an object in all three dimensions. Position values are measured with high precision (less then two nanometers) and controlled by closed loop control. Features at the object's surface are detected by fixed probe devices so the process delivers high precision position values for these features. Such machines are also known as 'Scanning Probe Microscopes' (SPM).

The models are mixed from discrete-event and continuous-time paradigms and exhibit a multi-domain nature. They are developed from a previously created model described in [6]. While the target system of the design process is an embedded multiprocessor system that mainly performs closed loop control algorithms and processing of sensor values, the mechanical and optical parts of the machine constitute the embedding environment that is to be included into the model too.

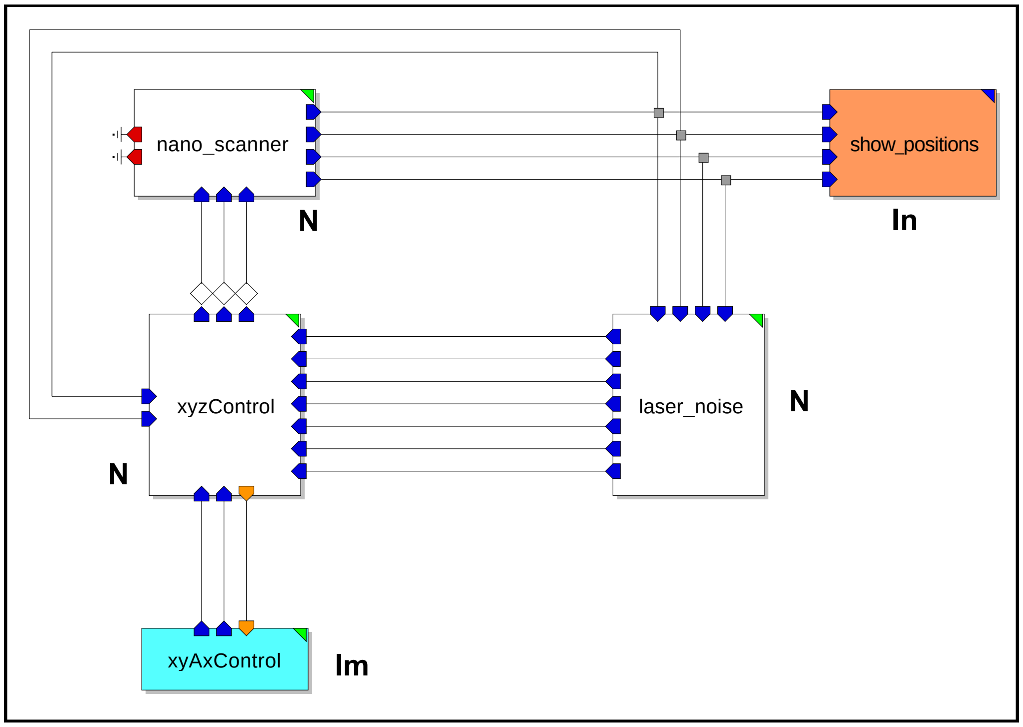

Figure 2: Top level of model, with blocks belonging to 'Instrumentation' (**In**) and 'Implementation' (**Im**) partitions as well as 'None' (**N**) blocks.

Some of the functions modeled are as follows:

- Control loops for high precision position control, including electro-magnetic actuators, mechanical assemblies, laser-optical position sensors, sensor signal processing, control algorithms.
- Initiation and control of function sequences.
- Controllable error and noise models.
- Recording and/or interactive display of values and states during simulation.

The model structure was based on the functional relationships of the blocks. The assignment of blocks to the model's partition was handled by using the concept developed. It became visible that the method was easily handled by the users and lead to useful, clearly arranged models.

Figure 2 shows the top level of the main model. Only five blocks appear. The block *'nano_scanner'* models the mechanical system of the measurement machine, consisting of movable parts that can be positioned with high precision. Embedded at deeper levels are components that simulate errors and noise influences and collect information during simulation. At this level partition membership of this block had not been defined (mapping value 'None').

The same is true for block *'laser_noise'*, containing elements of the measurement system and assigned error models, and for block *'xyzControl'*, containing main parts for closed loop control of positions.

The block *'xyAxControl'* mainly contains software components for sequence control and operation management of the machine that belong to the 'Implementation' partition. So this mapping property has already been defined at this level and has automatically been propagated into deeper levels.

The block *'show_positions'* contains elements that collect and visualize position values during simulation and is therefore assigned to the 'Instrumentation' partition.

Remark: These figures are screenshots taken from the actual modelling tool. Since the visualization of mapping values originally done by colors does not apply to monochrome figures these values are clarified by letter symbols (inserted manually). Please refer to figure captions for explanation.

Figure 3 shows the refinement of the block *'nano_scanner'* containing model components for the mechanical system (e.g. *'3d_system'* and *'cantilever'*, belonging to 'Environment' partition), 'Instrumentation' elements (e.g. sources for parameters that are controllable during simulation) and a number of 'Miscellaneous' blocks such as forks and blackholes (dummy destinations). While this block *'nano_scanner'* basically belongs to the model environment, handling of blocks with other mapping properties is not hampered inside this level.

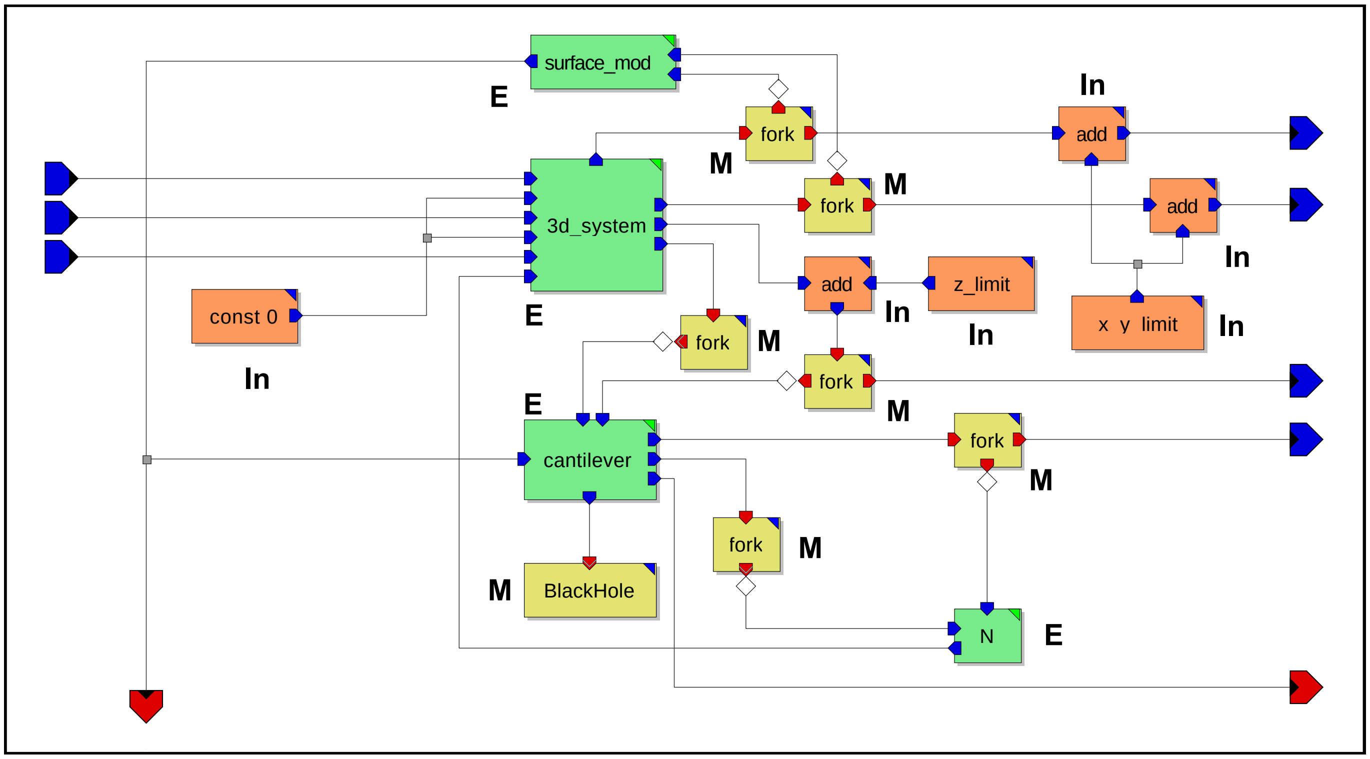

Figure 3: Refinement of block *'nano_scanner'*, with blocks belonging to 'Environment' (**E**) and 'Instrumentation' (**In**) partitions as well as 'Miscellaneous' (**M**) blocks.

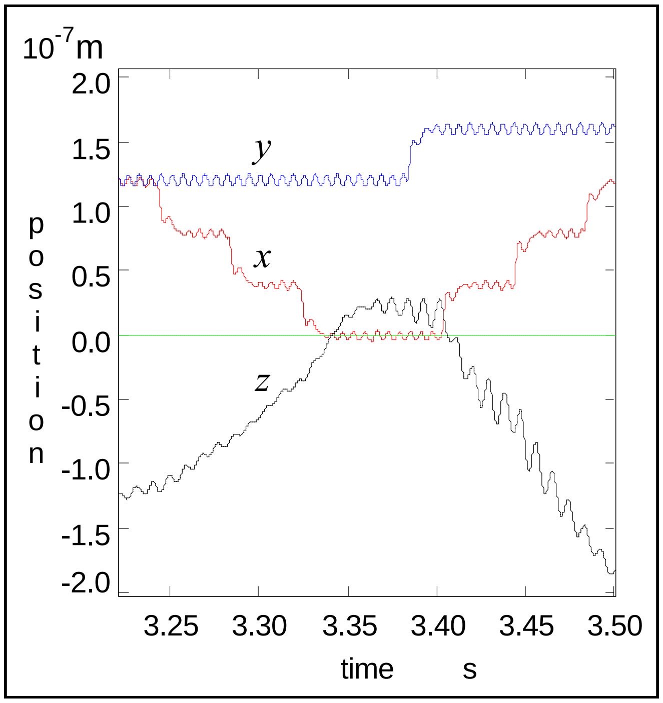

Figure 4: Example position display from simulation (*x*, *y*, *z* position values, partly shown).

Figure 4 illustrates simulation results by showing a part of the interactive plot of position values delivered by elements inside the *'show_positions'* block. Intended stepwise changes of *x* and *y* values are visible as well as unwanted vibration and noise errors in all three axis.

5. Results and Conclusion

In this paper a flexible concept for assigning a model's components to different model partitions has been defined and explained. The concept has been implemented and examined experimentally with nontrivial models from a real project. The main expectations concerning flexible handling during model development have been met.

Further work will include refining the method and fully integrating it into the modelling tool. The method can be further developed by generalizing the predefined set of model partitions into a dynamically extensible partition concept. This allows supporting partitioning problems such as hardware/software partitioning or workload partitioning for multiprocessor systems by the same approach.

Moreover the inclusion of methods of multitasking software behavior [10] and code generation for target systems [3] will be explored in order to contribute to an integrated design framework for embedded systems.

6. Acknowledgements

This work is supported by *Deutsche Forschungsgemeinschaft* (DFG) under SFB 622.

MLDesigner © 2004 MLDesign Technologies, Inc. All rights reserved. *http://www.mldesigner.com/*

References

[1] F. Balarin, P. Giusto, A. Jurecska, C. Passerone, E. Sentovich, B. Tabbara, M. Chiodo, H. Hsieh, L. Lavagno, A. L. Sangiovanni-Vincentelli, & K. Suzuki, Hardware-Software Co-Design of Embedded Systems. The POLIS Approach, Boston, USA, Kluwer Academic Publishers, 1997.

[2] H. Rath, & H. Salzwedel, ANSI C Code Synthesis for MLDesigner Finite State Machines, Synergies between Information and Automation, 49th International Scientific Colloquium, Aachen: Shaker, Germany, 2004, 107-112.

[3] B. Däne, & W. Fengler, Implementing Mixed Discrete-Continuous Models into Realtime Environments, MIC 2004, The 23rd IASTED International Conference on Modelling, Identification, and Control, Grindelwald, Switzerland, 2004, 583-588.

[4] J. Nützel, B. Däne, & W. Fengler, Object Nets for the Design and Verification of Distributed and Embedded Applications, Proc. EHPC'98, 3rd International Workshop on Embedded High Performance Computing at the First Merged Symposium IPPS/SPDP'98, Orlando, USA, 1998, 953-962.

[5] E. Kaufmann, Modelling Methodology for Embedded DSP Systems Design (Diploma Thesis 2004-05-03/048/IN96/2231), Department of Computer Architectures, Ilmenau Technical University, Ilmenau, Germany, 2004.

[6] W. Fengler, B. Däne, & V. Duridanova, Design Methodology for an Embedded System for High-Performance Computing, Proc. WRTP'03, 27th IFAC/IFIP/IEEE Workshop on Real-Time Programming, Lagow, Poland, 2003, 123-128.

[7] V. Zerbe, U. Freund, & H. Salzwedel, Mission Level Design of Control Systems, Proc. SCI/ISAS'99 Multiconference on Systemics, Cybernetics, Informatics, Orlando, USA, 1999, vol. 7, 237-243.

[8] J. T. Buck, S. Ha, E. A. Lee, & D. G. Messerschmitt, Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems, Int. Journal of Computer Simulation, 22(4), 1994, 155-182.

[9] G. Jäger, E. Manske, T. Hausotte, W. Schott, Operation and Analysis of Nanopositioning and Nanomeasuring Machine, Proceedings 17th Annual Meeting, ASPE, St. Louis, USA, 2002, 299 – 304.

[10] B. Däne, W. Fengler, & F. Berger, Modelling and Simulation of Operating System Behavior, Proc. MSO 2003, IASTED International Conference on Modelling, Simulation and Optimization, Banff, Canada, 2003, 78-81.