Modeling Production Lines with Blocking, Batch Processing and Unreliable Machines Using the Markov Analyzer MOSES

Gunter Bolch
University Erlangen-Nuernberg
IMMD IV
Martensstrasse 1
D – 91058 Erlangen
bolch@informatik.uni-erlangen.de

Stefan Greiner
Duke University
Dept. of Electrical Engineering
Durham, NC 27708-0291, USA
Box 90291
greiner@ee.duke.edu

Abstract

The analysis of todays production systems gets more and more important because of growing system complexity as well as increasing production costs. Therefore methods and tools are needed that allow a simple system specification and analysis. The purpose of this article is to give a short introduction to the specification of production systems. At first the Markov analysis tool **MOSES**(**MO**delling, **S**pecification and **E**valuation of Computer **S**ystems) and the model description language **MOSLANG** are introduced. Then different types of characteristics of production systems are shortly explained. For each of these characteristics it is shown how to specify it using MOSLANG. After that different combinations of these characteristics are explained and specified. These specifications are used to show how to model a simple example of a production system.

1 Introduction/Motivation

Performance evaluation of todays production lines [1] becomes more and more important because of the growing system complexity. It is often the case that system measurements are not possible or very difficult to do. Therefore a system model has to be developed. To analyze such a model, three different techniques are possible:

- Discrete event simulation.
- Markovian analysis.
- Product form queueing network oriented analysis [2].

If we use discrete event simulation as analysis technique then the model behavior can be described very exactly but computation time is usually extremely high. If we choose on the other side a queueing network approach then only very simple production lines can be specified because of the limiting assumptions of queueing networks. It is for example difficult to model production lines with finite buffer, unreliable server and batch processing using queueing network solution techniques. But especially these features are of particular interest when modeling production lines. Therefore we use Markovian modeling techniques to model production lines. Markov models are very useful to evaluate these types of systems. Since it is nearly impossible to construct the underlying state space by hand, the Markov analyzer **MOSES** and the model description language **MOSLANG** are used [3, 4, 5]. It is of course also possible to use Petri nets to model production systems because Petri nets as well as the MOSLANG code are transformed into the underlying Markov chain which is then solved. In this article the model description language **MOSLANG** is introduced and applied to some examples of production lines with unreliable servers, finite buffers and batch processing. The MOSLANG specification of the system is translated by MOSES into the Markovian system of equations. For the solution of these equations MOSES offers six different methods which can be chosen by the user interactively.

This article is organized as follows. At first the different characteristics of production systems like batch processing, finite buffer and unreliable server are introduced. Their characteristics are explained shortly and then it is shown how to specify these characteristics using MOSES. In these specifications at first each characteristic is considered separately from each other and then it is shown how to combine these characteristics with each other. For each specification we also represent some results to show a possible system behavior. Finally it is shown how to specify a simple example of a production system that contains all the different explained characteristics. It is assumed that all times are exponentially distributed. Throughout this article the following parameters are used:

Parameter	Explanation
num	actual number of jobs
K	maximum number of jobs
a	minimum batch size
b	maximum batch size
mttr	mean repair time
mtbf	mean time between failures
capacity	length of the finite queue
service_time_1	service time of machine 1
service_time_2	service time of machine 2
service_time_3	service time of machine 3
arrival	arrival rate at the system

2 The Model Description Language MOSLANG

The language MOSLANG consists of a series of constructs which can be summarized in four parts, the declaration part, the vector description part, the rules part and the results part.
In this section the semantic of each part is briefly explained.

Declaration part: This part contains the declarations. Symbolic names can be given to each component of the state vector. Constants can be defined as in C. All parameters and the required performance measures must be declared here. The type and name of the parameter or performance measure must be specified here.

Vector description part: In this part the dimension and the range of the state space is determined. Also the prohibited states are specified.

Rules part: In the rules part the state transitions are specified. This can be done by specifying a number of RULE structures. The RULE structure consists of a condition and an action part. If all conditions are fulfilled, the specified actions will be executed. The action part also contains the transition rate and the transition probability. One RULE structure specifies several state transitions. It is not necessary to specify an individual RULE structure for every state transition. It is also possible to use *immediate transitions* (timeless transitions). By using timeless transitions a decoupling of different nodes is achieved which results in a considerable reduction of the system description.

Results part: In the results part the performance measures are specified. The results part consists of RESULT structures and FINAL structures. With the RESULT structure the basic performance measures which can be derived directly from the state probabilities are computed. With the FINAL structure the performance measures which are functions of the basic performance measures and the system parameters, for example average waiting times, utilization, throughput, can be computed.

3 Application of the Markov Analyzer MOSES

Now we want to show how to specify production lines with unreliable server, finite buffer and batch processing. Our survey is based on the model shown in Fig. 1:

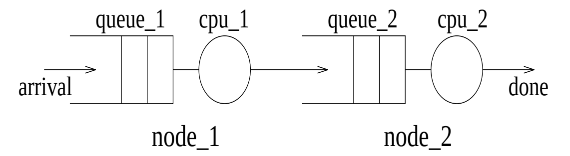

Figure 1: Basic model

As can be seen each *node* consists of a *queue* and a *cpu*. In the following specifications it is sometimes necessary to specify the behavior of a node and a cpu separately (cpu_i, node_i, i=1..2) and sometimes it is enough to specify the behavior of the whole node (using only node_i, i=1..2). The system is open and jobs arrive at the system with rate *arrival*. As soon as they are processed they leave the system. All times in this article are given in the quantity of *time units*.

3.1 Batch Processing

3.1.1 Description of the Batch Processing Modes

Batch processing is an integral part of many production systems. We want to consider two different kinds of batch processing modes:

Full batch systems (FB): In this case the machine does not start to work until the batch is full (assuming a batch of size b). As soon as the batch is full the machine starts and all b jobs are processed in parallel. When the jobs are served the next b jobs can enter the machine to be processed.

Minimum batch systems (MB): In this case the machine already starts to work when at least $a \leq b$

jobs are in the batch. If jobs arrive later at the system and the batch is not full then these jobs are also served but they only get the rest service time of the batch.

3.1.2 Specification of the Full Batch Processing Mode

The core of the specification in MOSLANG for the full batch processing mode is shown in Fig. 2. Here node_1 (consisting of queue_1 and cpu_1) is a full batch node while node_2 is a regular M/M/1 node.

```
1  RULE {queue_1 >= b}{cpu_1 == idle} ->
          cpu_1 = busy : immediate : 1.0           ;
2  RULE {cpu_1 == busy} ->
          queue_1 -= b node_2 += b
          cpu_1 = idle : 1/service_time_1 : 1.0   ;
3  RULE {node_2 > 0} ->
          node_2 -= 1 : 1/service_time_2 : 1.0    ;
```

Figure 2: MOSES core for the full batch mode

These lines can be interpreted as follows:

Line 1: If there are at least b jobs in *queue_1* (the batch is full) and *cpu_1* is *idle* then the batch can start. Therefore *cpu_1* is set *busy*. This transition is immediate and takes place with probability 1.0 .

Line 2: As soon as *cpu_1* is busy b jobs are removed from *queue_1*. These jobs are then processed. After that they are moved to *node_2* and *cpu_1* is set *idle* again because the batch has been processed. This transition takes place with rate $1/service\_time\_1$ and probability 1.0.

Line 3: As soon as there are jobs in *node_2* the rule in line 3 is fulfilled and a job can be processed. In this case the processed job is removed from *node_2*. This transition takes place with rate $1/service\_time\_2$ and probability 1.0.

3.1.3 Graphical Representation of the Results

In the plot shown in Fig. 3 the batch size b is varied from 1 to 20 and the throughput of the system is plotted as a function of the batch size b. The other parameter are given as follows:

service_time_1	=	8.0	K	=	20
service_time_2	=	0.2			

As can be seen in Fig. 3 for each arrival rate it can be found an optimal batch size b so that the system throughput is maximum.

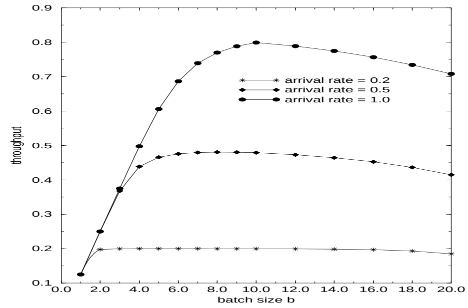

Figure 3: Results for full batch processing

3.1.4 Specification of the Minimum Batch Processing Mode

The core of the specification in MOSLANG for the minimum batch processing mode is shown in Fig. 4. Here node_1 (consisting of queue_1 and cpu_1) is a minimum batch node while node_2 is a regular M/M/1 node.

```
1  RULE {queue_1 >= a}{cpu_1 == idle} ->
          cpu_1 = busy : immediate : 1.0         ;
2  RULE {cpu_1 == busy} ->
          cpu_1 = wait  : 1/service_time_1 :
          1.0                                    ;
3  RULE {cpu_1 == wait}{queue_1 >= b} ->
          queue_1 -= b node_2 += b
          cpu_1 = idle : immediate : 1.0         ;
4  RULE {cpu_1 == wait}{queue_1 > 0}
        {queue_1 < b} ->
          queue_1 -= 1 node_2 += 1 :
          immediate : 1.0 ;
5  RULE {cpu_1 == wait}{queue_1 == 0} ->
          cpu_1 = idle : immediate : 1.0         ;
6  RULE {node_2 > 0} ->
          node_2 -= 1
          num -= 1 : 1/service_time_2 : 1.0      ;
```

Figure 4: MOSES core for minimum batch mode

These lines can be interpreted as follows:

Line 1: If there are at least a jobs in the batch and *cpu_1* is idle then *cpu_1* can start to work. This transition is immediate and takes place with probability 1.0.

Line 2: If *cpu_1* is *busy* then *cpu_1* switches to the pseudo *wait* state. This transition takes place with rate $1/service\_time\_1$ and probability 1.0. This is to simulate that jobs are served. The problem is that during the service of the jobs other jobs can arrive at the system and still get service. Therefore we use the *wait* state to determine how many jobs finally got served.

Line 3: If *cpu_1* is in the *wait* state and there are more than b jobs in *queue_1* then exactly b jobs get served and are moved to *node_2*. The *cpu_1* is then set *idle* again.

Line 4: If there were less than b jobs in the system then all jobs that arrived got served and have to be moved to *node_2*.

Line 5: As soon as in the case of line 4 all jobs are moved to *node_2*, *cpu_1* is set idle.

Line 6: This rule again shows that jobs are served at *node_2* with rate $1/service\_time\_2$.

As can be seen in this description the full batch system is a special case of the minimum batch system. If we set in the minimum batch system the batch sizes $a = b$ then we get a full batch system.

3.1.5 Graphical Representation of the Results

In the plot shown in Fig. 5 the batch size a is varied from 1 to 18 while b is 18. The other parameter are as in the example before. As can be seen the bigger a the lower is the throughput of the system. This can be explained very easily since with bigger a jobs have to wait longer until there are at least a jobs in the queue so that the system can start.

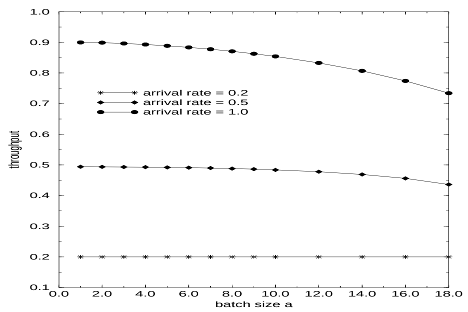

Figure 5: Results for minimum batch processing mode

3.2 Systems with Finite Buffer

3.2.1 Description of the Blocking Strategy

Assuming a system with finite buffer, blocking can occur in the system if a buffer is full. Many different blocking strategies are possible but in the area of production systems it is usual to consider blocking after service. In this case *node_1* is blocked as soon as a served job at *cpu_1* wants to go to *node_2* which is already full. Then the already served job has to wait in *cpu_1* until there is space available at *node_2*.

3.2.2 Specification of a System with Finite Buffer

The core of the specification in MOSLANG for the system with finite buffer is shown in Fig. 6. Here node_1 (consisting of queue_1 and cpu_1) is a regular M/M/1 node while node_2 is a finite buffer node.

```
1  RULE {queue_1 > 0}{cpu_1 == idle} ->
          cpu_1 = busy queue_1 -= 1 :
          immediate : 1.0                         ;

2  RULE {cpu_1 == busy} ->
          cpu_1 = wait :
          1/service_time_1 : 1.0                  ;

3  RULE {cpu_1 == wait}{node_2 < capacity} ->
          cpu_1 = idle node_2 += 1 :
          immediate : 1.0                         ;

4  RULE {node_2 > 0} ->
          node_2 -= 1 num -= 1 :
          1/service_time_2 : 1.0                  ;
```

Figure 6: MOSES core for a finite buffer

These lines can be interpreted as follows:

Line 1: If there are jobs waiting at *queue_1* and *cpu_1* is idle then a job is removed from *queue_1* and *cpu_1* is set *busy*.

Line 2: As soon as the cpu is busy the job is served and the cpu is set in the pseudo state *wait*. This state is necessary because we have to check if the *capacity* of the following node is already exceeded.

Line 3: If the capacity of *node_2* is not exceeded yet then the served job can be moved to *node_2* and *cpu_1* is set *idle*. In the other case the job stays in *cpu_1* and the node is blocked.

Line 4: The processing of a job at *node_2* is described in that line.

3.2.3 Graphical Representation of the Results

In the plot shown in Fig. 7 the capacity of the node is varied from 1 to 20 and the system throughput is plotted as a function of the capacity. In this case we have chosen the following parameters:

service_time_1	=	8.0	K	=	20
service_time_2	=	0.2			

As expected, the system throughput reaches some kind of steady state with increasing capacity. This is because the higher the capacity the lower is the probability that blocking occurs. For a high capacity the system behaves nearly like a system without blocking.

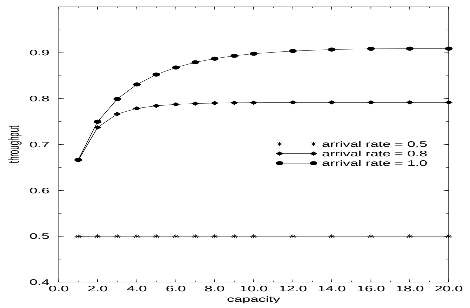

Figure 7: Results for finite buffer

3.3 Systems with Unreliable Machines

3.3.1 Description of the System with Unreliable Machines

In general we have to assume unreliable machines for a production system. This means a machine can go down and has to be repaired. During this time no job can be served and all jobs have to wait in the queue but jobs can still arrive at the system and have to wait at the queue. It is assumed that if a machine goes down the job just in service has to be served again. To indicate if a machine is still available the two states *up* and *down* are introduced.

3.3.2 Specification of a System with Unreliable Machines

The core of the specification in MOSLANG for a system with unreliable machines is shown in Fig. 8. Here node_1 (consisting of queue_1 and cpu_1) is an unreliable node while node_2 is a regular M/M/1 node.

```
1  RULE {state_cpu_1 == up} ->
         state_cpu_1 = down : 1/mtbf : 1.0      ;
2  RULE {state_cpu_1 == down} ->
         state_cpu_1 = up : 1/mttr : 1.0        ;
3  RULE {queue_1 > 0}{cpu_1 == idle}
       {state_cpu_1 == up} ->
         cpu_1 = busy : immediate : 1.0         ;
4  RULE {cpu_1 == busy}{state_cpu_1 == up} ->
         cpu_1 = wait : 1/service_time_1 : 1.0 ;
5  RULE {cpu_1 == wait}{state_cpu_1 == up} ->
         queue_1 -= 1 node_2 += 1
         cpu_1 = idle : immediate : 1.0         ;
6  RULE {node_2 > 0} ->
         node_2 -= 1 num -= 1 :
         1/service_time_2 : 1.0                 ;
```

Figure 8: MOSES core for an unreliable machine

These lines can be interpreted as follows:

Line 1: To simulate the failure of a machine the state of the machine is set to *down* after an exponentially distributed time $mtbf$.

Line 2: If a machine is *down*, it is repaired with rate $1/mttr$ and the state of the machine is set to *up* again.

Line 3: As soon as there are jobs in *queue_1* and *cpu_1* is *idle* and the machine is *up* then *cpu_1* is set *busy*.

Line 4: Again the pseudo state *wait* is used to serve the job. If during the service time the machine goes down, the job stays in the cpu and has to wait there until the machine is repaired.

Line 5: If the job has been served successfully then it is moved to *node_2*.

Line 6: This rule again describes that a job is served at *node_2*.

3.3.3 Graphical Representation of the Results

In the plot shown in Fig. 9 the mean repair time of a node is varied from 100 to 1000 (time units) while the mean time between two failures is 2000. All other system parameters are the same as in the example before. The system throughput is then plotted as a function of the mean repair time.

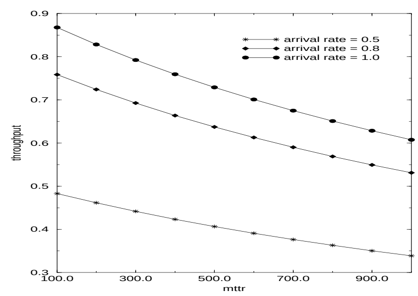

Figure 9: Results for unreliable machines

3.4 Combination of the Presented Node Types

Up to now we have assumed that a node is either unreliable or has finite buffer or uses a certain batch strategy. In reality normally these characteristics appear in all possible combinations. We want to show now how to specify nodes with these combined characteristics. As before at first the core of the specification is introduced and shortly explained and afterwards some plots are presented.

3.4.1 Full Batch and Unreliable Machines

In this case we have the full batch service strategy in combination with an unreliable machine. The corresponding core of the MOSES specification is shown in Fig. 10. Here node_1 (consisting of queue_1 and cpu_1) is an unreliable node with full batch service strategy while node_2 is a regular M/M/1 node.

```
1  RULE {state_cpu_1 == up} ->
        state_cpu_1 = down : 1/mtbf : 1.0       ;
2  RULE {state_cpu_1 == down} ->
        state_cpu_1 = up : 1/mttr : 1.0         ;
3  RULE {queue_1 >= b}{cpu_1 == idle}
       {state_cpu_1 == up} ->
        cpu_1 = busy : immediate : 1.0          ;
4  RULE {cpu_1 == busy}{state_cpu_1 == up} ->
        cpu_1 = wait : 1/service_time_1 : 1.0 ;
5  RULE {cpu_1 == wait}{state_cpu_1 == up} ->
        queue_1 -= b node_2 += b
        cpu_1 = idle : immediate : 1.0          ;
6  RULE {node_2 > 0} ->
        node_2 -= 1 num -= 1 :
        1/service_time_2 : 1.0                  ;
```

Figure 10: MOSES core for an unreliable machines with full batch strategy

As can easily be seen, the main structure of this specification is basically the same as the two single specifications of the characteristics. The only difference is that if we want to use *cpu_1* we first have to check the state of *cpu_1* if it is *up* or *down*. This is done in lines 3, 4 and 5. If *cpu_1* is *up* then the system behaves like the two single nodes (full batch, finite buffer). But if on the other side *cpu_1* is *down* then no job can move from *cpu_1* to *node_2* and also no job can be processed at *cpu_1*. After an exponentially distributed time *mttr* *cpu_1* is repaired again and can work.

3.4.2 Graphical Representation of the Results

In Fig. 11 the system throughput is given as a function of the batch size b. As in the case of a full batch service strategy, an optimal batch size b can be found so that the system throughput is maximal. The following parameters were chosen:

service_time_1	=	8.0	K	=	20
service_time_2	=	0.2	arrival	=	1.0
mtbf	=	2000			

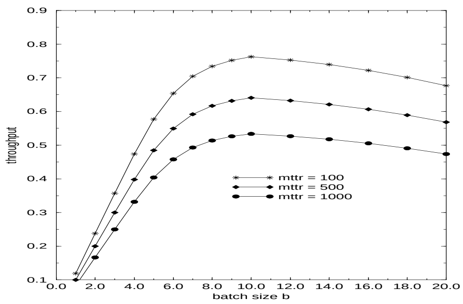

Figure 11: Results for full batch and unreliable machines

3.4.3 Minimum Batch and Finite Buffer

The core of the MOSES specification for this combination is shown in Fig. 12. Here node_1 (consisting of queue_1 and cpu_1) is a regular M/M/1 node while node_2 (consisting of queue_2 and cpu_2) is a minimum batch node with finite buffer.

```
1  RULE {queue_1 > 0}{cpu_1 == idle} ->
        cpu_1 = busy queue_1 -= 1 :
        immediate : 1.0                          ;
2  RULE {cpu_1 == busy} ->
        cpu_1 = wait : 1/service_time_1 : 1.0    ;
3  RULE {cpu_1 == wait}{queue_2 < capacity} ->
        cpu_1 = idle queue_2 += 1 :
        immediate : 1.0                          ;
4  RULE {queue_2 >= a}{cpu_2 == idle} ->
        cpu_2 = busy : immediate : 1.0           ;
5  RULE {cpu_2 == busy} ->
        cpu_2 = wait  : 1/service_time_2 : 1.0   ;
6  RULE {cpu_2 == wait}{queue_2 >= b} ->
        queue_2 -= b num += b cpu_2 = idle  :
        immediate : 1.0                          ;
7  RULE {cpu_2 == wait}{queue_2 > 0}
       {queue_2 < b} ->
        queue_2 -= 1 num += 1 : immediate : 1.0 ;
8  RULE {cpu_2 == wait}{queue_2 == 0} ->
        cpu_2 = idle : immediate : 1.0           ;
```

Figure 12: MOSES core for a finite buffer with minimum batch strategy

As can be seen the only difference between this specification and the original two specifications is the expression $\{queue\_2 < capacity\}$ which determines that as soon as the capacity of the blocking node is exceeded the corresponding node is blocked and the job has to wait in the server.

3.4.4 Graphical Representation of the Results

The results for this combination of characteristics are shown in Fig. 13. The following parameters were chosen:

service_time_1	=	1.0	K	=	20
service_time_2	=	1.0	b	=	10
capacity	=	15			

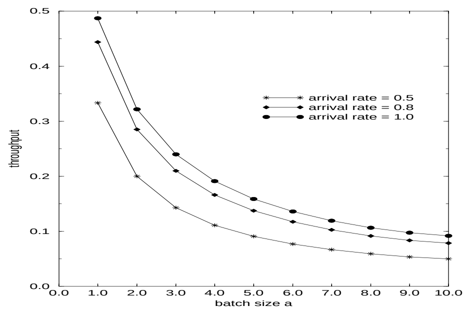

Figure 13: Results for minimum batch and blocking

4 Example of a Production System

With these simple specifications in mind it is now easy to specify even larger production systems using any kind of combinations of the presented characteristics. We chose for example the combined characteristics presented above. This means our system consists of three nodes where node_1 is a combination of full batch and unreliable node, node_2 is a combination of minimum batch and finite buffer and the third node is a regular M/M/1 node. The following parameters are used to analyze the system:

service_time_1	=	8.0	K	=	20
service_time_2	=	1.0	a	=	5
service_time_3	=	0.1	b	=	5...20
mttr	=	500	mtbf	=	2000
capacity	=	16			

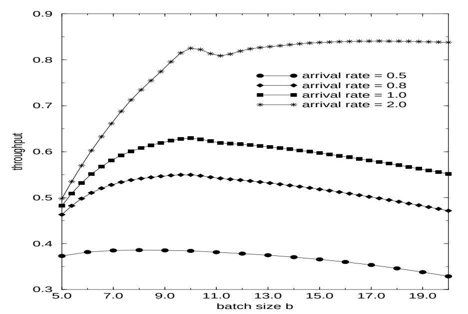

Figure 14: Results for the production system

5 Conclusion

With the presented analysis tool MOSES it is possible to specify and analyze production systems with unreliable machines, finite buffers and batch processing. The modeler is free to combine these and other system characteristics as necessary. It is of course possible to specify not only single nodes but also more complex systems with all different kinds of nodes (blocking, batch, M/M/1, M/M/m, Infinite server, different distributions at the nodes, ...). Currently macros are developed for production systems (blocking, batch type, unreliable machines, M/M/m nodes, Infinite server nodes, different distributions, ...) to simplify the specification process. The MOSLANG specification of the node is then generated automatically. Also a new front end language called MOSEL (MOdeling Specification and Evaluation Language) is developed at the moment [5]. This new front end language makes system description much easier and understandable compared to MOSES since it reflects the system structure much better than MOSES. As shown, it is very easy to specify production systems using our new tool. If we consider for example a production system with several parallel unreliable machines and finite buffers, then we can specify this system with a minimum amount of lines of code while the state space is very big. In this sense MOSES allows a very elegant system description. The analysis of each of the presented models took less than one second. Because of this speed it is very easy to do a several program runs to optimize the system (find for example the optimal batch size, ...). MOSES is therefore very well suited for applications in the area of production system analysis.

References

[1] Alvarez-Vargas R., Dallery Y. and David R.: *A Study of the Continuous Flow Model of Production Lines with Unreliable Machines and Finite Buffers*, Journal of Manufacturing Systems, Volume 13/No.13.

[2] Bolch, G.: *Leistungsbewertung von Rechensystemen mittels analytischer Warteschlangenmodelle*, Teubner, Stuttgart, 1989. Springer, Berlin, 1965.

[3] Bolch G., Greiner S., Jung H., Zimmer R.: *The Markov Analyzer MOSES*, Technical Report TR-I4-10-94, Institute of Mathematical Machines and

[4] Bolch G., Greiner S., Roessler M.: *Modellierung und Analyse Komplexer Systeme unter Verwendung des Markov-Analysetools MOSES* , Published at the journal "it+ti–Informationstechnik und technische Informatik", pp 28-33, Vol. 37, 3/95.

[5] Bolch G., Herold H.: *MOSEL–A New Language for the Markov Analyzer MOSES* Technical Report TR-I4-95-02, Institute of Mathematical Machines and Data Processing IV University Erlangen-Nuernberg, March 1995.