MOSEL
MOdeling Specification and Evaluation Language

Gunter Bolch
Helmut Herold

March 1995 TR-I4-95-02

Technical Report

Computer
Science Department

Operating Systems - IMMD IV

Friedrich-Alexander-University
Erlangen-Nürnberg, Germany

MOSEL

MOdeling Specification and Evaluation Language

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

Helmut Herold
Siemens AG
PD 3 TS
Zeppelinstrasse 10
D - 91052 Erlangen
ea296@fim.uni-erlangen.de

Abstract

In this paper the new model description language MOSEL (**Mo**deling **S**pecification and **E**valuation **L**anguage) - developed at the Institute for Operating Systems at the University of Erlangen-Nuernberg - is described using many examples. To evaluate the performance of systems, like computer systems, networks, production lines and so on, the system has to be specified. Currently there already exist many different performance evaluation tools, which all have different input languages. The input languages for these tools are not only different, but they are also hard to use, since they all have a syntax which is oriented on the characteristics of the tool and not on the system which the user wants to describe. They are comparable with assembly languages for specific processors. The basic idea of the new language MOSEL is, to have only one language which is easy to use and reflects the real structure of the system the user wants to describe. MOSEL could be compared with a higher programming language which is problem and no longer machine orientated. MOSEL will be a common frontend for many performance evaluation tools. The user has only to become familiar with MOSEL and has not to learn all the different tool specific languages to use these tools. After the user has specified its system in MOSEL, the MOSEL-Compiler will produce the appropiate input description for the specific tools. By giving options the user may specify which tools he wants to use. Nevertheless MOSEL also decides which tools are applicable for the given system specification. Currently MOSEL can produce input descriptions for the tools MOSES [MOSES94], which is a Markov analyzer, and the tool SPNP [SPNP91, TRIV91], which is a Stochastic Petri Net Package. It is planned to use MOSEL as a common description language for other already implemented performance evaluation tools, like PEPSY [PEPSY94] or SHARPE [SHARPE86]. The MOSEL-Compiler is implemented in the programming language C by using the scanner and parser tools lex and yacc [HERO95].

1 Introduction

System performance evaluation becomes more and more important with growing system complexity. The analysis of already existing systems often fails because of the costs that this would require. This is one reason for the growing importance of tools which uses models to analyze systems. Such evaluation tools like MOSES or SPNP become even more valuable during the development of a system, where measurement is not possible at all.
This paper concentrates on the description of the language MOSEL in its currently existing and already implemented state. It is very likely that MOSEL will undergo some minor changes while the development of this language goes further and the usability to other already implemented analyzing tools like PEPSY [PEPSY94] or SHARPE [SHARPE86] is tested. Nevertheless it is unlikely that there will be major changes in the basical structure of the language MOSEL as it is described in this paper.
In the next sections we first give short descriptions of MOSLANG, which is the input language of the tool MOSES, and of CSPL, which is the input language for the tool SPNP, before we give a more detailed desription of the new language MOSEL and the calling syntax of the MOSEL-Compiler. Finally, a lot of examples demonstrate the facilities of the new language MOSEL. The complete syntax of the new language MOSEL can be found in the Appendix where it is given in a combination of Backus-Naur-Form and meta characters of UNIX regular expressions.

2 The Model Description Language MOSLANG for MOSES

MOSLANG was developed at the Institute for Operating Systems at the University of Erlangen-Nuernberg and its usability has been proved for many large and complicated systems [MOSES94]. The Markov analyzer MOSES facilitates the input of MOSLANG and subsequently creates the Markovian system of equations automatically. For solving this system of equations currently six different methods are provided. MOSES calculates the state probabilities and derives from them the performance measures which are currently specified using 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. A detailled description of MOSLANG can be found in [MOSES94].

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 parameters or performance measures must be specified here also.

Vector description part

In this part the dimension and the range of the state space is fixed.
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 constructions.
The RULE construction consists of a condition and an action part. If all conditions are fulfiled, the specified actions will be executed.
The action part also contains the transition rate and the transition probability.
One RULE construction specifies several state transitions. It is not necessary to specify an individual RULE construction for every state transition.
It is possible to use *immediate transitions*. These are executed immediately, when the conditions are fulfiled. Thus modeling preemption and priorities is simple. Immediate transitions are characterized by the transition rate -1.0. By using immediate transitions one achieves a decoupling of the different nodes on the system. This causes a very powerful compression of the model description.

Results part

In the results part the performance measures are specified.
The results part consists of RESULT constructions, RESULT MEAN constructions, RESULT DIST constructions and FINAL constructions. In the RESULT construction the basic performance measures which can be deduced directly from the state probabilities are computed. In the RESULT MEAN construction and in the RESULT DIST construction the average number of jobs in a station and the probability of each possible queue length in a station can be determined. In the FINAL construction performance measures which are functions of the basic performance measures and the system parameters, for example waiting times, utilization, throughputs, can be computed.

3 The Model Description Language CSPL for SPNP

CSPL (*C-based Stochastic Petri net Language*) is the input language for the tool SPNP (*Stochastic Petri Net Pakkage*) which was developed at the Duke University of Durham in North Carolina and its usability has also been proved for many large and complicated systems [SPNP91, TRIV91]. The package is composed of several C files. The SPN to be studied must be described in a CSPL file, which is a C file specifying the structure of the SPN (Stochastic Petri Net) and the desired outputs, by means of predefined functions. The CSPL file is compiled, linked to the other files constituting the package (usually kept compiled). The syntax and semantic of CSPL is widely equivalent to the programming language C. CSPL only differs from C in that way that it offers a set of predefined functions, data types and constants, which are needed for the implementation of a SPN. In this section only a short description of CSPL is given. A detailled description of CSPL can be found in [SPNP91,TRIV91].

3.1 Predefined functions

A CSPL file must specify the following functions: *parameters*, *net*, *assert*, *ac_init*, *ac_reach* and *ac_final*.

parameters

In the function *parameters* the user may set certain options or specify parameters which have to be specified at run time.

net

In the function *net* the user has to specify the complete structure of the SPN. Therefore predefined functions exist, like

place	defines a place in the petri net
init	initializes the number of tokens in a certain place
trans	defines a transition in the petri net
iarc	defines an input arc for a transition
oarc	defines an output arc for a transition
harc	defines an inhibitor arc from a transition to a place
miarc	defines an input arc with multiplicity for a transition
moarc	defines an output arc with multiplicityfor a transition
mharc	defines a multiple inhibitor arc from a transition to a place
rateval	defines the rate of a transition as a constant value
ratedep	defines the rate of a transition which depends on the number of tokens in a place
ratefun	defines the rate of a transition as a function
probval	defines the probability of a transition as a constant value
probdep	defines the probability of a transition which depends on the number of tokens in a place
probfun	defines the probability of a transition as a function
priority	defines the priority of a transition
guard	defines an enabling function for a transition
enbaling	defines also an enabling function for a transition

......

assert

The function *assert* allows the evaluation of a logical condition on a marking of the SPN. *assert* is called during the reachability graph construction to check the validity of each newly found marking. It must return RES_ERROR if the marking is illegal or RES_NOERR if the marking is legal.

ac_init

The function *ac_init* is called before starting the reachability graph construction. To get data about the SPN, the function *pr_net_info* should be called in this function.

ac_reach

The function *ac_reach* is called after the reachability graph construction has completed. To get data about the reachability graph, the function *pr_rg_info* should be called in this function.

ac_final

In this function the user may specify the output measures which should be calculated. Therefore predefined functions are available, like

pr_expected	prints the expected value of a so called reward function, which the user must define
pr_value	prints the value of specified variable
expected	returns the expected value of a so called reward function, which the user must define

.....

3.2 Predefined data types

CSPL knows four predefined data types:

enabling_type	is defined as **int**; it is the return type for enabling functions
probability_type	is defined as **double**; it is the type for probability variables or functions
rate_type	is defined as **double**; it is the type for rate variables or functions
reward_type	is defined as **double**; it is the type for reward variables or functions

3.3 Predefined constants

There are a lots of predefined constants which are options for the SPNP. These options may be set in the function *parameters* by using the predefined functions *iopt* (for integer options) or *fopt* (for double options).

4 The New Language MOSEL

The new language MOSEL (**Mo**deling **S**pecification and **E**valuation **L**anguage) consists of a series of constructs which can be summarized in 6 parts

1. Declaration part
2. Vector description part
3. Rules part
4. Result part
5. Preprocessing directives
6. Comments.

Whereas the first four parts must be given in the above order, the last two parts (*Preprocessing directives* and *Comments*) can be located at any place in the MOSEL input file. In this chapter the semantic of each part is briefly explained. A complete description of the MOSEL syntax can be found in the Appendix where the syntax of MOSEL is given in a combination of Backus-Naur-Form and the UNIX meta characters for regular expressions. Further sections of this chapter describe additional features of MOSEL like *loops, parameterizing of macros* and *multidimensional nodes*.

4.1 Declaration part

In this optional part *constants* may be defined and *variables* may be declared.

4.1.1 Constant definitions

This optional part contains constants, which can be defined in two different ways:

#define *identifier value*

This kind of definition is similar to the corresponding construction in the programming language C except that it allows only numerical values (integer and floating point) and no expressions or other kind of text replacement.

enum *enum_name* **{** *enumconst*[**=***int_value*] [[,] *enumconst*[**=***int_value*]]... **};**

This construction allows the definition of a whole set of constants which are members of the self-defined data type called *enum_name*. If one doesn't assign any integer value to an constant name in the enumeration list, then this constant gets as default value an integer, which is one higher than the value of the predecessor in the given list. The default value for the first constant is 0.
For example, the following two enum declarations are both identical

```
enum  cpu_state { idle=0, user=1, kernel=2, driver=3 };
enum  cpu_state { idle, user, kernel, driver };
```

and the following enum-declaration would result in three constants (a=0, b=20, c=21):

```
enum  abc  { a, b=20, c };
```

The naming rules for *identifiers*, *enum_name* and *enumconst* corresponds exactly to the naming rules of C. Whereas at **#define** double and integer constants are allowed, in the enum-list only integer values are allowed.

4.1.2 Variable declarations

This optional part contains the variable declarations. All used variable names can be declared here. It must be stated that this is not necessary in MOSEL since this tool generates automatically the declarations for all used variables. MOSEL determines the kind and type of the variables from the context in them they are used. Nevertheless the user has also the possibility to declare its variables:

VAR	for input variables
OUTPUT	for output variables
HELP	for help variables

As type can be used **int**, **double** or **prob** (probability type). For example:

```
VAR     int     K;        /* declares an input variable K of type int */
OUTPUT  double  lambda;   /* declares an output variable lambda of type double */
```

An initialization of a variable is also possible, like

```
VAR int   K=5;
```

4.1.3 Function declarations

In the declaration part also Functions may be declared in the same way as that is done in the programming language C. Such functions may be used to specify rates, probabilities or output measures to be calculated. This feature is not yet implemented but this will be done soon.

4.2 Vector description part

This part consists of the *NODE part*, the *START part* and the *NOT part*.

4.2.1 NODE part

This part specifies the components of the state vector. For each component the name and its capacity must be given. For example

```
NODE  driverq[K];              /* Range: 0..K */
NODE  cpu[cpu_state];          /* Range: 0..3;(idle, user, kernel, driver); see  4.1.1*/
NODE  cpuactive[user..kernel]; /* Range: 1..3; (user, kernel, driver); see 4.1.1 */
NODE  abcnode[a..c];           /* Range: 0..21  */
```

These four lines declare four nodes (*driverq*, *cpu*, *cpuactive*, *abcnode*) with the ranges given in the comment. Using these ranges, the state space can be automatically generated by MOSEL. As we have seen, there are a lot of possibilities for the specification of the capacity. In the following all these possibilities are summarized

- *const_name* Range: 0..*const_name* (value of *const_name*; if *const_name* is a member of an enum type, then as lower value not 0 but the minimal constant of that enum type is taken)·
- *enum_name* Range: *min_value*..*max_value* (of *enum_name*)·
- *int_nr* Range: 0..*int_nr*·
- *int_nr1*..*int_nr2* Range: *int_nr1*..*int_nr2*·
- *int_nr*..*const_name* Range: *int_nr*..*const_name* (*const_name*-value as upper boundary)·
- *const_name*..*int_nr* Range: *const_name*..*int_nr* (*const_name*-value as lower boundary)·
- *const_name1*..*const_name2* Range: *const_name1*.. *const_name2* (values of these constants)

If the specification of the capacity contains an identifier which is neither a constant name nor an enum name, MOSEL assumes that this name must be an input name and will automatically generate a corresponding variable declaration (*VAR int*) for that name, if that name is not already declared by the user.

At the declaration of a node its initialization also is possible. Therefore only the corresponding value or arithmetic expression must be assigned to the node. For example:

```
NODE driverq[K] = 2;
NODE cpu[cpu_state] = kernel;
```

The initial value or expression specifies the start state of that node. If for the same node in the START part (see below) also a value or expression is specified, this START value/expression will be ignored by MOSEL.

There exists also the possibility to define subnodes in a NODE-Declaration, like

```
NODE  N1[0..K; Phase1[1]; Phase2[1]];    /* Node is a M/E2/1 server */
```

Subnodes can be used for example for components with Erlang-distribution. Later we will see an example for this (see section 7.3). Initialization of the subnodes is also possible, like

```
NODE  N1[0..K; Phase1[1]=1; Phase2[1]=0] = 5;    /* Node is a M/E2/1 server */
```

If the user does not specify any capacity for a node, MOSEL assumes infinite capacity (currently 100), e.g.

```
NODE  driverq;
NODE  userq;
```

corresponds to the following NODE declarations:

```
NODE  driverq[100];
NODE  userq[100];
```

IF the user does not want this *infinite capacity* for nodes without a capacity, it may set the predefined constant **DEFAULT_CAPACITY** to a value which should be taken instead of *infinite capacity* (100), e.g.

```
#define DEFAULT_CAPACITY  K

NODE  driverq;
NODE  userq;
```

would correspond to the following NODE declarations:

```
NODE  driverq[K];
NODE  userq[K];
```

4.2.2 START part

This optional part specifies a valid start state and facilitates the building of the graph which is describing the Markovian model. The start states must be given in the same order as the NODE declarations and the start states may be specified by values, constant names, variables or arithmetic expressions. If the user specifies less values than nodes, MOSEL assumes for the remaining states (nodes) their minimum values.
If two different start values are specified for a node: one at the node declaration (initialization) and one in the START part, the START value will be ignored. That means that an initialization in a node declaration has higher priority than a START specification for that node.

4.2.3 NOT part

This optional construct allows the specifying of prohibited states. That means that these states are not allowed to appear in the Markov chain. If any state in the state space fulfils the NOT construct then it is a prohibited state for the specified system. The user can specify as many NOT constructs as necessary. A NOT construct consists of one or more conditions which are implicitly ANDed by MOSEL. That means that the whole condition is true when all single conditions of the NOT construct are true. The keyword AND may be used to connect single conditions what actually is not necessary but makes the specification better readable.

4.3 Rules part

In this part the rules which determine the behaviour of the system are specified. Each rule consists of a global part and optionally of a local part which must be enclosed in curly parentheses. For example the following specification shows three rules. The first two rules only consist of a global part, whereas the third rule also has a local part with three local rules:

```
FROM user  TO kernel  W 1.0/usertime  IF state_CPU==up_CPU;

FROM N1  TO Phase1;
```

```
FROM kernel  W 1.0/kerneltime  IF state_CPU==up_CPU  {
       TO user                   P 1.0-pio-pdone;
       TO driver                 P pio;
       TO idle  TO kernelq       P pdone;
}
```

The first rule specifies a transition from node *user* to node *kernel* with the transition rate *1.0/usertime* if the condition *state_CPU==up_CPU* is true.
The second rule specifies an immediate transition from node *N1* to node *Phase1*.
The third rule specifies a couple of transitions. First of all it specifies in the global part that the transition source is the node kernel with the transition rate *1.0/kerneltime*. In the three local rules it specifies the targets of the transitions with different transition probabilities. The third local rule specifies even two targets. Since in the local rules no new transition rate is specified, the transition rate (*1.0/kerneltime*) of the global part is taken. Also the globally specified condition (*state_CPU==up_CPU*) is taken for the local rules. It would be also possible to specify additional conditions for the single local rules. For example, MOSEL would generate the following five MOSLANG rules from these three MOSEL-rules.

```
RULE  {cpu == user} {state_CPU==up_CPU}  ->
         cpu = kernel  : 1.0/usertime : 1.0;

RULE  {N1 >= 1} {Phase1 <= 0}  ->
         N1 -= 1 Phase1 += 1  : -1.0 : 1.0;

RULE  {cpu == kernel} {state_CPU==up_CPU}  ->
         cpu = user  : 1.0/kerneltime : 1.0-pio-pdone;
RULE  {cpu == kernel} {state_CPU==up_CPU}  ->
         cpu = driver  : 1.0/kerneltime : pio;
RULE  {cpu == kernel} {state_CPU==up_CPU} {kernelq <= K-1}  ->
         cpu = idle kernelq += 1  : 1.0/kerneltime : pdone;
```

The keywords **FROM** and **TO** induce MOSEL to generate the appropriate actions and conditions in the corresponding tool languages. For example the following specification

```
NODE  driverq[K1];
NODE  kernelq[K2];
NODE  cpu[cpu_state];
.....
FROM kernel   TO idle  TO kernelq;
FROM kernelq  TO driverq;
```

will be translated in the following MOSLANG specification by MOSEL:

```
RULE  {cpu == kernel} {kernelq <= K2-1}  ->
         cpu = idle kernelq += 1  : -1.0 : 1.0 ;

RULE  {kernelq >= 1} {driverq <= K1-1}  ->
         driverq += 1 kernelq -= 1  : -1.0 : 1.0 ;
```

For most of the applications these automatically generated checks and movements are exactly what the user wants but there might be also situations where this automatical checks or movements should be switched off. Therefore the other keywords **FROMC**, **FROMM**, **FROME**, **TOC**, **TOM** and **TOE** exist.
The keywords **FROMM** and **TOM** can be used in cases, where the automatically generated check for enough elements in a "FROM-Node" or for enough place in a "TO-node" should be switched off. This is for example the case in the following rule, where an explicit condition for the node active (*active==2*) is given:

```
FROMM active  TO waitq  W 1*mue  IF active==2 AND working==1;
```

For this rule it would be redundant when MOSEL would generate its automatic check *active>=1*, since the user has already given an explicit check for *active*. For cases like this the keywords **FROMM** or **TOM** (**M** stands for

"***M**ove only without automatic check*") should be used.

As counterpart to **FROMM** and **TOM** exist also the keywords **FROMC** and **TOC** (**C** stands for "***C**heck only without movement*") which causes MOSEL to generate only the appropriate checks (conditions) but no movements (actions) for the corresponding node.

Besides these keywords there exists also the keywords **FROME** and **TOE** (**E** stands for "***E**xternal node*"). **FROME** or **TOE** must be used when a transition from or to an external node takes place. Thereby an arbitrary node name may be used which should not be declared in the NODE-part. The names after **FROME** and **TOE** can also be omitted.
Table 1 summarizes the effects of the different FROM/TO keywords.

	automatically generated check (condition)	automatically generated movement (action)
FROM/TO	yes	yes
FROMC/TOC	yes	no
FROMM/TOM	no	yes
FROME/TOE	no	no

Table 1: Effects of the different FROM/TO keywords

Additionally it is possible to specify the transition of an arbitrary number of elements between the nodes. Therefore a parenthesized number or variable must follow the corresponding node name. If no number is specified, 1 is the default. For example the following rule

```
FROM kernelq(5)  TO driverq(7);
```

will be translated into the following MOSLANG rule by MOSEL

```
RULE  {kernelq >= 5} {driverq <= K1-7}  ->
          driverq += 7 kernelq -= 5  : -1.0 : 1.0 ;
```

Besides numbers there are also variables allowed to specify the transition of an arbitrary number of elements between nodes. For example the following rule

```
FROM kernelq(x)  TO driverq(y);
```

will be translated into the following MOSLANG rule by MOSEL

```
RULE  {kernelq >= x} {driverq <= K1-y}  ->
          driverq += y kernelq -= x  : -1.0 : 1.0 ;
```

If the user has to specify more than one FROM- or TO-nodes in a rule, like in

```
FROM a(1)  FROM b(var)  TO x(10) TO y(tovar) TO z   IF count == 5
```

he has not always to repeat the corresponding FROM and TO-keywords but can specify these keywords only once and then give the list of the corresponding node names, like

```
FROM a(1) b(var)  TO x(10) y(tovar) z   IF count == 5
```

For better readabilility commas may be used to spearate the single names in the according list, like

```
FROM a(1),b(var)  TO x(10), y(tovar), z   IF count == 5
```

Often enum names or enum constant names are used to specify the range of a node, like

```
enum cpu_state { idle, user, kernel, driver };
....
NODE  cpu[cpu_state];
```

In cases like that not the real node name must be given after a FROM/TO-keyword but one of the enum constant names, like

```
FROM kernel  TO idle  TO queue;
```

which will be translated e.g. to following MOSLANG output

```
RULE  {cpu == kernel} {queue <= K-1}  ->
          cpu = idle queue += 1  : -1.0 : 1.0;
```

Sometimes identical enum names or enum constant names are used in the range specification of different nodes, like

```
NODE  cpu1[cpu_state];
NODE  cpu2[cpu_state];
```

In cases like that it is not sufficient to specify an enum constant name after a FROM/TO-keyword but it must also the corresponding node be specified, like

```
FROM cpu2[kernel]  TO cpu2[idle]  TO queue;
```

which for example will be translated in the following MOSLANG construct

```
RULE  {cpu2 == kernel} {queue <= K-1}  ->
          cpu2 = idle queue += 1  : -1.0 : 1.0;
```

This construction could also be used for the previous example (where it is not necessary):

```
FROM cpu[kernel]  TO cpu[idle]  TO queue;
```

In the global part and also in the local part additionally priorities can be specified. It must be stated that priorities doesn't have any effect on the MOSLANG output, but only on the CSPL output. There exist two possibilities to specify priorities for transitions or probabilities:

- By using the keyword **PRIO**, e.g.

  ```
  FROM A  TO B  PRIO 10;
  FROM A  TO B {
              FROM X  TO Y  PRIO a;
              FROM U  TO V;
  }
  ```

  In this case, all immediate transitions for the rules or local rules, where **PRIO** is specified, get the specified priority (value or variable) in the CSPL file.

- By specifying the appropriate priority (value or variable) enclosed in brackets behind the transition rate or probability, like

  ```
  FROM A  TO B  W mue[10];

  FROM A  TO B {
              FROM X  TO Y  W mue1[a]  P p11[100];
              FROM U  TO V  W mue2     P p12[20];
  }
  ```

In this case, the corresponding timed transitions which will be generated for the rates (**W**) get the specified priority (e.g. *a*). Accordingly the corresponding immediate transitions which will be generated for the probabilities (**P**) get also the user defined priority (e.g. *100* and *20*). The keyword **PRIO** can never override such a locally specified transition or probability priority.

If no priorities are specified, MOSEL always sets the priority for immediate transitions to 10000 in the CSPL file. Timed transitions don't get any priority by default.

4.4 Result part

In this optional part the basic performance measures are specified. The result part consists of RESULT constructions, RESULT MEAN constructions and RESULT DIST constructions. In the RESULT construction the basic performance measures which can be deduced directly from the state probabilities are computed. In the RESULT MEAN construction and in the RESULT DIST construction the average number of jobs in a node and the probability of each possible queue length in a station can be determined. If computations are depending on certain conditions, these conditions can be specified similar to the if-construction in the programming language C:

```
IF (condition1 AND condition2)  { action1 action2 }
```

Whereas the keyword AND and the curly parentheses around the actions are optional, the parentheses around the condition part must always be specified.

Performance measures which shall be given as output on the screen or in a file must be specified using the keyword **RESULT>>** for each performance measure. The syntax of this construct is identical to the **RESULT** syntax except that here also single variables (which were be calculated in a RESULT constructions) can be specified.

4.5 Preprocessor Directives

MOSEL allows the definitions of text macros. A macro definition has the following form:

```
#string  macroname  macrotext:
```

The naming rule for a *macroname* corresponds exactly to the naming rules of C.
The macrotext starts at the first non white character (white characters = blanks and tabs) and ends with a colon (**:**).
In the *macrotext* all characters are allowed, e.g. also blanks, newlines or semicolons.
On places where the *macrotext* should be inserted only the *macroname* with a preceeding dollar sign (**$**) must be given:

$*macroname*

MOSEL uses an own preprocessing pass which only purpose it is to perform such macro expansion.

In a macrotext also the replacement of other previously defined macros is possible, e.g.

```
#string CONDI1    cpu == idle :
#string CONDI2    $CONDI1  driverq == 0 :
#string RULE1     FROM a  TO b  IF $CONDI2 :
#string RULE2     FROM x  TO y  $RULE1:
```

After these macro definitions the following line

```
FROM node_1 $RULE1;
```

would be expanded by the preprocessor of MOSEL to the following MOSEL specification:

```
FROM node_1 FROM a  TO b  IF cpu == idle  driverq == 0;
```

and the following line

```
FROM node_2 $RULE2;
```

would be expanded to

```
FROM node_2  FROM x  TO y  FROM a  TO b  IF cpu == idle  driverq == 0;
```

Macro replacement (with **$***macroname*) is only possible for already defined macros. That means that macros must always be defined before they can be called (with **$***macroname*).

Macro definitions can be placed at any location in the MOSEL input file.

When calling a macro also arguments for that macro can be specified (see also section 4.8)

4.6 Comments

Comments must - like in the programming language C - be enclosed in **/* .. */**. Comments can be placed at any place in a MOSEL file. There are two special cases for comments:

- If a comment starts with **/****, MOSEL places this comment before the appropriate construct in its generated output file.
- If a comment starts with **/*****, MOSEL considers that comment as a global comment. MOSEL places such global comments always at the beginning in its generated output file.

All other comments which start only with **/*** will be ignored by MOSEL.

Besides C-Comments which are enclosed in **/* .. */** and can be used to comment out more than one line also the C++ comment sign **//** may be used when only one line or the remainder of a line should be commented out. Such C++ like comments will always be ignored by MOSEL and - like a /*..*/ comment - not be taken to output file.

4.7 Loops in MOSEL

Some specifications happens to be very similar. For situation like that, MOSEL offers a new construct, the so called loops. This section describes the main topics of loops.

4.7.1 Loop indices

For example, if the following specifications have to be made:

```
FROM N2  TO N1  W mue2 ;
FROM N3  TO N1  W mue3 ;
FROM N4  TO N1  W mue4 ;
```

it is very cumbersome to repeat every single specification which always differs from the previous rule just in a number. With the new available loops these applications could be shortened to

```
<2..4>  FROM N#  TO N1  W mue# ;
```

As one can see, the loop indices must be enclosed in <...> and must be specified at the beginning of the corresponding specification. The loop indices can be specified in three different ways:

int_nr	for an index *int_nr*
int_nr1..int_nr2	for the index range from *int_nr1..int_nr2*
identifier	for an index *identifier*

Instead of *int_nr* also integer constant names may be used. In this case the integer value of this constant will be taken by MOSEL.

It is also possible to specify a list of such loop indices in which case the single specifications of indices must be separated by comma. For example, the following loop construct

```
<2..5, user, kernel, 10..12, 15>
```

defines the indices *2, 3, 4, 5, user, kernel, 10, 11, 12, 15.*
The number of indices determines how many specifications result from such a specification.

The character # can be used in names to specify that this name will change in every generated specification. The character # is thereby replaced by the actual index of the loop.

4.7.2 Nested loops

MOSEL also allows nested loops. Nested loops can be specified by giving more than one loop list. For example, the following loop construct

```
<1..4> <user, kernel, driver> <_> <a,b>
```

results in the following indices

```
1user_a, 1user_b, 1kernel_a, 1kernel_b, 1driver_a, 1driver_b,
2user_a, 2user_b, 2kernel_a, 2kernel_b, 2driver_a, 2driver_b,
3user_a, 3user_b, 3kernel_a, 3kernel_b, 3driver_a, 3driver_b,
4user_a, 4user_b, 4kernel_a, 4kernel_b, 4driver_a, 4driver_b
```

That means that 24 (4*3*1*2) specifications results from the one specification where this loop construct is used. In each of these 24 specification one of the above indices is used to replace the character # in the names.

4.7.3 Calculations in Loops

Usually the replacement of # by indices is just a text replacement. Nevertheless there exists also the possibility to make calculations by indices. For example, the following specification

```
<1..4>  FROM N#   TO N<#+1>;
```

is a short form for the following four specifications

```
FROM N1   TO N2;
FROM N2   TO N3;
FROM N3   TO N4;
FROM N4   TO N5;
```

As one can see, to let MOSEL perform calculation, the # with the expression in the names must be enclosed in <...>.

4.7.4 Partial replacement

Sometimes it is useful to replace only a certain index of the loop and not the whole composed index string, e.g.

```
<a,b,c><1..3> NODE  node_# [K<#2>];
```

which corresponds to the following MOSEL specifications

```
NODE  node_a1[K1];
NODE  node_a2[K2];
NODE  node_a3[K3];
NODE  node_b1[K1];
NODE  node_b2[K2];
NODE  node_b3[K3];
NODE  node_c1[K1];
NODE  node_c2[K2];
NODE  node_c3[K3];
```

As one can see, to let MOSEL replace only a certain index and not the whole composed actual index text, the required index number preceeded by # must be enclosed in <...>.

4.7.5 Constructs in which loops are allowed

It does not make sense, to allow loops for each MOSEL construct. Therefore loop lists are only allowed before the following keywords.

enum
#define
VAR, OUTPUT, HELP
NODE
NOT
RESULT
RESULT>>

Loop lists are also allowed at the beginning of each global or local rule, or as an argument when calling a macro (see also section 4.8)

4.8 Calling Macros with a loop list argument

It is also possible to specify loop arguments when calling a macro. In this case the macrotext will be inserted at the calling place as often as specified in the loop argument, whereby all #-signs in the macrotext will be replaced by the actual index of the loop, e.g.

```
#string nodes node_# :

FROM node_a  TO $nodes(<1..5>)
```

would be expanded by MOSEL to

```
FROM node_a  TO node_1 node_2 node_3 node_4 node_5
```

The next more complex example shows more advanced features of the parameterizing in macro calls, like nested parameterized macros.

```
#define n     5

#string queues queue#(cc) :
#string K_queue K#(n) $queues(<2..n><a,b,c>) :
#string nodes  node_# :
#string in_queues  queue_# :

<2..n><a,b,c>  NODE queue#[K];
               NODE node_a[K] = 0;
       <1..n>  NODE node_#[K] = 0;
       <1..n>  NODE queue_#[K] = 0;
               NODE node_b[K] = 0;
  <1..3><x,y>  NODE K#[k];

FROM $K_queue(<1..3><x,y>) TO node_1;

FROME  TO node_a  W arrival;

        FROM node_a  TO $nodes(<1..n>)  W mue_a;
<1..n>  FROM node_#  TO queue_#  W mue_#;
        FROM $in_queues(<1..n>)  TO node_b;
        FROM node_b  TOE  W mue_b;

        RESULT    IF (node_b > 0) rho_b += PROB;
        RESULT>>  throughput = rho_b * mue_b;
```

This MOSEL input would be expanded by the MOSEL preprocessor to the following

```
#define n     5



<2..n><a,b,c>  NODE queue#[K];
               NODE node_a[K] = 0;
       <1..n>  NODE node_#[K] = 0;
       <1..n>  NODE queue_#[K] = 0;
               NODE node_b[K] = 0;
  <1..3><x,y>  NODE K#[k];

FROM  #line 15  K1x(n)  queue2a(cc)  queue2b(cc)  queue2c(cc)  queue3a(cc)  queue3b(cc)
queue3c(cc)  queue4a(cc)  queue4b(cc)  queue4c(cc)  queue5a(cc)  queue5b(cc)  queue5c(cc)
K1y(n)  queue2a(cc)  queue2b(cc)  queue2c(cc)  queue3a(cc)  queue3b(cc)  queue3c(cc)
queue4a(cc)  queue4b(cc)  queue4c(cc)  queue5a(cc)  queue5b(cc)  queue5c(cc)   K2x(n)
queue2a(cc)  queue2b(cc)  queue2c(cc)  queue3a(cc)  queue3b(cc)  queue3c(cc)  queue4a(cc)
queue4b(cc)  queue4c(cc)  queue5a(cc)  queue5b(cc)  queue5c(cc)   K2y(n)  queue2a(cc)
queue2b(cc)  queue2c(cc)  queue3a(cc)  queue3b(cc)  queue3c(cc)  queue4a(cc)  queue4b(cc)
queue4c(cc)  queue5a(cc)  queue5b(cc)  queue5c(cc)   K3x(n)  queue2a(cc)  queue2b(cc)
queue2c(cc)  queue3a(cc)  queue3b(cc)  queue3c(cc)  queue4a(cc)  queue4b(cc)  queue4c(cc)
queue5a(cc)  queue5b(cc)  queue5c(cc)   K3y(n)  queue2a(cc)  queue2b(cc)  queue2c(cc)
queue3a(cc)  queue3b(cc)  queue3c(cc)  queue4a(cc)  queue4b(cc)  queue4c(cc)  queue5a(cc)
queue5b(cc)  queue5c(cc)   #line 15  TO node_1;

FROME  TO node_a  W arrival;

        FROM node_a  TO  #line 19  node_1  node_2  node_3  node_4  node_5  #line 19   W
mue_a;
<1..n>  FROM node_#  TO queue_#  W mue_#;
        FROM  #line 21  queue_1  queue_2  queue_3  queue_4  queue_5  #line 21   TO node_b;
        FROM node_b  TOE  W mue_b;

        RESULT    IF (node_b > 0) rho_b += PROB;
        RESULT>>  throughput = rho_b * mue_b;
```

The **#line**-construct is here generated by MOSEL for error reporting purposes. Thus MOSEL also knows after the preprocessing pass to which original line the current line corresponds.

4.9 Multidimensional nodes

Sometimes it happens that states belong together like in

```
enum cpu_state {idle, busy, wait};
enum updown    {up, down};

NODE cpu_1[cpu_state];
NODE cpu_2[updown];
```

In situations like that, a description reflects the real structure of the system much more, when the user specifies a multidimensional node, like

```
enum cpu_state {idle, busy, wait};
enum updown    {up, down};

NODE cpu[cpu_state][updown];
```

Both specifications result in the same output. It is also possible to define multidimensional nodes which do not have enum constants as ranges, like

```
NODE queue[K1][K2][K3];
```

When multidimensional nodes are used, a number of restrictions must be observed:

- A START part is not allowed. Initialization at node declaration should be used instead, like
```
NODE cpu[cpu_state][updown] = idle, up;
```

- Multidimensional nodes with ranges of enum names/constants and numerical indices are not allowed in one node declaration, e.g. for
```
NODE cpu[cpu_state][K];
```
MOSEL would report an error.

- In a multidimensional node declaration the same enum names or constants of the same enum name must not be used in different ranges, e.g.
```
NODE cpu[cpu_state][cpu_state];
```
is not allowed. Instead of that different node declarations must be used, like
```
NODE cpu1[cpu_state];
NODE cpu2[cpu_state];
```

- Whereas nodes with enum names or enum constants as ranges may be indexed in rules, an indexing of nodes with numerical ranges is not allowed, like
```
enum cpu_state {idle, user, kernel, driver };
NODE cpu[cpu_state];
NODE kernelq[K1][K2];
NODE driverq[K1][K2];

FROM cpu[idle]  TO cpu[kernel]  W 1/kerneltime;   /* is allowed */
FROM kernelq[5]  TO driverq[4][8]  W 1/x;       /* is not allowed */
```

- Multidimensional nodes with enum names or enum constants as ranges must always be indexed in rules, like
```
enum cpu_state {idle, user, kernel, driver };
enum updown {up, down};

NODE cpu[cpu_state][updown];
NODE kernelq[K1][K2];
NODE driverq[K1][K2];

FROM cpu[idle][up]  TO cpu[kernel]  W 1/kerneltime;    /* is allowed */
FROM cpu  TO cpu                                /* is not allowed */
FROM kernelq  TO driverq  W 1/x;                /* is  allowed */
FROM kernelq[5]  TO driverq[4][8]  W 1/x;       /* is not allowed */
```

- For each multidimensional node MOSEL generates the corresponding number of one dimensional nodes. The basenames of these nodes are identical to the name of the multidimensional node, but each onedimensional gets a suffix, which is _ (underscore) followed by the dimension number, e.g.
```
enum cpu_state {idle, user, kernel, driver };
enum updown {up, down};

NODE cpu[cpu_state][updown];
NODE kernelq[K1][K2];
NODE driverq[K1][K2][K3];
```

  is identical to to the following specification

```
enum cpu_state {idle, user, kernel, driver };
enum updown {up, down};

NODE cpu_1[cpu_state];
NODE cpu_2[updown];
NODE kernelq_1[K1];
NODE kernelq_2[K2];
NODE driverq_1[K1];
NODE driverq_2[K2];
NODE driverq_3[K3];
```

Knowing this convention makes is possible, to name a certain dimension of a node, e.g.

```
enum cpu_state {idle, user, kernel, driver };
enum updown {up, down};

NODE cpu[cpu_state][updown];
NODE kernelq[K1][K2];
NODE driverq[K1][K2][K3];

FROM cpu_2[up]  TO cpu_2[down]  W 1/mtbf;
       /* also possible: FROM cpu[up]  TO cpu[down]  W 1/mtbf; */
FROM kernelq_2  TO driverq_1  W 1/kerneltime;
....
RESULT>> (IF kernelq_2 > 0) rhoq_2 += PROB;
```

It must be stated that this naming convention is only valid for multidimensional nodes. Onedimensional nodes always get exactly the name that was given at declaration time.

The next complex and artifical example shows numerous kinds of multidimensional and nested nodes, and how they can be used. It also shows, how multidimensional nodes can be initialized.

```
enum noise         {calm, loud, music};
enum price         {cheap, expensive};
enum wheather      {sunny, rainy, cloudy};
enum temperature   {hot, warm, cold};

NODE europe[ 1000;
             germany[100] [ 50;
                           bavaria[ K;
                                    munich[20]=2;
                                    pub[noise][price][temperature]=loud,cheap,hot
                                  ]=1
                          ]=10,5
           ]=100;

NODE festival[wheather][temperature];
NODE X[2..10][4..20]= 3,5;
NODE Y[100..1000][200..2000][300..3000] = 500, 1500, 2500;

FROM sunny  TO rainy, festival[hot], X                           ;
FROM pub[cold]  TO pub[warm]                                     ;
FROM europe  TO germany  W 1/a                                   ;
FROM bavaria  TO munich                                          ;
FROM X(5)  TO Y(600)                                             ;
FROM pub[hot][music]  TO pub[warm]                               ;
FROM Y TO X                                                      ;
FROM X_2 TO Y_1(10),Y_3(20)                                      ;
FROM pub[loud][expensive] TO pub[calm][cheap][warm]              ;
FROM pub_1[music], pub_3[cold]   TO pub_1[calm], pub_3[hot]      ;
```

This MOSEL input would result e.g. in the following MOSLANG output:

```
                                           /*--- generated from `xxx.ein' ---*/
/*=========== Nodes ====================*/
#define  europe      VAL[0]
#define  bavaria     VAL[1]
#define  munich      VAL[2]
#define  germany_1   VAL[3]
#define  germany_2   VAL[4]
#define  pub_1       VAL[5]
#define  pub_2       VAL[6]
#define  pub_3       VAL[7]
#define  festival_1  VAL[8]
#define  festival_2  VAL[9]
#define  X_1         VAL[10]
#define  X_2         VAL[11]
#define  Y_1         VAL[12]
#define  Y_2         VAL[13]
#define  Y_3         VAL[14]

/*=========== Constants =================*/
#define  calm        0
#define  loud        1
#define  music       2
#define  cheap       0
#define  expensive   1
#define  sunny       0
#define  rainy       1
#define  cloudy      2
#define  hot         0
#define  warm        1
#define  cold        2

/*=========== Variables =================*/
VAR  int      K;
VAR  double   a;  /* double-Variable assumed */


/*=========== state space =================*/
DIM    15;
MIN   0        0        0        0        0        calm      cheap     hot       sunny
hot      2         4         100       200       300       ;
MAX   1000     K        20       100      50       music     expensive cold      cloudy
cold      10        20        1000      2000      3000      ;

/*=========== START =======================*/
START  100      1        2        10       5        loud      cheap     hot       sunny
hot       3         5         500       1500      2500      ;

/*========================== RULES ==========================*/
                                        /*............ MOSEL-Rule 1 .....*/
RULE  {festival_1 == sunny} {X_1 <= 9} {X_2 <= 19}  ->
          festival_1 = rainy festival_2 = hot X_1 += 1 X_2 += 1  : -1.0 : 1.0;
                                        /*............ MOSEL-Rule 2 .....*/
RULE  {pub_3 == cold}  ->
          pub_3 = warm  : -1.0 : 1.0;
                                        /*............ MOSEL-Rule 3 .....*/
RULE  {europe >= 1} {germany_1 <= 99} {germany_2 <= 49}  ->
          europe -= 1 germany_1 += 1 germany_2 += 1  : 1/a : 1.0;
                                        /*............ MOSEL-Rule 4 .....*/
RULE  {bavaria >= 1} {munich <= 19}  ->
          bavaria -= 1 munich += 1  : -1.0 : 1.0;
                                        /*............ MOSEL-Rule 5 .....*/
RULE  {X_1 >= 7} {X_2 >= 9} {Y_1 <= 400} {Y_2 <= 1400} {Y_3 <= 2400}  ->
          X_1 -= 5 X_2 -= 5 Y_1 += 600 Y_2 += 600 Y_3 += 600  : -1.0 : 1.0;
                                        /*............ MOSEL-Rule 6 .....*/
RULE  {pub_3 == hot} {pub_1 == music}  ->
          pub_3 = warm  : -1.0 : 1.0;

                 /*............ MOSEL-Rule 7 .....*/
```

```
RULE  {Y_1 >= 101} {Y_2 >= 201} {Y_3 >= 301} {X_1 <= 9} {X_2 <= 19}  ->
          Y_1 -= 1 Y_2 -= 1 Y_3 -= 1 X_1 += 1 X_2 += 1  : -1.0 : 1.0;
                                        /*........... MOSEL-Rule 8 .....*/
RULE  {X_2 >= 5} {Y_1 <= 990} {Y_3 <= 2980}  ->
          X_2 -= 1 Y_1 += 10 Y_3 += 20  : -1.0 : 1.0;
                                        /*........... MOSEL-Rule 9 .....*/
RULE  {pub_1 == loud} {pub_2 == expensive}  ->
          pub_1 = calm pub_2 = cheap pub_3 = warm  : -1.0 : 1.0;
                                        /*........... MOSEL-Rule 10 .....*/
RULE  {pub_1 == music} {pub_3 == cold}  ->
          pub_1 = calm pub_3 = hot  : -1.0 : 1.0;
```

5 Calling syntax of the MOSEL compiler

If you call MOSEL without any arguments it displays the calling syntax on the screen, like

```
                                    ../feb1/src/mosel-Version: 0.1
---------------------------------------------------------------
usage: ../feb1/src/mosel [option(s)] moselfile

   Options:
      -c           create CSPL-File (extension '.c')
      -m           create MOSLANG-File (extension '.spec')
      -o outfile   if this option is not specified,
                    the output file gets the same name as the
                    input file with the appropriate extension
      -h           prints this help information
      -p           preprocessing output is kept in a file
                    (preprocessing output file gets the same name as the
                     input file with the extension '.pp')
      -d           dumps the parser results
      -v           prints the version number
---------------------------------------------------------------
```

It must be stated that the combination of more than one option also is possible, like

mosel -mc queuenet.mos   or
mosel -m -c queuenet.mos

Both calls tell MOSEL to generate a MOSLANG file (*queuenet.spec*) and a CSPL file (*queuenet.c*). If the user wants to specify an own name for the output file, it must use the option **-o**, like

mosel -m queuenet.mos -o queuenet.mos.spec or
mosel -c queuenet.mos -o queuenet.cspl

If the user specifies more than one tool and specifies an output filename, MOSEL will not write both generated outputs to the specified output file, but will produce two different files, like in

mosel -mc queuenet.mos -o queuenet

MOSEL will write the MOSLANG specification to the required output file *queuenet* and the CSPL specification to the file *queuenet.c* (which will be reported to the user).

If a required output cannot be produced, MOSEL will report this. MOSEL even reports the reasons why it is not possible, e.g.

```
MOSLANG output not possible, because
  - HELP variables are used in superm.ein
  - variable probabilities are used together with immediate rates in superm.ein
........no output file produced
```

6 Restrictions and rules for the different tools

Since MOSEL is a general language for a number of tools. There exist MOSEL constructs and features which are not supported by certain tools. Currently the following rules and restrictions for the different tool languages are known.

6.1 Restrictions in MOSLANG

6.1.1 No Functions

If functions are declared in the MOSEL input file, MOSEL will not generate a MOSLANG output, since there is no possibility to specify functions in MOSLANG.

6.1.2 No HELP variables

If **HELP** variables are declared in the MOSEL input file, MOSEL will not generate a MOSLANG output, since there is no possibility to specify **HELP** variables in MOSLANG. In MOSLANG only input and output variables exist.

6.1.3 No Initializing of variables

If variables are initialized in the MOSEL input file, MOSEL will not generate a MOSLANG output, since there is no possibility to initialize variables in MOSLANG.

6.1.4 Expressions for probabilities

If a rule contains a probability which is specified as an expression, MOSEL will not generate a MOSLANG output, since there is no possibility to specify expressions for probabilities in MOSLANG.

6.1.5 Variable probability with an immediate transition

If a rule contains at the same time a variable probability (not a constant) but no transition, MOSEL will not generate a MOSLANG output, since there is no possibility to specify such a construct in MOSLANG.

6.1.6 Expressions in START construct or at node initialization

If a START construct or an initialization at a node declaration contains an expression and not only a variable or a constant, MOSEL will not generate a MOSLANG output, since there is no possibility to specify such a construct in MOSLANG.

6.2 Rules for CSPL

6.2.1 Rules with a common rate and different probabilities

If transitions have a common rate but there are different probabilities, these probabilities must be specfied in one rule by using local rules with a global rate, e.g.

```
FROM kernel  W 1.0/kerneltime {
       TO user                  P 1.0-pio-pdone;
       TO driver                P pio;
       TO idle  TO kernelq      P pdone;
}
```

If this specification would be given by using different global rules like in

```
FROM kernel  TO user  W 1.0/kerneltime  P 1.0-pio-pdone;
FROM kernel  TO driver  W 1.0/kerneltime  P pio;
FROM kernel  TO idle  TO kernelq  W 1.0/kerneltime  P pdone;
```

the generated CSPL file would not provide the correct results. By the way, in MOSLANG it doesn't matter if such a specification is given by using local or global rules.

6.2.2 Global rates with local FROMs

If a global rate is specified then MOSEL will report an error when in local rules additional FROMs are given, like in

```
FROM kernel  W 1.0/kerneltime {
       FROM userq TO user       P 1.0-pio-pdone;
       TO driver                P pio;
       TO idle  TO kernelq      P pdone;
}
```

This restrictions hold only for the CSPL and not for the MOSLANG generation.

6.2.3 Node initialization in closed networks

Whereas in open systems an initialization of nodes is not necessary, in closed networks an initialization of the corresponding nodes is necessary. Otherwise, there wouldn't exist any tokens in the closed network and the petri net would never fire (move any tokens). Thus, the calculated results cannot be correct.

6.2.4 Naming conventions

CSPL uses own names which have an underscore (_) at the beginning and MOSEL uses names for internally created places and variables with an underscore (_) at the end. Therefore the user should not use names which begin or end with an underscore (_) to avoid conflicts with names which are internally created by MOSEL or CSPL.

7 Examples

Here we use numerous examples to demonstrate the new Model description language MOSEL.

7.1 The Central-Server-Model

As first example we take a Central-Server-Model as it is given in [BOLCH89]. This example contains N nodes and K jobs. The service time of a job in the node i (i=1,2,3,4) is exponentially distributed.
By using MOSEL the model from Fig. 1 can be specified in many different ways. We first show a longer version where no loops are used at all, before we give a much more shorter MOSEL description for the same central server model. Both specifications will result in the same MOSLANG or CSPL output which are shown at the end of this section.

Fig. 1: Central-Server-model

7.1.1 A long MOSEL version for the Central-Server-Model

Here we describe the Central-Server-model from Fig. 1 by using MOSEL. In the following listing we do not use any loops nor do we declare any variables. All used variables will be automatically detected by MOSEL and correspondingly declared in the MOSLANG output.

```
/***====== Central-Server-model ==================*/

/**------------- States -------------------------------------------*/
NODE N1[K]=K;
NODE N2[K];
NODE N3[K];
NODE N4[K];

/**------------- Prohibited states ---------------------------------*/
NOT  N1 + N2 + N3 + N4 != K;

/*------------- Rules --------------------------------------------*/
FROM N1  WITH mue1 {
     TO N2  P p12;
     TO N3  P p13;
     TO N4  P p14;
}

FROM N2  TO N1  WITH mue2;
FROM N3  TO N1  WITH mue3;
FROM N4  TO N1  WITH mue4;

/**------------- RESULTs -------------------------------------------*/
RESULT>>  IF (N1>0) rho1 += PROB;
RESULT>>  IF (N2>0) rho2 += PROB;
RESULT>>  IF (N3>0) rho3 += PROB;
RESULT>>  IF (N4>0) rho4 += PROB;

RESULT  nquer1 = MEAN N1;
```

```
RESULT  nquer2 = MEAN N2;
RESULT  nquer3 = MEAN N3;
RESULT  nquer4 = MEAN N4;

RESULT  DIST N1;
RESULT  DIST N2;
RESULT  DIST N3;
RESULT  DIST N4;

RESULT>>  lambda1 = rho1 * mue1;
RESULT>>  lambda2 = rho2 * mue2;
RESULT>>  lambda3 = rho3 * mue3;
RESULT>>  lambda4 = rho4 * mue4;

RESULT>>  tquer1 = nquer1 / lambda1;
RESULT>>  tquer2 = nquer2 / lambda2;
RESULT>>  tquer3 = nquer3 / lambda3;
RESULT>>  tquer4 = nquer4 / lambda4;
```

7.1.2 A short MOSEL version for the Central-Server-Model

Here we describe the same Central-Server-model in MOSEL by using loops. Thus we get a much shorter MOSEL description file.

```
/***====== Central-Server-model ===================*/

/**------------- States --------------------------------------------*/
       NODE N1[K]=K;
<2..4> NODE N#[K];

/**------------- Prohibited states ---------------------------------*/
NOT  N1 + N2 + N3 + N4 != K;

/*------------- Rules ----------------------------------------------*/
FROM N1  WITH mue1 {
     <2..4> TO N#  P p1#;
}

<2,3,4> FROM N#  TO N1  WITH mue#;

/**------------- RESULTs --------------------------------------------*/
<1..4>  RESULT>>  IF (N#>0) rho# += PROB;
<1..4>  RESULT    nquer# = MEAN N#;
<1..4>  RESULT    DIST N#;
<1..4>  RESULT>>  lambda# = rho# * mue#;
<1..4>  RESULT>>  tquer# = nquer# / lambda#;
```

7.1.3 The generated MOSLANG file

For both of the previous description files MOSEL produces - except for the automatically generated comments - the same MOSLANG output.

```
                                        /*--- generated from 'bol109.ein' ---*/
/***====== Central-Server-model ===================*/

/**------------- States --------------------------------------------*/
#define  N1          VAL[0]
#define  N2          VAL[1]
#define  N3          VAL[2]
#define  N4          VAL[3]

/*=========== Variables ================*/
VAR  int      K;
VAR  double   mue1;  /* double-Variable assumed */
```

```
VAR  double   mue2;  /* double-Variable assumed */
VAR  double   mue3;  /* double-Variable assumed */
VAR  double   mue4;  /* double-Variable assumed */
VAR  prob     p12;
VAR  prob     p13;
VAR  prob     p14;

OUTPUT  double  rho1;  /*  double assumed */
OUTPUT  double  rho2;  /*  double assumed */
OUTPUT  double  rho3;  /*  double assumed */
OUTPUT  double  rho4;  /*  double assumed */
OUTPUT  double  nquer1;  /*  double assumed */
OUTPUT  double  nquer2;  /*  double assumed */
OUTPUT  double  nquer3;  /*  double assumed */
OUTPUT  double  nquer4;  /*  double assumed */
OUTPUT  double  lambda1;  /*  double assumed */
OUTPUT  double  lambda2;  /*  double assumed */
OUTPUT  double  lambda3;  /*  double assumed */
OUTPUT  double  lambda4;  /*  double assumed */
OUTPUT  double  tquer1;  /*  double assumed */
OUTPUT  double  tquer2;  /*  double assumed */
OUTPUT  double  tquer3;  /*  double assumed */
OUTPUT  double  tquer4;  /*  double assumed */

/*=========== state space =================*/
DIM    4;
MIN    0         0         0         0         ;
MAX    K         K         K         K         ;

/*=========== START =======================*/
START  K         0         0         0         ;

/**------------- Prohibited states ---------------------------------*/
NOT  {N1+N2+N3+N4 != K};

/*========================== RULES ===========================*/
                                            /*........... MOSEL-Rule 1 .....*/
RULE  {N1 >= 1} {N2 <= K-1}  ->
          N1 -= 1 N2 += 1  : mue1 : p12;
RULE  {N1 >= 1} {N3 <= K-1}  ->
          N1 -= 1 N3 += 1  : mue1 : p13;
RULE  {N1 >= 1} {N4 <= K-1}  ->
          N1 -= 1 N4 += 1  : mue1 : p14;
                                            /*........... MOSEL-Rule 2 .....*/
RULE  {N2 >= 1} {N1 <= K-1}  ->
          N2 -= 1 N1 += 1  : mue2 : 1.0;
                                            /*........... MOSEL-Rule 3 .....*/
RULE  {N3 >= 1} {N1 <= K-1}  ->
          N3 -= 1 N1 += 1  : mue3 : 1.0;
                                            /*........... MOSEL-Rule 4 .....*/
RULE  {N4 >= 1} {N1 <= K-1}  ->
          N4 -= 1 N1 += 1  : mue4 : 1.0;

/**------------- RESULTs -------------------------------------------*/
RESULT  {N1 > 0} -> rho1 += PROB ;
RESULT  {N2 > 0} -> rho2 += PROB ;
RESULT  {N3 > 0} -> rho3 += PROB ;
RESULT  {N4 > 0} -> rho4 += PROB ;
RESULT  nquer1 = MEAN N1 ;
RESULT  nquer2 = MEAN N2 ;
RESULT  nquer3 = MEAN N3 ;
RESULT  nquer4 = MEAN N4 ;
RESULT  DIST N1 ;
RESULT  DIST N2 ;
RESULT  DIST N3 ;
RESULT  DIST N4 ;

FINAL   rho1;
FINAL   rho2;
```

```
FINAL   rho3;
FINAL   rho4;
FINAL   lambda1 = rho1*mue1;
FINAL   lambda2 = rho2*mue2;
FINAL   lambda3 = rho3*mue3;
FINAL   lambda4 = rho4*mue4;
FINAL   tquer1 = nquer1/lambda1;
FINAL   tquer2 = nquer2/lambda2;
FINAL   tquer3 = nquer3/lambda3;
FINAL   tquer4 = nquer4/lambda4;
```

7.1.4 The generated CSPL file

For both of the previous description files MOSEL produces - except for the automatically generated comments - the same CSPL output.

```
                                        /*--- generated from 'bol109.ein' ---*/
/***====== Central-Server-model ===================*/

/*========== global part ================================================*/
#include   <sys/types.h>
#include   <stdio.h>
#include   <stdlib.h>
#include   <ctype.h>
#include   <string.h>
#include   "user.h"

/*========== Variables ==================================================*/

            /*------------------- INPUT-Variables ---*/
int                 K;
double              mue1;  /* double-Variable assumed */
double              mue2;  /* double-Variable assumed */
double              mue3;  /* double-Variable assumed */
double              mue4;  /* double-Variable assumed */
probability_type    p12;
probability_type    p13;
probability_type    p14;

            /*------------------- OUTPUT-Variables ---*/
reward_type         rho1;  /*  reward_type assumed */
reward_type         rho2;  /*  reward_type assumed */
reward_type         rho3;  /*  reward_type assumed */
reward_type         rho4;  /*  reward_type assumed */
reward_type         nquer1;  /*  reward_type assumed */
reward_type         nquer2;  /*  reward_type assumed */
reward_type         nquer3;  /*  reward_type assumed */
reward_type         nquer4;  /*  reward_type assumed */
reward_type         lambda1;  /*  reward_type assumed */
reward_type         lambda2;  /*  reward_type assumed */
reward_type         lambda3;  /*  reward_type assumed */
reward_type         lambda4;  /*  reward_type assumed */
reward_type         tquer1;  /*  reward_type assumed */
reward_type         tquer2;  /*  reward_type assumed */
reward_type         tquer3;  /*  reward_type assumed */
reward_type         tquer4;  /*  reward_type assumed */

            /*------------ DISTRIBUTION-Variables ---*/
double              dist_N1_;
double              dist_N2_;
double              dist_N3_;
double              dist_N4_;


/*========== enabling-functions ==========================================*/
enabling_type  enable_1_(void)      /*-------------------  enable_1_ ---*/
{
```

```
   if (mark("N1")>=1)
      return(1);
   else
      return(0);
}

enabling_type  enable_1_1_(void)      /*------------------  enable_1_1_ ---*/
{
   if (mark("N2")<=K-1)
      return(1);
   else
      return(0);
}

enabling_type  enable_1_2_(void)      /*------------------  enable_1_2_ ---*/
{
   if (mark("N3")<=K-1)
      return(1);
   else
      return(0);
}

enabling_type  enable_1_3_(void)      /*------------------  enable_1_3_ ---*/
{
   if (mark("N4")<=K-1)
      return(1);
   else
      return(0);
}

enabling_type  enable_2_(void)      /*------------------  enable_2_ ---*/
{
   if (mark("N2")>=1 &&
       mark("N1")<=K-1)
      return(1);
   else
      return(0);
}

enabling_type  enable_3_(void)      /*------------------  enable_3_ ---*/
{
   if (mark("N3")>=1 &&
       mark("N1")<=K-1)
      return(1);
   else
      return(0);
}

enabling_type  enable_4_(void)      /*------------------  enable_4_ ---*/
{
   if (mark("N4")>=1 &&
       mark("N1")<=K-1)
      return(1);
   else
      return(0);
}

/*========== reward-functions ============================================*/
reward_type  resultfunc_1_(void)       /*--------------  resultfunc_1_ ---*/
{
   if (mark("N1")>0) {
      return(1);
   } else {
      return(0);
   }
}

reward_type  resultfunc_2_(void)       /*--------------  resultfunc_2_ ---*/
{
```

```
   if (mark("N2")>0) {
      return(1);
   } else {
      return(0);
   }
}

reward_type  resultfunc_3_(void)       /*--------------  resultfunc_3_ ---*/
{
   if (mark("N3")>0) {
      return(1);
   } else {
      return(0);
   }
}

reward_type  resultfunc_4_(void)       /*--------------  resultfunc_4_ ---*/
{
   if (mark("N4")>0) {
      return(1);
   } else {
      return(0);
   }
}

reward_type  resultfunc_5_(void)       /*--------------  resultfunc_5_ ---*/
{
   return(mark("N1") );
}

reward_type  resultfunc_6_(void)       /*--------------  resultfunc_6_ ---*/
{
   return(mark("N2") );
}

reward_type  resultfunc_7_(void)       /*--------------  resultfunc_7_ ---*/
{
   return(mark("N3") );
}

reward_type  resultfunc_8_(void)       /*--------------  resultfunc_8_ ---*/
{
   return(mark("N4") );
}

reward_type  resultfunc_9_(void)       /*--------------  resultfunc_9_ ---*/
{
   if (mark("N1") == dist_N1_)
      return(1.0);
   else
      return(0.0);
}

reward_type  resultfunc_9__(void)       /*-------------  resultfunc_9__ ---*/
{
   return(mark("N1"));
}

reward_type  resultfunc_10_(void)       /*--------------  resultfunc_10_ ---*/
{
   if (mark("N2") == dist_N2_)
      return(1.0);
   else
      return(0.0);
}

reward_type  resultfunc_10__(void)       /*-------------  resultfunc_10__ ---*/
{
   return(mark("N2"));
```

```
}

reward_type  resultfunc_11_(void)       /*--------------  resultfunc_11_ ---*/
{
   if (mark("N3") == dist_N3_)
      return(1.0);
   else
      return(0.0);
}

reward_type  resultfunc_11__(void)       /*-------------  resultfunc_11__ ---*/
{
   return(mark("N3"));
}

reward_type  resultfunc_12_(void)       /*--------------  resultfunc_12_ ---*/
{
   if (mark("N4") == dist_N4_)
      return(1.0);
   else
      return(0.0);
}

reward_type  resultfunc_12__(void)       /*-------------  resultfunc_12__ ---*/
{
   return(mark("N4"));
}

/*========== parameters ==================================================*/
void parameters(void)
{
   K = input("K");
   mue1 = input("mue1");
   mue2 = input("mue2");
   mue3 = input("mue3");
   mue4 = input("mue4");
   p12 = input("p12");
   p13 = input("p13");
   p14 = input("p14");
}

/*========== net =========================================================*/
void net(void)
{
   /**------------- States --------------------------------------------*/
   place("N1");    init("N1", K);
   place("N2");
   place("N3");
   place("N4");

   /*----------- transforming rules ---------------------------*/
                                        /*............ MOSEL-Rule 1 .....*/
      /* FROM N1  W mue1   {
            TO N2  P p12
            TO N3  P p13
            TO N4  P p14   }  */
   trans("timedtrans_1_1_");
    rateval("timedtrans_1_1_", mue1);
   place("ghelp_place_1_1_");
    oarc("timedtrans_1_1_", "ghelp_place_1_1_");
    harc("timedtrans_1_1_", "ghelp_place_1_1_");
    iarc("timedtrans_1_1_", "N1");
    enabling("timedtrans_1_1_", enable_1_);
   trans("immedtrans_1_1_");
    priority("immedtrans_1_1_", 10000);
    probval("immedtrans_1_1_", p12);
    enabling("immedtrans_1_1_", enable_1_1_);
    iarc("immedtrans_1_1_", "ghelp_place_1_1_");
    oarc("immedtrans_1_1_", "N2");
```

```
   trans("immedtrans_1_2_");
    priority("immedtrans_1_2_", 10000);
    probval("immedtrans_1_2_", p13);
    enabling("immedtrans_1_2_", enable_1_2_);
    iarc("immedtrans_1_2_", "ghelp_place_1_1_");
    oarc("immedtrans_1_2_", "N3");
   trans("immedtrans_1_3_");
    priority("immedtrans_1_3_", 10000);
    probval("immedtrans_1_3_", p14);
    enabling("immedtrans_1_3_", enable_1_3_);
    iarc("immedtrans_1_3_", "ghelp_place_1_1_");
    oarc("immedtrans_1_3_", "N4");
                                    /*........... MOSEL-Rule 2 .....*/
      /* FROM N2  TO N1  W mue2  */
   trans("timedtrans_2_");
    rateval("timedtrans_2_", mue2);
    iarc("timedtrans_2_", "N2");
    oarc("timedtrans_2_", "N1");
    enabling("timedtrans_2_", enable_2_);
                                    /*........... MOSEL-Rule 3 .....*/
      /* FROM N3  TO N1  W mue3  */
   trans("timedtrans_3_");
    rateval("timedtrans_3_", mue3);
    iarc("timedtrans_3_", "N3");
    oarc("timedtrans_3_", "N1");
    enabling("timedtrans_3_", enable_3_);
                                    /*........... MOSEL-Rule 4 .....*/
      /* FROM N4  TO N1  W mue4  */
   trans("timedtrans_4_");
    rateval("timedtrans_4_", mue4);
    iarc("timedtrans_4_", "N4");
    oarc("timedtrans_4_", "N1");
    enabling("timedtrans_4_", enable_4_);
}

/*========== assert ======================================================*/
int assert(void)
{
   /**------------- Prohibited states ---------------------------------*/
   int   returnval;

   if ((mark("N1")+mark("ghelp_place_1_1_"))+mark("N2")+mark("N3")+mark("N4")!=K)
      returnval = RES_ERROR;
   else
      returnval = RES_NOERR;
   if (returnval == RES_ERROR)
      return(RES_ERROR);

   return(returnval);
}

/*========== ac_init =====================================================*/
void ac_init(void)
{
   pr_net_info();
}

/*========== ac_reach ====================================================*/
void ac_reach(void)
{
   pr_rg_info();
}

/*========== ac_final ====================================================*/
void ac_final(void)
{
   int  i;
   /**------------- RESULTs -------------------------------------------*/
     rho1 += expected(resultfunc_1_);
```

```
    rho2 += expected(resultfunc_2_);
    rho3 += expected(resultfunc_3_);
    rho4 += expected(resultfunc_4_);
    nquer1 = expected(resultfunc_5_);
    nquer2 = expected(resultfunc_6_);
    nquer3 = expected(resultfunc_7_);
    nquer4 = expected(resultfunc_8_);
  pr_expected("__________MEAN N1", resultfunc_9__);
  for (i=0; i<=K; i++) {
     char  string[100];
     sprintf(string, "DIST N1[%d]", i);
     dist_N1_ = i;
     pr_expected(string, resultfunc_9_);
  }
  pr_expected("__________MEAN N2", resultfunc_10__);
  for (i=0; i<=K; i++) {
     char  string[100];
     sprintf(string, "DIST N2[%d]", i);
     dist_N2_ = i;
     pr_expected(string, resultfunc_10_);
  }
  pr_expected("__________MEAN N3", resultfunc_11__);
  for (i=0; i<=K; i++) {
     char  string[100];
     sprintf(string, "DIST N3[%d]", i);
     dist_N3_ = i;
     pr_expected(string, resultfunc_11_);
  }
  pr_expected("__________MEAN N4", resultfunc_12__);
  for (i=0; i<=K; i++) {
     char  string[100];
     sprintf(string, "DIST N4[%d]", i);
     dist_N4_ = i;
     pr_expected(string, resultfunc_12_);
  }
  pr_value("rho1", rho1);
  pr_value("rho2", rho2);
  pr_value("rho3", rho3);
  pr_value("rho4", rho4);
    lambda1 = rho1 * mue1 ;
  pr_value("lambda1", lambda1);
    lambda2 = rho2 * mue2 ;
  pr_value("lambda2", lambda2);
    lambda3 = rho3 * mue3 ;
  pr_value("lambda3", lambda3);
    lambda4 = rho4 * mue4 ;
  pr_value("lambda4", lambda4);
    tquer1 = nquer1 / lambda1 ;
  pr_value("tquer1", tquer1);
    tquer2 = nquer2 / lambda2 ;
  pr_value("tquer2", tquer2);
    tquer3 = nquer3 / lambda3 ;
  pr_value("tquer3", tquer3);
    tquer4 = nquer4 / lambda4 ;
  pr_value("tquer4", tquer4);
}
```

7.2 Net with more than one job class

Sometimes it makes sense to have a model where the jobs belong to different job classes, what means that the jobs are differently served at the service units. For example, Fig. 2 shows a model with five nodes and two job classes. Interchange between the job classes is not allowed.

Fig. 2: Closed net with five nodes and two job classes (no interchange between the two job classes)

This example was taken from [GRÜTZ94]. By using MOSEL the model from Fig. 2 can be specified in many different ways. We first show a longer version where no loops are used at all, before we give a much more shorter MOSEL description for the same model. Both specification will result in the same MOSLANG and CSPL output which are shown at the end of this section.

7.2.1 A long MOSEL version for a closed network with more job classes

Here we describe the network with more job classes (see Fig. 2) by using MOSEL. In the following listing we do not use any loops nor do we declare any variables. All used variables will be automatically detected by MOSEL and correspondingly declared in MOSLANG and CSPL.

```
/***================== net with more classes ==========================*/

/*--------------- Nodes --------------------------------*/
NODE  N1_1 [K1]=K1                                                      ;
NODE  N1_2 [K2]=K2                                                      ;

NODE  W2_1[K1]                                                          ;
NODE  W3_1[K1]                                                          ;
NODE  W4_1[K1]                                                          ;
NODE  W5_1[K1]                                                          ;

NODE  W2_2[K2]                                                          ;
NODE  W3_2[K2]                                                          ;
NODE  W4_2[K2]                                                          ;
NODE  W5_2[K2]                                                          ;

NODE  B2_1[1]                                                           ;
NODE  B2_2[1]                                                           ;
NODE  B3_1[1]                                                           ;
NODE  B3_2[1]                                                           ;
NODE  B4_1[1]                                                           ;
NODE  B4_2[1]                                                           ;
NODE  B5_1[1]                                                           ;
NODE  B5_2[1]                                                           ;

/*--------------- START --------------------------------*/
START K1 K2  0 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0                         ;

/*--------------- NOT ----------------------------------*/
NOT  N1_1 + W2_1 + B2_1 + W3_1 + B3_1 + W4_1 + B4_1 + W5_1 + B5_1  != K1      ;
NOT  N1_2 + W2_2 + B2_2 + W3_2 + B3_2 + W4_2 + B4_2 + W5_2 + B5_2  != K2      ;
```

```
/*--------------- Rules --------------------------------*/
FROM N1_2  TO W2_2  W mue1_2*N1_2/(N1_1+N1_2)                                    ;

FROM N1_1  W mue1_1*N1_1/(N1_1+N1_2)
{      TO W3_1  P 0.35                                                           ;
       TO W4_1  P 0.35                                                           ;
       TO W5_1  P 0.3                                                            ;
}

FROM W2_1  TO B2_1  IF B2_1+B2_2 < 1                                             ;
FROM W2_2  TO B2_2  IF B2_1+B2_2 < 1                                             ;
FROM W3_1  TO B3_1  IF B3_1+B3_2 < 1                                             ;
FROM W3_2  TO B3_2  IF B3_1+B3_2 < 1                                             ;
FROM W4_1  TO B4_1  IF B4_1+B4_2 < 1                                             ;
FROM W4_2  TO B4_2  IF B4_1+B4_2 < 1                                             ;
FROM W5_1  TO B5_1  IF B5_1+B5_2 < 1                                             ;
FROM W5_2  TO B5_2  IF B5_1+B5_2 < 1                                             ;

FROM B2_1  TO N1_1  W B2_1*mue2                                                  ;
FROM B2_2  TO N1_2  W B2_2*mue2                                                  ;
FROM B3_1  TO N1_1  W B3_1*mue3                                                  ;
FROM B3_2  TO N1_2  W B3_2*mue3                                                  ;
FROM B4_1  TO N1_1  W B4_1*mue4                                                  ;
FROM B4_2  TO N1_2  W B4_2*mue4                                                  ;
FROM B5_1  TO N1_1  W B5_1*mue5                                                  ;
FROM B5_2  TO N1_2  W B5_2*mue5                                                  ;

/*--------------- RESULTs ------------------------------*/
RESULT  IF (N1_1 > 0)  rho1_1 += PROB * N1_1/(N1_1+N1_2)                         ;
RESULT  IF (N1_2 > 0)  rho1_2 += PROB * N1_2/(N1_1+N1_2)                         ;

RESULT  IF (B2_1 > 0)  rho2_1 += B2_1*PROB                                       ;
RESULT  IF (B2_2 > 0)  rho2_2 += B2_2*PROB                                       ;
RESULT  IF (B3_1 > 0)  rho3_1 += B3_1*PROB                                       ;
RESULT  IF (B3_2 > 0)  rho3_2 += B3_2*PROB                                       ;
RESULT  IF (B4_1 > 0)  rho4_1 += B4_1*PROB                                       ;
RESULT  IF (B4_2 > 0)  rho4_2 += B4_2*PROB                                       ;
RESULT  IF (B5_1 > 0)  rho5_1 += B5_1*PROB                                       ;
RESULT  IF (B5_2 > 0)  rho5_2 += B5_2*PROB                                       ;

/*--------------- Output ------------------------------*/
RESULT>>  rho1 = rho1_1+rho1_2                                                   ;
RESULT>>  rho2 = rho2_1+rho2_2                                                   ;
RESULT>>  rho3 = rho3_1+rho3_2                                                   ;
RESULT>>  rho4 = rho4_1+rho4_2                                                   ;
RESULT>>  rho5 = rho5_1+rho5_2                                                   ;
```

7.2.2 A short MOSEL version for a closed network with more job classes

Here we describe the same closed network with more job classes in MOSEL by using loops and we get a really short description file.

```
/***================== net with more classes ==========================*/

/*--------------- Nodes --------------------------------*/
<1,2>           NODE  N1_# [K#] = K#                                              ;
<1..2> <2..5>   NODE  W<#2>_<#1>[K<#1>] = 0                                       ;
<2..5> <_1,_2>  NODE  B#[1] = 0                                                   ;

/*--------------- NOT ----------------------------------*/
<1..2> NOT  N1_# + W2_# + B2_# + W3_# + B3_# + W4_# + B4_# + W5_# + B5_#  != K#;

/*--------------- Rules --------------------------------*/
FROM N1_2  TO W2_2  W mue1_2*N1_2/(N1_1+N1_2)                                     ;

FROM N1_1  W mue1_1*N1_1/(N1_1+N1_2) {
      TO W3_1  P 0.35                                                             ;
```

```
      TO W4_1  P 0.35                                                            ;
      TO W5_1  P 0.3                                                             ;
}
<2..5> <_1,_2>  FROM W#  TO B#      IF B<#1>_1 + B<#1>_2 < 1                     ;
<2..5> <_1,_2>  FROM B#  TO N1<#2>  W  B#*mue<#1>                                ;

/*--------------- RESULTs -----------------------------*/
<1..2>          RESULT  IF (N1_# > 0) rho1_# += PROB * N1_#/(N1_1+N1_2)          ;
<2..5> <_1,_2>  RESULT  IF (B# > 0)   rho# += B#*PROB                            ;

/*--------------- Output -----------------------------*/
<1..5>  RESULT>>   rho# = rho#_1+rho#_2                                          ;
```

7.3 A closed queuing network with a M/E2/1 and a M/M/1 server

Here we show the usability of MOSEL for a model of a closed queuing network of the following kind.·

- the system is closed with K jobs·
- node one is a M/E2/1 server·
- node two is a M/M/1 server

Fig. 3 shows this model which was taken from [MOSES94].
Fig.3: Simple example of a non productform queuing network

Fig.3: Simple example of a non productform queuing network

7.3.1 MOSEL version for the closed queuing network with a M/E2/1 and a M/M/1 server

The following listing shows a possibility to describe this model from Fig. 3. Thereby subnodes are used to describe the M/E2/1 node.

```
/***================== M/E2/1 server  and  M/M/1 server ================*/

/*---------------- NODES ---------------------------------------*/
NODE  N1[K; Phase1[1]; Phase2[1]]=K;
NODE  N2[K]=0;

/*---------------- NOT -----------------------------------------*/
NOT  N1+Phase1+Phase2+N2 != K;

/*---------------- Rules ---------------------------------------*/
FROM  N1      TOM Phase1  IF Phase1+Phase2==0;
FROM  Phase1  TO  Phase2  W mue11;
FROM  Phase2  TO  N2      W mue12;
FROM  N2      TO  N1      W mue2;

/*---------------- RESULTs -------------------------------------*/
RESULT  IF (N2!=0)  rho2 += PROB;
RESULT  MEAN N1;
RESULT  DIST N2;

RESULT>>  lambda2 = rho2*mue2;
```

7.4 A Fault Tolerant UNIX System

In this chapter it is shown how MOSEL can be used to specify the model of a fault tolerant UNIX system. The model of this system is described in [JUNG91].
The typical live cycle of a job in a UNIX system can be described with four states:

- a user job is executed (*user*)
- a system call is handled (*kernel*)
- a driver routine is executed (*driver*)
- waiting for the end of an I/O demand (*io*)

The model of the monoprocessor system is shown in Fig. 4. The system consists of three queues *userq*, *kernelq*, *driverq* and the four states *user*, *kernel*, *driver* and *io*. The live cycle of a job always starts in the kernel context and if the job is finished it will be replaced by a new job. That means that the number of jobs in the system is always constant (closed queueing network). More information about this model can be found in [JUNG91]
.

Fig. 4: The Monoprocessor UNIX Model

By using MOSEL the model from Fig. 4 can be specified in many different ways. We show only one of the possible specifications and the corresponding MOSLANG output.

7.4.1 A MOSEL version for a Fault Tolerant UNIX System

Here we describe the model of the fault tolerant system (see Fig. 4) by using MOSEL. In the following listing we do not use any loops nor do we declare any variables. All used variables will be automatically detected by MOSEL and correspondingly declared in the MOSLANG and CSPL output.

```
/***================ Fault Tolerant UNIX system ==========================*/

/*----- enum- und and constants- definition ------------------------*/
enum cpu_state { idle=0, user=1, kernel=2, driver=3 };
enum cpu_up_down {down_CPU, up_CPU };
enum io_up_down {down_IO, up_IO };

/**--- NODE-specification -----------------------------------------*/
NODE cpu[cpu_state]=idle;  // also possible: NODE cpu[idle..driver];
NODE userq[K]=K;           // # jobs waiting user context
NODE kernelq[K];           // # jobs waiting kernel context
NODE driverq[K];           // # jobs waiting driver context
NODE io[K];                // # jobs waiting IO completion
NODE state_CPU[down_CPU..up_CPU] = up_CPU;
NODE state_IO[io_up_down] = up_IO;

/**--- Prohibited states -------------------------------------------*/
NOT cpu==idle  AND  userq+kernelq+driverq+io != K;
NOT cpu!=idle  userq+kernelq+driverq+io != K-1;

/**--- Rules for the failure of components -------------------------*/
FROM up_CPU  TO down_CPU  W 1.0/MTBF_CPU;
FROM up_IO   TO down_IO   W 1.0/MTBF_IO ;
```

```
/**--- Rules for repairing of components ---------------------------*/
FROM down_CPU  TO up_CPU  W 1.0/MTTR_CPU  P coverage_CPU;
FROM down_IO   TO up_IO   W 1.0/MTTR_IO   P coverage_IO ;

/**--- I/O- behaviour ----------------------------------------------*/
FROM io  TO driverq  IF state_IO==up_IO
         W  io==1 ? 1.0/28.00,
            io==2 ? 1.0/18.667,
            io==3 ? 1.0/15.5553,
            io==4 ? 1.0/13.998,
            io==5 ? 1.0/13.0668,
            io==6 ? 1.0/12.4443,
            io==7 ? 1.0/11.99991,
            io==8 ? 1.0/11.6669,
            io==9 ? 1.0/11.4037,
            11.20;
/**--- Normal behaviour of a process --------------------------------*/
FROM user  TO kernel  W 1.0/usertime  IF state_CPU==up_CPU;
FROM kernel  W 1.0/kerneltime  IF state_CPU==up_CPU {
      TO user              P 1.0-pio-pdone;
      TO driver            P pio;
      TO idle  TO kernelq  P pdone;
}
FROM driver  TO idle  W 1.0/drivertime  IF state_CPU==up_CPU  {
      TO kernelq  P pdrivdone;
      TO io       P 1-pdrivdone  IF state_IO==up_IO;
}

/**--- Preemption  -------------------------------------------------*/
FROM user  TO userq  IF state_CPU==up_CPU {
      FROM driverq  TO driver;
      FROM kernelq  TO kernel  IF driverq==0;
}

/**--- Priorities  -------------------------------------------------*/
FROM idle  IF state_CPU==up_CPU {
      FROM driverq  TO driver;
      FROM kernelq  TO kernel  IF driverq==0;
      FROM userq    TO user    IF driverq==0 kernelq==0;
}

/**--- Results -----------------------------------------------------*/
RESULT    MEAN  userq;
RESULT    IF (cpu==kernel AND state_CPU==up_CPU)  rho_cpu_kernel += PROB;
RESULT>>  lambda = 1000*(pdone*rho_cpu_kernel/kerneltime);
```

By using loops here, for example

```
                              NODE cpu[cpu_state];
<userq, kernelq, driverq, io> NODE #[K];
<CPU, IO>                     NODE state_#[down_#..up_#];
```

instead of

```
NODE userq[K];    /* # jobs waiting user context */
NODE kernelq[K];  /* # jobs waiting kernel context */
NODE driverq[K];  /* # jobs waiting driver context */
NODE io[K];       /* # jobs waiting IO completion */
NODE state_CPU[down_CPU..up_CPU]; /* also possible: NODE state_cpu[cpu_up_down]; */
NODE state_IO[io_up_down]; /* also possible: NODE state_io[down_IO..up_IO]; */
```

we won’t get any remarkable savings.

7.5 Another Fault Tolerant Multiprocessor System

Here we show the usability of MOSEL for an easier model of a fault tolerant multiprocessor system. This model was taken from [BOLCH95]. Fig. 5 shows a multiprocessor system with a finite puffer capacity.

Fig. 5: An example of a multiprocessor system

It is assumed that the failure and repair of servers will happen independently from each other.

- The system has a finite arrival queue of length L.
- The arrival rate in this queue is λ .
- The mean time between failure is negative exponentially distributed with the rate *1/mtbf*.
- The mean time to repair is also negative exponentially distributed with the rate *1/mttr.*
- If a server, who is currently working on job, fails, then this job is started again on a free server.
- If no server is currently available, then a job must wait until a server is repaired or becomes free.

7.5.1 The MOSEL version for the Fault Tolerant multiprocessor system

The following listing shows a possibility to describe this model of the fault tolerant multiprocessor system (see Fig. 5) by using MOSEL.

```
/***==================== Fault tolerant multiprozessor systems =========*/
NODE  queue[L] = 0;
NODE  active[3] = 0;
NODE  working[3] = 3;

NOT  queue+active > L+3;

/* Here is shown that the names after the keywords FROME and TOE
   are not necessary and if they are given they are only for documentation */
FROME extern  TO queue    W lambda;
FROM  working TOE senke   W 1/mtbf;
FROME         TO working  W 1/mttr  P coverage;

FROM queue  TOM active  IF working-active > 0;

<1..3>  FROMM active  TOE extern  W #*mue  IF active==# AND working>=#;

FROMM active  TOE extern    W 1*mue  IF active==2 AND working==1;
FROMM active  TOE anyplace  W 1*mue  IF active==3 AND working==1;
FROMM active  TOE           W 2*mue  IF active==3 AND working==2;

RESULT  IF (queue+active > 0)  rho += PROB;
RESULT  IF (working > 0)       p_working += PROB;
RESULT  DIST queue;
RESULT  qquer = MEAN queue;

RESULT>>  throughput = 3*rho*mue*p_working;
```

7.6 A Fork-Join System

Here we show the usability of MOSEL for a Fork-Join-System. Fork-Join systems are able to split a job in multiple subjobs which are processed parallely. For the splitting the corresponding system uses Fork- and Join-Operations (e.g. *parbegin/parend* constructs). Fig. 6 shows such a Fork-Join-System.

Fig. 6: A Fork-Join-System

7.6.1 The MOSEL version for this system

The following listing shows a possibility to describe this model.

```
#define n      5

#string nodes  node_#:
#string queues queue_#:

        NODE  node_a[K] = 0;
<1..n>  NODE  node_#[K] = 0;
<1..n>  NODE  queue_#[K] = 0;
        NODE  node_b[K] = 0;

FROME  TO node_a  W arrival;

        FROM node_a  TO $nodes(<1..n>)  W mue_a;
<1..n>  FROM node_#  TO queue_#  W mue_#;
        FROM $queues(<1..n>) TO node_b;
        FROM node_b  TOE  W mue_b;

        RESULT    IF (node_b > 0) rho_b += PROB;
        RESULT>>  throughput = rho_b * mue_b;
 <1..n> RESULT>>  IF (node_# > 0)  rho_# += PROB;
 <1..n> RESULT>>  throughput_# = rho_# * mue_#;
```

7.7 A Minimum Batch system with Blocking Node

Here we show the usability of MOSEL for a Minimum Batch System with Blocking Node. This example was taken from [BOLGREI95].

7.7.1 The MOSEL version for this system

The following listing shows a possibility to describe this model.

```
/*** This is a simple example of the combination of different     */
/*** node characteristics.                                         */
/***  node 1 is a regular M/M/1 node                               */
/***  node 2 is a combination of a minimum batch service strategy */
/***         and finite queue                                      */

/**----- cpu_states -----------------------------------------------*/
enum cpu_state {idle, busy, wait}                                   ;

/**----- states ---------------------------------------------------*/
<1..2>  NODE  queue_#[K] = 0                                        ;
<1..2>  NODE  cpu_#[cpu_state] = idle                               ;
        NODE  num[K]                                                ;

/**----- Definition for the arrival of jobs ------------------------*/
FROME source  TO num  TOM queue_1  W arrival                        ;

/**----- Definition of the behavior of node 1 ----------------------*/
FROM cpu_1[idle],queue_1  TO cpu_1[busy]                            ;
FROM cpu_1[busy]  TO cpu_1[wait]  W 1/service_time_1                ;
FROM cpu_1[wait]  TO cpu_1[idle]  TOM queue_2  IF queue_2 < capacity  ;

/**----- Definition of the behavior of node 2 ----------------------*/
FROM cpu_2[idle]  TO cpu_2[busy]  IF queue_2 >= a                   ;
FROM cpu_2[busy]  TO cpu_2[wait]  W 1/service_time_2                ;
FROM cpu_2[wait] {
   IF queue_2 >= b  FROMM queue_2(b), num(b)  TO cpu_2[idle]        ;
   IF queue_2 < b   FROM  queue_2   FROMM  num                      ;
   IF queue_2 == 0  TO cpu_2[idle]                                  ;
}

/**----- Definition of performance measures ------------------------*/
RESULT    IF (queue_2 >= a) rho_2 += PROB                           ;
RESULT>>  throughput = rho_2/service_time_2                         ;

   pr_net_info();
}

/*========== ac_reach ===================================================*/
void ac_reach(void)
{
   pr_rg_info();
}

/*========== ac_final ===================================================*/
void ac_final(void)
{
   int  i;
   /**----- Definition of performance measures ------------------------*/
     rho_2 += expected(resultfunc_1_);
     throughput = rho_2 / service_time_2 ;
   pr_value("throughput", throughput);
}
```

7.8 A Full Batch System with Unreliable Machines

This example was also taken from [BOLGREI95]. Fig. 7 shows this Full Batch system with unreliable machines.

Fig. 7: Queueing model for a full batch system with unreliable server

7.8.1 The MOSEL version for this system

The following listing shows a possibility to describe this model.

```
/***---------------------------------------------***/
/*** A Full Batch System with Unreliable Machine ***/
/***---------------------------------------------***/

enum cpu_state {idle, busy, wait};
enum up_down   {up, down};

<1,2> NODE node_#[K] = 0;
      NODE num[K] = 0;
      NODE cpu[cpu_state][up_down] = idle, up;

/**-- Definition for the arrival of jobs */
FROME  TO num, node_1  W arrival                         ;

FROM cpu[up]  TO cpu[down]  W 1/mtbf                     ;
FROM down     TO cpu[up]    W 1/mttr                     ;

/**-- Definition of the behavior of node 1 */
IF cpu==up {
    FROM idle             TO busy       IF node_1>=b     ;
    FROM cpu[busy]        TO cpu[wait]  W mue_1          ;
    FROM wait, node_1(b)  TO node_2(b)  idle             ;
}

/**-- Definition of the behavior of node 2 */
FROM node_2, num  TOE   W mue_2                          ;

/**-- Definition of performance measures */
RESULT>>   IF (node_2 > 0) rho_2 += PROB                 ;
RESULT>>   throughput = rho_2*mue_2                      ;
RESULT>>   IF (cpu == up) upprob += PROB                 ;
```

7.9 Monoprocessor model of BS2000

Here we show the usability of MOSEL for a Monoprocessor model of BS2000 where we assume only three user job class and fixed priorities. This example was taken from [GREINER95]. For this example the CSPL output is not given, since this output would be about 1800 lines.

7.9.1 The MOSEL version for this system

The following listing shows a possibility to describe this model.

```
/*** Monoprocessor model of BS2000 where we assume   */
/*** only three user job class and fixed priorities  */

/*---------------- Macros -----------------------------------------*/
#string NOTCOND
    userq_tra + userq_dia + userq_bat + sihq_tra + sihq_dia + sihq_bat +
    sysq_tra  + sysq_dia  + sysq_bat  + ioq_tra  + ioq_dia  + ioq_bat        :

#string  COND1             sihq_tra==0                                       :
#string  COND2     $COND1  sihq_dia==0                                       :
#string  COND3     $COND2  sihq_bat==0                                       :
#string  COND4     $COND3  sysq_tra==0                                       :
#string  COND5     $COND4  sysq_dia==0                                       :
#string  COND6     $COND5  sysq_bat==0                                       :
#string  COND7     $COND6  userq_tra==0                                      :
#string  COND8     $COND7  userq_dia==0                                      :

#string  TRANS1             FROM sihq_tra  TO sih_tra;                       :
#string  TRANS2    $TRANS1  FROM sihq_dia  TO sih_dia  IF $COND1;            :
#string  TRANS3    $TRANS2  FROM sihq_bat  TO sih_bat  IF $COND2;            :
#string  TRANS4    $TRANS3  FROM sysq_tra  TO sys_tra  IF $COND3;            :
#string  TRANS5    $TRANS4  FROM sysq_dia  TO sys_dia  IF $COND4;            :
#string  TRANS6    $TRANS5  FROM sysq_bat  TO sys_bat  IF $COND5;            :
#string  TRANS7    $TRANS6  FROM userq_tra TO user_tra IF $COND6;            :
#string  TRANS8    $TRANS7  FROM userq_dia TO user_dia IF $COND7;            :
#string  TRANS9    $TRANS8  FROM userq_bat TO user_bat IF $COND8;            :

/**------------ possible states of cpu/io --------------------------*/
enum  cpu_state { idle_cpu,
                  user_tra, user_dia, user_bat,
                  sih_tra, sih_dia, sih_bat,
                  sys_tra, sys_dia, sys_bat   }                               ;
enum  io_state  { idle_io, io_tra, io_dia, io_bat }                           ;

/**---- States -----------------------------------------------------*/
                                   NODE  cpu[cpu_state]=idle_cpu              ;
<tra,dia,bat>                      NODE  userq_#[K_#]=K_#                     ;
<sihq,sysq,ioq> <_tra,_dia,_bat>   NODE  #[K<#2>]=0                           ;
                                   NODE  io[io_state]=idle_io                 ;

/**---- Prohibited states ------------------------------------------*/
NOT  cpu != idle_cpu  io == idle_io  $NOTCOND  !=  K_tra + K_dia + K_bat - 1   ;
NOT  cpu == idle_cpu  io != idle_io  $NOTCOND  !=  K_tra + K_dia + K_bat - 1   ;
NOT  cpu != idle_cpu  io != idle_io  $NOTCOND  !=  K_tra + K_dia + K_bat - 2   ;

/**---- Normal behaviour of a job ----------------------------------*/
<tra,dia,bat>  FROM user_#  TO sih_#  W 1.0/usertime_#                        ;
<tra,dia,bat>  FROM sih_# {
      TO idle_cpu   TOM  ioq_#   W 1.0/sihtime  P pio                         ;
      TO idle_cpu   TOM  sihq_#  W 1.0/sihtime  P pdone                       ;
      TO user_#                  W 1.0/sihtime  P puser                       ;
      TO sys_#                   W 1.0/sihtime  P 1-puser-pio-pdone           ;
}
<tra,dia,bat> FROM sys_#   TO sih_#                W 1.0/systime              ;
<tra,dia,bat> FROM  io_#   TO idle_io TOM sihq_#   W 1/iotime                 ;
```

```
/**---- definition of the preemption mechanism --------------------*/
FROM sys_tra   TOM sysq_tra  { $TRANS3 }
FROM sys_dia   TOM sysq_dia  { $TRANS4 }
FROM sys_bat   TOM sysq_bat  { $TRANS5 }
FROM user_tra  TOM userq_tra { $TRANS6 }
FROM user_dia  TOM userq_dia { $TRANS7 }
FROM user_bat  TOM userq_bat { $TRANS8 }

/**---- definition for fetching jobs ------------------------------*/
FROM idle_cpu  { $TRANS9 }
FROM idle_io {
      FROM ioq_tra  TO io_tra                                       ;
      FROM ioq_dia  TO io_dia  IF ioq_tra==0                        ;
      FROM ioq_bat  TO io_bat  IF ioq_tra==0 AND ioq_dia==0         ;
}

<sihq_,sysq_,userq_,ioq_> <tra,dia,bat>  RESULT  MEAN #             ;
<sih_,sys_,user_> <tra,dia,bat>          RESULT  IF (cpu == #) rho_# += PROB   ;

<tra,dia,bat>                  RESULT>> throughput_# = pdone *rho_sih_#/sihtime;
<tra,dia,bat>                  RESULT>> responsetime_# = K_#/throughput_#      ;
<sih_,sys_,user_><tra,dia,bat> RESULT>> rho_#                                  ;
```

Conclusion

This paper introduced the new description language MOSEL, which is an easier to use frontend to MOSLANG and CSPL. By using examples it was shown that MOSEL makes the specification of a system easier and more readable, since it allows a description, which reflects the real structure of the system much more than MOSLANG or CSPL does. It is planned to use MOSEL as a common description language for other already implemented measurement tools, like PEPSY [PEPSY94] or SHARPE. It is very likely that MOSEL will get some minor changes when the development of this language goes further and its usability for these other already implemented analyzing tools will be proved. Nevertheless it is unlikely that there will be major changes in the basical structure of the language MOSEL as it is described in this paper.

Appendix The Syntax of the Model Description Language MOSEL

For the descripton of the MOSEL syntax a combination of Backus-Naur-Form and the meta chararacters of Unix regular expressions is used. The meta characters here used have the following meaning:

```
a?  zero or one occurence of a.
a*  zero or more occurences of a.
a+  one or more occurences of a.
```

```
/*=========== MOSEL-input ---------------------------------------------------*/
MOSEL-input :  decl*  node+  start?  not*  rule+  result*

/*=========== decl ----------------------------------------------------------*/
decl        :  loop* #define ID value
            |  loop* #define ID ID
            |  loop* enum ID { enumlist } ;
            |  var_decl ;
enumlist    :  enumconst
            |  enumconst ,? enumlist
enumconst   :  ID  |  ID = INT_NR
var_decl    :  loop* declkind decltype declname
declkind    :  VAR     |  OUTPUT  |  HELP
decltype    :  DOUBLE  |  INT     |  PROB
declname    :  ID  |  ID = value

/*=========== node ----------------------------------------------------------*/
node        :  loop*  NODE  nodename  capacities*  init? ;
capacities  :  [ capacity innodes* ]
innodes     :  ; ID  capacities*  init?
capacity    :  INT_NR |  INT_NR..INT_NR |  ID |  INT_NR..ID |  ID..ID |  ID..INT_NR
init        :  = startvals
startvals   :  arexpr  |  arexpr ,?  startvals

/*=========== start ---------------------------------------------------------*/
start       :  START  arexpr+ ;

/*=========== not -----------------------------------------------------------*/
not         :  loop* NOT condition ;

/*=========== rule ----------------------------------------------------------*/
rule        :  global_rule { local_rule* }
            |  global_rule ;
global_rule:   loop* rule_elem*
local_rule  :  loop* rule_elem+ ;
rule_elem   :  FROM namelist |  FROMC namelist |  FROMM namelist |  FROME ID?
            |  TO  namelist |  TOC namelist  |  TOM namelist  |  TOE ID?
            |  W  mue_rate  |  WITH  mue_rate
            |  P  prob_rate
            |  IF condition
            |  PRIO  ID  |  PRIO  INT_NR
namelist    :  nodespec
            |  namelist  ,?  nodespec
nodespec    :  nodename count?  |  nodename  nodeindex+
nodeindex   :  [ ID ]
mue_rate    :  arexpr  priospec?
            |  compar ? arexpr ,  mue_rate
prob_rate   :  arexpr  priospec?
            |  compar ? arexpr ,  prob_rate
priospec    :  [  ID  ]
            |  [  INT_NR  ]
count       :  ( INT_NR )  |  ( ID )
```

```
/*=========== result -----------------------------------------------------------*/
result     :  loop* RESULT   result_type ;
           |  loop* RESULT>> result_type ;
result_type:  MEAN  nodename
           |  DIST  nodename
           |  ID asexpr?
           |  ID asoper MEAN nodename
           |  IF ( condition ) action+
           |  IF ( condition ) { action+ }
/*----------- loop -----------------------------------------------------------*/
loop     :  < indexlist >
indexlist:  index
         |  indexlist ,?  index
index[1]   :   ID  |  INT_NR  |  INT_NR..INT_NR  |  INT_NR..ID  |  ID..INT_NR  |  ID..ID

/*----------- Conditions -------------------------------------------------------*/
condition :  condition AND? compar
          |  compar
compar    :  arexpr cooper arexpr
          |  ( arexpr cooper arexpr )
cooper    :  ==  |  !=  |  <=  |  >=  |  <  |  >

/*----------- Expressions ------------------------------------------------------*/
asexpr :  asoper arexpr
arexpr :  arexpr addop term
       |  term
term   :  term multop factor
       |  factor
factor :  ID
       |  value
       |  ( arexpr )
       |  PROB
addop  :  +  |  -
multop :  *  |  /
asoper :  =  |  +=  |  -=  |  *=  |  /=

/*----------- action -----------------------------------------------------------*/
action :  ID asoper arexpr

/*----------- nodename ---------------------------------------------------------*/
nodename:  ID
/*----------- value ------------------------------------------------------------*/
value  :  INT_NR  |  DOUBLE_NR  |  loop_nr
/*----------- ID ---------------------------------------------------------------*/
ID     :  [a-zA-Z_][a-zA-Z_0-9]*
/*----------- loop_nr ----------------------------------------------------------*/
loop_nr    :  loop_const  exp_loop?
           |  digit_loop  exp_loop
loop_const:  digit_loop? . digit_loop
           |  digit_loop .
digit_loop:  digit+  digit_loop*
           |  loop_expr  digit_loop*
loop_expr :  <  repl  >
           |  <  repl  op  digit  >
           |  repl
repl       :  #
           |  digit+
exp_loop   :  exp  plmi?  digit_loop
digit      :  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
op         :  + | - | * | /
plmi       :  + | -
exp        :  e | E
```

1. If here ID is a constant name, then the value of this integer constant will be taken by MOSEL.

This syntax description does not contain or does not reflect the real semantic of the following MOSELfeatures:

- Identifiers (ID) may contain the sign **#** when loops are used. In this case the sign # will be replaced by actual loop index. Notice that only text replacement will take place. Thus for example, the line
```
<1,2>  FROM  A#  TO B#+1
```
will result in the following two MOSEL specifications
```
FROM  A1  TO B1+1
FROM  A2  TO B2+1
```
and not in
```
FROM  A1  TO B2
FROM  A2  TO B3
```
as one might expect; see also the next point

- As previously mentioned, usually the replacement of **#** by indices is just a text replacement, but there exists also the possibility to make calculations with indices. The following specification e.g
```
<1,2>  FROM  A#  TO B<#+1>
```
is a short form for the following two specifications
```
FROM  A1  TO B2
FROM  A2  TO B3
```
As one can see, to let MOSEL perform calculations, the **#** with the expression in the names must be enclosed in <...>.·

- It is possible to replace only a certain index of the loop and not the whole composed index string, e.g.
```
<1..2><a,b> NODE  node_<#2><#1> [K<#1>];
```
which corresponds to the following MOSEL specifications
```
NODE  node_a1[K1];
NODE  node_b1[K1];
NODE  node_a2[K2];
NODE  node_b2[K2];
```
As one can see, to let MOSEL replace only a certain index and not the whole composed actual index text, the required index number preceeded by **#** must be enclosed in <...>.·

- When MOSEL detects a variable which is not explicitly declared by the user, it generates automatically a declaration for that variable. The kind and type in the declaration of such a variable MOSEL arranges from the context (last read keyword). Table 2 summarizes the rules used by MOSEL to determine the kind and type of a not explicitly declared variable.

Last keyword	**kind type**	**description**
NODE	**int** (input)	variable appears either as capacity or in the initialization expression
START	**int** (input)	variable appears in a expression which specifies a start state
FROM,**FROMC**,**FROMM**,**FROME** **TO**,**TOC**,**TOM**,**TOE**	**int** (input)	variable specifies the number of elements in the transition
W, **WITH**	**double** (input)	variable appears in a expression which specifies the transition rate
P	**prob** (input)	variable appears in a expression which specifies the transition probability
IF	**int** (input)	variable appears in a expression which specifies conditions for a transition
RESULT, **RESULT>>**	**double** (output)	variable appears in a expression which specifies performance measures to be calculated or to be outputed

Table 2: Determination of kind and type of automatically generated variable declaration

- In a global or local rule the possible keywords (like **FROM**, **TO**, **P**, **IF**, etc.) can be specified more than once. Table 3 summarizes for all these keywords, if they are accumulated or override the previous specification in the case of multiple specifications.

keyword	Effect on multiple specification in the same rule
FROM, **FROMC**, **FROMM**, **FROME** **TO**, **TOC**, **TOM**, **TOE**	accumulate
W, **WITH**	override
P	override
IF	override
PRIO	override

Table 3: Effect on multiple specification of the same keyword in one rule

Table 4 summarizes what happens when the same keyword is specified in a global and a local rule.

keyword	Effect in a local rule when also specified in global rule
FROM, **FROMC**, **FROMM**, **FROME** **TO**, **TOC**, **TOM**, **TOE**	accumulate
W, **WITH**	override
P	override
IF	accumulate
PRIO	override

Table 4: Effect of rule keywords in a local rule when also specified in the global rule

Priorities which are given for a specific rate or probability ([*value*] or [*ID*]) always override a **PRIO** specification.

- If the user does not specify any capacity for a node, MOSEL assumes infinite capacity (currently 100), e.g.

```
NODE  driverq;
NODE  userq;
```

corresponds to the following NODE declarations:

```
NODE  driverq[100];
NODE  userq[100];
```

IF the user does not want this *infinite capacity* for nodes without a capacity, it may set the predefined constant **DEFAULT_CAPACITY** to a value which should be taken instead of *infinite capacity* (100), e.g.

```
#define DEFAULT_CAPACITY  K

NODE  driverq;
NODE  userq;
```

would correspond to the following NODE declarations:

```
NODE  driverq[K];
NODE  userq[K];
```

- In the **START** part the start states must be given in the same order as the NODE declarations and the start states may be specified by values, constant names, variables or arithmetic expressions. If the user specifies less values than nodes, MOSEL assumes for the remaining states (nodes) their minimum values. If two different start values are specified for a node: one at the node declaration (initialization) and one in the START part, the START value will be ignored. That means that an initialization in a node declaration has higher priority than a START specification for that node.

- Table 5 summarizes the effects of the different FROM/TO keywords.

	automatically generated check (condition)	automatically generated movement (action)
FROM/TO	yes	yes
FROMC/TOC	yes	no
FROMM/TOM	no	yes
FROME/TOE	no	no

Table 5: Effects of the different FROM/TO keywords

- MOSEL allows the definitions of text macros. A macro definition has the following form:

```
#string  macroname  macrotext:
```

The *macrotext* starts at the first non white character (white characters = blanks and tabs) and ends with a colon (**:**). In the *macrotext* all characters except for colon are allowed.
On places where the *macrotext* should be inserted only the *macroname* with a preceeding dollar sign (**$**) must be given:

```
$macroname
```

In a *macrotext* also the replacement of other previously defined macros is possible.
Macros must always be defined before they can be called (with $*macroname*).
Macro definitions can be placed at any location in the MOSEL input file.
It is also possible to specify loop arguments when calling a macro. In this case the *macrotext* will be inserted at the calling place as often as specified in the loop argument, whereby all #-signs in the *macrotext* will be replaced by the actual index of the loop, e.g.

```
#string nodes node_# :
.....
FROM node_a  TO $nodes(<1..5>)
```

would be expanded by MOSEL to

```
FROM node_a  TO node_1 node_2 node_3 node_4 node_5
```

- Comments must - like in the programming language C - be enclosed in **/* .. */**. Comments can be placed at any place in a MOSEL file. There are two special cases for comments:
  - If a comment starts with **/****, MOSEL places this comment before the appropriate construct in its generated output file.
  - If a comment starts with **/*****, MOSEL considers that comment as a global comment. MOSEL places such global comments always at the beginning in its generated output file.

  All other comments which start only with **/*** will be ignored by MOSEL.
  Besides C-Comments which are enclosed in **/* .. */** and can be used to comment out more than one line also the C++ comment sign // may be used when only one line or the remainder of a line should be commented out. Such C++ like comments will always be ignored by MOSEL and - like a /*..*/ comment - not be taken to output file.

- When multidimensional nodes are used, a number of restrictions must be observed:
  - A START part is not allowed. Initialization at node declaration should be used instead, like

    ```
    NODE cpu[cpu_state][updown] = idle, up;
    ```

  - Multidimensional nodes with ranges of enum names/constants and numerical indices are not allowed in one node declaration, e.g. for

```
NODE cpu[cpu_state][K];
```

MOSEL would report an error.

- In a multidimensional node declaration the same enum names or constants of the same enum name must not be used in different ranges, e.g.

```
NODE cpu[cpu_state][cpu_state];
```

is not allowed. Instead of that different node declarations must be used, like

```
NODE cpu1[cpu_state];
NODE cpu2[cpu_state];
```

- Whereas nodes with enum names or enum constants as ranges may be indexed in rules, an indexing of nodes with numerical ranges is not allowed, like

```
enum cpu_state {idle, user, kernel, driver };
NODE cpu[cpu_state];
NODE kernelq[K1][K2];
NODE driverq[K1][K2];
......
FROM cpu[idle]  TO cpu[kernel]  W 1/kerneltime;   /* is allowed */
FROM kernelq[5]  TO driverq[4][8]  W 1/x;       /* is not allowed */
```

- Multidimensional nodes with enum names or enum constants as ranges must always be indexed in rules, like

```
enum cpu_state {idle, user, kernel, driver };
enum updown {up, down};
.....
NODE cpu[cpu_state][updown];
NODE kernelq[K1][K2];
NODE driverq[K1][K2];
......
FROM cpu[idle][up]  TO cpu[kernel]  W 1/kerneltime;    /* is allowed */
FROM cpu  TO cpu                                /* is not allowed */
FROM kernelq  TO driverq  W 1/x;                /* is  allowed */
FROM kernelq[5]  TO driverq[4][8]  W 1/x;       /* is not allowed */
```

- For each multidimensional node MOSEL generates internally the corresponding number of one dimensional nodes. The basenames of these nodes are identical to the name of the multidimensional node, but each onedimensional gets a suffix, which is _ (underscore) followed by the dimension number, e.g.

```
enum cpu_state {idle, user, kernel, driver };
enum updown {up, down};
.....
NODE cpu[cpu_state][updown];
NODE kernelq[K1][K2];
NODE driverq[K1][K2][K3];
```

is identical to to the following specification

```
enum cpu_state {idle, user, kernel, driver };
enum updown {up, down};
....
NODE cpu_1[cpu_state];
NODE cpu_2[updown];
NODE kernelq_1[K1];
NODE kernelq_2[K2];
NODE driverq_1[K1];
NODE driverq_2[K2];
NODE driverq_3[K3];
```

Knowing this convention makes is possible, to name a certain dimension of a node, e.g.

```
enum cpu_state {idle, user, kernel, driver };
```

```
enum updown {up, down};

NODE cpu[cpu_state][updown];
NODE kernelq[K1][K2];
NODE driverq[K1][K2][K3];

FROM cpu_2[up]  TO cpu_2[down]  W 1/mtbf;
       /* also possible: FROM cpu[up]  TO cpu[down]  W 1/mtbf; */
FROM kernelq_2  TO driverq_1  W 1/kerneltime;
....
RESULT>> (IF kernelq_2 > 0) rhoq_2 += PROB;
```

It must be stated that this naming convention is only valid for multidimensional nodes. Onedimensional nodes always get exactly the name that was given at declaration time.

- There exist restrictions and rules for the different tools (see also section 6).

References

[BOLCH89] Bolch, G: Leistungsbewertung von Rechensystemen mittels analytischer Warteschlangenmodelle, B.G. Teubner, Stuttgart, 1989.

[BOLCH94] Bolch, G; Roessler, M; Zimmer, R: Leistungsbewertung mit PEPSY-QNS und MOSES, Aufsatz 1995

[BOLGREI95] Bolch, G; Greiner, S: Modeling Production Lines with Blocking, Batch Processing and Unreliable Machines Using the Markov Analyzer MOSES, 1995

[GRÜTZ94] Grütz, B. A.: Spezifikation und Analyse von Warteschlangennetzen mit dem Markov Analyser MOSES,Diplomarbeit, IMMD IV, Erlangen-Nürnberg, 1994

[GREINER95] Greiner, S: Ein Analytisches Modell für das Betriebssystem BS2000, Diplomarbeit, IMMD IV, Erlangen-Nürnberg, 1995

[HERO95] Herold, H.: lex und yacc, Addison-Wesley, Bonn 1995

[JUNG91] Jung, H: Leistungsbewertung UNIX-basierter Betriebssysteme für Multiprozessoren mit globalem Speicher, Dissertation, IMMD IV, Erlangen-Nürnberg, 1991

[MOSES94] Bolch, G.; Greiner, S; Jung, H.; Zimmer, R: The Markov Analyzer MOSES,
Technical Report (TR-14-10-94), IMMD IV, Erlangen-Nürnberg, 1994

[SPNP91] Ciardo, G; Muppala, J.K.: Manual for the SPNP Package Version 3.1,
Duke University Durham, North Carolina, 1991

[PEPSY94] Kirschnick, M: PEPSY-QNS,
Technical Report (TR-14-18-94), IMMD IV, Erlangen-Nürnberg, 1994

[SHARPE86] Sahner, R A.; Trivedi, K S.: SHARPE: Symbolic Hierarchical Automated Reliability and Performance Evaluator, Duke University Durham, North Carolina, 1986

[TRIV91] Trivedi, K S.; Ciardo, G.: A Decomposition Approach for Stochastic Reward Net Models, Duke University Durham, North Carolina, 1991