MOSEL - Modeling, Specification and Evaluation Language

Jörg Barner, Khalid Begain, Björn Beutel, Gunter Bolch and Helmut Herold
University Erlangen, 2003

Outline

- Motivation
- MOSEL
  - Structure of the MOSEL Modelling Environment
  - Constructs of MOSEL
- Production Line Examples
  - Fundamental Systems: Basic Model, Multiple Machines, Finite Buffer, Batch Processing, Unreliable Machines
  - Preemption in a Multitasking environment
- Other Real Life Examples
- IGL Intermediate Graphic Language
- Related Work and Future Work
Motivation (1)

- Discrete Event Systems:
  - Dynamic evolution of the system proceeds from one *discrete* state to another at arbitrary moments in time.
  - State changes (Transitions) are triggered by *Events*

- Two different *modelling paradigms* for DES are popular
  - *Process-Activity* based modelling
  - *State-Transition* based modelling

- MOSEL can be used for Performance and Reliability Evaluation of DES

- MOSEL follows the state-transition based modelling paradigm!

Motivation (2)

- Performance and Reliability Evaluation Methods
  - Analytical Methods:
    *Very fast, but restrictive*
  - Simulation:
    *Very time consuming, but universal*
  - Numerical Methods:
    *Faster than Simulation, but not applicable in many situations*
  - Measurement:
    *Expensive, only for already existing systems; time consuming*

- Performance and Reliability Evaluation with MOSEL employs numerical and simulative Evaluation Methods

- Based on a total or partial generation of the state-space of the DES
Motivation (3)

State-Space based Performance and Reliability Evaluation

- **Create a Model** of the DES using a (Formal) Description Technique containing
  - Specification of **structure** and **dynamic behaviour** of the DES
  - Specification of the **performance measures** of interest
  - Course of time can be modelled **stochastically** (continuous random variables are associated with the state changes of the DES)

- Use a **tool** supporting the chosen modelling formalism for evaluation:
  - automatically generates a **state-space**-level dynamic model
  - maps onto a **stochastic process** or a **Discrete Event Simulation** model.
  - calculates **state probabilities** numerically or by simulation
  - derives the **performance measures** as specified in the high-level description from the state probabilities

Motivation (4)

Performance and Reliability Evaluation Modelling Formalisms

- Queueing Network Models
- Petri Net Models
- Precedence Graph Models
- Fault Trees
- Markov Models
- Stochastic Activity Networks
- Stochastic Process Algebras
Motivation (5)

- Tools for Performance and Reliability Evaluation
  - Queueing Network Tools
    - QNAP, RESQ, PEPSY, ...
  - Petri Net Tools
    - SPNP, TimeNET, PETS!, GreatSPN, ...
  - Tools based on Stochastic Process Algebras
    - EMPA, PEPA, TIPP, ...

Motivation (6)

- Observations:
  - Learning more than one modeling language is very time consuming
  - Many tools for performance evaluation with well-tested solution methods already exist
  - Graphical modelling formalisms are sometimes confusing for larger systems with complex synchronisation schemes

- Conclusion:
  - Provide the performance modeller with an easy-to-learn, textual modelling language in which he can describe the relevant system properties directly
  - Don’t re-invent the wheel concerning the solution methods, reuse the power of existing performance and reliability tools

MOSEL: MOdeling, Specification and Evaluation Language
MOSEL - Modeling Specification and Evaluation Language

**Performance Modelling with the MOSEL-environment**

Real System

High-Level System Description

MOSES SPNP TimeNET ...

Semantic Model (State Space)

Stochastic Process

CTMC, GSMP

Results

Numerical analysis methods

| tool | specific | result | ... | files |

MOSEL - Modeling Specification and Evaluation Language

- SPNP 6.1 - Stochastic Petri Net Package (Duke University)

Generation and Solution of CTMCs and GSMCs

- Input: Description of the System in CSPL (C based Petri Net Language)
  - Generated automatically from the MOSEL Description of the System by the MOSEL-CSPL translation component

- Output: State Probabilities, reward based measures
- Solution Methods
  - Numerical Solution of CTMCs (steady state and transient)
  - Discrete Event Simulation of CTMCs and GSMCs (steady state and transient)
TimeNET 3.0 - Timed Net Evaluation Tool
(technical university Berlin)

Support for different types of extended Petri net classes
• eDSPN: extended Deterministic and Stochastic Petri Nets

Generation and Solution of the Markov Chain
• Input: Description of the system in proprietary .tn-format
  – Generated automatically from the MOSEL description of the system by the MOSEL-.tn translation component.
• Output: State probabilities, reward based measures
  Solution Methods
  – Various numerical solution algorithms, discrete event simulation

Structure of a MOSEL file (1)

• Parameter declaration part
  – System parameter sets
  – Constants

• Component definition part (node part)
  – Components of the system (called NODES), each NODE can hold an integer number of jobs or token
  – Range of the local state spaces (each NODE has a fixed capacity)
  – Initial number of token in each node (the global system start state)
  – Specification of prohibited system states (optional)
Structure of a MOSEL file (2)

- **Transition definition part (rules part):**
  Specification of the transitions (called rules) which determine the dynamic behaviour of the system
  - Condition part (specify the circumstances under which the rule can be "executed", complex synchronisation schemes are possible)
  - Action part (stochastic timing, branching probabilities, priorities, reenabling policies)

- **Results part:** Specification of the performance measures (state probabilities, mean response times, MTTF, reward-based measures, etc.)

- **Picture part:** Specification of the graphical representation of the results

---

**MOSEL - Queueing Network Examples**

### Open Tandem Network (1)

```
/* Definition of the system parameters */
CONST arrival := 0.25;
CONST ServiceRate_1 := 0.28;
CONST ServiceRate_2 := 0.22;
```

![Open Tandem Network Diagram]

1. **arrival**
2. **station_1**
3. **station_2**
MOSEL - Queueing Network Examples

Open Tandem Network (2)

```
/* Definition of the system components */
NODE station_1 [K] := 0;
NODE station_2 [K] := 0;
NODE num [K] := 0;

/* Definition of the arrival of jobs */
FROM EXTERN TO station_1, num RATE arrival;
```

```
/* Definition of the behavior of the stations */
FROM station_1 TO station_2 RATE ServiceRate_1;
FROM station_2, num TO EXTERN RATE ServiceRate_2;
```

```
/* Definition of the performance measures */
PRINT rho1 := PROB (station_1 > 0);
PRINT WIP := MEAN (num);
PRINT system_time := WIP / arrival;
```
Closed Queueing Network

/* Definition of the system components */

NODE station_1 [K] := K;
NODE station_2 [K] := 0;
NODE station_3 [K] := 0;
ASSERT station_1 + station_2 + station_3 = K; ( <=K )

/* Definition of the behavior of the stations */

FROM station_1 RATE ServiceRate_1
THEN { TO station_2 WEIGHT p_2;
         TO station_3 WEIGHT p_3; }
FROM station_2 TO station_1 RATE ServiceRate_2;
FROM station_3 TO station_1 RATE ServiceRate_3;

/* Specification of the performance measures */

PRINT utilization_1 := UTIL(station_1);
PRINT throughput_1 := utilization_1 * ServiceRate_1;
A bridged MOSEL Version using "loops"

/* Definition of the system components */

NODE station_1 [K] := K;
@<2,3> { NODE station_# [K] := 0; }

ASSERT station_1 + station_2 + station_3 = K;

/* Definition of the behavior of the stations */

FROM station_1 RATE ServiceRate_1
THEN { TO station_2 WEIGHT p_2;
TO station_3 WEIGHT p_3; }
@<2,3> { FROM station_# TO station_1 RATE ServiceRate_#; }

/* Specification of the performance measures */

@<1,2,3>{ PRINT utilization_# = UTIL(station_#); }
@<1,2,3>{ PRINT throughput_# = utilization_# * ServiceRate_#; }
Non-Productform Queueing Network

```plaintext
/* Definition of the state vector */
NODE queue_1 [K] := K;
NODE phase_1 [1] := 0;
NODE phase_2 [1] := 0;
NODE node_2 [K] := 0;

ASSERT queue_1 + phase_1 + phase_2 + node_2 = K;
```

Nonproductform Queueing Network (2)

```plaintext
/* Behavior of node_1 */
IF (phase_1 + phase_2 = 0) FROM queue_1 TO phase_1;
FROM phase_1 TO phase_2 RATE mue_11;
FROM phase_2 TO node_2 RATE mue_12;

/* Behavior of node_2 */
FROM node_2 TO queue_1 RATE mue_2;
```
MOSEL Specification:

/*---------------------------------- Constants and Parameter sets------*/

CONST K := 10;
CONST mtbf := 10000;
CONST mttr := 10;
PARAMETER lambda := 0.1, 0.2, 0.3, 0.4, 0.5, 0.6 0.8, 1;
PARAMETER mue := 0.5, 0.8, 1, 1.5, 1.8, 2, 3;
ENUM cpu_state := { down, up };

/*------------------------------------------ Node definitions -------*/

NODE queue[K] := 0;
NODE server[cpu_state] := up;

/*--------------------------- Arrival and service of the jobs -------*/

FROM EXTERN TO queue RATE lambda;
IF (server = up) FROM queue TO EXTERN RATE mue ;
**Petri Net Example (3)**

**MOSEL Specification**

```mosel
/*------------------------------ Repair/Failure of the server -------*/
FROM server[up]  TO server[down] RATE 1/mtbf;
FROM server[down] TO server[up] RATE 1/mttr;

/*--------------------------------------------------- Results -------*/
PRINT server_idle := PROB (queue == 0) ;
PRINT server_reject := PROB (queue == K) ;
PRINT rate_reject := lambda *server_reject;
PRINT mean_qlength := MEAN (queue) ;
PRINT util_server := UTIL (queue) ;
PRINT throughput := util_server*mue ;

/*-------------------------------------------------- Pictures -------*/
PICTURE "Mean queue length"
PARAMETER lambda
CURVE mean_qlength ;
```

**Petri Net Example (4)**

**Invoking the MOSEL-ENVIRONMENT:**

```
mosel2 -cs petri.msl --> petri.res --> petri.igl
igl petri.igl --> pictures
```

**Results:**

```
........... lambda = 0.1, mue = 0.5 ........... ........... lambda = 1, mue = 2 ...........
  server_idle = 0.665874058769  server_idle = 0.499527144294
  server_reject = 1.82017918692e-06  server_reject = 0.000962398971526
  rate_reject = 1.82017918692e-07  rate_reject = 0.000962398971526
  mean_qlength = 0.251555031241  mean_qlength = 1.00283434233
  util_server = 0.20059915656  thruput = 1.00094571141

........... lambda = 0.1, mue = 0.8 ...........  ........... lambda = 1, mue = 3 ...........
  server_idle = 0.874438162634  server_idle = 0.665874058769
  server_reject = 1.33052923469e-06  server_reject = 1.82017918692e-06
  rate_reject = ......  rate_reject = ......
```

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

MOSEL - Modeling Specification and Evaluation Language
© Jörg Barner, Khalid Begain, Björn Beutel, Gunter Bolch, Helmut Herold • Erlangen, 2003
mosel_tut03.fm 2004-01-29 19:18 .25

MOSEL - Modeling Specification and Evaluation Language
© Jörg Barner, Khalid Begain, Björn Beutel, Gunter Bolch, Helmut Herold • Erlangen, 2003
mosel_tut03.fm 2004-01-29 19:18 .26
Production Line Examples

Basic Model

/* Declaration part*/

CONST arrival := 0.5;
CONST ServiceRate_1 := 0.75;
CONST ServiceRate_2 := 0.6;

/* System components part */

NODE buffer_1 [K] := 0;
NODE machine_1 [1] := 0;
NODE station_2 [K] := 0;
NODE num [K] := 0;

/* Rules part */

FROM EXTERN TO buffer_1, num RATE arrival;
FROM buffer_1 TO machine_1;
FROM machine_1 TO station_2 RATE ServiceRate_1;
FROM station_2, num TO EXTERN RATE ServiceRate_2;
Production Line Examples

- Basic Model (3)

/* Results part */

PRINT utilization_1 := UTIL (machine_1);
PRINT utilization_2 := UTIL (station_2);
PRINT WIP := MEAN (num);
PRINT T := WIP / Arrival;

Production Line Examples

- Multiple Machines

CONST m := 4;

NODE buffer_1 [K] := 0;
NODE machine [m] := 0;
NODE station_2 [K] := 0;
NODE num [K] := 0;
Production Line Examples

Multiple Machines (2)

```mosel
FROM EXTERN TO buffer_1, num RATE arrival;
FROM buffer_1 TO machine;
@<1..m> { IF (machine = #) FROM machine TO station_2
  RATE #*ServiceRate_1; }
FROM station_2, num TO EXTERN RATE ServiceRate_2;
```

Production Line Examples

Multiple Machines (3)

```mosel
/* Mean number of active machines in station_1 */
PRINT A := MEAN (machine);

/* Utilization of station_1 */
PRINT utilization_1 := A/m;
```
Production Line Examples

Finite Buffer (Blocking)

/* Definition of the system components */

NODE station_1 [K] := 0;
NODE block [1] := 0;
NODE station_2 [Capacity] := 0;
NODE num [K];

/* Definition of the arrivals */

FROM EXTERN TO station_1 RATE arrival;

/* Definition of the behavior of station_1 */

FROM station_1 TO block RATE ServiceRate_1;
FROM block TO station_2:

/* Definition of the behavior of station_2 */

FROM station_2 TO EXTERN RATE ServiceRate_2;
Production Line Examples

Finite Buffer (Blocking) (3)

![Finite Buffer Diagram]

```mosel
/* Definition of the Results */
PRINT utilization_2 := UTIL (station_2);
PRINT throughput := utilization_2 * ServiceRate_2;
PRINT blockprob := PROB (block = 1);
PRINT utilization_1 := PROB (station_1 > 0 OR block = 1);
PRINT WIP = MEAN (num);
```

Production Line Examples

Batch Processing

![Batch Processing Diagram]

```mosel
/* Declaration part*/
CONST b = 5; /* Batch size */;

/*Definition of the system components */
NODE buffer_1 [K] := 0;
NODE machine_1 [b] := 0;
NODE station_2 [K] := 0;
NODE num [K] := 0;
```
Production Line Examples

Batch Processing (2)

/* Arrival of jobs */
FROM EXTERN TO num, buffer_1 RATE arrival;

/* station_1 */
IF (buffer_1 >= b) FROM buffer_1 (b) TO machine_1 (b);
FROM machine_1 (b) TO station_2 (b) RATE ServiceRate_1;

/* station_2 */
FROM station_2, num TO EXTERN RATE ServiceRate_2;

Production Line Examples

Batch Processing (3)

/* Performance measures */
PRINT utilization_2 := UTIL (station_2);
PRINT throughput := utilization_2 * ServiceRate_2;
PRINT p_zero := PROB (machine_1 = 0);
PRINT utilization_1 := 1 - p_zero;
PRINT WIP := MEAN (num);
PRINT DIST num;
Production Line Examples

- Unreliable Machines

```
/* Declaration part */
ENUM state_a := {up, down};

/* Definition of the system components */
NODE station_1[K] := 0;
NODE station_2[K] := 0;
NODE num[K] := 0;
NODE server[state] := up;
```

/* Arrival of jobs */
FROM EXTERN TO num, station_1 RATE arrival;

/* Failure and repair */
FROM server[up] TO server[down] RATE 1/mtbf;
FROM server[down] TO server[up] RATE 1/mttr;
```
Production Line Examples

Unreliable Machines (3)

```mosel
/* station_1 */
IF (server = up) FROM station_1 TO station_2 RATE ServiceRate_1;

/* station_2 */
FROM station_2, num TO EXTERN RATE ServiceRate_2;

/* Performance Measures */
PRINT utilization_2 := UTIL (station_2);
PRINT throughput_2 := utilization_2 * ServiceRate_2;
PRINT upprob := PROB (server = up);
```

Preemption in a multitasking environment

Block 1
- user
  - U_time
  - system
  - S_time

Block 2
- cons
- buff1
  - producer
    - work
      - end
      - stop

Block 3
- busy_prod
  - slot_1
  - busy_2
  - slot_2
  - idle_prod
  - idle_2

Preemption in a multitasking environment
Preemption in a multitasking environment

/* Constant declarations*/
CONST N := 3;
CONST prod_start := 0.5;
CONST prod_end := 1.5;
CONST U_time := 3;
CONST S_time := 1.0;
CONST busy_prod := 0.1;
CONST busy2 := 0.1;
CONST cons1 := 0.1;

/* Nodes for Block 1 */
NODE user[1] := 1;
NODE system[1];

/* Nodes for Block 2 */
NODE work[N] := 3;
NODE buff1[N];

/* Nodes for Block 3 */
NODE slot1[1] := 1;
NODE slot2[1];

/* Node indicating completion of service */
NODE stop[1] := 1;

/* Condition under which the user may continue his work */
COND user_may_work := system = 0 AND stop = 0;
/* Dynamic behaviour of the System */

IF user_may_work FROM user TO system RATE U_time PRD;
IF (stop = 0 AND user = 0) FROM system TO user RATE S_time PRD;

IF (slot2 = 0 AND system = 0) FROM work TO buff1 AFTER prod_start..prod_end PRD;
IF (slot1 = 0 AND system = 0) FROM buff1 RATE cons1;

IF user_may_work FROM slot1 TO slot2 AFTER busy_prod PRD;
IF (user_may_work AND work = 0) FROM slot1 TO slot2 PRIO 1;
IF user_may_work FROM slot2 TO slot1 AFTER busy2 PRD;
IF (user_may_work AND buff1 = 0) FROM slot2 TO slot1 PRIO 1;

IF (work = 0 AND buff1 = 0) TO stop PRIO 2;
IF stop = 1 /* do nothing, just fire */ RATE 1;

/* Definition of time points to be used in transient analysis */
TIME 20..500 STEP 20;

/* Results */

/* Mean number of jobs to be processed */
PRINT work_left := MEAN (work);

/* Probability that the work is completed */
PRINT prob_work_complete := PROB (stop > 0);

/* Graphical presentation */
PICTURE "service completion probability "
PARAMETER TIME
CURVE prob_work_complete;
/*========== Central-Server-model ==================*/

/*---------- Parameter declaration part--------------------------*/

CONST K := 10;
CONST mue1 := 3.5;
CONST mue2 := 0.9;
CONST mue3 := 2.3;
CONST mue4 := 1.2;
CONST p12 := 0.25;
CONST p13 := 0.35;
CONST p14 := 0.4;

/*****************************************************************************/

/*---------- System component definition part -------------------------*/

NODE N1[K] := K;
@<2..4> { NODE N#[K]; }

/*****************************************************************************/

/*---------- Prohibited states --------------------------------------*/

ASSERT N1 + N2 + N3 + N4 = K;
/*--------Transition definition part-----------------------------------------------*/

FROM N1 RATE mue1 THEN {
    @<2..4> { TO N# WEIGHT p1#; }
}

FROM N# TO N1 RATE mue#;

/*-------- Result part----------------------------------------------------------*/

PRINT rho# := PROB (N#>0); // utilization
PRINT n# := MEAN (N#); // mean queue length
PRINT DIST N#; // distribution (probability of each possible queue length)

PRINT lambda# := rho# * mue#; // throughput
PRINT t# := n# / lambda#; // mean time a job is in a node

/*-------- Picture part---------------------------------------------------------*/

PICTURE "Distribution for queue lengths"
    CURVE DIST N1, N2, N3, N4 ;

PICTURE "lamdas_and_rhos"
    LIST lambda1, lambda2, lambda3, lambda4, rho1, rho2, rho3, rho4

Invoking the MOSEL-ENVIRONMENT:

mosel2 -cs centralserver.msl  -->  centralserver.res  -->  centralserver.igl
igl centralserver.igl  -->  pictures
Results provided by the tool `SPNP'

Constants:
- $K = 10$
- $\mu_1 = 3.5$
- $\mu_2 = 0.9$
- $\mu_3 = 2.3$
- $\mu_4 = 1.2$
- $p_{12} = 0.25$
- $p_{13} = 0.35$
- $p_{14} = 0.4$

Results:

__DIST N1__

- $0$: 0.226490510324
- $10$: 0.00523876082994

__DIST N2__

- $0$: 0.247975421288
- $10$: 0.00395268685905

__DIST N3__

- $0$: 0.588022114438
- $10$: 9.62301477005e-06

__DIST N4__

- $0$: 0.0975734220455
- $10$: 0.0244732959731

- $\rho_1 = 0.773509489676$
- $\rho_2 = 0.752024578712$
- $\rho_3 = 0.411977885562$
- $\rho_4 = 0.902426577954$

- $\lambda_1 = 2.70728321387$
- $\lambda_2 = 0.676822120841$
- $\lambda_3 = 0.947549136792$
- $\lambda_4 = 1.08291189355$

- $t_1 = 0.965895587825$
- $t_2 = 3.56550339435$
- $t_3 = 0.720806197492$
- $t_4 = 3.96046645608$
MOSEL - Modeling Specification and Evaluation Language

IGL Intermediate Graphic Language
Real Life Examples

- MOSEL has been successfully used to model and to analyze systems from the following domains:
  
  
  - **Communication Systems**: Cellular Mobil Networks, ATM-Multiplexer, Internet Router, Retrial Systems,
  
Remarks

- **Availability**: MOSEL is working on Solaris and Linux platforms. An earlier version had been compiled successfully under Windows NT. The IGL-interpreter is written in Tcl/Tk. MOSEL is implemented in C. The MOSEL-Tool (MOSEL, IGL) is freeware.


- **Current State**: MOSEL contains translators to CSPL (suitable for SPNP v. 6.x) and to .TN (suitable for TimeNET 3.0), and a translator to the Markov Chain Solver MOSES.

http://www4.informatik.uni-erlangen.de/Projects/MOSEL/

Future Work and Related Work

- Modularization: Decomposition of the monolithic MOSEL model into functional components

- Integration of other Tools into the MOSEL environment

- Extend the language for the support of Fluid Stochastic Petri Nets

- Application to Special Systems
  - Computer Systems
  - Operating Systems
  - Communication Systems (Ethernet (CSMA/CD, Tokenring), FDDI, ATM, Cellular Mobil Networks)
  - Fault Tolerant Systems

- Computer Systems
- Operating Systems
- Communication Systems (Ethernet (CSMA/CD, Tokenring), FDDI, ATM, Cellular Mobil Networks)
- Fault Tolerant Systems