MOSEL
MOdeling Specification and Evaluation Language

Jörg Barner and Gunter Bolch
University Erlangen / Germany

Helmut Herold
University of Applied Sciences
Nürnberg / Germany

Khalid Begain
University Bradford / UK

Outline

- Motivation

- MOSEL
  - Structure of the MOSEL System
  - Constructs of MOSEL
  - Queueing Network Examples, Petri Net Example

- Production Line Examples
  - Fundamental Systems: Basic Model, Multiple Machines, Finite Buffer, Batch Processing, Unreliable Machines
  - Wafer Production System

- Other Real Life Examples

- IGL Intermediate Graphic Language

- Related Work and Future Work
Motivation (1)

- Performance and Reliability Evaluation is important for
  - Design
  - Planning
  - Tuning
  - Comparison
  - Selection
  of
  - Manufacturing Systems
  - Computer Systems
  - Operating Systems
  - Communication Systems
  - Workflow Management Systems

Motivation (2)

- Performance and Reliability Evaluation Methods
  - Measurement:  
    *Expensive, only for already existing systems*
  - Simulation:  
    *Very time consuming, but universal*
  - Analytical Methods:  
    *Very fast, but restrictive*
  - Numerical Methods:  
    *Faster than Simulation*
Motivation (3)

- Performance and Reliability Evaluation Models
  - Queueing Network Models
  - Petri Net Models
  - Precedence Graph Models
  - Fault Trees
  - Markov Models
  - Modeling Languages

Motivation (4)

- Tools for Performance and Reliability Evaluation
  - Queueing Network Tools
    - QNAP, RESQ, PEPSY, ...
  - Petri Net Tools
    - SPNP, TIMENET, PETS, GreatSPN, ...
  - Tools based on Modeling Languages
    - MOSEL - Tool, SHARPE, QNAP, SPNP, ...
Motivation (5)

- **Problems:**
  - Learning more than one modeling language is very time consuming
  - Some systems are difficult to describe in a particular language
  - Syntax of a modeling language is oriented to the specific characteristics of the particular tool

- **Solution:**
  - Design a model description language that allows the user to describe the system directly without any knowledge of the underlying methods or tools
  - Provide translators that transform this model description into the input languages/models needed by specific already existing tools.

MOSEL: MOdeling, Specification and Evaluation Language

MOSEL

Structure of the MOSEL System
MOSEL

MOdeling Specification and Evaluation Language

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

- Generation and Solution of the Markov Chain
  - Input: Description of the System in CSPL
    - Generated automatically from the MOSEL Description of the System by a Translator
  - Output: State Probabilities
  - Solution Methods
    - Power (Iterative)
    - Gauss Seidel (Iterative)
    - SOR (Iterative)
    - Uniformization (Transient Solution)
MOSEL
MOdeling Specification and Evaluation Language

- **MOSEL - Tool** (University of Erlangen)
- Generation and Solution of the Markov Chain
  - Input: Description of the System in CSPL
    - Generated automatically from the MOSEL Description of the System by a Translater
  - Output: State Probabilities
  - Solution Methods
    - Power, SOR
    - Gauss Seidel
    - Multi Level
    - Uniformization
    - (Simulation)

- Structure of a MOSEL file (1)
  - **Parameter declaration part** (optional)
    - Variables
    - Constants
  - **System state definition part** (Vector description part)
    - Components of the state vector
    - Range of the state space
    - Start vector (optional)
    - Specification of the prohibited states (optional)
Structure of a MOSEL file (2)

- **Transition definition part (Rules part)**
  - Specification of the transitions between the states
    - Condition part
    - Action part (rate, probability)
- **Results part**
  - Specification of the performance measures
- **Picture part**
  - Specification of the values to be plotted

Queueing Network Examples

**Open Tandem Network (1)**

```
/* Definition of the state vector */
NODE station_1 [K] = 0;
NODE station_2 [K] = 0;
NODE num [K] = 0;

/* Definition of the arrival of jobs */
FROME TO station_1, num W arrival;
```
MOSEL
Queueing Network Examples

- **Open Tandem Network (2)**

![Open Tandem Network Diagram]

/* Definition of the behavior of the stations */
FROM station_1 TO station_2 W ServiceRate_1;
FROM station_2, num TOE W ServiceRate_2;

/* Definition of the performance measures */
RESULT>> k_1 = MEAN station_1;
RESULT>> WIP = MEAN num;
RESULT>> system_time = WIP / arrival;

---

MOSEL
Queueing Network Examples

- **Closed Queueing Network (1)**

![Closed Queueing Network Diagram]

Description in MOSEL

/* Definition of the state vector */

```
NODE station_1 [K] = K;
NODE station_2 [K] = 0;
NODE station_3 [K] = 0;
NOT station_1 + station_2 + station_3 != K;
```
MOSEL
Queueing Network Examples

- Closed Queueing Network (2)

/* Definition of the behavior of the stations */
FROM station_1 W ServiceRate_1
{ TO station_2 P p_2;
  TO station_3 P p_3;}
FROM station_2 TO station_1 W ServiceRate_2;
FROM station_3 TO station_1 W ServiceRate_3;

/* Definition of the performance measures */
RESULT>> utilization_1 = UTIL station_1;
RESULT>> throughput_1 = utilization_1 * ServiceRate_1;

MOSEL MOdeling Specification and Evaluation Language
J. Barner, K. Begain, G. Bolch and H. Herold, ESM02, 2002

MOSEL
Queueing Network Examples

- Short MOSEL Version

/* Definition of the state vector */

/* Definition of the behavior of the stations */
FROM station_1 W ServiceRate_1
{ TO station_2 P p_2;
  TO station_3 P p_3;}
FROM station_2 TO station_1 W ServiceRate_2;
FROM station_3 TO station_1 W ServiceRate_3;

/* Definition of the performance measures */
RESULT>> utilization_# = UTIL station_#;
RESULT>> throughput_# = utilization_# * ServiceRate_#;
Nonproductform Queueing Network (1)

/* Definition of the state vector */

NODE queue_1 [K] = 0;
NODE phase_1 [1] = 0;
NODE phase_2 [1] = 0;
NODE node_2 [K] = 0;

NOT queue_1 + phase_1 + phase_2 + node_2 != K;

Nonproductform Queueing Network (2)

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

/* Behavior of node_2 */
FROM node_2 TO queue_1 W mue_2;
/*---------------------------------- Different values (loops) -------*/
#define K 10
#define mtbf 10000
#define mttr 10
#define lambda 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8, 1
#define mue 0.5, 0.8, 1, 1.5, 1.8, 2, 3

/*------------------------------------------ Node definitions -------*/
enum down_up { down, up };
NODE queue[K] = 0;
NODE server[down_up] = up;

/*------------------------ Arrival and service of the jobs ------*/
FROME TO queue W lambda;
IF server == up FROM queue TOE W mue ;
Petri Net Example (2)

MOSEL Specification

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

/*---------------------------------------------------
Results -------*/
RESULT >> IF (queue == 0) server_idle += PROB;
RESULT >> IF (queue == K) server_reject += PROB;
RESULT >> rate_reject = lambda * server_reject;
RESULT >> meanqlength = MEAN queue;
RESULT >> thruput = util_server * mue;
RESULT >> util_server = UTIL queue;

/*---------------------------------------------------
Pictures -------*/
PICTURE "Mean queue length" -BACKGROUND gray90
XSCALE lambda -MIDDLE -HIGHVALUE 1.4 -LABELNR 7
CURVE meanqlength -DIFF THICK, POINT

Petri Net Example (3)

Results

Call of the MOSEL-TOOL: mosel -cs petri.msl --> petri.res --> petri.igl
igl petri.igl --> pictures

Results:

......... lambda = 0.1, mue = 0.5 ...........
server_idle = 0.665874058769
server_reject = 1.82017918692e-06
rate_reject = 1.82017918692e-07
meanqlength = 0.251555031241
thruput = 0.10029957828
util_server = 0.20059915656

......... lambda = 0.1, mue = 0.8 ...........
server_idle = 0.874438162634
server_reject = 1.33052923469e-06
rate_reject = .........
meanqlength = 1.00283434233
thruput = 1.00094571141
util_server = 0.500472855706

......... lambda = 1, mue = 2 ...........
server_idle = 0.499527144294
server_reject = 0.000962398971526
rate_reject = 0.000962398971526
meanqlength = 1.00094571141
thruput = 0.500472855706
util_server = .........

......... lambda = 1, mue = 3 ...........
server_idle = 1.82017918692e-06
server_reject = 1.82017918692e-06
rate_reject = .........
Petri Net Example (4)

PETRI NET EXAMPLE (5)
Production Line Examples
Basic Model (1)

/* Declaration part*/
enum machine_state {idle, busy};

/* Vector Description part*/
NODE buffer_1 [K] = 0;
NODE machine_1 [1] = 0;
NODE station_2 [K] = 0;
NODE num [K] = 0;

/* Rules part */
FROME TO buffer_1, num W arrival;
FROM buffer_1 TO machine_1;
FROM machine_1 TO station_2 W ServiceRate_1;
FROM station_2, num TOE W ServiceRate_2;

/* Results part */
RESULT>> utilization_1 = UTIL machine_1;
RESULT>> utilization_2 = UTIL station_2;
RESULT>> WIP = MEAN num;
RESULT>> T = WIP / Arrival;

Production Line Examples
Basic Model (2)
Production Line Examples
Multiple Machines (1)

```mosel
define m 4;

NODE buffer_1 [K];
NODE machine [m];
NODE station_2 [K];
NODE num [K];

FROME TO buffer_1, num W arrival;
```

Production Line Examples
Multiple Machines (2)

```mosel```
FROM buffer_1 TO machine;
<1..m> FROM machine TO station_2 W

```mosel```
FROM station_2, num TOE W ServiceRate_2;

/* Mean number of active machines in station_1 */
RESULT>> A = MEAN machine;

/* Utilization of station_1 */
RESULT>> utilization_1 = A/m;
Production Line Examples
Finite Buffer (Blocking) (1)

/* Definition of the state vector */

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

Production Line Examples
Finite Buffer (Blocking) (2)

/* Definition of the arrivals */
FROME TO station_1 W arrival;

/* Definition of the behavior of station_1 */
FROM station_1 TO block W ServiceRate_1;
FROM block TO station_2:

/* Definition of the behavior of station_2 */
FROM station_2 TOE W ServiceRate_2;
Production Line Examples
Finite Buffer (Blocking) (3)

/* Definition of the Results */

RESULT>> utilization_2 = UTIL station_2;
RESULT>> throughput = utilization_2 * ServiceRate_2;
RESULT>> IF (block == 1) blockprob + = PROB;
RESULT>> IF (station_1 > 0 OR block == 1) utilization_1 + = PROB
RESULT>> WIP = MEAN num;

Production Line Examples
Batch Processing (1)

/* Declaration part*/

VAR int b; /* Batch size */

/*Definition of the state vector */

NODE buffer_1 [K];
NODE machine_1 [b];
NODE station_2 [K];
NODE num [K];
Production Line Examples
Batch Processing (2)

/* Arrival of jobs */
FROME TO num, buffer_1 W arrival;

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

/* station_2 */
FROM station_2, num TOE W ServiceRate_2;

Production Line Examples
Batch Processing (3)

/* Performance measures */
RESULT >> utilization_2 = UTIL station_2;
RESULT >> throughput = utilization_2 * ServiceRate_2;
RESULT >> IF (machine_1 = 0) P_0 += PROB;
RESULT >> utilization_1 = 1 - P_0;
RESULT >> WIP = MEAN num;
RESULT >> DIST num;
Production Line Examples
Unreliable Machines (1)

/* Declaration part */
enum state_a {up, down};

/* Definition of the state vector */
NODE station_1[K] = 0;
NODE station_2[K] = 0;
NODE num[K] = 0;
NODE server[state] = up;

Production Line Examples
Unreliable Machines (2)

/* Arrival of jobs */
FROME TO num, station_1 W arrival;

/* Failure and repair */
FROM server[up] TO server[down] W 1/mtbf;
FROM server[down] TO server[up] W 1/mtr;
### Production Line Examples

**Unreliable Machines (3)**

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

/* station_2 */
FROM station_2, num TOE W ServiceRate_2;

/* Performance Measures */
RESULT >> utilization_2 = UTIL station_2;
RESULT >> throughput_2 = utilization_2 * ServiceRate_2;
RESULT >> IF (server == up) upprob += PROB;
```

### Wafer Production System (1)

**Model of the wafer production system**

```
/* Nodes */
<1..5> NODE Buffer_# [K];
<1..5> NODE Active_# [m_#];
NODE Station_6 [K];
NODE num [K];
```
Wafer Production System (2)

/* Rules */
FROM TO Buffer_1, num W Arrival;
<1..5> FROM Buffer_# TO Active_#;
<1..m_1> FROM Active_1 TO Buffer_2 W #*mue_1 P p12 IF Active_1 == #;
<1..m_1> FROM Active_1 TO Buffer_3 W #*mue_1 P p13 IF Active_1 == #;
<1..m_2> FROM Active_2 TO Buffer_4 W #*mue_2 P p24 IF Active_2 == #;
<1..m_2> FROM Active_2 TO Buffer_5 W #*mue_2 P p25 IF Active_2 == #;
<1..m_3> FROM Active_3 TO Buffer_4 W #*mue_3 P p34 IF Active_3 == #;
<1..m_3> FROM Active_3 TO Buffer_5 W #*mue_3 P p35 IF Active_3 == #;
<1..m_4> FROM Active_4, num TOE W #*mue_4 P p40 IF Active_4 == #;
<1..m_4> FROM Active_4 TO Station_6 W #*mue_4 P p46 IF Active_4 == #;
FROM Active_5 TO Buffer_1 W mue_5;
FROM Station_6, num TOE P p60;
FROM Station_6 TO Buffer_1 P p61;

Wafer Production System (3)

/* Results */

/* Mean number of active machines */
<1..5> RESULT> A_# = MEAN Active_#;

/* Utilization of the machines */
<1..5> RESULT> utilization_# = A_#/m_#;

/* Mean buffer length */
<1..5> RESULT> Q_# = MEAN Buffer_#;

/* Mean number of wafers (WIP) */
RESULT> WIP = MEAN num;

/* Mean system time */
RESULT> T = WIP / Arrival;
Wafer Production System (4)

IGL Intermediate Graphic Language (1)
IGL Intermediate Graphic Language (2)

```c
/**====== Central-Server-mode ======*/
/**--------- Parameter declaration part------------------------*/
#define K 10
#define mue1 3.5
#define mue2 0.9
#define mue3 2.3
#define mue4 1.2
#define p12 0.25
#define p13 0.35
#define p14 0.4

/**--------- System state definition part ---------------------*/

NODE N1[K] = K;
<2..4> NODE N#[K];

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

IGL Intermediate Graphic Language (3)

`--------- Transition definition part -----------------------*/
FROM N1 WITH mue1 {
    <2..4> TO N# P p1#;
}
<2,3,4> FROM N# TO N1 WITH mue#;

`--------- Result part --------------------------------------*/
<1..4> RESULT >> IF (N#>0) rho# += PROB; // utilization
<1..4> RESULT nquer# = MEAN N#; // mean queue length
<1..4> RESULT DIST N#; // distribution (probability of each possible queue length)
<1..4> RESULT lambda# = rho# * mue#; // throughput
<1..4> RESULT tquer# = nquer# / lambda#; // mean time a job is in a node

`--------- Picture part--------------------------------------*/
PICTURE "Distribution for queue lengths"
    FONTSIZE 16
    CURVE DIST N1, N2, N3, N4
PICTURE lambdas_and_rhos
    FONT courier -FONTSIZE 20
    LIST lambda1, lambda2, lambda3, lambda4, rho1, rho2, rho3, rho4
LEGEND -NOVISIBLE
```
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:

<table>
<thead>
<tr>
<th>DIST</th>
<th>N1</th>
<th>0: 0.226490510324</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10: 0.00523876082994</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIST</th>
<th>N2</th>
<th>0: 0.247975421288</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10: 0.0039526685905</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIST</th>
<th>N3</th>
<th>0: 0.588022144338</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10: 0.62301477005e-06</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIST</th>
<th>N4</th>
<th>0: 0.0975734220455</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10: 0.0244732959731</td>
<td></td>
</tr>
</tbody>
</table>

\[ \rho_1 = 0.773509489676 \]
\[ \rho_2 = 0.752024578712 \]
\[ \rho_3 = 0.411977885562 \]
\[ \rho_4 = 0.902426577954 \]

\[ \lambda_1 = 2.70728321387 \]
\[ \lambda_2 = 0.676822120841 \]
\[ \lambda_3 = 0.947549136792 \]
\[ \lambda_4 = 1.08291189355 \]

\[ t_{quer1} = 0.9658958587825 \]
\[ t_{quer2} = 3.56550339435 \]
\[ t_{quer3} = 0.720806197492 \]
\[ t_{quer4} = 3.96046645608 \]
IGL Intermediate Graphic Language (6)

IGL Intermediate Graphic Language (7)
Real Life Examples

- MOSEL has been successfully used to model and to analyze systems from the following domains:
  - **Computer Systems:** UNIX Operating System, Polling Systems, Fork Join Systems, Terminal Systems, Multithreaded Architecture, Client Server, Multi Processor Systems
  - **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 a translator to CSPL (suitable for SPNP v. 6.x, and a Petri Net based analysis module which was recently added to the MOSEL-Tool)

- [http://www4.informatik.uni-erlangen.de/Projects/MOSEL/](http://www4.informatik.uni-erlangen.de/Projects/MOSEL/)
Future Work and Related Work

- Macros
- Parallel Solver
- Translator to other Tools
- Application to Special Systems
  - Computer Systems
  - Operating Systems
  - Communication Systems (Ethernet (CSMA/CD, Tokenring), FDDI, ATM, Cellular Mobil Networks)
  - Fault Tolerant Systems
- Simulation
- Non Phase Type Distribution