MoToR: The MoDeST Tool EnviRonment

H. Bohnenkamp, P.R. D'Argenio, H. Hermanns, A. Hartmans, J-P Katoen, J. Klaren, and many more people

MOVES,RWTH Aachen University, D
FMT, University Twente, NL
Saarbrücken Universit, D
FaMAF, Universidad Nacional de Córdoba, AR

VOSS 2, Leiden, 2007

MoDeST

MoDeST

Modeling and Description Language for Stochastic and Timed Systems

Supported Concepts:

- Clocks
- Variables
- Samples
- Delays and urgency

- Local probabilistic branching
- Processes
- Actions
- Parallel composition
- Synchronisation

- Formal semantics: Stochastic Timed Automata (STA)

MoDeST

MoDeST

Modeling and Description Language for Stochastic and Timed Systems

Supported Concepts:

- Clocks
- Variables
- Samples
- Delays and urgency

- Local probabilistic branching
- Processes
- Actions
- Parallel composition
- Synchronisation

- Formal semantics: Stochastic Timed Automata (STA)

MoDeST

MoDeST

Modeling and Description Language for Stochastic and Timed Systems

Supported Concepts:

- Clocks
- Variables
- Samples
- Delays and urgency

- Local probabilistic branching
- Processes
- Actions
- Parallel composition
- Synchronisation

- Formal semantics: Stochastic Timed Automata (STA)

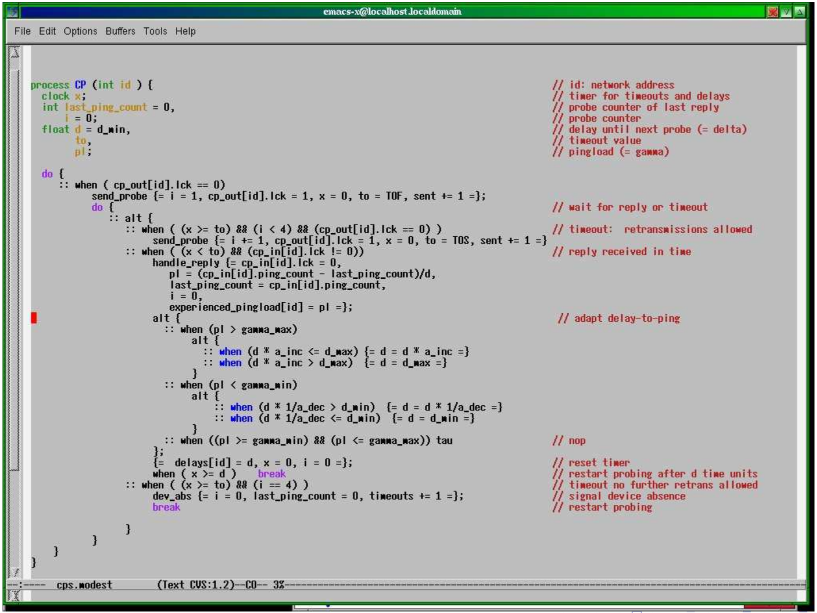
MoDeST

A MoDeST example:

```
clock c1, c2, c3;
float x1, x2, x3;

x1 = Exponential(1.0); c1 = 0;
x2 = Exponential(10.0); c2 = 0;
x3 = Uniform(1.0, 100.0); c3 = 0;
alt{
   :: when (c1 >= x1) P1()
   :: when (c2 >= x2) P2()
   :: when (c3 >= x3) P3()
}
```

A slightly more MoDeST example $\Longrightarrow$


emacs-x@localhost.localdomain
File Edit Options Buffers Tools Help

```
process CP (int id ) {                                                                        // id: network address
  clock x;                                                                                    // timer for timeouts and delays
  int last_ping_count = 0,                                                                    // probe counter of last reply
      i = 0;                                                                                  // probe counter
  float d = d_min,                                                                            // delay until next probe (= delta)
        to,                                                                                   // timeout value
        pl;                                                                                   // pingload (= gamma)

  do {
     :: when ( cp_out[id].lck == 0)
           send_probe {= i = 1, cp_out[id].lck = 1, x = 0, to = TOF, sent += 1 =};
           do {                                                                               // wait for reply or timeout
              :: alt {
                 :: when ( (x >= to) && (i < 4) && (cp_out[id].lck == 0) )                    // timeout:  retransmissions allowed
                      send_probe {= i += 1, cp_out[id].lck = 1, x = 0, to = TOS, sent += 1 =}
                 :: when ( (x < to) && (cp_in[id].lck != 0))                                  // reply received in time
                      handle_reply {= cp_in[id].lck = 0,
                         pl = (cp_in[id].ping_count - last_ping_count)/d,
                         last_ping_count = cp_in[id].ping_count,
                         i = 0,
                         experienced_pingload[id] = pl =};
                      alt {                                                                    // adapt delay-to-ping
                        :: when (pl > gamma_max)
                             alt {
                               :: when (d * a_inc <= d_max) {= d = d * a_inc =}
                               :: when (d * a_inc > d_max)  {= d = d_max =}
                             }
                        :: when (pl < gamma_min)
                             alt {
                                 :: when (d * 1/a_dec > d_min)  {= d = d * 1/a_dec =}
                                 :: when (d * 1/a_dec <= d_min)  {= d = d_min =}
                             }
                        :: when ((pl >= gamma_min) && (pl <= gamma_max)) tau                  // nop
                      };
                      {=  delays[id] = d, x = 0, i = 0 =};                                    // reset timer
                      when ( x >= d )    break                                                // restart probing after d time units
                 :: when ( (x >= to) && (i == 4) )                                            // timeout no further retrans allowed
                      dev_abs {= i = 0, last_ping_count = 0, timeouts += 1 =};                // signal device absence
                      break                                                                   // restart probing

                 }
           }
    }
}
--:----  cps.modest       (Text CVS:1.2)--C0-- 3%----------------------------------------------------------------------
```

MoDeST

Model classes

- Labelled transition systems
- Timed automata
- Stochastic processes ($\geq$ class GSMPs)
- Probabilistic automata
- Markov decision processes
- Stochastic automata

and some combinations

Analysis of MoDeST models

Single-formalism, multi-solution approach

- One all-encompassing model
- Extraction of simpler models
- Analysis with existing tools

Example

- Abstract from stochastic and probabilistic information
- Feed resulting timed automaton into UPPAAL

Analysis of MoDeST models

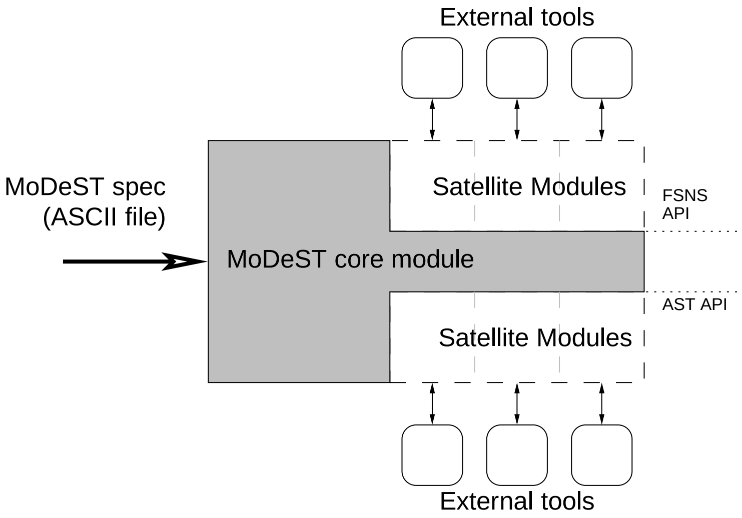
Single-formalism, multi-solution approach

- One all-encompassing model
- Extraction of simpler models
- Analysis with existing tools

Example

- Abstract from stochastic and probabilistic information
- Feed resulting timed automaton into UPPAAL

MoToR


External tools
MoDeST spec
(ASCII file)
Satellite Modules
FSNS
API
MoDeST core module
AST API
Satellite Modules
External tools

MoToR

The Möbius-connection

- Möbius provides DE simulator and statistical evaluation
- largest MoDeST model class covered
  - non-determinism is a problem
- global MoDeST variables become reward variables

Types of measures:

- Mean values, variances, distributions
- Point measures, cumulative rewards, steady state

MoToR

The Möbius-connection

- Möbius provides DE simulator and statistical evaluation
- largest MoDeST model class covered
  - non-determinism is a problem
- global MoDeST variables become reward variables

Types of measures:

- Mean values, variances, distributions
- Point measures, cumulative rewards, steady state

MoToR

The Möbius-connection

- Möbius provides DE simulator and statistical evaluation
- largest MoDeST model class covered
  - non-determinism is a problem
- global MoDeST variables become reward variables

Types of measures:

- Mean values, variances, distributions
- Point measures, cumulative rewards, steady state

MoToR

The CADP connection

- Plain LTS can be generated
- Output in bcg-format
- Analysis with CADP possible
- Very slow and immature

MoToR

UPPAAL connection (Univ. Saarbrücken)

- Protoype translator from MoDeST to Network of TA (UPPAAL)
- Not quite finished

Open issues

- MoDeST synchronisation vs. UPPAAL communication
- different notions of urgency

MoToR

MoDeST extension: value passing (Univ. Saarbrücken)

- Extension of MoDeST (and MoToR) with value passing (LOTOS-style)
- More convenient modeling (before: shared variables)
- Changes in semantics are currently incorporated into MoToR

Sideshows:

Eclipse plugin (Univ. Saarbrücken)

- Editor plugin for MoDeST in Eclipse
- Implemented in JAVA, not part of MoToR
- Single-step simulator with syntactic highlighting
- Currently: data-abstract evaluation
- Data evaluation in development

Availability

`http://www.purl.org/net/motor`