VIATRA/GRASS: Graph Transformation-based Stochastic Simulation

(1) Paolo Torrini, Reiko Heckel, (2) Istvan Rath

pt95,reiko@mcs.le.ac.uk, rath@mit.bmi.hu

(1) University of Leicester,

(2) Budapest University of Technology and Economics

Stochastic graph transformation

- Modelling: graph transformation
- Validation: stochastic simulation
  - runs depends on random numbers
  - useful when models are too large for model checking
- Generalised stochastic graph transformation:
  events associated with general probability distributions
- Discrete event system semantics
- Implementation based on incremental pattern matching

Some history

- 2004-07: Heckel, Lajios, Menge — seminal work on stochastic graph transformation
- rules associated with exponential distributions, translated to labelled transition systems, analysed as Markov chains (probabilistic model checking)
- 2007: Heckel: application to P2P networks
- 2007: Lajos, Kosiuczenko — outline extension, rules with general distributions, semi-Markov processes, unfolding semantics (global name-space needed)

... and more recently

- 2009: Heckel, Torrini — rule matches with general distributions,
  concrete semantics (numbered graphs) and extensions
- Torrini, Rath — implementation based on VIATRA
- Ajab Kahn — application to Skype (PhD topic)
- Kahn, Torrini, Heckel — ICGT 2008 Doctoral Symposium
- Torrini, Heckel, Rath — paper submitted at FASE 2010

GTS with probability

- Stochastic Graph Transformation:
  rules associated with exponential probability distributions
- Generalised Stochastic Graph Transformation:
  rule matches associated with general probability distributions
- Rule matches as equivalence classes — identity through transformation
  — cardinality restrictions to ensure they are a proper set (numbered graphs)
- Probabilistic rather than indeterministic actions
- Continuous time: waiting times as independent random variables — no parallelism

Probability distributions

- Possible rule application associated with expected delay D
- D (waiting time) — random variable associated to a probability distribution function $F_D(x) = P(D \leq x)$
  F_D determines the probability of the delay being less than x
- Markov property — process depends on present state only, $P(D > x + z | D > z) = P(D > x)$
  Semi-Markov process: next state may depend on time spent in current state
- Exponential distribution: determined by a rate, can express how “fast” is a Markov process.
- Normal distribution: mean and variance — process with meaningful average value and deviation

GSMS

Discrete event systems semantics — stochastic models based on Generalised Semi-Markov Schemes (more general than Markov chains, general distributions)

$$\begin{aligned} GSMS \quad = \langle \quad & States \\ & Events \\ & ActiveEvents : State \rightarrow \wp Event \\ & Transition : State \times Event \rightarrow State \\ & DistrAssign : Event \rightarrow Distribution \\ & InitState : State \qquad \rangle \end{aligned}$$

where $Distribution \; = \; \mathcal{R} \rightarrow [1, 0]$

GSGTS as GSMS

A Generalised Stochastic GTS defines a GSMS, where $\Delta(\langle r, m\rangle)(d)$ is the probability that the waiting time for rule r at match m is less than d

$$
\begin{aligned}
GSGTS \quad = \langle \quad & ReachGraphs \\
& RuleMatches \qquad \text{(equivalence classes)} \\
& EnabledMatches : ReachGraph \rightarrow \wp RuleMatch \\
& GraphTrans : \\
& \qquad ReachGraph \times RuleMatch \rightarrow ReachGraph \\
& \Delta : RuleMatch \rightarrow (\mathcal{R} \rightarrow [1, 0]) \\
& InitialGraph : ReachGraph \qquad \rangle
\end{aligned}
$$

GSMS-based simulation

- GSMS execution based on event scheduling scheme
- At each step
  - the event with the shortest waiting time is executed
  - the simulation time is updated
  - enabled matches are computed
  - scheduling times of new matches are determined by random number generator given Δ
  - waiting times of old matches decrease

Computational aspect

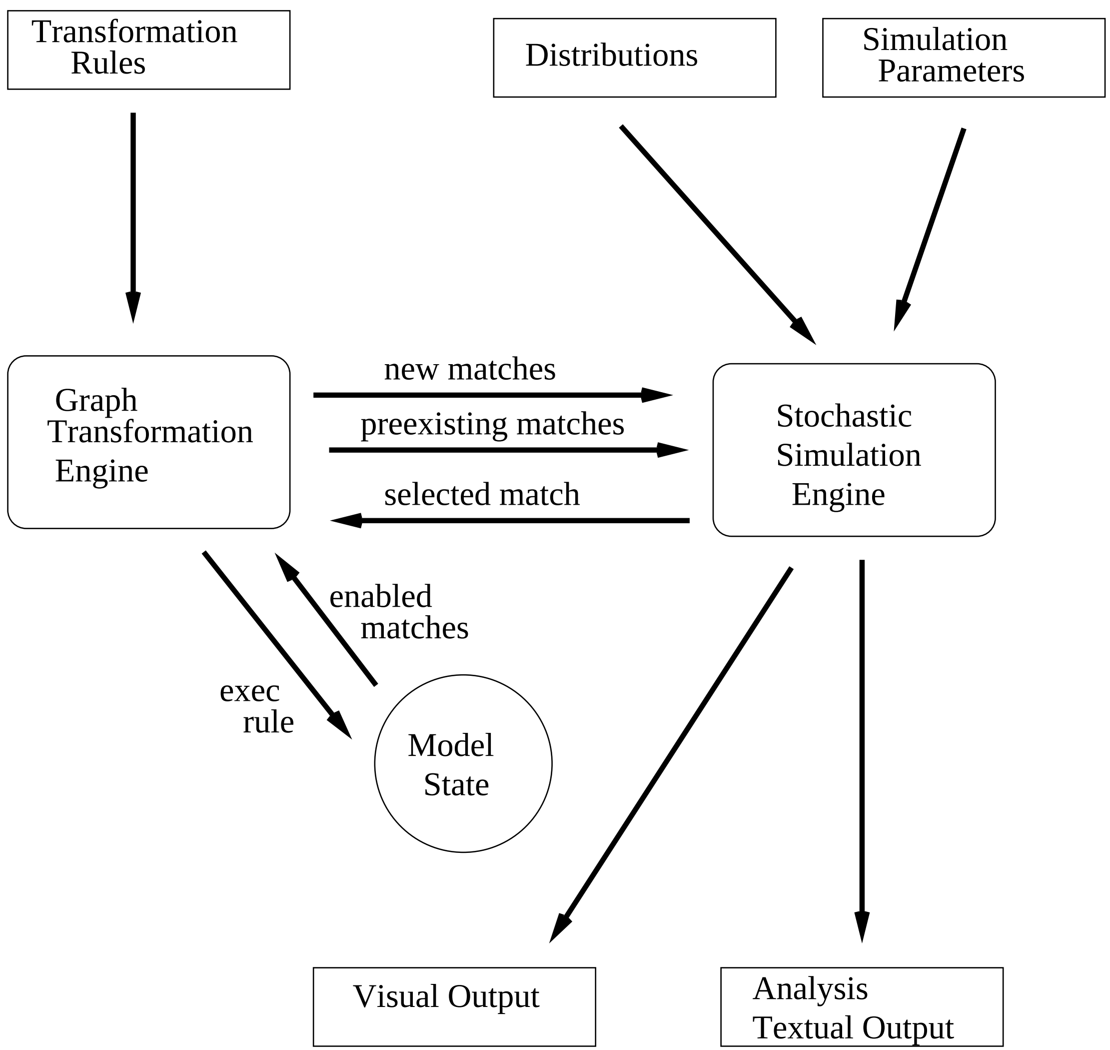
- Substantial problem: computing all matches at each step, needed because
  - not enough to know that a rule is enabled — number of matches makes difference in actual probability of rule application
  - waiting times may depend on local values of attributes
- Moreover, we need to retain identity of matches — so we cannot recompute the matches at each step

Incremental Pattern Matching

- Incremental approach (RETE algorithm):
  pattern-matching problem constant in model size, polynomial in rule number — after initialisation phase, which can be hard (subgraph homomorphism problem known to be NP-complete)
- Standard approach: update constant in model size and rule number
- IPM useful when rules have complex LHS and when all matches are needed
- VIATRA (Eclipse plugin) — graph transformation engine that implements IPM

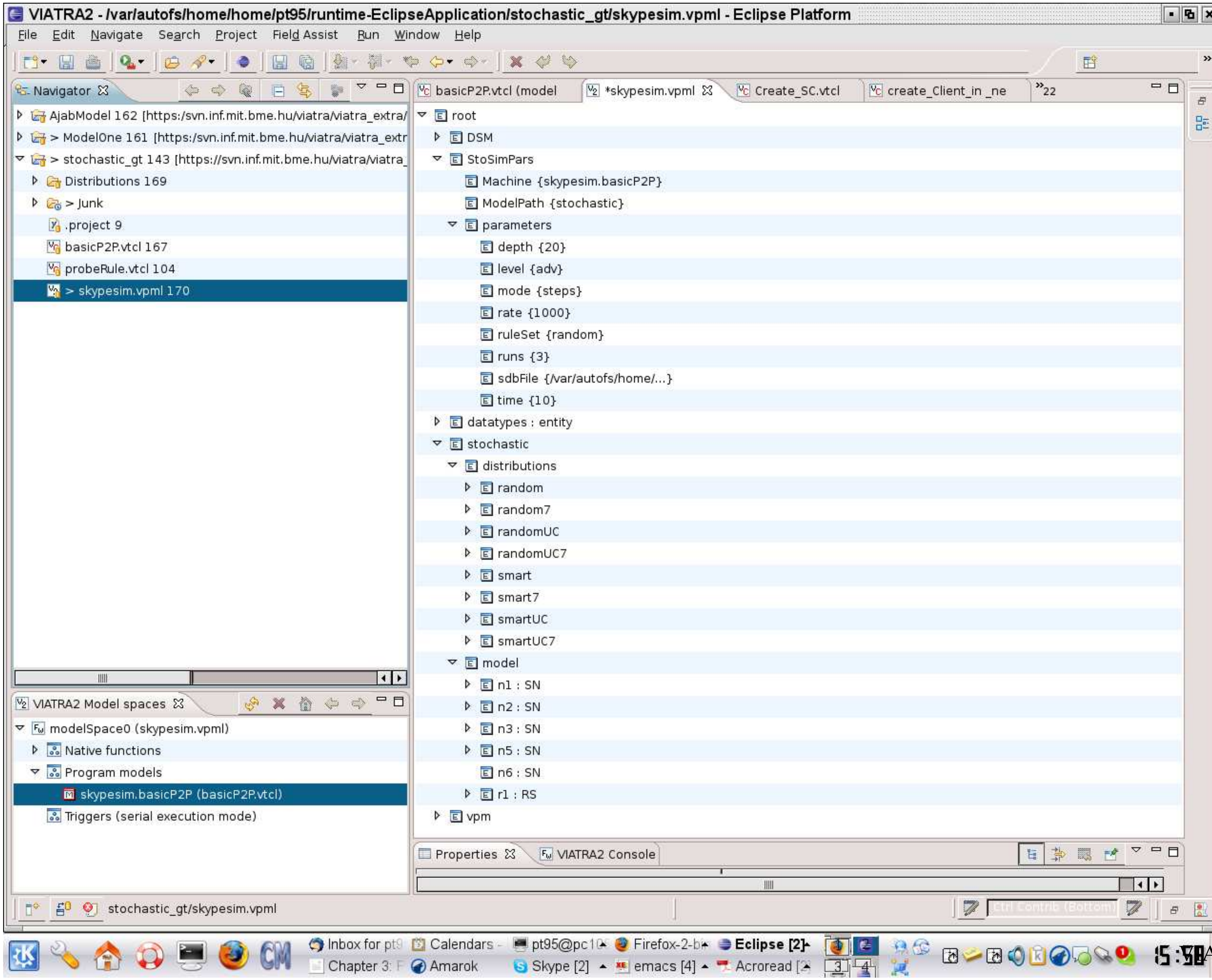
Architecture of the tool

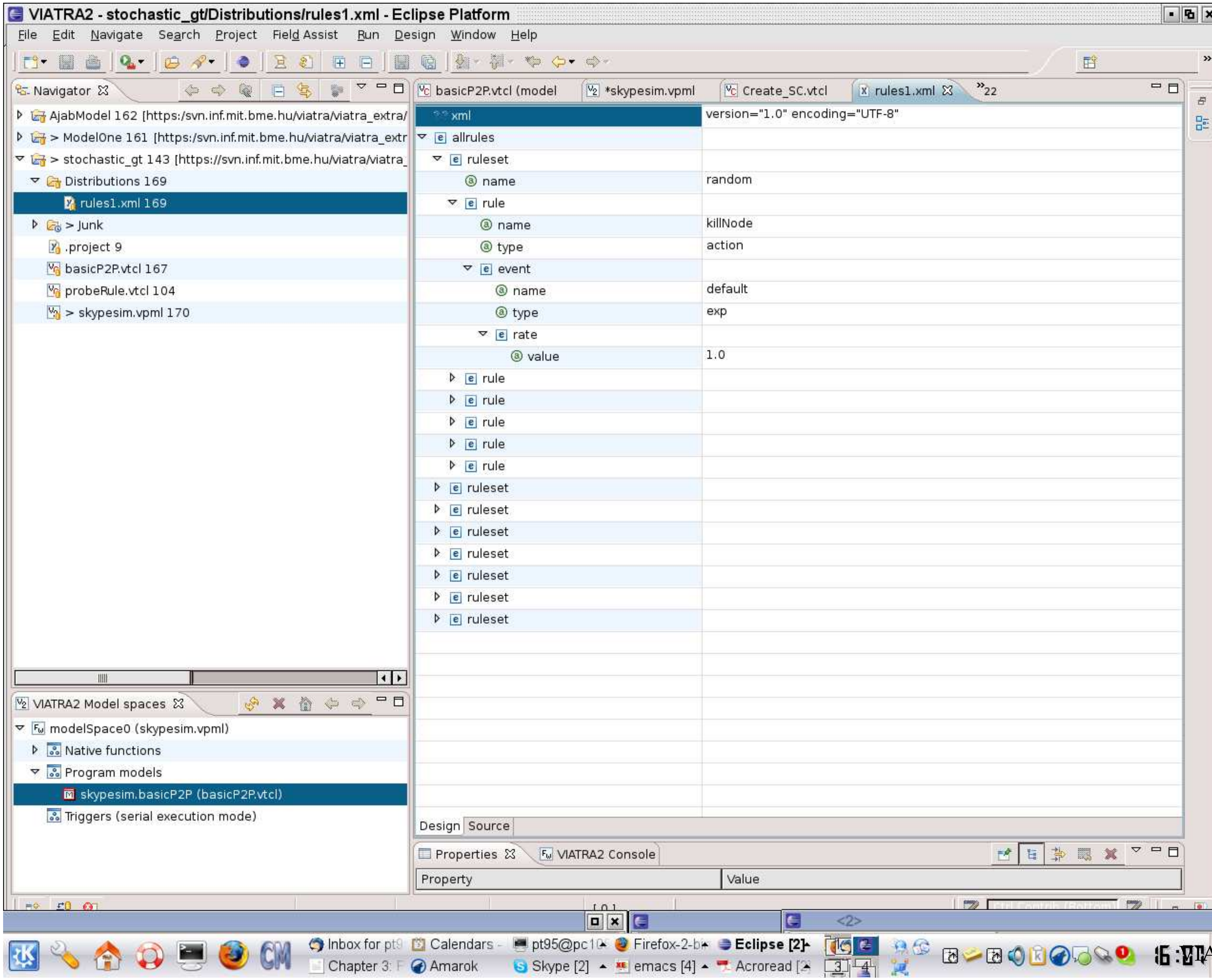
- Graph transformation engine
  - computes matches
  - executes selected rule match
- Simulation engine
  - determines waiting time, relying on SSJ random number generation
  - manages waiting times
  - selects rule matches for execution
  - extracts statistics, relying on SSJ tally classes
  - controls textual and visual output

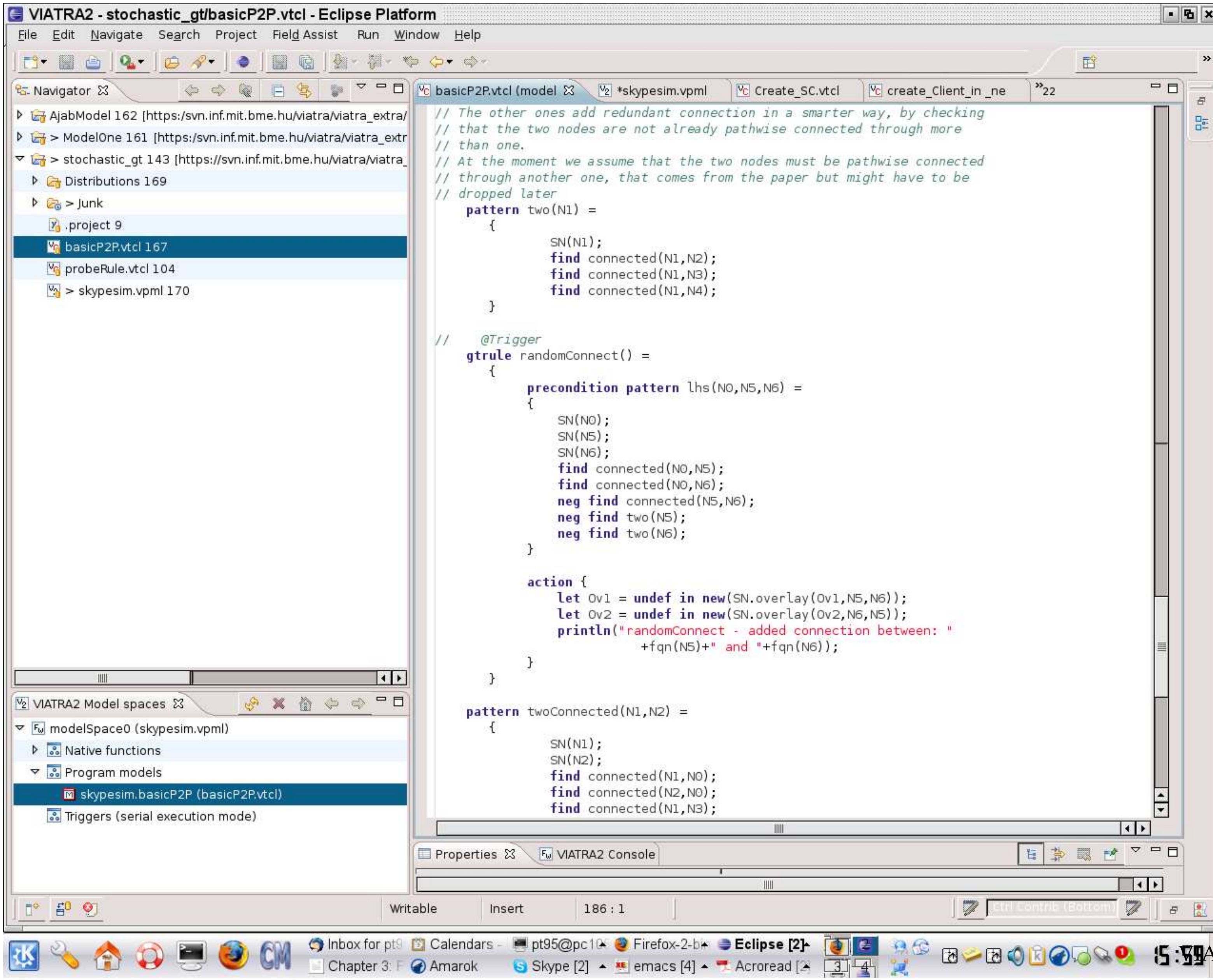

Transformation Rules
Distributions
Simulation Parameters
Graph Transformation Engine
new matches
preexisting matches
selected match
Stochastic Simulation Engine
enabled matches
exec rule
Model State
Visual Output
Analysis Textual Output

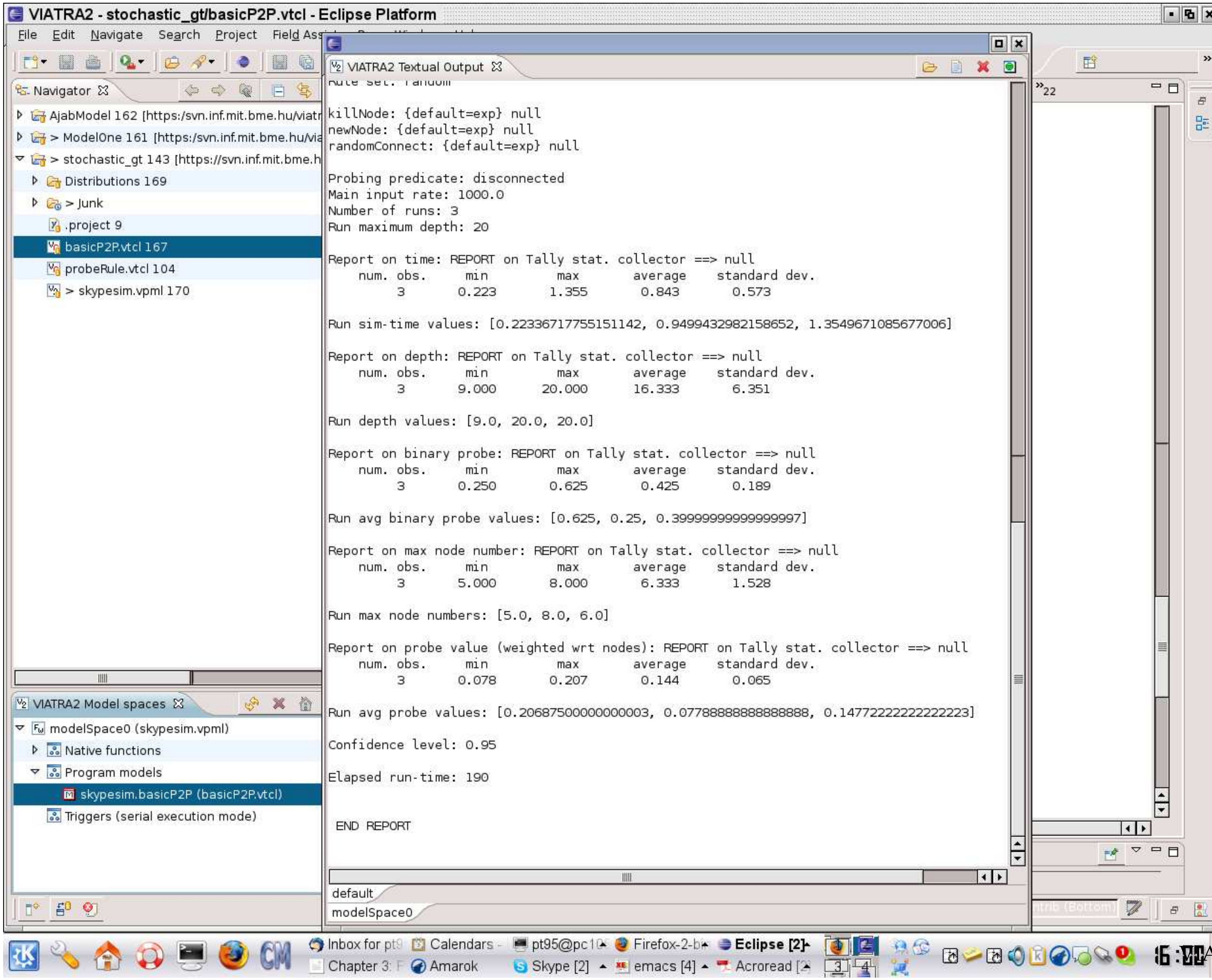
VIATRA/GRASS

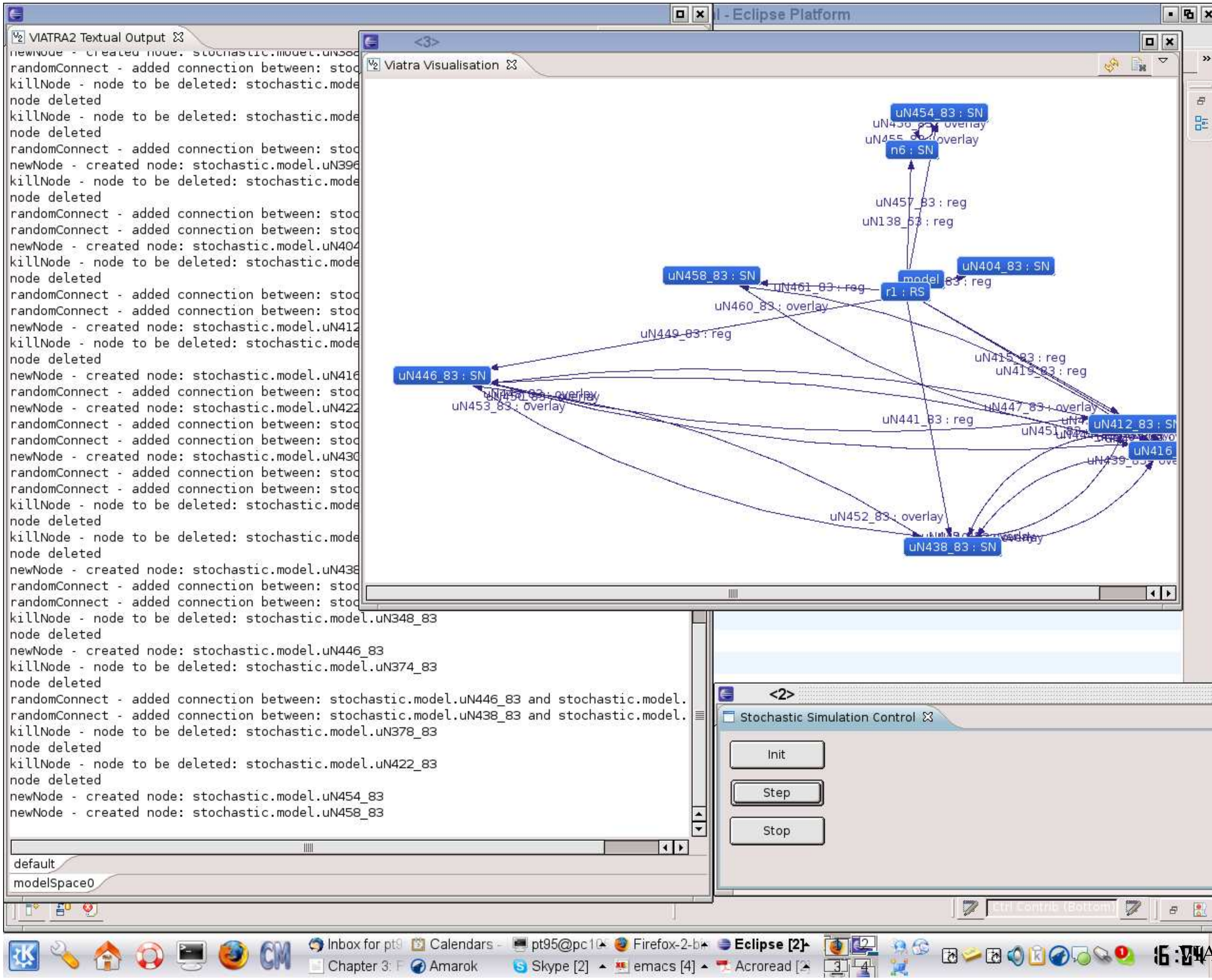
- Stochastic simulation implemented in Java on top of VIATRA — uses Java-SSJ libraries
- inputs: G (GTS), Δ (distribution assignment)
- G: loaded in VIATRA — model-space (*vpml*) and rules (*vtcl*)
- Δ: case-defined in an XML file, automatically translated to a model-space entity
- probe rules: extract information for stochastic analysis — collected in tally class objects, displayed as textual output
- additional simulation parameters: e.g. number of runs, max depth of each run (steps or time)
- visualisation for staged execution


VIATRA2 - /var/autofs/home/home/pt95/runtime-EclipseApplication/stochastic_gt/skypesim.vpml - Eclipse Platform
File Edit Navigate Search Project Field Assist Run Window Help
Navigator
AjabModel 162 [https:/svn.inf.mit.bme.hu/viatra/viatra_extra/
> ModelOne 161 [https:/svn.inf.mit.bme.hu/viatra/viatra_extr
> stochastic_gt 143 [https://svn.inf.mit.bme.hu/viatra/viatra_
Distributions 169
> Junk
.project 9
basicP2P.vtcl 167
probeRule.vtcl 104
> skypesim.vpml 170
basicP2P.vtcl (model
*skypesim.vpml
Create_SC.vtcl
create_Client_in_ne
22
root
DSM
StoSimPars
Machine {skypesim.basicP2P}
ModelPath {stochastic}
parameters
depth {20}
level {adv}
mode {steps}
rate {1000}
ruleSet {random}
runs {3}
sdbFile {/var/autofs/home/...}
time {10}
datatypes : entity
stochastic
distributions
random
random7
randomUC
randomUC7
smart
smart7
smartUC
smartUC7
model
n1 : SN
n2 : SN
n3 : SN
n5 : SN
n6 : SN
r1 : RS
vpm
VIATRA2 Model spaces
modelSpace0 (skypesim.vpml)
Native functions
Program models
skypesim.basicP2P (basicP2P.vtcl)
Triggers (serial execution mode)
Properties
VIATRA2 Console
stochastic_gt/skypesim.vpml


VIATRA2 - stochastic_gt/Distributions/rules1.xml - Eclipse Platform
File Edit Navigate Search Project Field Assist Run Design Window Help
Navigator
AjabModel 162 [https://svn.inf.mit.bme.hu/viatra/viatra_extra/
> ModelOne 161 [https://svn.inf.mit.bme.hu/viatra/viatra_extr
> stochastic_gt 143 [https://svn.inf.mit.bme.hu/viatra/viatra_
Distributions 169
rules1.xml 169
> Junk
.project 9
basicP2P.vtcl 167
probeRule.vtcl 104
> skypesim.vpml 170
VIATRA2 Model spaces
modelSpace0 (skypesim.vpml)
Native functions
Program models
skypesim.basicP2P (basicP2P.vtcl)
Triggers (serial execution mode)
basicP2P.vtcl (model
*skypesim.vpml
Create_SC.vtcl
rules1.xml
xml
version="1.0" encoding="UTF-8"
allrules
ruleset
name
random
rule
name
killNode
type
action
event
name
default
type
exp
rate
value
1.0
rule
rule
rule
rule
rule
ruleset
ruleset
ruleset
ruleset
ruleset
ruleset
ruleset
Design
Source
Properties
VIATRA2 Console
Property
Value
Inbox for pt9
Calendars
pt95@pc1
Firefox-2-b
Eclipse [2]
Chapter 3
Amarok
Skype [2]
emacs [4]
Acroread

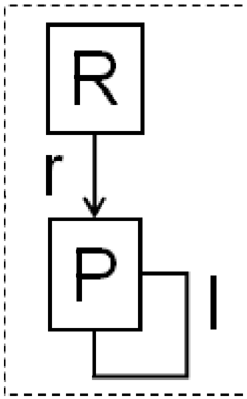

VIATRA2 - stochastic_gt/basicP2P.vtcl - Eclipse Platform
File Edit Navigate Search Project Field Assist Run Window Help
Navigator
AjabModel 162 [https:/svn.inf.mit.bme.hu/viatra/viatra_extra/
> ModelOne 161 [https:/svn.inf.mit.bme.hu/viatra/viatra_extr
> stochastic_gt 143 [https://svn.inf.mit.bme.hu/viatra/viatra_
Distributions 169
> Junk
.project 9
basicP2P.vtcl 167
probeRule.vtcl 104
> skypesim.vpml 170
VIATRA2 Model spaces
modelSpace0 (skypesim.vpml)
Native functions
Program models
skypesim.basicP2P (basicP2P.vtcl)
Triggers (serial execution mode)
basicP2P.vtcl (model
*skypesim.vpml
Create_SC.vtcl
create_Client_in _ne
// The other ones add redundant connection in a smarter way, by checking
// that the two nodes are not already pathwise connected through more
// than one.
// At the moment we assume that the two nodes must be pathwise connected
// through another one, that comes from the paper but might have to be
// dropped later
pattern two(N1) =
{
SN(N1);
find connected(N1,N2);
find connected(N1,N3);
find connected(N1,N4);
}
// @Trigger
gtrule randomConnect() =
{
precondition pattern lhs(N0,N5,N6) =
{
SN(N0);
SN(N5);
SN(N6);
find connected(N0,N5);
find connected(N0,N6);
neg find connected(N5,N6);
neg find two(N5);
neg find two(N6);
}
action {
let Ov1 = undef in new(SN.overlay(Ov1,N5,N6));
let Ov2 = undef in new(SN.overlay(Ov2,N6,N5));
println("randomConnect - added connection between: "
+fqn(N5)+" and "+fqn(N6));
}
}
pattern twoConnected(N1,N2) =
{
SN(N1);
SN(N2);
find connected(N1,N0);
find connected(N2,N0);
find connected(N1,N3);
Properties
VIATRA2 Console
Writable
Insert
186 : 1
Inbox for pt9
Calendars
pt95@pc1
Firefox-2-b
Eclipse [2]
Chapter 3
Amarok
Skype [2]
emacs [4]
Acroread


VIATRA2 - stochastic_gt/basicP2P.vtcl - Eclipse Platform
File Edit Navigate Search Project
Navigator
AjabModel 162 [https:/svn.inf.mit.bme.hu/viatr
> ModelOne 161 [https:/svn.inf.mit.bme.hu/via
> stochastic_gt 143 [https://svn.inf.mit.bme.h
Distributions 169
> Junk
.project 9
basicP2P.vtcl 167
probeRule.vtcl 104
> skypesim.vpml 170
VIATRA2 Model spaces
modelSpace0 (skypesim.vpml)
Native functions
Program models
skypesim.basicP2P (basicP2P.vtcl)
Triggers (serial execution mode)
VIATRA2 Textual Output
killNode: {default=exp} null
newNode: {default=exp} null
randomConnect: {default=exp} null
Probing predicate: disconnected
Main input rate: 1000.0
Number of runs: 3
Run maximum depth: 20
Report on time: REPORT on Tally stat. collector ==> null
num. obs. min max average standard dev.
3 0.223 1.355 0.843 0.573
Run sim-time values: [0.22336717755151142, 0.9499432982158652, 1.3549671085677006]
Report on depth: REPORT on Tally stat. collector ==> null
num. obs. min max average standard dev.
3 9.000 20.000 16.333 6.351
Run depth values: [9.0, 20.0, 20.0]
Report on binary probe: REPORT on Tally stat. collector ==> null
num. obs. min max average standard dev.
3 0.250 0.625 0.425 0.189
Run avg binary probe values: [0.625, 0.25, 0.39999999999999997]
Report on max node number: REPORT on Tally stat. collector ==> null
num. obs. min max average standard dev.
3 5.000 8.000 6.333 1.528
Run max node numbers: [5.0, 8.0, 6.0]
Report on probe value (weighted wrt nodes): REPORT on Tally stat. collector ==> null
num. obs. min max average standard dev.
3 0.078 0.207 0.144 0.065
Run avg probe values: [0.20687500000000003, 0.07788888888888888, 0.14772222222222223]
Confidence level: 0.95
Elapsed run-time: 190
END REPORT
default
modelSpace0
Inbox for pt
Calendars
pt95@pc1
Firefox-2-b
Eclipse [2]
Chapter 3
Amarok
Skype [2]
emacs [4]
Acroread

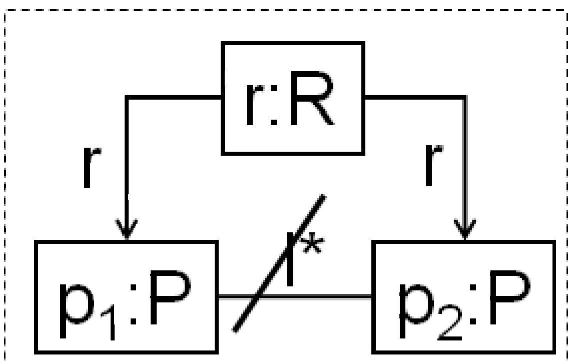

Eclipse Platform
VIATRA2 Textual Output
randomConnect - added connection between: stoc
killNode - node to be deleted: stochastic.mode
node deleted
newNode - created node: stochastic.model.uN396
newNode - created node: stochastic.model.uN404
newNode - created node: stochastic.model.uN412
newNode - created node: stochastic.model.uN416
newNode - created node: stochastic.model.uN422
newNode - created node: stochastic.model.uN430
newNode - created node: stochastic.model.uN438
killNode - node to be deleted: stochastic.model.uN348_83
node deleted
newNode - created node: stochastic.model.uN446_83
killNode - node to be deleted: stochastic.model.uN374_83
node deleted
randomConnect - added connection between: stochastic.model.uN446_83 and stochastic.model.
randomConnect - added connection between: stochastic.model.uN438_83 and stochastic.model.
killNode - node to be deleted: stochastic.model.uN378_83
node deleted
killNode - node to be deleted: stochastic.model.uN422_83
node deleted
newNode - created node: stochastic.model.uN454_83
newNode - created node: stochastic.model.uN458_83
default
modelSpace0
<3>
Viatra Visualisation
uN454_83 : SN
n6 : SN
uN457_83 : reg
uN138_83 : reg
uN458_83 : SN
uN404_83 : SN
r1 : RS
uN461_83 : reg
uN460_83 : overlay
uN449_83 : reg
uN446_83 : SN
uN415_83 : reg
uN419_83 : reg
uN453_83 : overlay
uN447_83 : overlay
uN441_83 : reg
uN412_83 : SN
uN416
uN452_83 : overlay
uN438_83 : SN
<2>
Stochastic Simulation Control
Init
Step
Stop

Case study: P2P

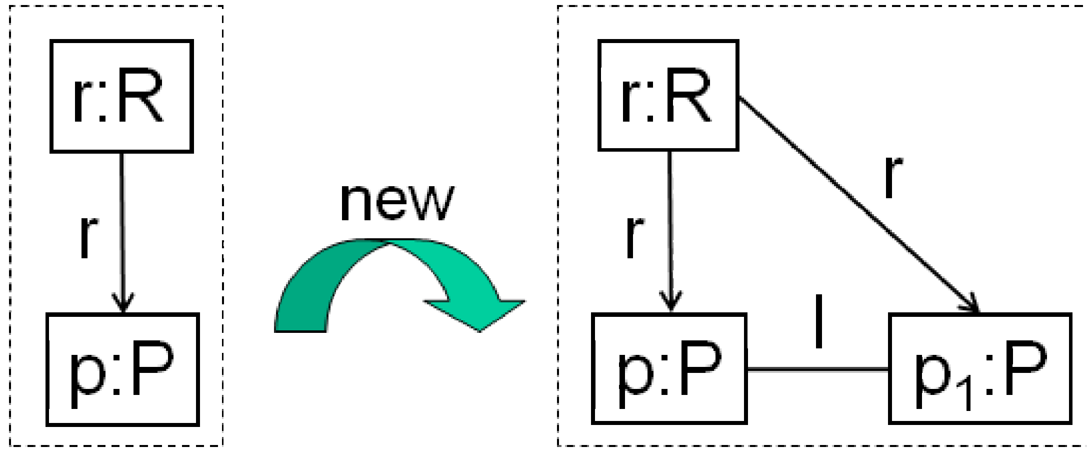
- Simulation of P2P network
- Basic example — rules with exponential distributions
- Two behavioural rules: create node, kill node
- Two alternative reconfiguration rules: randomly/smartly create redundant connection
- Stochastic analysis based on number of pathwise disconnected nodes at each step, on varying the rate of reconfiguration
- Small model, few seconds execution time

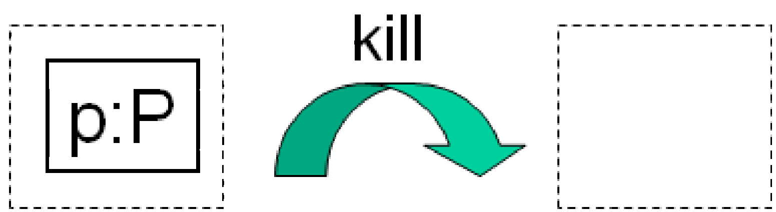


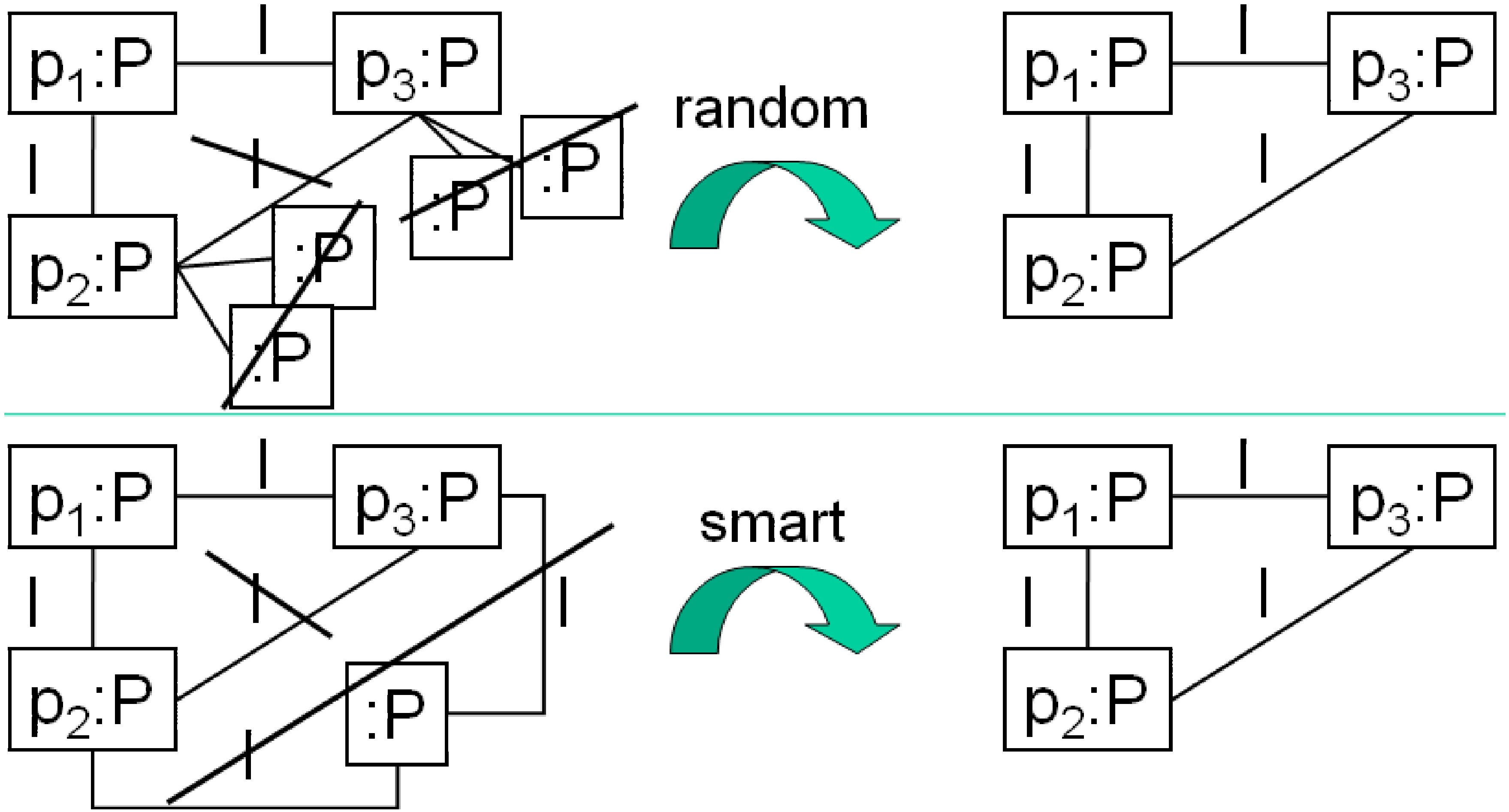
Type Graph
P = peer
R = registry



disconnected







p1:P
l
p3:P
l
l
l
l
l
p2:P
:P
:P
:P
:P
random
p1:P
l
p3:P
l
l
p2:P
p1:P
l
p3:P
l
l
l
l
p2:P
:P
smart
p1:P
l
p3:P
l
l
p2:P

```
pattern pathEx(N1,N2) = {
                 SN(N1);
                 SN(N2);
                 find connected(N1,N2); }
            or { SN(N1);
              SN(N2);
              SN(N0);
              find connected(N1,N0);
              find pathEx(N0,N2); }
```

```
pattern noPathEx(N1,N2) = {
       SN(N1);
               SN(N2);
               neg find pathEx(N1, N2); }

 gtrule disconnected() = {
       precondition pattern lhs(N1, N2) = {
            SN(N1);
            SN(N2);
            find noPathEx(N1, N2); }
       action {println("..."+fqn(N1)+fqn(N2));}}
```

Stochastic analysis

- Program can execute given number of simulation runs up to given depth value, expressed either as sim-time or step number
- returns average number of probe matches, max number of nodes, number of steps, and simulation time for each run
- average, min, max and deviation over all the runs
- more specific — wrt to the P2P probe: statistics on M/N^2 with M number of probe matches, N number of nodes
- Basic method: run many simulations long enough, until similar results for probe matches are obtained
- Hypothesis on behaviour tested by changing reconfiguration rates, comparing models, etc..

Sample results

Model: P2P	Disconnected	Number of steps	Max number of peers	Runtime
random:1	0.46	33	6	5
random:10	0.62	71	8	8
random:100	0.55	86	8	7
random:1000	0.89	284	20	10
random:10,000	0.46	116	8	9
smart:1	1.33	18	5	1
smart:10	0.01	90	8	4
smart:100	0.00	3561	48	10
smart:1000	0.00	998	24	10
smart:10,000	0.00	62	8	3

Extending the model

- Distributions may depend on attributes of match elements
- global variables — e.g. simulation time
- Derived attributes, computed when needed
  — depending on global variables,
  depending on local information — incoming/outgoing edges
- Distributions depending on derived attributes
- Spatial dependencies — matches “in the same region”
- Distributions depending on attributes of nearby matches
- trying to take advantage of incremental pattern matching

Further work

- Handling general distributions (beyond exponential and normal ones)
- Refining stochastic analysis
- Improving scalability
- Modelling VoIP networks (with Ajab Khan)
- Synchronisation of textual and visual output
- Comparison with other tools/approaches