Spike - a command line tool for continuous, stochastic & hybrid simulation of (coloured) Petri nets

Jacek Chodak, Monika Heiner (supervisor)
Computer Science Institute, Brandenburg University of Technology
Cottbus, Germany
jacek.chodak@b-tu.de, monika.heiner@b-tu.de
http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Spike
Agenda

- PetriNuts framework
- Why Spike
- Spike functionality
  - Simulation
  - Conversion
  - Reduction
  - Reproducibility
- Architecture
- Use cases
- Present status
- Future works
**PetriNuts Framework**

**CHARLIE [1]**
- Structural analysis
- Analyses building on incidence matrix
- Reachability graph analysis
- Siphon/trap computation
- Explicit CTL model checking
- Structural reduction

**SNOOPY [3, 4]**
- Modelling
- Simulation:
  - stochastic,
  - continuous,
  - hybrid
- Animation

**MARCE [2]**
- Symbolic analysis (PN)
  - State space analysis
  - CTL model checking
- Symbolic analysis (SPN)
  - Standard numerical analysis,
  - CSRL model checking
- Simulative analysis (SPN)
  - Standard numerical analysis
  - PLTe model checking
  - Explicit approximative numerical analysis

**SPIKE**
- Distributed simulation
  - stochastic,
  - continuous,
  - hybrid

**PATTY [5]**
- web based animation

---

**References:**
why Spike

* Speedup simulation Models can contain dozens of thousands of nodes. To speed up simulation, a model can be reduced or divided into modules (spatial decomposition, decomposition by node types) and simulated in a distributed way.

* Reproducibility There are many parameters: model parameters (e.g., initial marking, kinetic constants); simulations parameters (e.g., type of algm, length of trace, number of stochastic runs). Experiments with Spike are documented by configuration files.

* Simplifying workflow of simulation experiments CLI controlling multiple simulations without user intervention; typically for different model configurations and/or simulator configurations.
why Spike

* Speedup simulation  Models can contain dozens of thousands of nodes. To speed up simulation, a model can be reduced or divided into modules (spatial decomposition, decomposition by node types) and simulated in a distributed way.

* Reproducibility  There are many parameters: model parameters (e.g., initial marking, kinetic constants); simulations parameters (e.g., type of algm, length of trace, number of stochastic runs). Experiments with Spike are documented by configuration files.

* Simplifying workflow of simulation experiments  CLI controlling multiple simulations without user intervention; typically for different model configurations and/or simulator configurations.
why Spike

* **Speedup simulation** Models can contain dozens of thousands of nodes. To speed up simulation, a model can be reduced or divided into modules (spatial decomposition, decomposition by node types) and simulated in a distributed way.

* **Reproducibility** There are many parameters: model parameters (e.g., initial marking, kinetic constants); simulations parameters (e.g., type of algm, length of trace, number of stochastic runs). Experiments with Spike are documented by configuration files.

* **Simplifying workflow of simulation experiments** CLI controlling multiple simulations without user intervention; typically for different model configurations and/or simulator configurations.
Simulation

Supported (coloured) PN classes:
* SPN - Stochastic Petri Net (CONTINUOUS TIME MARKOV CHAINS)
* CPN - Continuous Petri Net (ODEs)
* HPN - Hybrid Petri Net
Simulation

Supported (coloured) PN classes:
- SPN - Stochastic Petri Net (Continuous Time Markov Chains)
- CPN - Continuous Petri Net (ODEs)
- HPN - Hybrid Petri Net

\[
\text{SPN} \iff \text{CPN} \iff \text{HPN}
\]
Simulation

* Stochastic

* Continuous

* Hybrid

CPN, HPN => SPN
SPN, HPN => CPN
SPN, CPN => HPN
Simulation

* Stochastic

CPN, HPN => SPN

* Continuous

SPN, HPN => CPN

* Hybrid

SPN, CPN => HPN
Conversion

human readable formats for Petri nets and Coloured Petri nets, respectively, used internally by the PetriNuts framework

(the Systems Biology Markup Language) - an XML-based representation format designed to exchange computational models of biological processes

an XML-based interchange format for Petri nets (no time)

a tool for the evaluation and reduction of chemical reaction networks readded as ordinary differential equations (ODEs)
Conversion

Import

Spike

Export

CANDL
SBML
PNI
ERODE
ANO
ERODE
Reduction

* Clean siphons
* Constant places

* ERODE Further reductions may be applied by converting a model to the ERODE format, if the model is to be read as ordinary differential equations (ODEs).

* Further general reduction rules under consideration
Reduction

- Clean siphons
- Constant places

- ERODE Further reductions may be applied by converting a model to the ERODE format, if the model is to be read as ordinary differential equations (ODEs).

- Further general reduction rules under consideration
Reproducibility

* model configuration
  ** constants
  *(arc weights, initial markings, kinetic constants)*
  ** initial states
  ** observers

* simulation configurations
  ** exports
Reproducibility

configuration: {
model: {
  constants: {
    ...
    }
  places: {
    ...
    }
  observers: {
    ...
    }
  }
  simulation: {
    export: {
      ...
      }
    }
  }
}

* model configuration
  ** constants
    (arc weights, initial markings, kinetic constants)
  ** initial states
  ** observers

* simulation configurations
  ** exports
Reproducibility

* Definition of constants

```plaintext
constants: ∂
  // name of a group
  all: ∂
  /* if constant does not exist
     * then it will be created and
     * can be used in the configuration,
     * for example in defining a place marking
     */
  M: "D/2 + 1"
```

Reproducibility

* Set marking for places

places: {
    // example of use of the newly created constant M
    P: "1000*(M,M)"
    P_2_2: 500
    P_3: 500
}
Reproducibility

* Definition of auxiliary variables (observers) which allow for extra measures by defining numerical functions; depending on the type of observer, it can be defined for places, transitions or simultaneously for places, transitions and constants

observers: \( \xi \)
place: \( \xi \)
\( O_P01: \xi \)
function: "\( P_{1\_1} + P_{2\_3} \)"
Reproducibility

* Defining multiple simulation configurations, which permits to run multiple experiments for one model configuration

```plaintext
simulation:
    name: "Diffusion" // Name of a simulation
    type: stochastic // [continuous, stochastic, hybrid]
    solver: direct // [BDF, classic, direct, delta-leaping static, staticAcc]
    threads: 0 // 0 - try to get the number of concurrent threads supported by the hardware
    interval: 1:1:1000// start : step : end (startTimePoint : intervalSize : endTimePoint)
    runs: 10 // Applicable only for stochastic sim
    // continues conf
    iniStep: 0.1
    linSolver: CVDense // [CVDense, CVSpgmr, CVDiag, CVSbpbcg, and CVSptfqr]
    relTol: 1.0e-5
    absTol: 1.0e-10
    // Export
    export:
        places: [] // Array of places to save (if empty export all)
        transitions: []
        to: "testCase-diffusion-stochastic.csv"
```

// simulation:
```
Reproducibility

* Defining multiple exports of simulation results by use of regular expressions over the nodes; any combination of places, transitions and observers, coloured and uncoloured, in one file

```{}
// Array of places to save,
// including colored places like P
// in this example (if empty, export all)
places: ["P_1_1", "OP01", "Grid.*", "D", "P"]

// Array of transitions to save,
// including colored transitions
transitions: ["t3_1_1_1_2", "t3", "t3"]

// Array of observers to save (if empty, export all)
observers: ["M01", "OT01"]
to: "sim01-file01.csv"
```
# Architecture

* **Modularity**

<table>
<thead>
<tr>
<th>Module</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td>version</td>
<td>display version of Spike</td>
</tr>
<tr>
<td>CLI</td>
<td>help</td>
<td>display help for a given command</td>
</tr>
<tr>
<td>Configuration</td>
<td>exe</td>
<td>execute configuration script</td>
</tr>
<tr>
<td>Converter</td>
<td>load</td>
<td>load a model from a given file</td>
</tr>
<tr>
<td></td>
<td>save</td>
<td>save a model to a given file</td>
</tr>
<tr>
<td></td>
<td>prune</td>
<td>prune a model</td>
</tr>
<tr>
<td></td>
<td>eval</td>
<td>evaluate constants and places</td>
</tr>
<tr>
<td></td>
<td>unfold</td>
<td>unfold a coloured model</td>
</tr>
<tr>
<td>Simulation</td>
<td>sim</td>
<td>run a simulation of the model</td>
</tr>
</tbody>
</table>
Architecture

* Command queue, a typical use case
Architecture

* Command queue, a typical use case

Command queue
* Command queue, a typical use case

```
user cmd: spike exe -f conffile
```
Architecture

* Command queue, a typical use case

CLI → \texttt{exe} → \textbf{Command queue} → \texttt{exe}

user cmd: spike exe -f confFile
Architecture

* Command queue, a typical use case

user cmd: spike exe -f confFile
Architecture

* Command queue, a typical use case

**CLI** `exe` **Command queue**

```plaintext
exe
load
```

**exe** `load` **Configuration**

user cmd: `spike exe -f conffile`
Architecture

* Command queue, a typical use case

CLI \(\rightarrow\) exe

Command queue

\[\begin{array}{c}
\text{exe} \\
\text{load} \\
\text{unfold}
\end{array}\]

exe \(\rightarrow\) Configuration

user cmd: spike exe -f conffile
## Architecture

* Command queue, a typical use case

**CLI** \(\rightarrow\) **Command queue** \(\rightarrow\) **Configuration**

<table>
<thead>
<tr>
<th>Command queue</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>exe</strong></td>
<td><strong>exe</strong></td>
</tr>
<tr>
<td><strong>load</strong></td>
<td><strong>eval</strong></td>
</tr>
<tr>
<td><strong>unfold</strong></td>
<td></td>
</tr>
<tr>
<td><strong>eval</strong></td>
<td></td>
</tr>
</tbody>
</table>

**user cmd:** `spike exe -f conffile`
Architecture

* Command queue, a typical use case

user cmd: spike exe -f conffile
Architecture

* Command queue, a typical use case

CLI  exe  Command queue

<table>
<thead>
<tr>
<th>exe</th>
<th>load</th>
<th>unfold</th>
<th>eval</th>
<th>sim</th>
</tr>
</thead>
</table>

exe  load, unfold, eval, sim  Configuration

load  Converter

user cmd: spike exe -f confFile
Architecture

* Command queue, a typical use case

CLI \(\rightarrow\) exe \(\rightarrow\) Command queue

- exe
- load
- unfold
- eval
- sim

exe \(\rightarrow\) Configuration
- load, unfold, eval, sim

unfold \(\rightarrow\) Converter

user cmd: spike exe -f confFile
* Command queue, a typical use case

```
user cmd: spike exe -f confFile
```
Architecture

* Command queue, a typical use case

CLI \(\rightarrow\) **exe** **\rightarrow** Command queue

<table>
<thead>
<tr>
<th>exe</th>
<th>load</th>
<th>unfold</th>
<th>eval</th>
<th>sim</th>
</tr>
</thead>
</table>

**exe** **\rightarrow** Configuration

**exe** **\rightarrow** Converter

**load, unfold, eval, sim** **\rightarrow** Simulation

user cmd: spike exe -f conffile
use case

Compare how a model behaves under different types of simulation algorithms or under different configurations of a given simulation algorithm.

1. Load model
2. Set model configuration
3. Determine set of simulation configurations
4. For( each simulation configuration ) { 
5. Run simulation
6. Save results of the simulation
7. }

Present status
Present status

Model and simulation configuration

Model

Reduced Model

Decomposition

Reduction

Components with interfaces

Finding interface nodes

Distributed model

Simulation

Traces

Export

Trace Analysis

Reproducible simulation
Present status

Model and simulation configuration

Model

Reduced Model

Decomposition

Reduction

Components

Finding interface tables

Distribution

Components with interfaces

Reproducible simulation

Trace Analysis

Export

Traces

Simulation

Reduced model

Reduction

Decomposition

Reproducible simulation
Present status

Model and simulation configuration

Model

Decomposition

Decomposition

Reduction

Reduced model

Components

Simulation

Simulation with interfaces

Distributed model

Distributed model

Traces

Traces

Export

Export

Trace Analysis

Trace Analysis

Reproducible simulation

Finding interface nodes

Finding interface nodes

Components with interfaces

Components with interfaces
Future works
Future works

Diagram illustrating future works:

- Model and simulation configuration
  - Reduced model
    - Decomposition
      - Reduction
        - Components with interfaces
          - Simulation
            - Distributed model
              - Distribution
                - Finding interface nodes
      - Reproducible simulation
        - Export
          - Traces
            - Trace analysis
- X
- ✓
Future works

Model and simulation configuration

Model

Decomposition

Reduction

Components

Finding interface nodes

Distributed model synchronization

Simulation

Traces

Export

Trace analysis

Reproducible simulation

Components with interfaces

Reduction

Decomposition

Reduction

X

X

X

X
Thank you for your attention

Any questions are welcome

http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Spike