Configuration Dynamics Verification Using UPPAAL

David Fabian   Radek Mařík

Department of Mathematics
Faculty of Nuclear Sciences and Physical Engineering
Czech Technical University in Prague

30/08/2013

Outline

Motivation

- Software applications become more and more complex
- Internal dynamics can be very complicated and hard to maintain
- Imperative style of programming not optimal
- Needs for *(semi)automatic verification* of soundness and completeness of the implementation

Software Configuration

1. Module composition
   - Composition of software modules into an application that fulfills requirements
2. Options settings
   - Deployment and maintenance of a finished application
   - Adjustments to a fixed set of configuration options (keys)
   - There exist general-purpose configuration tools to help with configuration changes such as KConfigXT and Freeconf

Configuration Dynamics

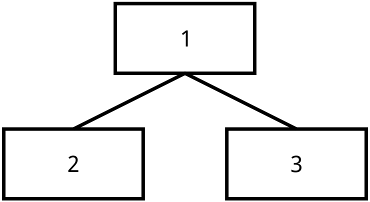
- Keys are usually organized into hierarchical structures
- Each key has some private properties — *internal key state*
- The user can interact with the tool and change values of keys
- Any change can lead to other changes depending on the semantics of configuration options
- Dynamical behavior gets complicated for tools with many internal key states

Verification

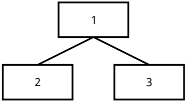
1. Configuration model
2. Declarative description of the dynamics
3. Model-checker

Configuration Hierarchical Model

- Hierarchical model is a rooted acyclic graph
- Every node has a unique ID, its parent ID, and a list of its successor IDs
- Nodes have internal states
- The internal state is a set of Boolean and bounded integer properties

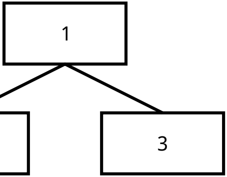


Propagation Rules



- Propagation rules describe dynamical changes of the hierarchical model
- They are of the form $\mathcal{A} \rightarrow \mathcal{B}$, $\mathcal{A}$ is the head, $\mathcal{B}$ is the body
- Head is always bound to a specific node
- Body is a non-empty set of variables assignments
- If the head is satisfied, the rule fires and the body is executed
- $++$ and $--$ syntactic sugar is present to raise or lower the value of a variable by one

Example Model



$$M = \{ \ \big(1, \emptyset, \{2,3\}, \big(bool_1^1, bool_2^1, int_1^1, \{0,1,2\}\big)\big), \\ \big(2, 1, \emptyset, \big(bool_1^2, bool_2^2, int_1^2, \{0,1,2\}\big)\big), \\ \big(3, 1, \emptyset, \big(bool_1^3, bool_2^3, int_1^3, \{0,1,2\}\big)\big) \ \} .$$

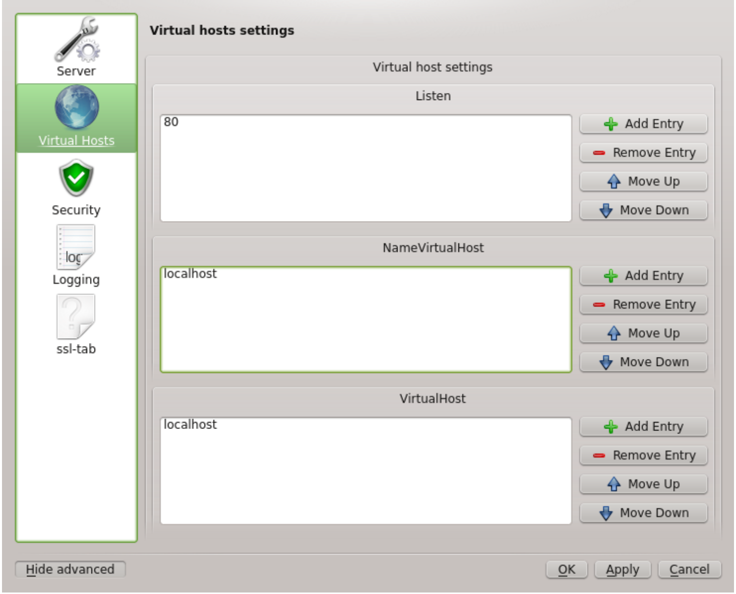
- Whenever $bool_1$ is *false* for node two, $bool_2$ must also be *false* for that particular node
- Whenever $bool_2$ is *true* and int_1 is greater than one in node three, the value of the parent's int_1 must be two

$$\neg bool_1^2 \ \rightarrow \ bool_2^2 = false$$
$$bool_2^3 \wedge int_1^3 > 1 \ \rightarrow \ int_1^1 = 2$$

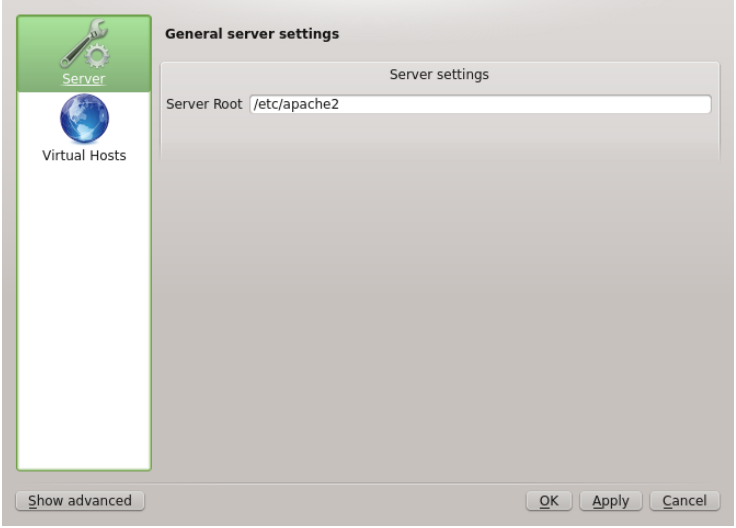
Freeconf

- Multi-platform configuration utility developed at FNSPE
- Organizes keys into configuration sections
- Support for hundreds or thousands of keys
- GUI must be clear and simple, optional keys should be hidden
- Every key has a set of properties that describes its importance (mandatory, active, inconsistent, etc.)

Freeconf GUI (full detail)



Freeconf GUI (simplified)



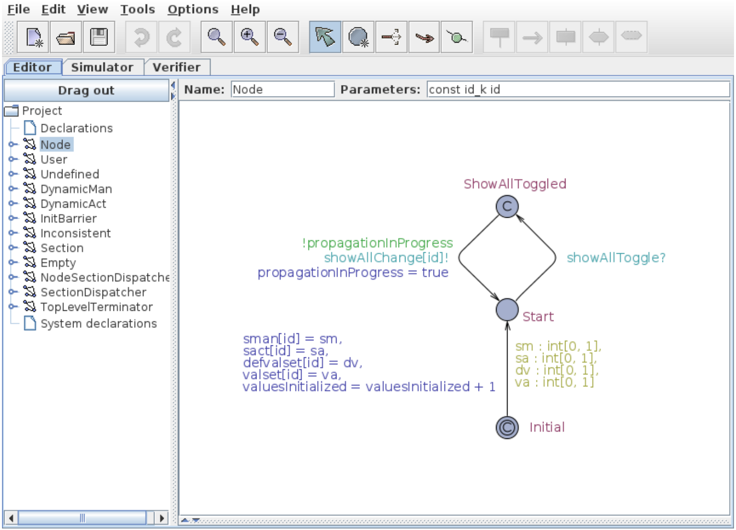
Freeconf Configuration Model

- Straightforward to encode Freeconf model as a hierarchical configuration model
- Two types of nodes — *configuration keys* and *configuration sections*
- Key internal state formed by eight Boolean variables
- Section internal state formed by three Boolean and four integer counters
- Propagation rules expressive enough to describe Freeconf's dynamics

UPPAAL Model-Checker

- Joint project of Upsalla and Aalborg University
- Model-checker utility of real-time dynamic systems with Java GUI
- Visual modeling and C-like programming
- Support for integer and Boolean variables, arrays, and automata templates
- Automata can be synchronized by channels
- Operates on a subset of Timed Computational Tree Logic (TCTL)
- Custom query language

UPPAAL Graphical User Interface



Queries

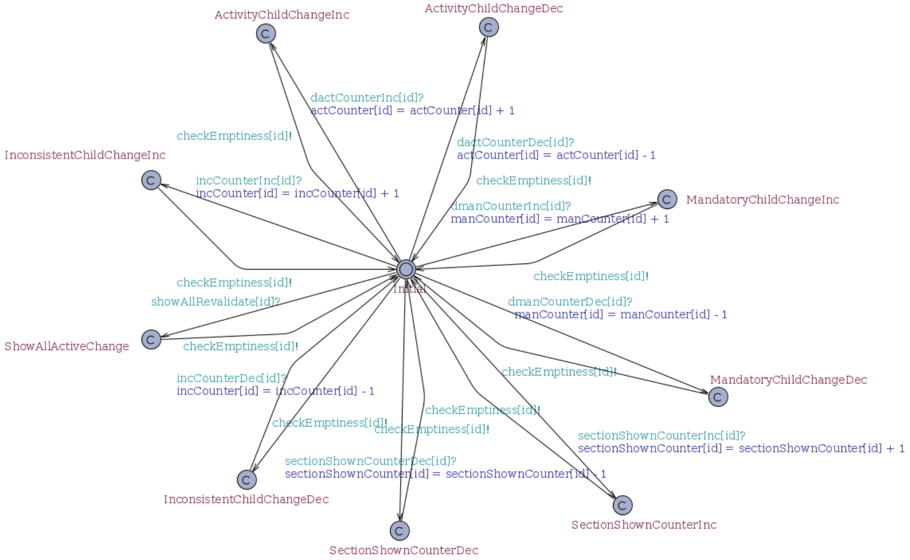
$$\mathrm{E} <> \mathit{forall}(i : \mathit{id\_s})\mathit{manCounter}[i] < 0$$

- Query usually starts with a quantifier and a path modality
- Array indexing is supported
- UPPAAL can be set to produce counter-examples
- Nested modalities are not supported

Freeconf model in UPPAAL

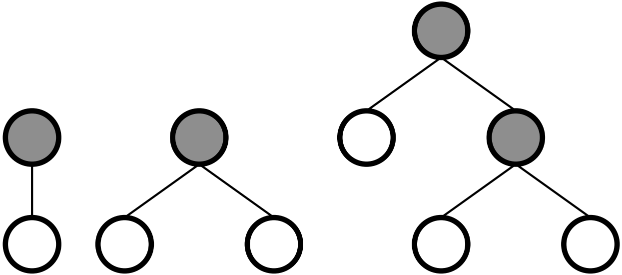
- Key properties modeled as global arrays
- Hierarchy nodes modeled as automata templates `Node` and `Section`
- Hierarchy structure encoded as 2D arrays
- Properties propagation modeled using channel synchronization and global variables
- Rule heads and bodies *hard-wired* as automata
- Auxiliary data structures needed to enforce causality

Section Automaton



Tested Instances

- Freeconf model can be arbitrarily large
- Only a small subset of models tested
- Deficiencies in Freeconf revealed by the verification



(a) (b) (c)

Results

Model	Time (s)	Memory (KiB)	# of states
a	0.07	6889	16384
b	1.67	24572	21233664
c	189.1	2147932	17592186044416

- Configuration hierarchical model defined
- Freeconf configuration dynamics encoded into the hierarchical model
- Freeconf model encoded into UPPAAL
- Several Freeconf instances verified by UPPAAL on Intel Core 2 Quad Q9550 CPU at 2.83 GHz, 4 GiB RAM, running 64 bit Linux 3.1.10

Conclusion

- UPPAAL easy to use but too general
- Substantial amount of auxiliary code necessary, re-verification problematic
- Custom domain-specific model-checker needed in the future
- Attempts to design the model-checker in Constraint Handling Rules (CHR)
- All propagation rules should be held at one place
- Re-verification should be easy

Thank you for your attention!