Softwaretechnik / Software-Engineering
Lecture 14: Behavioural Software Modelling

2018-06-28

Prof. Dr. Andreas Podelski,
Dr. Bernd Westphal
Albert-Ludwigs-Universität Freiburg, Germany

Topic Area Architecture & Design: Content

- Introduction and Vocabulary
- Software Modelling
  - model; views / viewpoints; 4+1 view
- Modelling structure
  - (simplified) class & object diagrams
  - (simplified) object constraint logic (OCL)
- Principles of Design
  - modularity, separation of concerns
  - information hiding and data encapsulation
  - abstract data types, object orientation
- Design Patterns
- Modelling behaviour
  - communicating finite automata (CFA)
  - Uppaal query language
- CFA vs. Software
  - Unified Modelling Language (UML)
    - basic state-machines
    - an outlook on hierarchical state-machines
- Model-driven/-based Software Engineering

Communicating Finite Automata (CFA)
- concrete and abstract syntax,
- networks of CFA,
- operational semantics.

Transition Sequences
- Deadlock, Reachability
- Uppaal
  - tool demo (simulator),
  - query language,
  - CFA model-checking.

CFA at Work
- drive to configuration, scenarios, invariants
  - tool demo (verifier).

Uppaal Architecture

Example

half_idle

request_sent

tea_selected

soft_selected

water_selected

idle

DOK?

OK!

water_enabled := false, soft_enabled := false, tea_enabled := false

DTEA!

DWATER!

DSOFT!

tea_enabled

TEA?

soft_enabled

SOFT?

water_enabled

WATER?
That is, modifications are executed sequentially from left to right. We set
\[ v^n = \begin{cases} a & \text{if } \phi_i \text{ is an action}, \\ \ell & \text{if } \phi_i \text{ is a location}, \\ \end{cases} \]

of the variables, \( \ell \) is called the initial location of \( \phi_i \).

For each channel \( \text{Chan} \), be communicating finite automata.

\[ \text{Effect of modification } D \subseteq \text{Act} \]

\[ \text{In the example, } R \in \text{DSOFT}! \]
A configuration is reachable, i.e. $\langle \vec{\ell}, \nu \rangle$ with $\vec{\ell} := \vec{\ell}$ such that $\nu$ and $\vec{\ell}, \nu$ is reachable, i.e. $\langle \vec{\ell}, \nu \rangle$ is called $A$ (simplified) configuration of network $C$.

Reachability

An $A$ (simplified) configuration $C$ of network $C$ is a deadlock if and only if there is a reachable configuration $\langle \vec{\ell}, \nu \rangle$, i.e. if and only if there is a transition sequence of the form $\langle \vec{\ell}, \nu \rangle \leadsto \langle \vec{\ell}', \nu' \rangle$, where $\vec{\ell}'$ is the result of applying first $\nu$ on $\vec{\ell}$ and then $\vec{r}$ and then $\nu$ on $\vec{r}$, and $\vec{\ell}'$ and $\nu'$ are valuation.

Example

A transition sequence $\langle \vec{\ell}, \nu \rangle \leadsto \langle \vec{\ell}', \nu' \rangle$ is a synchronisation transition if $\nu' = \nu \land \vec{\ell}' = \vec{\ell}$, i.e. if the source valuation satisfies guard (!).
\[ F \] for model-checking problem

\[ \text{We say } = | \text{, if and only if } = | N |, \text{ denoted by } F \]

\[ N \text{ of } A \]

\[ \text{Always globally } \]

\[ \text{Exists globally } \]

\[ \text{Satisfaction of Uppaal Queries by Configurations} \]
Model Architecture — Who Talks to Whom

CoinValidator
User
ChoicePanel
WaterDispenser
SoftDispenser
TeaDispenser
Service
C50, E1
WATER, SOFT, TEA
OK
DWATER
DSOFT
DTEA
DOK
FILLUP

ENVIRONMENT SYSTEM

• Shared variables:

  • bool water_enabled, soft_enabled, tea_enabled;

  • int w = 3, s = 3, t = 3;

Note: Our model does not use scopes (“information hiding”) for channels.

Design Sanity Check: Drive to Configuration

• Question: Is it (at all) possible to have no water in the vending machine model? (Otherwise, the design is definitely broken.)

• Approach: Check whether a configuration satisfying $w = 0$ is reachable, i.e. check $\exists \diamond w = 0$ for the vending machine model $\mathcal{N}_{VM}$.

Design Check: Scenarios

• Question: Is the following existential LSC satisfied by the model? (Otherwise, the design is definitely broken.)

LSC: buy tea

  AM: initial
  I: permissive

• Approach: Use the following newly created CFA 'Scenario ' end_of_scenario instead of User and check whether location end_of_scenario is reachable, i.e. check $\exists \diamond \text{Scenario\.end_of_scenario}$ for the modified vending machine model $\mathcal{N}'_{VM}$.

Design Verification: Invariants

• Question: Is it the case that the "tea" button is only enabled if there is e1.50 in the machine? (Otherwise, the design is broken.)

• Approach: Check whether the implication tea_enabled $\Rightarrow$ Coin Validator. have_c150 holds in all reachable configurations, i.e. check $\forall \Box \text{tea_enabled imply Coin Validator. have_c150}$ for the vending machine model $\mathcal{N}_{VM}$.

Design Verification: Sanity Check

• Question: Is the "tea" button ever enabled? (Otherwise, the considered invariant tea_enabled $\Rightarrow$ Coin Validator. have_c150 holds vacuously.)

• Approach: Check whether a configuration satisfying tea_enabled = 1 is reachable. Exactly like we did with $w = 0$ earlier.

Design Verification: Another Invariant

• Question: Is it the case that, if there is money in the machine and water in stock, that the "water" button is enabled?

• Approach: Check $\forall \Box (\text{Coin Validator. have_c50 or Coin Validator. have_c100 or Coin Validator. have_c150)}$ imply water_enabled for the vending machine model $\mathcal{N}_{VM}$.
Recall: Universal LSC Example

LSC: buy water
AC: true
AM: invariant
I: strict

User
CoinValidator
ChoicePanel
Dispenser

C
50
WATER

¬ (C50 ! ∨ E1 ! ∨ pSOFT ! ∨ pTEA ! ∨ pFILLUP !)

water
in
stock

dWATER
OK

¬ (dSoft ! ∨ dTEA !)

Uppaal

• tool demo (simulator),
• query language,
• CFA model-checking.

CFA at Work
• drive to configuration, scenarios, invariants
• tool demo (verifier).

Uppaal Architecture

server
verifyta

Tell Them What You've Told Them...

• A network of communicating finite automata
• describes a labelled transition system,
• can be used to model software behaviour.

• The Uppaal Query Language
• can be used to formalize reachability (∃♦ CF, ∀□ CF, . . . ) and
• leadsto (CF1 −→ CF2) properties.

• Since the model-checking problem of CFA is decidable,
• there are tools which automatically check whether a network of CFA satisfies a given query.

• Use model-checking, e.g., to
• obtain a computation path to a certain configuration (drive-to-configuration),
• check whether a scenario is possible,
• check whether an invariant is satified.

• If not, analyse the design further using the obtained counter-example.

References

