Softwaretechnik / Software-Engineering

Lecture 14: Behavioural Software Modelling

2019-07-01

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)

- **Modelling behaviour**
  - Communicating Finite Automata (CFA)
  - Uppaal query language

- **Unified Modelling Language (UML)**
  - basic state-machines
  - an outlook on hierarchical state-machines

- **Principles of Design**
  - modularity, separation of concerns
  - information hiding and data encapsulation
  - abstract data types, object orientation

- **Design Patterns**
- **Model-driven/-based Software Engineering**
Content

- 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
\[ \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \xrightarrow{\alpha_3} \sigma_3 \]

Analyst

Software Modelling

Main

Input

Game Logic

Output

• Keyboard
• Joystick
• ...

• player scores
• interface inputs/engine

• Graphics (from ASCII to bitmap; native or via API)
• Sound
• ...

Engine

• keyboard
• joystick
• ...

• world
• ...

• notify
• update

User Game

• n
• w
• e
• s
• resigned

\[ LSC: \text{name} \]
\[ AC: \text{true} \]
\[ AM: \text{invariant} \]
\[ I: \text{permissive} \]
Communicating Finite Automata

presentation follows (Olderog and Dierks, 2008)
**ChoicePanel:**
(simplified)

- **Input Action:**
  - WATER?
  - SOFT?
  - TEA?

- **Location:**
  - idle
  - soft_selected
  - tea_selected
  - request_sent

- **Initial Location:**
  - idle

- **Guards:**
  - water_enabled := false
  - soft_enabled := false
  - tea_enabled := false

- **Update:**
  - output action
  - update vector

- **Internal Action:**
  - OK!

- **Guard:**
  - true

- **Update vector:**
  - OK!

- **Output Action:**
  - DOK?
  - OK!
To define communicating finite automata, we need the following sets of symbols:

- A set \((a, b \in \text{Chan})\) of **channel names** or **channels**.
- For each channel \(a \in \text{Chan}\), two **visible actions**: \(a?\) and \(a!\) denote **input** and **output** on the channel \((a?, a! \notin \text{Chan})\).
- \(\tau \notin \text{Chan}\) represents an **internal action**, not visible from outside.
- \((\alpha, \beta \in \text{Act})\) \(\text{Act} := \{a? \mid a \in \text{Chan}\} \cup \{a! \mid a \in \text{Chan}\} \cup \{\tau\}\) is the set of **actions**.

- An **alphabet** \(B\) is a set of **channels**, i.e. \(B \subseteq \text{Chan}\).
- For each alphabet \(B\), we define the corresponding **action set**

\[
B?! := \{a? \mid a \in B\} \cup \{a! \mid a \in B\} \cup \{\tau\}.
\]

**Note:** \(\text{Chan}?! = \text{Act}\).
Let \((v, w \in V)\) be a set of ((\textit{finite domain}) integer) variables.

By \((\varphi \in \Psi(V))\) we denote the set of \textit{integer expressions} over \(V\) using function symbols \(+, −, \ldots\) and relation symbols \(<, \leq, \ldots\).

A \textbf{modification} on \(v \in V\) is of the form

\[
v := \varphi, \quad v \in V, \quad \varphi \in \Psi(V).
\]

By \(R(V)\) we denote the set of all modifications.

By \(\bar{r}\) we denote a finite list \(\langle r_1, \ldots, r_n \rangle, n \in \mathbb{N}_0\), of modifications \(r_i \in R(V)\). \(\bar{r}\) is called \textbf{reset vector} (or \textbf{update vector}).

\(\langle \rangle\) is the empty list \((n = 0)\).

By \(R(V)^*\) we denote the set of all such finite lists of modifications.
Definition. A **communicating finite automaton** is a structure

\[
\mathcal{A} = (L, B, V, E, \ell_{ini})
\]

where

- \( (\ell \in) L \) is a finite set of **locations** (or **control states**),
- \( B \subseteq \text{Chan} \),
- \( V \): a set of data variables,
- \( E \subseteq L \times B \times \Phi(V) \times R(V)^* \times L \): a finite set of **directed edges** such that \((\ell, \alpha, \varphi, \vec{r}, \ell') \in E \land \text{chan}(\alpha) \in U \implies \varphi = \text{true}\).

Edges \((\ell, \alpha, \varphi, \vec{r}, \ell')\) from location \( \ell \) to \( \ell' \) are labelled with an **action** \( \alpha \), a **guard** \( \varphi \), and a list \( \vec{r} \) of **modifications**.

- \( \ell_{ini} \in L \) is the **initial location**.
Example

Abstract syntax: \[ A = (L, B, V, E, \ell_{ini}) \]

\[ A_1 : \]

\[ L = \{ l_0, l_1, l_2 \} \]
\[ B = \{ \text{A} \} \]
\[ V = \{ x \} \]
\[ \ell_{ini} = l_0 \]
\[ E = \{ (l_0, \tau, x := 0, x := 27, l_1), (l_1, A!, \text{true}, \langle 7, l_2 \rangle), \ldots \} \]
Definition.
Let \( A_i = (L_i, B_i, V_i, E_i, \ell_{ini,i}), 1 \leq i \leq n \), be communicating finite automata.

The **operational semantics** of the **network** of CFA \( C(A_1, \ldots, A_n) \)

is the labelled transition system

\[
T(C(A_1, \ldots, A_n)) = (Conf, Chan \cup \{\tau\}, \{\lambda \mapsto \lambda \mid \lambda \in Chan \cup \{\tau\}\}, C_{ini})
\]

where

- \( V = \bigcup_{i=1}^{n} V_i \),
- \( Conf = \{ \langle \ell, \nu \rangle \mid \ell \in L_i, \nu : V \to \mathcal{D}(V) \} \),
- \( C_{ini} = \langle \ell_{ini}, \nu_{ini} \rangle \) with \( \nu_{ini}(v) = 0 \) for all \( v \in V \).

The transition relation consists of transitions of the following two types.
• \( \nu : V \to \mathcal{D}(V) \) is a **valuation** of the variables,

• A valuation \( \nu \) of the variables canonically assigns an integer value \( \nu(\varphi) \) to each integer expression \( \varphi \in \Phi(V) \).

• \( \models \subseteq (V \to \mathcal{D}(V)) \times \Phi(V) \) is the canonical **satisfaction relation** between valuations and integer expressions from \( \Phi(V) \).

**Effect of modification** \( r \in R(V) \) on \( \nu \), denoted by \( \nu[r] \):

\[
\nu[v := \varphi](a) := \begin{cases} 
\nu(\varphi), & \text{if } a = v, \\
\nu(a), & \text{otherwise}
\end{cases}
\]

• We set \( \nu[\langle r_1, \ldots, r_n \rangle] := \nu[r_1] \ldots [r_n] = ((\nu[r_1])[r_2]) \ldots [r_n] \).

That is, modifications are executed sequentially from left to right.
An internal transition \( \langle \vec{l}, \nu \rangle \xrightarrow{\tau} \langle \vec{l}', \nu' \rangle \) occurs if there is \( i \in \{1, \ldots, n\} \) and there is a \( \tau \)-edge \((\ell_i, \tau, \varphi, \vec{r}, \ell'_i) \in E_i \) such that

- \( \nu \models \varphi \),

  “source valuation satisfies guard”

- \( \vec{l}' = \vec{l}[\ell_i := \ell'_i] \),

  “automaton \( i \) changes location”

- \( \nu' = \nu[\vec{r}] \),

  “\( \nu' \) is the result of applying \( \vec{r} \) on \( \nu \)”
An internal transition $\langle \vec{l}, \nu \rangle \xrightarrow{\tau} \langle \vec{l}', \nu' \rangle$ occurs if there is $i \in \{1, \ldots, n\}$ and
- there is a $\tau$-edge $(\ell_i, \tau, \varphi_i, \vec{r}_i, \ell'_i) \in E_i$ such that
  - $\nu \models \varphi_i$, “source valuation satisfies guard”
  - $\vec{l}' = \vec{l}[\ell_i := \ell'_i]$, “automaton $i$ changes location”
  - $\nu' = \nu[\vec{r}_i][\vec{r}_j]$, “$\nu'$ is the result of applying $\vec{r}_i$ on $\nu$”

A synchronisation transition $\langle \vec{l}, \nu \rangle \xrightarrow{\text{b}} \langle \vec{l}', \nu' \rangle$ occurs if there are $i, j \in \{1, \ldots, n\}$ with $i \neq j$ and
- there are edges $(\ell_i, \text{b}!, \varphi_i, \vec{r}_i, \ell'_i) \in E_i$ and $(\ell_j, \text{b}?, \varphi_j, \vec{r}_j, \ell'_j) \in E_j$ such that
  - $\nu \models \varphi_i \land \varphi_j$, “source valuation satisfies guards (!)”
  - $\vec{l}' = \vec{l}[\ell_i := \ell'_i][\ell_j := \ell'_j]$, “automaton $i$ and $j$ change location”
  - $\nu' = \nu[\vec{r}_i][\vec{r}_j]$, “$\nu'$ is the result of applying first $\vec{r}_i$ and then $\vec{r}_j$ on $\nu$”

This style of communication is known under the names “rendezvous”, “synchronous”, “blocking” communication (and possibly many others).
Example

\[
\begin{align*}
&\langle (l_0, m_0, m_0), x = 0 \rangle \\
&\quad\downarrow \tau \\
&\langle (l_1, m_0, m_0), x = 27 \rangle \\
&\quad\downarrow A \\
&\langle (l_2, m_1, m_0), x = 27 \rangle
\end{align*}
\]
Transition Sequences

- A transition sequence of $C(A_1, \ldots, A_n)$ is any (in)finite sequence of the form

  \[
  \langle \vec{\ell}_0, \nu_0 \rangle \xrightarrow{\lambda_1} \langle \vec{\ell}_1, \nu_1 \rangle \xrightarrow{\lambda_2} \langle \vec{\ell}_2, \nu_2 \rangle \xrightarrow{\lambda_3} \ldots
  \]

  with

  - $\langle \vec{\ell}_0, \nu_0 \rangle = C_{ini}$,
  - for all $i \in \mathbb{N}$, there is $\xrightarrow{\lambda_{i+1}}$ in $T(C(A_1, \ldots, A_n))$ with $\langle \vec{\ell}_i, \nu_i \rangle \xrightarrow{\lambda_{i+1}} \langle \vec{\ell}_{i+1}, \nu_{i+1} \rangle$. 

Reachability

- A configuration \(\langle \vec{l}, \nu \rangle\) is called \textit{reachable} (in \(C(A_1, \ldots, A_n)\)) \textit{from} \(\langle \vec{l}_0, \nu_0 \rangle\) if and only if there is a transition sequence of the form

\[
\langle \vec{l}_0, \nu_0 \rangle \xrightarrow{\lambda_1} \langle \vec{l}_1, \nu_1 \rangle \xrightarrow{\lambda_2} \langle \vec{l}_2, \nu_2 \rangle \xrightarrow{\lambda_3} \ldots \xrightarrow{\lambda_n} \langle \vec{l}_n, \nu_n \rangle = \langle \vec{l}, \nu \rangle.
\]

- A configuration \(\langle \vec{l}, \nu \rangle\) is called \textit{reachable} (without “from”!) if and only if it is reachable from \(C_{\text{ini}}\).

- A location \(\ell \in L_i\) is called \textit{reachable} if and only if \textit{any} configuration \(\langle \vec{l}, \nu \rangle\) with \(\ell_i = \ell\) is reachable, i.e. there exist \(\vec{l}\) and \(\nu\) such that \(\ell_i = \ell\) and \(\langle \vec{l}, \nu \rangle\) is reachable.
Deadlock

- A configuration $\langle \ell, \nu \rangle$ of $C(A_1, \ldots, A_n)$ is called **deadlock** if and only if there are no transitions from $\langle \ell, \nu \rangle$, i.e. if

$$\neg (\exists \lambda \in \Lambda \exists \langle \ell', \nu' \rangle \in Conf \bullet (\ell, \nu) \xrightarrow{\lambda} (\ell', \nu')).$$

The **network** $C(A_1, \ldots, A_n)$ is said to **have a deadlock** if and only if there is a reachable configuration $\langle \ell, \nu \rangle$ which is a deadlock.
Uppaal

(Larsen et al., 1997; Behrmann et al., 2004)
Consider $\mathcal{N} = C(A_1, \ldots, A_n)$ over data variables $V$.

- **basic formula:**

  $$\text{atom ::= } A_i.\ell \mid \varphi \mid \text{deadlock}$$

  where $\ell \in L_i$ is a location and $\varphi$ an expression over $V$.

- **configuration formulae:**

  $$\text{term ::= atom \mid \text{not term} \mid term_1 \text{ and } term_2}$$

- **existential path formulae:**

  $$\text{e-formula ::= } \exists \diamond term \quad \text{(exists finally)}$$

  $$\quad \exists \square term \quad \text{(exists globally)}$$

- **universal path formulae:**

  $$\text{a-formula ::= } \forall \diamond term \quad \text{(always finally)}$$

  $$\quad \forall \square term \quad \text{(always globally)}$$

  $$\quad \sim term_1 \rightarrow term_2 \quad \text{(leads to)}$$

- **formulae (or queries):**

  $$F ::= e\text{-formula} \mid a\text{-formula}$$
The satisfaction relation

\[ \langle \vec{\ell}, \nu \rangle \models F \]

between configurations

\[ \langle \vec{\ell}, \nu \rangle = \langle (\ell_1, \ldots, \ell_n), \nu \rangle \]

of a network \( C(\mathcal{A}_1, \ldots, \mathcal{A}_n) \) and formulae \( F \) of the Uppaal logic is defined inductively as follows:

- \( \langle \vec{\ell}, \nu \rangle \models \text{deadlock} \)
  - iff \( \ell_0, i \) is a deadlock configuration

- \( \langle \vec{\ell}, \nu \rangle \models \mathcal{A}_i \cdot \ell \)
  - iff \( \ell_0, i = \ell \)

- \( \langle \vec{\ell}, \nu \rangle \models \varphi \)
  - iff \( \nu \models \varphi \)

- \( \langle \vec{\ell}, \nu \rangle \models \text{not term} \)
  - iff \( \langle \vec{\ell}, \nu \rangle \not\models \text{term}_1 \) and \( \langle \vec{\ell}, \nu \rangle \not\models \text{term}_2 \)

- \( \langle \vec{\ell}, \nu \rangle \models \text{term}_1 \text{ and term}_2 \)
  - iff \( \nu \models \text{term}_1 \) and \( \nu \models \text{term}_2 \)
Example: Computation Paths vs. Computation Tree

\[ \langle (l0, m0, m0), \quad x = 0 \rangle \]
\[ \tau \]
\[ A \langle (l1, m0, m0), \quad x = 27 \rangle \]
\[ A \]
\[ \langle (l2, m1, m0), \quad x = 27 \rangle \]
\[ A \]
\[ \langle (l1, m1, m1), \quad x = 27 \rangle \]
\[ \langle (l2, m0, m1), \quad x = 27 \rangle \]
\[ A \]
\[ \langle (l1, m1, m1), \quad x = 27 \rangle \]
Example: Computation Paths vs. Computation Graph
(or: Transition Graph)

\[ \langle (l0, m0, m0), \ x = 0 \ \rangle \]
\[ \tau \] 
\[ \langle (l1, m0, m0), \ x = 27 \ \rangle \]
\[ A \]
\[ \langle (l2, m0, m1), \ x = 27 \ \rangle \]
\[ A \]
\[ \langle (l2, m1, m0), \ x = 27 \ \rangle \]
\[ A \]
\[ \langle (l1, m1, m1), \ x = 27 \ \rangle \]
\[ A \]
Satisfaction of Uppaal Queries by Configurations

Exists finally:

• \( \langle \ell_0, \nu_0 \rangle \models \exists \Diamond \text{term} \) iff \( \exists \text{path } \xi \text{ of } \mathcal{N} \text{ starting in } \langle \ell_0, \nu_0 \rangle \) \( \exists i \in \mathbb{N}_0 \bullet x_i \models \text{term} \)

"some configuration satisfying term is reachable"

Example: \( \langle \ell_0, \nu_0 \rangle \models \exists \Diamond \varphi \)
Exists globally:

- \( \langle \vec{l}_0, \nu_0 \rangle \models \exists \square \text{term} \)
  \[ \text{iff } \exists \text{ path } \xi \text{ of } \mathcal{N} \text{ starting in } \langle \vec{l}_0, \nu_0 \rangle \]
  \[ \forall i \in \mathbb{N}_0 \cdot \xi_i \models \text{term} \]

“on some computation path, all configurations satisfy \text{term}”

Example: \( \langle \vec{l}_0, \nu_0 \rangle \models \exists \square \phi \)
Satisfaction of Uppaal Queries by Configurations

- **Always globally:**

  \[
  \langle \ell_0, \nu_0 \rangle \models \forall \square \text{term} \iff \langle \ell_0, \nu_0 \rangle \not\models \exists \diamond \neg \text{term}
  \]

  “not (some configuration satisfying \(\neg \text{term}\) is reachable)”
  or: “all reachable configurations satisfy \(\text{term}\)”

- **Always finally:**

  \[
  \langle \ell_0, \nu_0 \rangle \models \forall \Diamond \text{term} \iff \langle \ell_0, \nu_0 \rangle \not\models \exists \Box \neg \text{term}
  \]

  “not (on some computation path, all configurations satisfy \(\neg \text{term}\))”
  or: “on all computation paths, there is a configuration satisfying \(\text{term}\)”
Leads to:

- \( \langle \vec{l}, \nu_0 \rangle \models term_1 \rightarrow term_2 \) iff \( \forall \) path \( \xi \) of \( \mathcal{N} \) starting in \( \langle \vec{l}, \nu_0 \rangle \) \( \forall \) \( i \) \( \in \mathbb{N}_0 \) •
  \( \xi_i \models term_1 \implies \xi_i \models \forall \Diamond term_2 \)

“on all paths, from each configuration satisfying \( term_1 \),
a configuration satisfying \( term_2 \) is reachable” (response pattern)

Example: \( \langle \vec{l}, \nu_0 \rangle \models \varphi_1 \rightarrow \varphi_2 \)
Definition. Let $\mathcal{N} = C(A_1, \ldots, A_n)$ be a network and $F$ a query.

(i) We say $\mathcal{N}$ satisfies $F$, denoted by $\mathcal{N} \models F$, if and only if $C_{ini} \models F$.

(ii) The model-checking problem for $\mathcal{N}$ and $F$ is to decide whether $(\mathcal{N}, F) \in \models$.

Proposition. The model-checking problem for communicating finite automata is decidable.
Content

- 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
CFA and Queries at Work
**Model Architecture — Who Talks What to Whom**

- **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. That is, ‘Service’ could send ‘WATER’ if the modeler wanted to.
**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 whether

\[ \mathcal{N}_{VM} \models \exists \diamondsuit w = 0. \]

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.)

- **Approach**: Use the following newly created CFA ‘Scenario’ instead of User and check whether location end_of_scenario is reachable, i.e. check whether

\[ \mathcal{N}_{VM}' \models \exists \Diamond \text{Scenario}.end\_of\_scenario. \]

for the modified vending machine model \( \mathcal{N}_{VM}' \).
• **Question**: Is it the case that the “tea” button is **only** enabled if there is €1.50 in the machine? (Otherwise, the design is broken.)

• **Approach**: Check whether the implication

\[ \text{tea\_enabled} \implies \text{CoinValidator\_have\_c150} \]

holds in all reachable configurations, i.e. check whether

\[ \mathcal{N}_{VM} \models \forall \Diamond (\text{tea\_enabled} \implies \text{CoinValidator\_have\_c150}) \]

for the vending machine model \( \mathcal{N}_{VM} \).
• **Question**: Is the “tea” button ever enabled? (Otherwise, the considered invariant

\[
\text{tea\_enabled} \implies \text{CoinValidator\_have\_c150}
\]

holds vacuously.)

• **Approach**: Check whether a configuration satisfying \(\text{water\_enabled} = 1\) is reachable.

Exactly like we did with \(w = 0\) earlier (i.e. check whether \(\mathcal{N}_{VM} \models \exists! \text{water\_enabled} = 1\)).
**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

\[ \mathcal{N}_{VM} \models \forall \square (\text{CoinValidator} \cdot \text{have\_c50} \lor \text{CoinValidator} \cdot \text{have\_c100} \lor \text{CoinValidator} \cdot \text{have\_c150}) \quad \text{imply} \quad \text{water\_enabled}. \]
Recall: Universal LSC Example

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

User  CoinValidator  ChoicePanel  Dispenser

- C50
- pWATER
- water_in_stock
- dWATER
- OK

\neg (C50 \lor E1 \lor pSOFT \lor pTEA \lor pFILLUP)

\neg (dSoft \lor dTEA)
What Can We Conclude From Verification Results?

- Assume that query $Q$ corresponds to a requirement on the system under development, and $\mathcal{N}$ is our design-idea model.
- Assume that the verification tool states $\mathcal{N} \models Q$. What can we conclude from that?

<table>
<thead>
<tr>
<th>the design idea</th>
<th>tool result</th>
<th>$\mathcal{N} \not\models Q$</th>
<th>$\mathcal{N} \models Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>sat. $Q$</td>
<td>false negative</td>
<td>true positive</td>
<td></td>
</tr>
<tr>
<td>does not sat. $Q$</td>
<td>true negative</td>
<td>false positive</td>
<td></td>
</tr>
</tbody>
</table>
Content

- **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**
Uppaal Architecture
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 ($\exists \diamond CF, \forall \Box CF, \ldots$) and
  - leadsto ($CF_1 \rightarrow CF_2$) 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 satisfied.

  (If not, analyse the design further using the obtained counter-example).
References
References


