• **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

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

• **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**
Software Modelling

Analyst

(`Σ × A)^ω`
Communicating Finite Automata

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

- idle
- request_sent
- tea_selected
- soft_selected
- water_selected

- input action
- channel
- edges
- location
- update vector

- guard
- initial location

- updates

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

- OK!
- DOK?
Channel Names and Actions

To define communicating finite automata, we need the following sets of symbols:

- A set \( (a, b \in) \) 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.

- \( (\alpha, \beta \in) \) Act := \( \{a? | a \in \text{Chan}\} \cup \{a! | a \in \text{Chan}\} \cup \{\tau\} \) 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**
  \[ B?! := \{a? | a \in B\} \cup \{a! | a \in B\} \cup \{\tau\}. \]

  **Note:** \( \text{Chan}?! = \text{Act} \).
• Let \((v, w \in V)\) be a set of (finite domain) integer variables. Including 0. \(v+w\)

By \((\varphi \in \Psi(V))\) we denote the set of integer expressions over \(V\) using function symbols +, −, . . . and relation symbols <, ≤, . . . . \(v<w\)

• A modification on \(v \in V\) is of the form

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

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

• By \(\vec{r}\) we denote a finite list \(\langle r_1, \ldots, r_n \rangle\), \(n \in \mathbb{N}_0\), of modifications \(r_i \in R(V)\). \(\vec{r}\) is called reset vector (or 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,
- \(E \subseteq L \times B_1 \times \Phi(V) \times R(V)^* \times L\): a finite set of **directed edges** such that
  \((\ell, \alpha, \varphi, \vec{r}, \ell') \in E \wedge \text{chan}(\alpha) \in U \implies \varphi = \text{true}\).

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

\( L = \{ \text{idle, water\_selected, ...} \} \)
\( B = \{ \text{WATER, OK, ...} \} \)
\( V = \{ \text{water\_enabled, ...} \} \)

ChoicePanel: (simplified)

\[
E = \{ (\text{idle, WATER?}, \text{water\_enabled}, \langle\rangle, \text{water\_selected}) , \\
(\text{request\_sent, n, true,}\langle\rangle, \text{half\_idle}), ... \} \\
L = B \cup V \cup R(V)^* \cup L \\
\text{l}\text{ini} = \text{idle}
\]
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

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

where

- \( V = \bigcup_{i=1}^{n} V_i \), valuation of \( V \)
- \( Conf = \{\langle \ell, \nu \rangle | \ell \in L_i, \nu: V \rightarrow \mathcal{D}(V)\} \), valuation of \( V \)
- \( C_{ini} = \langle \vec{\ell}_{ini}, \nu_{ini} \rangle \) with \( \nu_{ini}(v) = 0 \) for all \( v \in V \).

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

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

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

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

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

• 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.
Operational Semantics of Networks of CFA

\( \langle \vec{e}, \nu \rangle, \langle \vec{e}', \nu' \rangle \in \mathcal{E} \xrightarrow{\tau} \langle \vec{e}', \nu' \rangle \)

- An **internal transition** \( \langle \vec{e}, \nu \rangle \xrightarrow{\tau} \langle \vec{e}', \nu' \rangle \) occurs if there is \( i \in \{1, \ldots, n\} \) and there is a \( \tau \)-edge \((\vec{e}_i, \tau, \vec{r}, \vec{e}_i') \in E_i\) such that
  - \( \nu \models \varphi \), "source valuation satisfies guard"
  - \( \vec{e}' = \vec{e}[\vec{r}_i] \), "automaton \( i \) changes location"
  - \( \nu' = \nu[\vec{r}] \), "\( \nu' \) is the result of applying \( \vec{r} \) on \( \nu \)"

- A **synchronisation transition** \( \langle \vec{e}, \nu \rangle \xrightarrow{b} \langle \vec{e}', \nu' \rangle \) occurs if there are \( i, j \in \{1, \ldots, n\} \) with \( i \neq j \) and there are edges \((\vec{e}_i, b!, \varphi_i, \vec{r}_i, \vec{e}_i') \in E_i\) and \((\vec{e}_j, b?, \varphi_j, \vec{r}_j, \vec{e}_j') \in E_j\) such that
  - \( \nu \models \varphi_i \land \varphi_j \), "source valuation satisfies guards (!)"
  - \( \vec{e}' = \vec{e}[\vec{r}_i][\vec{r}_j] \), "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).
ChoicePanel: (simplified)

Example

User:

C50!

E1!

WATER!

SOFT!

TEA!

ChoicePanel:

idle

SOFT?

soft_enabled

TEA?

tea_enabled

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

half_idle

water_enabled := true

water_selected

soft_selected

request_sent

DOK?

OK!

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

DTEA!

DWATER!

DSOFT!

tea_enabled

TEA?

soft_enabled

SOFT?

water_enabled

WATER?

Cini:

< (idle, l), s = 0 >

< (idlle, l), t = 0 >

< (water_selected, l), s = 0 >

< (water_selected, l), t = 0 >

< (request_sent, l), s = 1 >

< (request_sent, l), t = 0 >

< (water_selected, l), s = 1 >

< (water_selected, l), t = 0 >

< (water_selected, l), s = 0 >

< (water_selected, l), t = 1 >
A transition sequence of $C(A_1, \ldots, A_n)$ is any (in)finite sequence of the form

$$(\vec{\ell}_0, \nu_0) \xrightarrow{\lambda_1} (\vec{\ell}_1, \nu_1) \xrightarrow{\lambda_2} (\vec{\ell}_2, \nu_2) \xrightarrow{\lambda_3} \ldots$$

with

- $(\vec{\ell}_0, \nu_0) = C_{\text{ini}}$ (without "start/ from")
- for all $i \in \mathbb{N}$, there is $\xrightarrow{\lambda_{i+1}}$ in $T(C(A_1, \ldots, A_n))$ with $(\vec{\ell}_i, \nu_i) \xrightarrow{\lambda_{i+1}} (\vec{\ell}_{i+1}, \nu_{i+1})$. 
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
\[
\neg \left( \exists \lambda \in \Lambda \ \exists \langle \ell', \nu' \rangle \in Conf \bullet \langle \ell, \nu \rangle \xrightarrow{\lambda} \langle \ell', \nu' \rangle \right).
\]

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.
Reachability

- A configuration $\langle \vec{l}, \nu \rangle$ is called **reachable** (in $C(A_1, \ldots, A_n)$) 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 **reachable** (without “from”!) if and only if it is reachable from $C_{ini}$.

- A location $l \in L_i$ is called **reachable** if and only if any configuration $\langle \vec{l}, \nu \rangle$ with $l_i = l$ is reachable, i.e. there exist $\vec{l}$ and $\nu$ such that $l_i = l$ and $\langle \vec{l}, \nu \rangle$ is reachable.
Uppaal

(Larsen et al., 1997; Behrmann et al., 2004)
The Uppaal Query Language

Consider $\mathcal{N} = C(A_1, \ldots, A_n)$ over data variables $V$.

- **basic formula:**
  
  
  \[
  atom ::= A_i.\ell \mid \varphi \mid \text{deadlock}
  \]

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

- **configuration formulae:**
  
  \[
  term ::= atom \mid \text{not} \ term \mid term_1 \text{ and } term_2
  \]

- **existential path formulae:**

  \[
  e\text{-}formula ::= \exists\lozenge term \quad \text{(exists finally)}
  \]

  \[
  | \exists\square term \quad \text{(exists globally)}
  \]

- **universal path formulae:**

  \[
  a\text{-}formula ::= \forall\lozenge term \quad \text{(always finally)}
  \]

  \[
  | \forall\square term \quad \text{(always globally)}
  \]

  \[
  | term_1 \rightarrow term_2 \quad \text{(leads to)}
  \]

- **formulae (or queries):**

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

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

between configurations

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

deadlock configuration

iff \( \ell_0, i \) is a deadlock configuration

iff \( \nu \models \chi \)

iff \( \langle \vec{\ell}, \nu \rangle \neq \text{term} \)

iff \( \langle \vec{\ell}, \nu \rangle \models \text{term}_1 \) and \( \text{term}_2 \)

iff \( \langle \vec{\ell}, \nu \rangle \models \text{term}_1 \), \( i = 1, 2 \)
Example: Computation Paths vs. Computation Tree

ChoicePanel:

User:

WATER? 

soft_selected

TEA?

water_selected

idle

SOFT?

request_sent

OK!

DOK?

OK!

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

DTEA!

DWATER!

DSOFT!

tea_enabled

TEA?

soft_enabled

SOFT?

water_enabled

WATER?

WATER

τ

SOFT

τ

C50!

l

WATER!

E1!

TEA!

WATER

\langle (\text{water\_selected}, l), \begin{array}{c} \text{we} = 1 \\ \text{se} = 1 \\ \text{te} = 0 \end{array} \rangle

SOFT

\langle (\text{soft\_selected}, l), \begin{array}{c} \text{we} = 1 \\ \text{se} = 1 \\ \text{te} = 0 \end{array} \rangle

\langle (\text{request\_sent}, l), \begin{array}{c} \text{we} = 1 \\ \text{se} = 1 \\ \text{te} = 0 \end{array} \rangle

\langle (\text{request\_sent}, l), \begin{array}{c} \text{we} = 1 \\ \text{se} = 1 \\ \text{te} = 0 \end{array} \rangle

\langle (\text{half\_idle}, l), \begin{array}{c} \text{we} = 1 \\ \text{se} = 1 \\ \text{te} = 0 \end{array} \rangle

\langle (\text{half\_idle}, l), \begin{array}{c} \text{we} = 1 \\ \text{se} = 1 \\ \text{te} = 0 \end{array} \rangle
Example: Computation Paths vs. Computation Graph

(or: Transition Graph)

ChoicePanel:

User:

C50!

E1!

WATER!

SOFT!

TEA!

\[ \langle \text{idle, l}, \begin{array}{c} \text{we} = 1 \\ \text{se} = 1 \\ \text{te} = 0 \end{array} \rangle \]

\[ \langle \text{water\_selected, l}, \begin{array}{c} \text{we} = 1 \\ \text{se} = 1 \\ \text{te} = 0 \end{array} \rangle \]

\[ \langle \text{request\_sent, l}, \begin{array}{c} \text{we} = 1 \\ \text{se} = 1 \\ \text{te} = 0 \end{array} \rangle \]

\[ \langle \text{half\_idle, l}, \begin{array}{c} \text{we} = 1 \\ \text{se} = 1 \\ \text{te} = 0 \end{array} \rangle \]

\[ \langle \text{soft\_selected, l}, \begin{array}{c} \text{we} = 1 \\ \text{se} = 1 \\ \text{te} = 0 \end{array} \rangle \]
**Satisfaction of Uppaal Queries by Configurations**

**Exists finally:**

- \( \langle \ell_0, \nu_0 \rangle \models \exists \Diamond \text{term} \)

  iff

  \[ \exists \text{path } \xi \text{ of } \mathcal{N} \text{ starting in } \langle \ell_0, \nu_0 \rangle \exists i \in \mathbb{N}_0 \cdot \xi^i \models \text{term} \]

  “some configuration satisfying term is reachable”

**Example:** \( \langle \ell_0, \nu_0 \rangle \models \exists \Diamond \varphi \)
Satisfaction of Uppaal Queries by Configurations

**Exists globally:**

- \( \langle \vec{\ell}_0, \nu_0 \rangle \models \exists \Box \text{term} \)

iff

\( \exists \text{path } \xi \text{ of } N \text{ starting in } \langle \vec{\ell}_0, \nu_0 \rangle \)

\( \forall i \in \mathbb{N}_0 \bullet \xi^i \models \text{term} \)

“on some computation path, all configurations satisfy term”

**Example:** \( \langle \vec{\ell}_0, \nu_0 \rangle \models \exists \Box \varphi \)
Satisfaction of Uppaal Queries by Configurations

- **Always globally:**

  \[
  \langle \vec{\ell}_0, \nu_0 \rangle \models \forall \Box \text{term} \quad \iff \quad \langle \vec{\ell}_0, \nu_0 \rangle \not\models \exists \Diamond \neg \text{term}
  \]

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

- **Always finally:**

  \[
  \langle \vec{\ell}_0, \nu_0 \rangle \models \forall \Diamond \text{term} \quad \iff \quad \langle \vec{\ell}_0, \nu_0 \rangle \not\models \exists \Box \neg \text{term}
  \]

  “not (on some computation path, all configurations satisfy \(\neg \text{term}\))”
  
  or: “on all computation paths, there is a configuration satisfying \(\text{term}\)”
Satisfaction of Uppaal Queries by Configurations

**Leads to:**

- \( \langle \vec{l}_0, \nu_0 \rangle \models \text{term}_1 \rightarrow \text{term}_2 \)  
  iff  
  \( \forall \) path \( \xi \) of \( \mathcal{N} \) starting in \( \langle \vec{l}_0, \nu_0 \rangle \)  
  \( \forall i \in \mathbb{N}_0 \)  
  \( \xi^i \models \text{term}_1 \rightarrow \xi^i \models \forall \diamond \text{term}_2 \)

“on all paths, from each configuration satisfying \( \text{term}_1 \), a configuration satifying \( \text{term}_2 \) is reachable” (response pattern)

**Example:** \( \langle \vec{l}_0, \nu_0 \rangle \models \varphi_1 \rightarrow \varphi_2 \)
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 (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

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

for the vending machine model \( \mathcal{N}_{VM} \).
**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

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

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 €1.50 in the machine? (Otherwise, the design is broken.)

- **Approach**: Check whether the implication

\[
tea\_enabled \implies \text{CoinValidator}\_\text{have}\_c150
\]

holds in all reachable configurations, i.e. check

\[
\mathcal{N}_{VM} \models \forall \Box tea\_enabled \implies \text{CoinValidator}\_\text{have}\_c150
\]

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.
**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

\[ \mathcal{N}_{VM} \models \forall \square (\text{CoinValidator} \cdot \text{have}_c\text{50} \lor \text{CoinValidator} \cdot \text{have}_c\text{100} \lor \text{CoinValidator} \cdot \text{have}_c\text{150}) \]

...imply \( \text{water}\_\text{enabled} \).
Recall: Universal LSC Example

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

User

CoinValidator

ChoicePanel

Dispenser

\[ \neg (C50 \lor E1 \lor p\text{SOFT} \lor p\text{TEA} \lor p\text{FILLUP}) \]

\[ \neg (d\text{Soft} \lor d\text{TEA}) \]

\[ \text{water\_in\_stock} \]

\[ p\text{WATER} \]

\[ d\text{WATER} \]

\[ \text{OK} \]
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
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 (∃◊ 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 satisfied.
    (If not, analyse the design further using the obtained counter-example).
References
References


