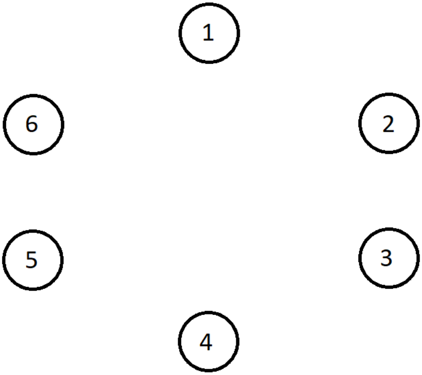
Reconfigurable vs. Static Communications

Mathieu Lehaut
Joint work with Nir Piterman

FM Retreat, 13/01/2023

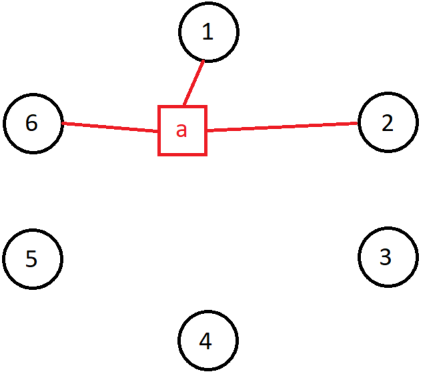
Distributed setting

Agents communicating over channels



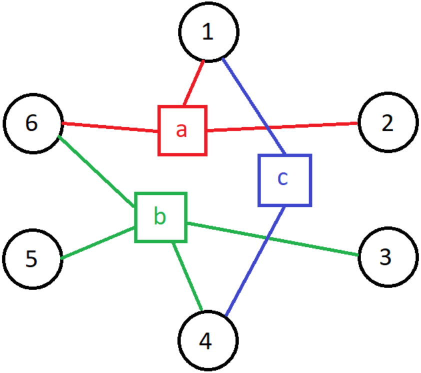
Distributed setting

Agents communicating over channels



Distributed setting

Agents communicating over channels



Static communications

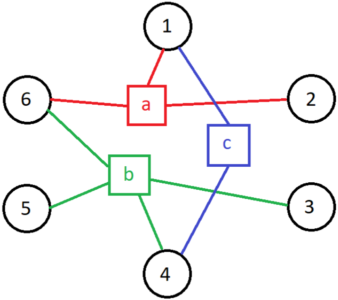
Fixed architecture

Domain function $dom : Channels \rightarrow 2^{Agents}$

Static communications

Fixed architecture

Domain function $dom : Channels \to 2^{Agents}$



$dom(a) = \{1, 2, 6\}, dom(b) = \{3, 4, 5, 6\}, dom(c) = \{1, 4\}$

Reconfigurable communications

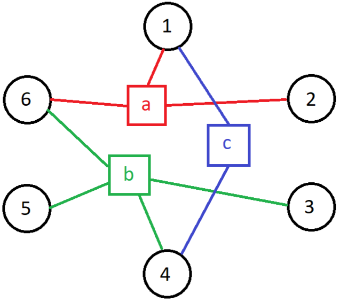
Reconfigurable architecture

Agents can change their interfaces after a communication

Reconfigurable communications

Reconfigurable architecture

Agents can change their interfaces after a communication

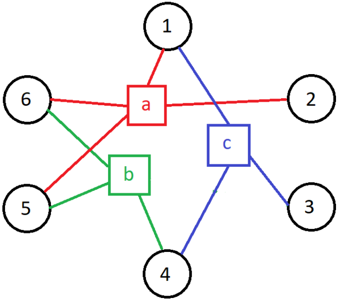


$dom(a) = \{1, 2, 6\}, dom(b) = \{3, 4, 5, 6\}, dom(c) = \{1, 4\}$

Reconfigurable communications

Reconfigurable architecture

Agents can change their interfaces after a communication



$$dom(a) = \{1, 2, 5, 6\}, dom(b) = \{\cancel{3}, 4, 5, 6\}, dom(c) = \{1, 3, 4\}$$

Communication language

- Sequence of channels used during execution, e.g. *ababc*

Communication language

- Sequence of channels used during execution, e.g. *ababc*
- Model for agents $\Rightarrow$ Language

Communication language

- Sequence of channels used during execution, e.g. *ababc*
- Model for agents ⇒ Language

Questions

- Difference between static/reconfigurable models?

Communication language

- Sequence of channels used during execution, e.g. *ababc*
- Model for agents ⇒ Language

Questions

- Difference between static/reconfigurable models?
- Reconstruct model for agents from language?

Zielonka's Asynchronous Automata

Fix $\mathbb{P}$ set of processes, Σ set of channels, *dom* a domain function.

Definition

An AA is a tuple $\mathcal{A} = ((S_p)_{p\in\mathbb{P}}, (s_p^0)_{p\in\mathbb{P}}, (\delta_a)_{a\in\Sigma}, \mathrm{Acc})$

Zielonka's Asynchronous Automata

Fix $\mathbb{P}$ set of processes, Σ set of channels, *dom* a domain function.

Definition

An AA is a tuple $\mathcal{A} = ((S_p)_{p\in\mathbb{P}}, (s_p^0)_{p\in\mathbb{P}}, (\delta_a)_{a\in\Sigma}, \mathrm{Acc})$

States

$S_1 = \underline{r_1}, s_1, t_1$
$S_2 = \underline{r_2}$
$S_3 = \underline{r_3}, s_3, t_3$
$S_4 = \underline{r_4}, s_4$

Zielonka's Asynchronous Automata

Fix $\mathbb{P}$ set of processes, Σ set of channels, *dom* a domain function.

Definition

An AA is a tuple $\mathcal{A} = ((S_p)_{p\in\mathbb{P}}, (s_p^0)_{p\in\mathbb{P}}, (\delta_a)_{a\in\Sigma}, \mathrm{Acc})$

States

$S_1 = \underline{r_1}, s_1, t_1$
$S_2 = \underline{r_2}$
$S_3 = \underline{r_3}, s_3, t_3$
$S_4 = \underline{r_4}, s_4$

Accepting condition

$\mathrm{Acc} = \{(t_1, r_2, t_3, s_4)\}$

Zielonka's Asynchronous Automata

Fix $\mathbb{P}$ set of processes, Σ set of channels, *dom* a domain function.

Definition

An AA is a tuple $\mathcal{A} = ((S_p)_{p\in\mathbb{P}}, (s_p^0)_{p\in\mathbb{P}}, (\delta_a)_{a\in\Sigma}, \mathrm{Acc})$

States

$S_1 = \underline{r_1}, s_1, t_1$
$S_2 = \underline{r_2}$
$S_3 = \underline{r_3}, s_3, t_3$
$S_4 = \underline{r_4}, s_4$

Accepting condition

$\mathrm{Acc} = \{(t_1, r_2, t_3, s_4)\}$

$\delta_a(1, 2)$

$(r_1, r_2) \to (s_1, r_2)$

$\delta_b(3, 4)$

$(r_3, r_4) \to (s_3, r_4) \mid (t_3, r_4) \to (t_3, s_4)$

$\delta_c(1, 3)$

$(s_1, s_3) \to (t_1, t_3)$

States

$S_1 = \underline{r_1}, s_1, t_1$
$S_2 = \underline{r_2}$
$S_3 = \underline{r_3}, s_3, t_3$
$S_4 = \underline{r_4}, s_4$

Accepting condition

$\mathrm{Acc} = \{(t_1, r_2, t_3, s_4)\}$

$\delta_a(1, 2)$

$(r_1, r_2) \to (s_1, r_2)$

$\delta_b(3, 4)$

$(r_3, r_4) \to (s_3, r_4) \mid (t_3, r_4) \to (t_3, s_4)$

$\delta_c(1, 3)$

$(s_1, s_3) \to (t_1, t_3)$

Configuration: r_1, r_2, r_3, r_4

$w =$

States

$S_1 = \underline{r_1}, s_1, t_1$
$S_2 = \underline{r_2}$
$S_3 = \underline{r_3}, s_3, t_3$
$S_4 = \underline{r_4}, s_4$

$\delta_a(1,2)$

$(r_1, r_2) \rightarrow (s_1, r_2)$

$\delta_b(3,4)$

$(r_3, r_4) \rightarrow (s_3, r_4) \mid (t_3, r_4) \rightarrow (t_3, s_4)$

Accepting condition

$\mathrm{Acc} = \{(t_1, r_2, t_3, s_4)\}$

$\delta_c(1,3)$

$(s_1, s_3) \rightarrow (t_1, t_3)$

Configuration: s_1, r_2, r_3, r_4

$w = a$

States

$S_1 = \underline{r_1}, s_1, t_1$
$S_2 = \underline{r_2}$
$S_3 = \underline{r_3}, s_3, t_3$
$S_4 = \underline{r_4}, s_4$

$\delta_a(1,2)$

$(r_1, r_2) \to (s_1, r_2)$

$\delta_b(3,4)$

$(r_3, r_4) \to (s_3, r_4) \mid (t_3, r_4) \to (t_3, s_4)$

Accepting condition

$\mathrm{Acc} = \{(t_1, r_2, t_3, s_4)\}$

$\delta_c(1,3)$

$(s_1, s_3) \to (t_1, t_3)$

Configuration: s_1, r_2, s_3, r_4

$w = ab$

States

$S_1 = \underline{r_1}, s_1, t_1$
$S_2 = \underline{r_2}$
$S_3 = \underline{r_3}, s_3, t_3$
$S_4 = \underline{r_4}, s_4$

Accepting condition

$\mathrm{Acc} = \{(t_1, r_2, t_3, s_4)\}$

$\delta_a(1, 2)$

$(r_1, r_2) \to (s_1, r_2)$

$\delta_b(3, 4)$

$(r_3, r_4) \to (s_3, r_4) \mid (t_3, r_4) \to (t_3, s_4)$

$\delta_c(1, 3)$

$(s_1, s_3) \to (t_1, t_3)$

Configuration: t_1, r_2, t_3, r_4

$w = abc$

States

$S_1 = \underline{r_1}, s_1, t_1$
$S_2 = \underline{r_2}$
$S_3 = \underline{r_3}, s_3, t_3$
$S_4 = \underline{r_4}, s_4$

$\delta_a(1,2)$

$(r_1, r_2) \to (s_1, r_2)$

$\delta_b(3,4)$

$(r_3, r_4) \to (s_3, r_4) \mid (t_3, r_4) \to (t_3, s_4)$

Accepting condition

$\mathrm{Acc} = \{(t_1, r_2, t_3, s_4)\}$

$\delta_c(1,3)$

$(s_1, s_3) \to (t_1, t_3)$

Configuration: t_1, r_2, t_3, s_4

$w = abcb \in \mathcal{L}$

States

$S_1 = \underline{r_1}, s_1, t_1$
$S_2 = \underline{r_2}$
$S_3 = \underline{r_3}, s_3, t_3$
$S_4 = \underline{r_4}, s_4$

Accepting condition

$\mathrm{Acc} = \{(t_1, r_2, t_3, s_4)\}$

$\delta_a(1, 2)$

$(r_1, r_2) \to (s_1, r_2)$

$\delta_b(3, 4)$

$(r_3, r_4) \to (s_3, r_4) \mid (t_3, r_4) \to (t_3, s_4)$

$\delta_c(1, 3)$

$(s_1, s_3) \to (t_1, t_3)$

Configuration: t_1, r_2, t_3, s_4

$w = abcb \in \mathcal{L}$ (and *bacb* too)

Reconfigurable automata

Fix $\mathbb{P}$ set of processes, Σ set of channels (but no *dom*!), and M set of message contents.

Reconfigurable automata

Fix $\mathbb{P}$ set of processes, Σ set of channels (but no *dom*!), and M set of message contents.

New stuff

- Listening function $L_p : S_p \to 2^\Sigma$

Reconfigurable automata

Fix $\mathbb{P}$ set of processes, Σ set of channels (but no *dom*!), and M set of message contents.

New stuff

- Listening function $L_p : S_p \to 2^{\Sigma}$
- Transitions of the form $s_p \xrightarrow{(a,m)} s'_p$

Example

From static to reconfigurable

Easy direction!

Theorem

Any AA can be simulated by a RA.

From static to reconfigurable

Easy direction!

Theorem

Any AA can be simulated by a RA.

Proof: All L_p are constant, set to $dom^{-1}(p)$

From static to reconfigurable

Easy direction!

Theorem

Any AA can be simulated by a RA.

Proof: All L_p are constant, set to $dom^{-1}(p)$
(Bonus: local transitions $\Rightarrow$ M not needed)

From reconfigurable to static

Theorem

There are RA which cannot be “nicely” simulated by any AA.

From reconfigurable to static

Theorem

There are RA which cannot be "nicely" simulated by any AA.

Proof idea: Set arbitrary subset of channels to be dependant

Conditions for distributability

Fix $\mathcal{A}$ over Σ and *dom*.

Diamond property

If $r \xrightarrow{a} s \xrightarrow{b} t$ with a, b independant,
then $r \xrightarrow{b} s' \xrightarrow{a} t$ also possible

Conditions for distributability

Fix $\mathcal{A}$ over Σ and *dom*.

Diamond property

If $r \xrightarrow{a} s \xrightarrow{b} t$ with a, b independant,
then $r \xrightarrow{b} s' \xrightarrow{a} t$ also possible

Theorem [W. Zielonka, '87]

If $\mathcal{A}$ satisfies the diamond property, then we can build an AA with the same language

Conditions for distributability

Fix $\mathcal{A}$ over Σ and *dom*.

Diamond property

If $r \xrightarrow{a} s \xrightarrow{b} t$ with a, b independant,
then $r \xrightarrow{b} s' \xrightarrow{a} t$ also possible

Theorem [W. Zielonka, '87]

If $\mathcal{A}$ satisfies the diamond property, then we can build an AA with the same language

Our goal: similar property for reconfigurable languages

Conditions for distributability

Fix $\mathcal{A}$ over Σ and *dom*.

Diamond property

If $r \xrightarrow{a} s \xrightarrow{b} t$ with a, b independant,
then $r \xrightarrow{b} s' \xrightarrow{a} t$ also possible

Theorem [W. Zielonka, '87]

If $\mathcal{A}$ satisfies the diamond property, then we can build an AA with the same language

Our goal: similar property for reconfigurable languages

Thanks, questions?