# Communication Requirements for Team Automata


Maurice H. ter Beek[1] Josep Carmona[2]
Rolf Hennicker[3] Jetty Kleijn[4]

[1]ISTI–CNR, Pisa, Italy

[2]Universitat Politècnica de Catalunya, Barcelona, Spain

[3]Ludwig-Maximilians-Universität München, Germany

[4]LIACS, Leiden University, The Netherlands


LIACS, Leiden University
7 May 2018

# Topics of the talk

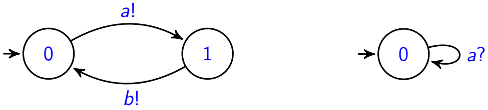
- Systems of communicating components which interact through message exchange.
- *Here:* Synchronous communication, i.e., outputs and inputs of the same message are performed simultaneously.
- Consideration of a wide range of synchronization types (peer-to-peer, multicast, broadcast, gathering, ...).

*Overall goal:*

Safe communication, i.e., avoidance of communication errors.

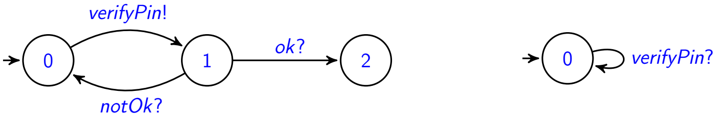
# Typical communication errors

- A component wants to send a message to another component which is not ready to receive it (missing reception).




*Literature:* [Brand, Zafiropulo 1983], [de Alfaro, Henzinger 2001], [Carmona, Cortadella 2002], [Larsen, Nyman, Wąsowski 2007], ...
Point-to-point communication

- A component waits for a message from another component which does not send it (missing response).




*Literature:* [Carrez, Fantechi, Najm 2003], [Durán, Ouederni, Salaün 2012]

# Contribution of our work

- Uniform formalism for the specification of various synchronization types.
- Investigation and formalization of *receptiveness* and *responsiveness requirements* (communication requirements).
- Communication requirements depend on the particular type of synchronization used in a system.
  - We show how to derive communication requirements from any particular synchronization type.
  - Generic theory for communication-safety (compatibility):

    synchronization type A $\mapsto$ compatibility notion A

    synchronization type B $\mapsto$ compatibility notion B

    ...

# Formal foundations

**Component automaton** is a tuple $\mathcal{A} = (Q, q^0, \Sigma, \rightarrow)$ such that

- $Q$ is a set of states, $q^0 \in Q$ is the initial state,
- $\Sigma$ is the disjoint union of sets $\Sigma_{inp}$, $\Sigma_{out}$, $\Sigma_{int}$ of input, output, and internal actions,
- $\rightarrow \subseteq Q \times \Sigma \times Q$ is a labelled transition relation.

# Formal foundations

**Component automaton** is a tuple $\mathcal{A} = (Q, q^0, \Sigma, \rightarrow)$ such that

- $Q$ is a set of states, $q^0 \in Q$ is the initial state,
- $\Sigma$ is the disjoint union of sets $\Sigma_{inp}$, $\Sigma_{out}$, $\Sigma_{int}$ of input, output, and internal actions,
- $\rightarrow \subseteq Q \times \Sigma \times Q$ is a labelled transition relation.

**Component system** is an indexed set $\mathcal{S} = \{ \mathcal{A}_i \mid i \in I \}$ of component automata $\mathcal{A}_i = (Q_i, q_i^0, \Sigma_i, \rightarrow_i)$ such that the internal actions of all $\mathcal{A}_i$ are unique within the system.

# Formal foundations

**Component automaton** is a tuple $\mathcal{A} = (Q, q^0, \Sigma, \rightarrow)$ such that

- $Q$ is a set of states, $q^0 \in Q$ is the initial state,
- $\Sigma$ is the disjoint union of sets $\Sigma_{inp}$, $\Sigma_{out}$, $\Sigma_{int}$ of input, output, and internal actions,
- $\rightarrow \subseteq Q \times \Sigma \times Q$ is a labelled transition relation.

**Component system** is an indexed set $\mathcal{S} = \{ \mathcal{A}_i \mid i \in I \}$ of component automata $\mathcal{A}_i = (Q_i, q_i^0, \Sigma_i, \rightarrow_i)$ such that the internal actions of all $\mathcal{A}_i$ are unique within the system.

**System state** is a tuple $(q_i)_{i \in I}$ with $q_i \in Q_i$ for all $i \in I$.

# Formal foundations

**Component automaton** is a tuple $\mathcal{A} = (Q, q^0, \Sigma, \rightarrow)$ such that

- $Q$ is a set of states, $q^0 \in Q$ is the initial state,
- $\Sigma$ is the disjoint union of sets $\Sigma_{inp}$, $\Sigma_{out}$, $\Sigma_{int}$ of input, output, and internal actions,
- $\rightarrow \subseteq Q \times \Sigma \times Q$ is a labelled transition relation.

**Component system** is an indexed set $\mathcal{S} = \{ \mathcal{A}_i \mid i \in I \}$ of component automata $\mathcal{A}_i = (Q_i, q_i^0, \Sigma_i, \rightarrow_i)$ such that the internal actions of all $\mathcal{A}_i$ are unique within the system.

**System state** is a tuple $(q_i)_{i \in I}$ with $q_i \in Q_i$ for all $i \in I$.

**System transition** is a transition between system states

$$(q_i)_{i \in I} \xrightarrow{a} (q_i')_{i \in I}$$

such that some components perform simultaneously a transition with shared action $a$ (and the others do nothing).

# Formal foundations

**Component automaton** is a tuple $\mathcal{A} = (Q, q^0, \Sigma, \rightarrow)$ such that

- $Q$ is a set of states, $q^0 \in Q$ is the initial state,
- $\Sigma$ is the disjoint union of sets $\Sigma_{inp}$, $\Sigma_{out}$, $\Sigma_{int}$ of input, output, and internal actions,
- $\rightarrow \subseteq Q \times \Sigma \times Q$ is a labelled transition relation.

**Component system** is an indexed set $\mathcal{S} = \{ \mathcal{A}_i \mid i \in I \}$ of component automata $\mathcal{A}_i = (Q_i, q_i^0, \Sigma_i, \rightarrow_i)$ such that the internal actions of all $\mathcal{A}_i$ are unique within the system.

**System state** is a tuple $(q_i)_{i \in I}$ with $q_i \in Q_i$ for all $i \in I$.

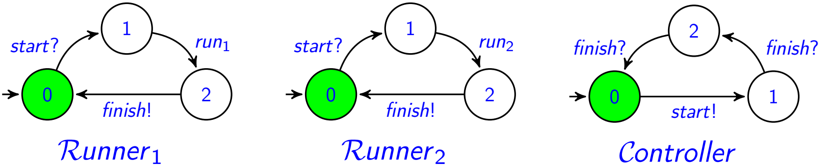
**System transition** is a transition between system states

$$(q_i)_{i \in I} \xrightarrow{a} (q_i')_{i \in I}$$

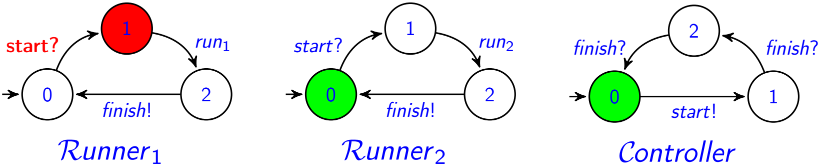
such that some components perform simultaneously a transition with shared action $a$ (and the others do nothing).

**Not all system transitions are meaningful!**

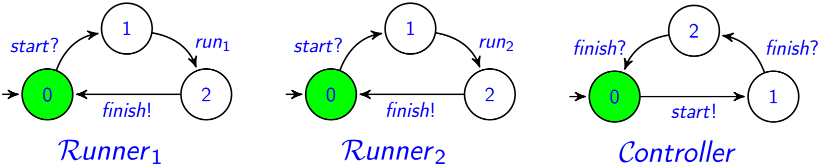
# Example: system transitions



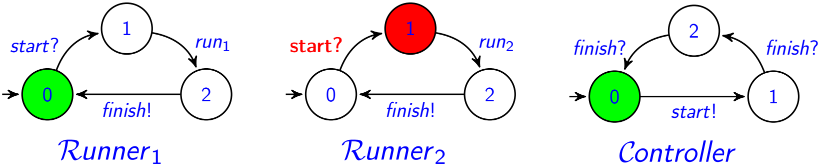
# Example: system transitions



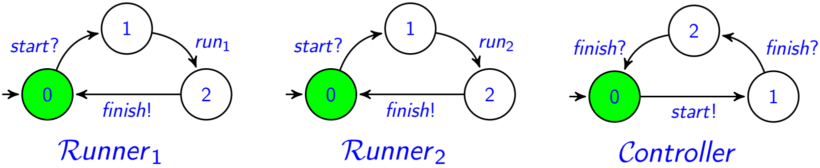
# Example: system transitions



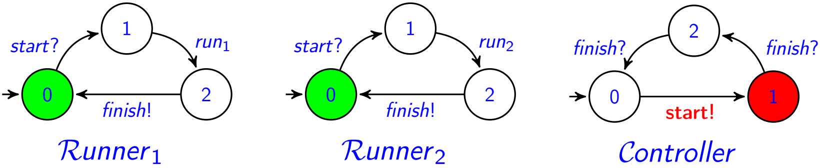
# Example: system transitions



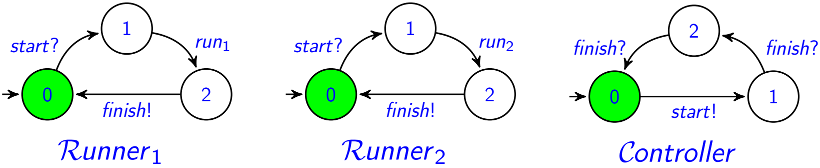
# Example: system transitions



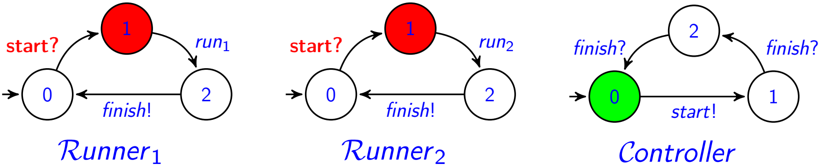
# Example: system transitions



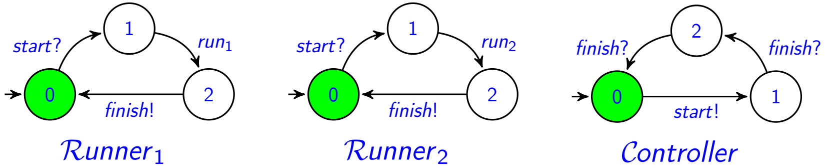
# Example: system transitions



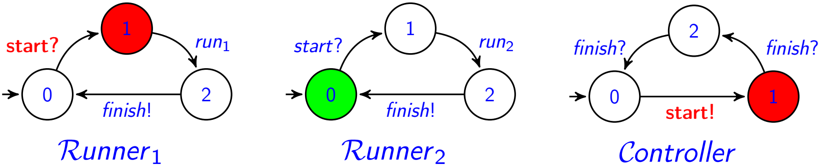
# Example: system transitions



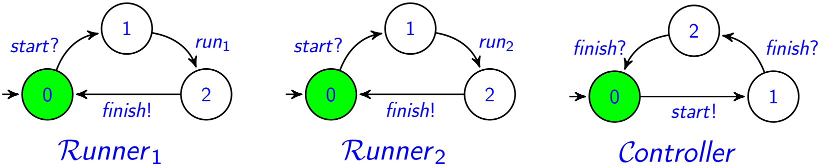
# Example: system transitions



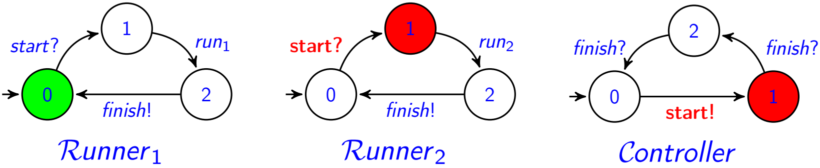
# Example: system transitions



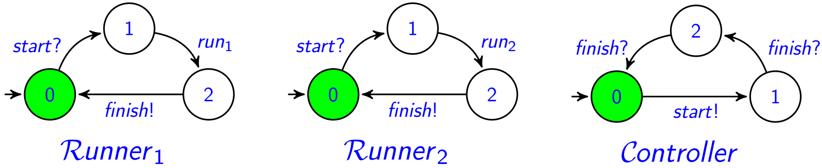
# Example: system transitions



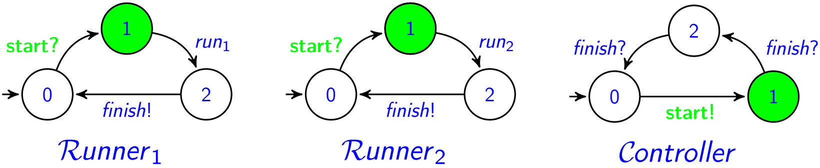
# Example: system transitions



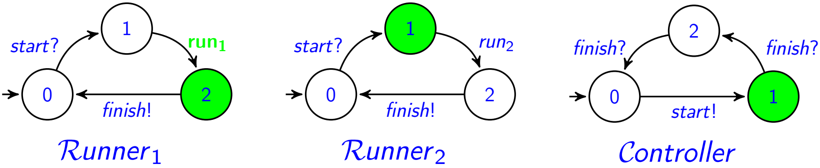
# Example: system transitions



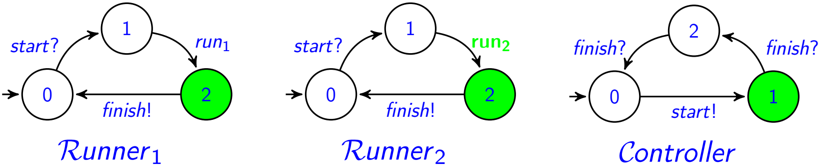
# Example: system transitions



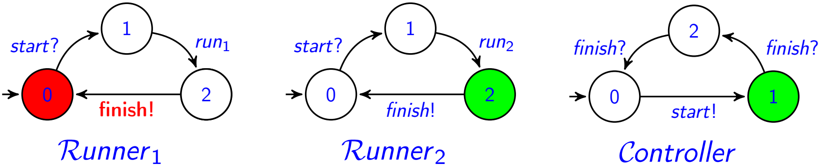
# Example: system transitions



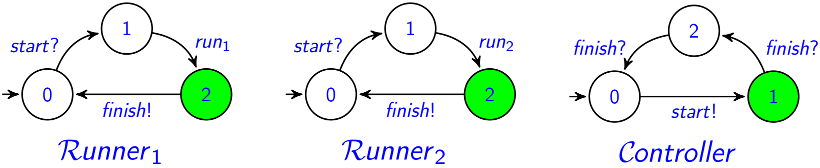
# Example: system transitions



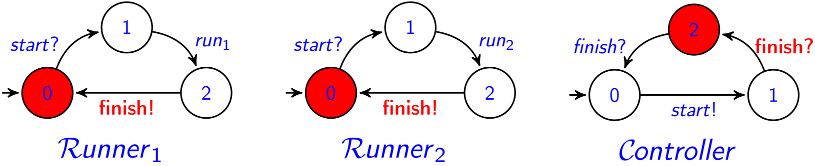
# Example: system transitions



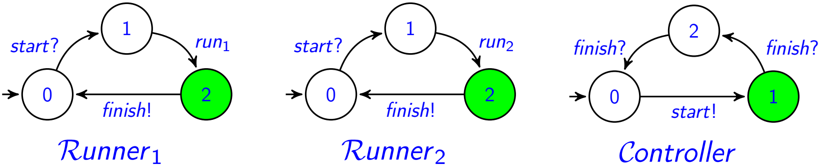
# Example: system transitions



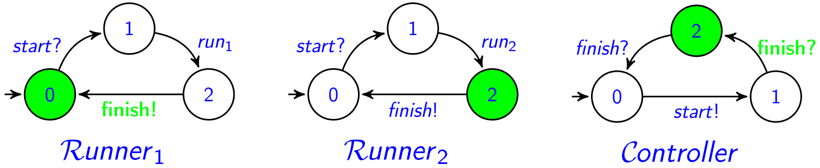
# Example: system transitions



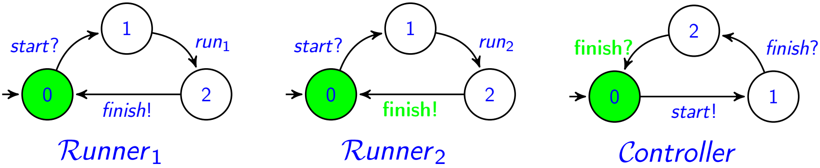
# Example: system transitions



# Example: system transitions



# Example: system transitions



## Team automata

Let $\mathcal{S} = \{ \mathcal{A}_i \mid i \in I \}$ be a component system and $\delta$ be a set of system transitions.

The **team automaton** over $\mathcal{S}$ with *synchronization policy* $\delta$ is the component automaton

$$\mathcal{T} = (\prod_{i \in I} Q_i, (q_i^0)_{i \in I}, \Sigma, \delta)$$

such that $\Sigma = \Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$ with

- $\Sigma_{inp} = (\bigcup_{i \in I} \Sigma_{i,inp}) \setminus \Sigma_{out}$,
- $\Sigma_{out} = \bigcup_{i \in I} \Sigma_{i,out}$,
- $\Sigma_{int} = \bigcup_{i \in I} \Sigma_{i,int}$.

Communication actions: $\Sigma_{com} = (\bigcup_{i \in I} \Sigma_{i,out}) \cap (\bigcup_{i \in I} \Sigma_{i,inp})$

# Team automata

Let $\mathcal{S} = \{ \mathcal{A}_i \mid i \in I \}$ be a component system and $\delta$ be a set of system transitions.

The **team automaton** over $\mathcal{S}$ with *synchronization policy* $\delta$ is the component automaton

$$\mathcal{T} = (\prod_{i \in I} Q_i, (q_i^0)_{i \in I}, \Sigma, \delta)$$

such that $\Sigma = \Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$ with

- $\Sigma_{inp} = (\bigcup_{i \in I} \Sigma_{i,inp}) \setminus \Sigma_{out}$,
- $\Sigma_{out} = \bigcup_{i \in I} \Sigma_{i,out}$,
- $\Sigma_{int} = \bigcup_{i \in I} \Sigma_{i,int}$.

Communication actions: $\Sigma_{com} = (\bigcup_{i \in I} \Sigma_{i,out}) \cap (\bigcup_{i \in I} \Sigma_{i,inp})$

**How to specify synchronization policies?**

# Synchronization types

*Idea:*

Specify how many senders and how many receivers are allowed to participate in a system transition $(q_i)_{i \in I} \xrightarrow{a} (q'_i)_{i \in I}$.

**Formally:**

A synchronization type is a pair $(\mathtt{snd}, \mathtt{rcv})$ consisting of

- a sending multiplicity $\mathtt{snd}$ and
- a receiving multiplicity $\mathtt{rcv}$

where a multiplicity has one of the following forms:

- $[\mathtt{min}, \mathtt{max}]$ with $\mathtt{min} \in \mathbb{N}, \mathtt{max} \in \mathbb{N} \cup \{*\}$ such that $\mathtt{min} \leq \mathtt{max}$,
- $\mathtt{ai}$ "action indispensible", or
- $\mathtt{si}$ "state indispensible".

# Team automata and synchronization types

- For each system transition
$$t : (q_i)_{i\in I} \xrightarrow{a} (q'_i)_{i\in I}$$
  - the group of components participating in $t$ by outputting $a$ is denoted by $Out(t) \subseteq \mathcal{S}$,
  - the group of components participating in $t$ by inputting $a$ is denoted by $In(t) \subseteq \mathcal{S}$.

# Team automata and synchronization types

- For each system transition
$$t : (q_i)_{i \in I} \xrightarrow{a} (q'_i)_{i \in I}$$
  - the group of components participating in $t$ by outputting $a$ is denoted by $Out(t) \subseteq \mathcal{S}$,
  - the group of components participating in $t$ by inputting $a$ is denoted by $In(t) \subseteq \mathcal{S}$.

(!) A team automaton $\mathcal{T}$ with synchronization policy $\delta$ is of type $(\mathtt{snd}, \mathtt{rcv})$ if

(i) for all transitions $t \in \delta$ labeled with $a \in \Sigma_{com}$, $Out(t)$ <u>fits to</u> $\mathtt{snd}$ and $In(t)$ <u>fits to</u> $\mathtt{rcv}$, <u>and</u>

(ii) $\delta$ contains all such transitions.

# Team automata and synchronization types

- For each system transition

$$t : (q_i)_{i \in I} \xrightarrow{a} (q'_i)_{i \in I}$$

  - the group of components participating in $t$ by outputting $a$ is denoted by $Out(t) \subseteq \mathcal{S}$,
  - the group of components participating in $t$ by inputting $a$ is denoted by $In(t) \subseteq \mathcal{S}$.

(!) A team automaton $\mathcal{T}$ with synchronization policy $\delta$ is of type $(\mathtt{snd}, \mathtt{rcv})$ if

(i) for all transitions $t \in \delta$ labeled with $a \in \Sigma_{com}$, $Out(t)$ <u>fits to</u> $\mathtt{snd}$ and $In(t)$ <u>fits to</u> $\mathtt{rcv}$, <u>and</u>

(ii) $\delta$ contains all such transitions.

- $Out(t)$ fits to $\mathtt{snd}$ if
  - if $\mathtt{snd} = [\mathtt{min}, \mathtt{max}]$ then $\mathtt{min} \leq |Out(t)| \leq \mathtt{max}$,
  - if $\mathtt{snd} = \mathtt{ai}$ then $Out(t) = \{ \mathcal{A}_i \in \mathcal{S} \mid a \in \Sigma_{i,out} \}$,
  - if $\mathtt{snd} = \mathtt{si}$ then "as above" + $a$ is enabled in $q_i$.

# Team automata and synchronization types

- For each system transition
$$t : (q_i)_{i\in I} \xrightarrow{a} (q'_i)_{i\in I}$$
  - the group of components participating in $t$ by outputting $a$ is denoted by $Out(t) \subseteq \mathcal{S}$,
  - the group of components participating in $t$ by inputting $a$ is denoted by $In(t) \subseteq \mathcal{S}$.

(!) A team automaton $\mathcal{T}$ with synchronization policy $\delta$ is of type $(\mathtt{snd}, \mathtt{rcv})$ if

  (i) for all transitions $t \in \delta$ labeled with $a \in \Sigma_{com}$, $Out(t)$ fits to $\mathtt{snd}$ and $In(t)$ fits to $\mathtt{rcv}$, and

  (ii) $\delta$ contains all such transitions.

- $Out(t)$ fits to $\mathtt{snd}$ if
  - if $\mathtt{snd} = [\mathtt{min}, \mathtt{max}]$ then $\mathtt{min} \leq |Out(t)| \leq \mathtt{max}$,
  - if $\mathtt{snd} = \mathtt{ai}$ then $Out(t) = \{\, \mathcal{A}_i \in \mathcal{S} \mid a \in \Sigma_{i,out} \}$,
  - if $\mathtt{snd} = \mathtt{si}$ then "as above" + $a$ is enabled in $q_i$.
- $In(t)$ fits to $\mathtt{rcv}$ is defined analogously.

# Teams with familiar synchronization types

$([1, 1], [1, 1])$ binary, peer-to-peer communication

# Teams with familiar synchronization types

$([1,1],[0,1])$ binary, peer-to-peer communication, lossy

# Teams with familiar synchronization types

| | |
|---|---|
| $([1,1],[0,1])$ | binary, peer-to-peer communication, lossy |
| $([0,1],[0,1])$ | non-blocking peer-to-peer (CCS) |

# Teams with familiar synchronization types

| | |
|---|---|
| $([1,1],[0,1])$ | binary, peer-to-peer communication, lossy |
| $([0,1],[0,1])$ | non-blocking peer-to-peer (CCS) |
| $([1,1],[0,*])$ | multicast |

# Teams with familiar synchronization types

| | |
|---|---|
| $([1, 1], [0, 1])$ | binary, peer-to-peer communication, lossy |
| $([0, 1], [0, 1])$ | non-blocking peer-to-peer (CCS) |
| $([1, 1], [1, *])$ | multicast, strong |

# Teams with familiar synchronization types

| | |
|---|---|
| $([1,1],[0,1])$ | binary, peer-to-peer communication, lossy |
| $([0,1],[0,1])$ | non-blocking peer-to-peer (CCS) |
| $([1,1],[1,*])$ | multicast, strong |
| $([1,1],\mathtt{si})$ | broadcast |

# Teams with familiar synchronization types

| | |
|---|---|
| $([1,1],[0,1])$ | binary, peer-to-peer communication, lossy |
| $([0,1],[0,1])$ | non-blocking peer-to-peer (CCS) |
| $([1,1],[1,*])$ | multicast, strong |
| $([1,1],\mathtt{ai})$ | broadcast, strong |

# Teams with familiar synchronization types

| | |
|---|---|
| $([1,1],[0,1])$ | binary, peer-to-peer communication, lossy |
| $([0,1],[0,1])$ | non-blocking peer-to-peer (CCS) |
| $([1,1],[1,*])$ | multicast, strong |
| $([1,1],\mathtt{ai})$ | broadcast, strong |
| $([1,*],[0,*])$ | master-slave communication,<br>'slaves never proceed alone' |

# Teams with familiar synchronization types

| | |
|---|---|
| $([1,1],[0,1])$ | binary, peer-to-peer communication, lossy |
| $([0,1],[0,1])$ | non-blocking peer-to-peer (CCS) |
| $([1,1],[1,*])$ | multicast, strong |
| $([1,1],\mathtt{ai})$ | broadcast, strong |
| $([1,*],[1,*])$ | master-slave communication, strong,<br>'slaves must obey (at least one)' |

# Teams with familiar synchronization types

| | |
|---|---|
| $([1,1],[0,1])$ | binary, peer-to-peer communication, lossy |
| $([0,1],[0,1])$ | non-blocking peer-to-peer (CCS) |
| $([1,1],[1,*])$ | multicast, strong |
| $([1,1],\mathtt{ai})$ | broadcast, strong |
| $([1,*],[1,*])$ | master-slave communication, strong, 'slaves must obey (at least one)' |
| $(\mathtt{ai},\mathtt{ai})$ | full synchronization (FSP) |

# Teams with familiar synchronization types

| | |
|---|---|
| $([1,1],[0,1])$ | binary, peer-to-peer communication, lossy |
| $([0,1],[0,1])$ | non-blocking peer-to-peer (CCS) |
| $([1,1],[1,*])$ | multicast, strong |
| $([1,1],\mathtt{ai})$ | broadcast, strong |
| $([1,*],[1,*])$ | master-slave communication, strong, 'slaves must obey (at least one)' |
| $(\mathtt{ai},\mathtt{ai})$ | full synchronization (FSP) |
| $([1,*],[1,1])$ | gathering |

# Teams with familiar synchronization types

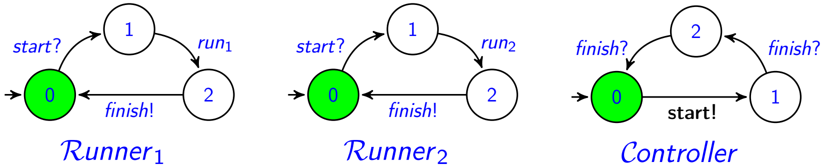
| | |
|---|---|
| $([1,1],[0,1])$ | binary, peer-to-peer communication, lossy |
| $([0,1],[0,1])$ | non-blocking peer-to-peer (CCS) |
| $([1,1],[1,*])$ | multicast, strong |
| $([1,1],\texttt{ai})$ | broadcast, strong |
| $([1,*],[1,*])$ | master-slave communication, strong, 'slaves must obey (at least one)' |
| $(\texttt{ai},\texttt{ai})$ | full synchronization (FSP) |
| $([1,*],[1,1])$ | gathering |
| $([0,*],[0,*])$ | all system transitions |

# Receptiveness requirements

*Basic idea:*

Whenever (a group of) components want to send a message, there should be (a group of) components ready to receive the message in conformance with the synchronization type.

**Example** with synchronization type $([1, 1], \mathtt{ai})$:

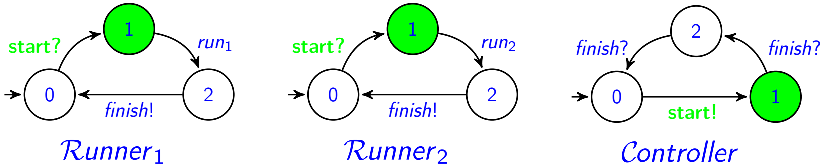


# Receptiveness requirements

*Basic idea:*

Whenever (a group of) components want to send a message, there should be (a group of) components ready to receive the message in conformance with the synchronization type.

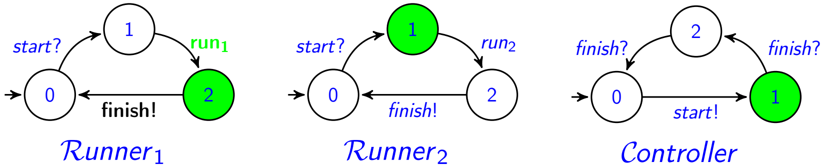
**Example** with synchronization type $([1,1], \mathtt{ai})$:



*Basic idea:*

Whenever (a group of) components want to send a message, there should be (a group of) components ready to receive the message in conformance with the synchronization type.

**Example** with synchronization type $([1,1], \mathtt{ai})$:

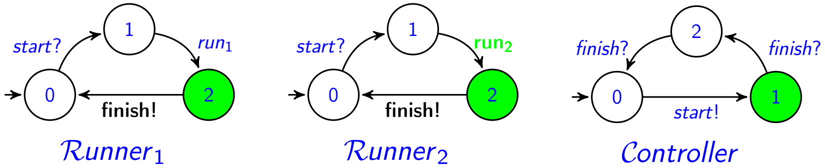


# Receptiveness requirements

*Basic idea:*

Whenever (a group of) components want to send a message, there should be (a group of) components ready to receive the message in conformance with the synchronization type.

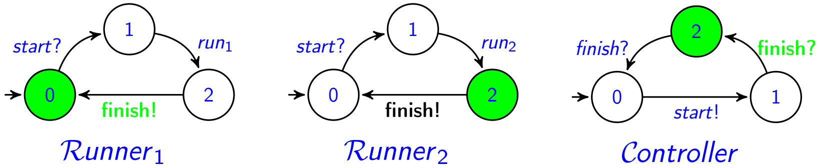
**Example** with synchronization type $([1,1], \mathtt{ai})$:



*Basic idea:*

Whenever (a group of) components want to send a message, there should be (a group of) components ready to receive the message in conformance with the synchronization type.

**Example** with synchronization type $([1, 1], \mathtt{ai})$:

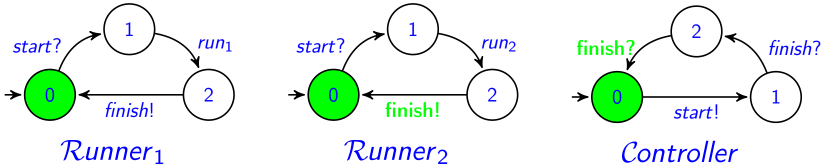


# Receptiveness requirements

*Basic idea:*

Whenever (a group of) components want to send a message, there should be (a group of) components ready to receive the message in conformance with the synchronization type.

**Example** with synchronization type $([1, 1], \mathtt{ai})$:



# Receptiveness requirements: definition

Let $\mathcal{S} = \{ \mathcal{A}_i \mid i \in I \}$ be a component system and $\mathcal{T}$ a team automaton with synchronization type $(\mathtt{snd}, \mathtt{rcv})$.

**Receptiveness requirement:**

For any reachable state $(q_i)_{i \in I}$ of $\mathcal{T}$ and for any action $a \in \Sigma_{com}$ we require:

If there is a group of components $\mathcal{G} = \{ \mathcal{A}_j \mid j \in J \subseteq I \}$ having $a$ as an <u>output</u> action such that

- $a$ is enabled in each local state $q_j$ with $j \in J$ <u>and</u>
- $\mathcal{G}$ fits to $\mathtt{snd}$ (!),

then there exists a group of components $\mathcal{H} = \{ \mathcal{A}_k \mid k \in K \subseteq I \}$ having $a$ as an <u>input</u> action such that

- $a$ is enabled in each local state $q_k$ with $k \in K$ <u>and</u>
- $\mathcal{H}$ fits to $\mathtt{rcv}$ (!).

# Receptiveness requirements: definition

Let $\mathcal{S} = \{ \mathcal{A}_i \mid i \in I \}$ be a component system and
$\mathcal{T}$ a team automaton with synchronization type $(\texttt{snd}, \texttt{rcv})$.

**Receptiveness requirement:**

For any reachable state $(q_i)_{i \in I}$ of $\mathcal{T}$ and for any action $a \in \Sigma_{com}$ we require:

If there is a group of components $\mathcal{G} = \{ \mathcal{A}_j \mid j \in J \subseteq I \}$ having $a$ as an <u>output</u> action such that

- $a$ is enabled in each local state $q_j$ with $j \in J$ <u>and</u>
- $\mathcal{G}$ fits to $\texttt{snd}$ (!),

then there exists a group of components $\mathcal{H} = \{ \mathcal{A}_k \mid k \in K \subseteq I \}$ having $a$ as an <u>input</u> action such that

- $a$ is enabled in each local state $q_k$ with $k \in K$ <u>and</u>
- $\mathcal{H}$ fits to $\texttt{rcv}$ (!).

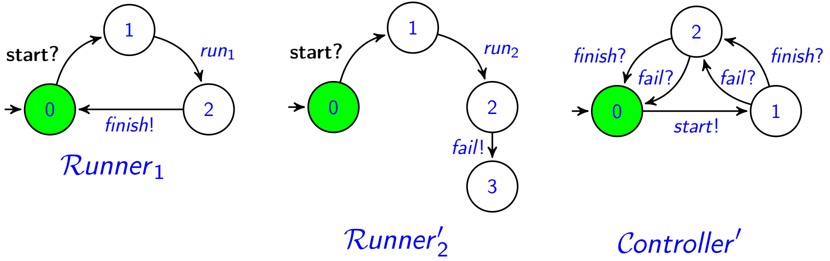
**Hence, the team can perform a transition** $(q_i)_{i \in I} \xrightarrow{a} (q'_i)_{i \in I}$

# Responsiveness requirements

*Basic idea:*

Whenever (a group of) components wait to receive one of the messages $a_1, \ldots, a_n$, there should be (a group of) components able to send (at least) one of the messages $a_1, \ldots, a_n$ in conformance with the synchronization type (*external choice*).

**Example** with synchronization type $([1, 1], \texttt{ai})$:

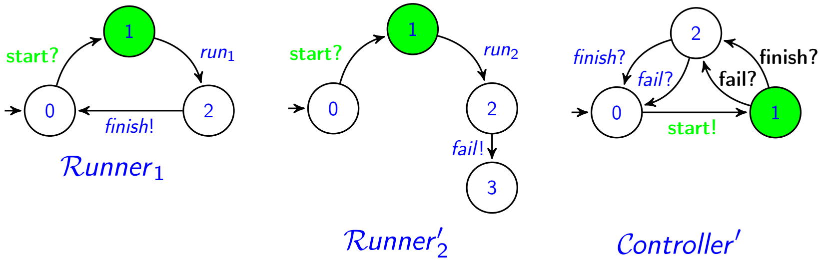


# Responsiveness requirements

*Basic idea:*

Whenever (a group of) components wait to receive one of the messages $a_1, \ldots, a_n$, there should be (a group of) components able to send (at least) one of the messages $a_1, \ldots, a_n$ in conformance with the synchronization type (*external choice*).

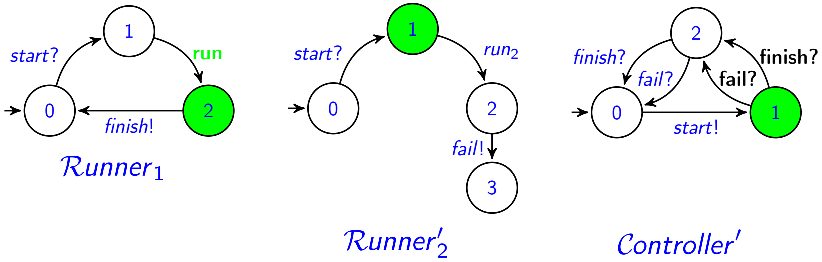
**Example** with synchronization type $([1, 1], \mathtt{ai})$:



*Basic idea:*

Whenever (a group of) components wait to receive one of the messages $a_1, \ldots, a_n$, there should be (a group of) components able to send (at least) one of the messages $a_1, \ldots, a_n$ in conformance with the synchronization type (*external choice*).

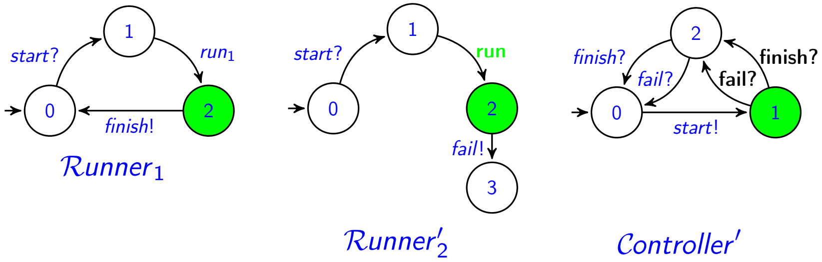
**Example** with synchronization type $([1, 1], \texttt{ai})$:



*Basic idea:*

Whenever (a group of) components wait to receive one of the messages $a_1, \ldots, a_n$, there should be (a group of) components able to send (at least) one of the messages $a_1, \ldots, a_n$ in conformance with the synchronization type (*external choice*).

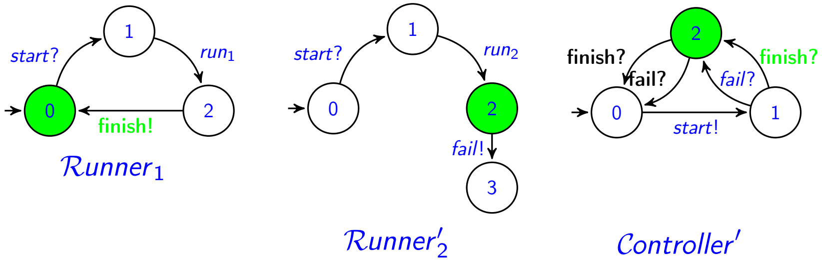
**Example** with synchronization type $([1,1], \mathtt{ai})$:



*Basic idea:*

Whenever (a group of) components wait to receive one of the messages $a_1, \ldots, a_n$, there should be (a group of) components able to send (at least) one of the messages $a_1, \ldots, a_n$ in conformance with the synchronization type (*external choice*).

**Example** with synchronization type $([1,1], \mathtt{ai})$:

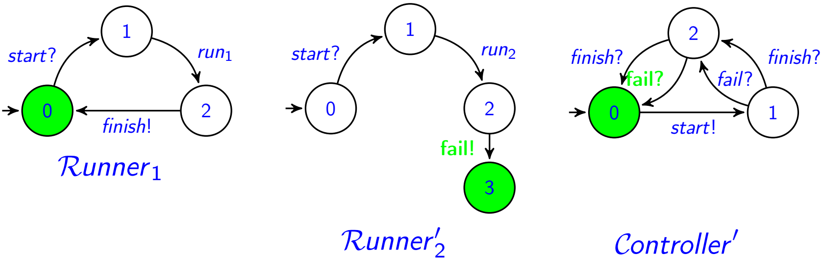


# Responsiveness requirements

*Basic idea:*

Whenever (a group of) components wait to receive one of the messages $a_1, \ldots, a_n$, there should be (a group of) components able to send (at least) one of the messages $a_1, \ldots, a_n$ in conformance with the synchronization type (*external choice*).

**Example** with synchronization type $([1, 1], \mathtt{ai})$:



# Responsiveness requirements: definition

Let $\mathcal{S} = \{ \mathcal{A}_i \mid i \in I \}$ be a component system and $\mathcal{T}$ a team automaton with synchronization type $(\mathtt{snd}, \mathtt{rcv})$.

**Responsiveness requirement:**

For any reachable state $(q_i)_{i \in I}$ of $\mathcal{T}$ and for any set of actions $a_1, \ldots, a_n \in \Sigma_{com}$ we require:

If there is a group of components $\mathcal{G} = \{ \mathcal{A}_j \mid j \in J \subseteq I \}$ having $a_1, \ldots, a_n$ as <u>input</u> actions such that

- exactly all $a_1, \ldots, a_n$ are enabled in each local state $q_j$ with $j \in J$ <u>and</u>
- $\mathcal{G}$ fits to $\mathtt{rcv}$ (!),

then there exists a group of components $\mathcal{H} = \{ \mathcal{A}_k \mid k \in K \subseteq I \}$ having (at least) one $a \in \{a_1, \ldots, a_n\}$ as <u>output</u> action such that

- $a$ is *weakly* enabled in each local state $q_k$ with $k \in K$ <u>and</u>
- $\mathcal{H}$ fits to $\mathtt{snd}$ (!).

## Responsiveness requirements: definition

Let $\mathcal{S} = \{ \mathcal{A}_i \mid i \in I \}$ be a component system and $\mathcal{T}$ a team automaton with synchronization type $(\texttt{snd}, \texttt{rcv})$.

**Responsiveness requirement:**

For any reachable state $(q_i)_{i \in I}$ of $\mathcal{T}$ and for any set of actions $a_1, \ldots, a_n \in \Sigma_{com}$ we require:

If there is a group of components $\mathcal{G} = \{ \mathcal{A}_j \mid j \in J \subseteq I \}$ having $a_1, \ldots, a_n$ as input actions such that

- exactly all $a_1, \ldots, a_n$ are enabled in each local state $q_j$ with $j \in J$ and
- $\mathcal{G}$ fits to $\texttt{rcv}$ (!),

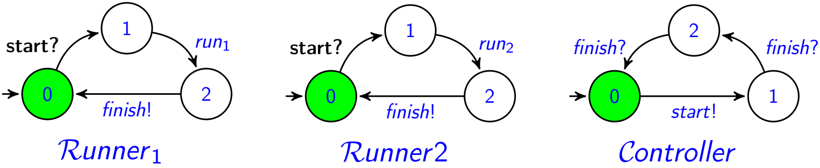
then there exists a group of components $\mathcal{H} = \{ \mathcal{A}_k \mid k \in K \subseteq I \}$ having (at least) one $a \in \{a_1, \ldots, a_n\}$ as output action such that

- $a$ is *weakly* enabled in each local state $q_k$ with $k \in K$ and
- $\mathcal{H}$ fits to $\texttt{snd}$ (!).

**Hence, the team can perform a transition** $(q_i)_{i \in I} \xrightarrow{a_i} (q'_i)_{i \in I}$

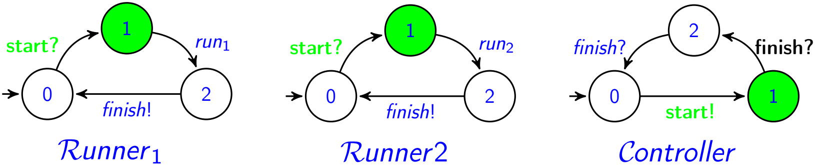
# Example: non-responsive system

Use synchronization type $([1,2], \mathtt{ai})$ instead of $([1,1], \mathtt{ai})$ in the system:



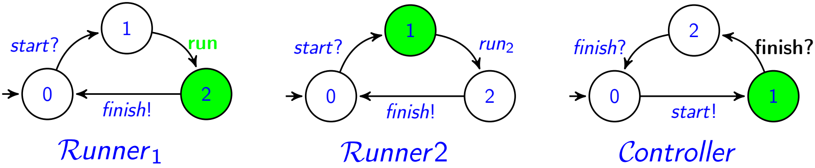
# Example: non-responsive system

Use synchronization type $([1,2], \mathtt{ai})$ instead of $([1,1], \mathtt{ai})$ in the system:



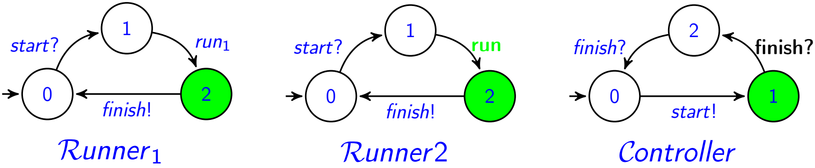
# Example: non-responsive system

Use synchronization type $([1,2], \texttt{ai})$ instead of $([1,1], \texttt{ai})$ in the system:



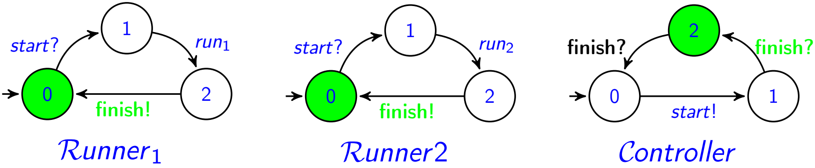
# Example: non-responsive system

Use synchronization type $([1,2],\texttt{ai})$ instead of $([1,1],\texttt{ai})$ in the system:



# Example: non-responsive system

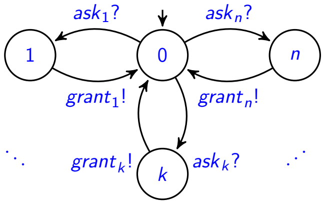
Use synchronization type $([1,2],\mathtt{ai})$ instead of $([1,1],\mathtt{ai})$ in the system:



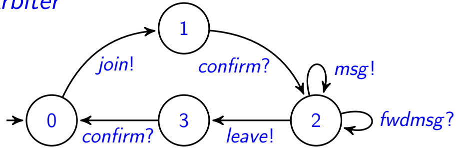
# Conclusion

- Meta-theory for communication-safety (compatibility) in multi-component systems applicable to many concrete synchronization types.
- Thus we have a means to compare compatibility notions from the literature (for closed/open systems, pessimistic/ optimistic receptiveness, strong/weak compatibility).
- Next steps:
  - larger case studies,
  - asynchronous communication,
  - synchronization types per action,
  - encapsulation and hierarchical constructions.

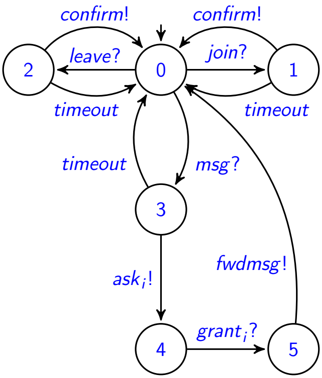
# Teaser: a distributed chat system




(a) $\mathcal{A}rbiter$




(b) $\mathcal{C}lient_i$

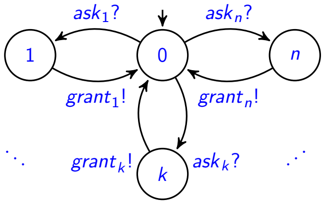



(c) $\mathcal{S}erver_j$

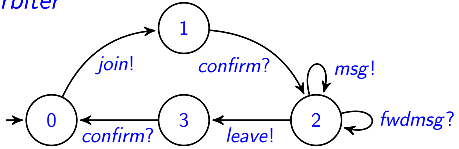
Automata $\mathcal{A}rbiter$, $\mathcal{C}lient_i$ and $\mathcal{S}erver_j$, with $1 \leq i \leq m$ and $1 \leq j \leq n$

Messaging protocol: clients communicate messages (*msg*) to the servers (several for robustness) which, upon approval by the arbiter, broadcast the messages (*fwdmsg*) to all clients in the chat.

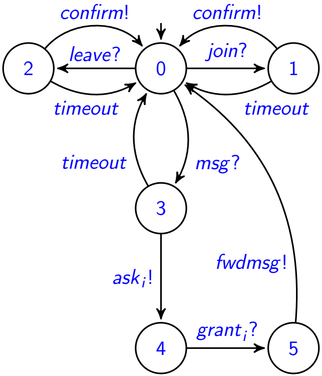
# Teaser: a distributed chat system




(a) $\mathcal{A}rbiter$




(b) $\mathcal{C}lient_i$

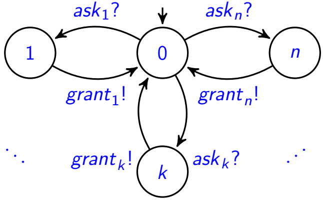



(c) $\mathcal{S}erver_j$

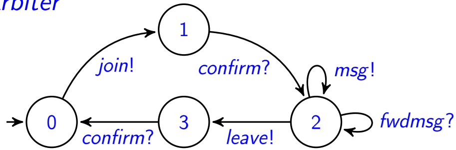
Problems with team $\mathcal{T}_{chat}$ with synchronisation type $([1,1],[1,*])$:

**1.** Server (after communication with client and arbiter) in state 5 not ready to receive *join?* from another client who wants to join! (The server would be allowed to move to state 0 with an internal action, but not upon communicating with other clients [Hennicker, Knapp 2015])

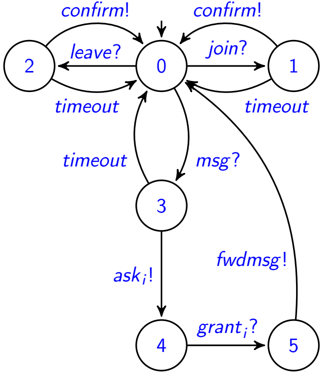
# Teaser: a distributed chat system




(a) $\mathcal{A}rbiter$




(b) $\mathcal{C}lient_i$




(c) $\mathcal{S}erver_j$

Problems with team $\mathcal{T}_{chat}$ with synchronisation type $([1,1],[1,*])$:

**2.** A client may send a message to two servers, who both forward it!

(This could be avoided by defining synchronization types per action, since this would allow to define synchronization type $([1,1],[1,1])$ for $msg$)