Communication Requirements for Team Automata

Maurice H. ter Beek\textsuperscript{1}  Josep Carmona\textsuperscript{2}
Rolf Hennicker\textsuperscript{3}  Jetty Kleijn\textsuperscript{4}

\textsuperscript{1}ISTI–CNR, Pisa, Italy
\textsuperscript{2}Universitat Politècnica de Catalunya, Barcelona, Spain
\textsuperscript{3}Ludwig-Maximilians-Universität München, Germany
\textsuperscript{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).

\[ \begin{array}{c}
0 \quad a! \rightarrow 1 \\
\quad b! \\
\end{array} \]

\[ \begin{array}{c}
0 \quad a? \\
\end{array} \]

*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).

\[ \begin{array}{c}
0 \quad \mathit{verifyPin}! \\
\quad \mathit{notOk}? \\
\end{array} \] \[ \begin{array}{c}
0 \quad \mathit{verifyPin}? \\
\end{array} \]

\[ \begin{array}{c}
0 \quad \mathit{ok}? \\
\quad \mathit{notOk}? \\
\end{array} \] \[ \begin{array}{c}
0 \quad 1 \quad 2 \\
\end{array} \]

*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 \leftrightarrow \text{compatibility notion A}
    
    synchronization type B \leftrightarrow \text{compatibility notion B}
    
    ...
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 $\mathcal{A}_i = (Q_i, q^0_i, \Sigma_i, \rightarrow_i)$ 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 $\Sigma_{\text{inp}}, \Sigma_{\text{out}}, \Sigma_{\text{int}}$ 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 $\mathcal{A}_i = (Q_i, q^0_i, \Sigma_i, \rightarrow_i)$ 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 $i \in I$. 
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 $\mathcal{A}_i = (Q_i, q^0_i, \Sigma_i, \rightarrow_i)$ 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).
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 $S = \{ \mathcal{A}_i \mid i \in I \}$ of component automata $\mathcal{A}_i = (Q_i, q^0_i, \Sigma_i, \rightarrow_i)$ 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

Runner₁

Runner₂

Controller

![Diagram of system transitions](image-url)
Example: system transitions

\[ \text{Runner}_1 \]

\[ \text{Runner}_2 \]

\[ \text{Controller} \]
Example: system transitions

Runner_1

Runner_2

Controller
Example: system transitions

Runner_1

Runner_2

Controller
Example: system transitions

Runner$_1$

Runner$_2$

Controller
Example: system transitions

Runner_1

Runner_2

Controller
Example: system transitions

Runner₁

Runner₂

Controller
Example: system transitions

Runner\textsubscript{1}

Runner\textsubscript{2}

Controller
Example: system transitions

Runner₁

Runner₂

Controller
Example: system transitions

\[ \text{Runner}_1 \quad \text{Runner}_2 \quad \text{Controller} \]
Example: system transitions

Runner_1

Runner_2

Controller
Example: system transitions

Runner_1

Runner_2

Controller
Example: system transitions

Runner₁

Runner₂

Controller
Example: system transitions

Runner₁

Runner₂

Controller
Example: system transitions

Runner₁

Runner₂

Controller
Example: system transitions

Runner₁

Runner₂

Controller
Example: system transitions
Example: system transitions

Runner\(_1\)

Runner\(_2\)

Controller
Example: system transitions

Runner_1

Runner_2

Controller
Example: system transitions

Runner₁

Runner₂

Controller
Example: system transitions

\[\text{Runner}_1\]

\[\text{Runner}_2\]

\[\text{Controller}\]
Example: system transitions

Runner₁

Runner₂

Controller
Team automata

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

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

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

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

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

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

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

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

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

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

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

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

**How to specify synchronization policies?**
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 \( (\text{snd}, \text{rcv}) \) consisting of
- a sending multiplicity \( \text{snd} \) and
- a receiving multiplicity \( \text{rcv} \)

where a multiplicity has one of the following forms:
- \([\text{min}, \text{max}]\) with \( \text{min} \in \mathbb{N}, \text{max} \in \mathbb{N} \cup \{\ast\} \) such that \( \text{min} \leq \text{max} \),
- \( \text{ai} \) “action indispensible”, or
- \( \text{si} \) “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 \( \text{Out}(t) \subseteq S \),
  - the group of components participating in \( t \) by inputting \( a \) is denoted by \( \text{In}(t) \subseteq S \).
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 \( Out(t) \subseteq S \),
  - the group of components participating in \( t \) by inputting \( a \) is denoted by \( In(t) \subseteq S \).

(!) A team automaton \( T \) with synchronization policy \( \delta \) is of type \( (\text{snd}, \text{rcv}) \) if
  (i) for all transitions \( t \in \delta \) labeled with \( a \in \Sigma_{\text{com}} \), \( Out(t) \) fits to \( \text{snd} \) and \( In(t) \) fits to \( \text{rcv} \), and
  (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 \( Out(t) \subseteq S \),
  - the group of components participating in \( t \) by inputting \( a \) is denoted by \( In(t) \subseteq S \).

(!) A team automaton \( \mathcal{T} \) with synchronization policy \( \delta \) is of type \( (\text{snd}, \text{rcv}) \) if
  (i) for all transitions \( t \in \delta \) labeled with \( a \in \Sigma_{\text{com}} \),
      \( Out(t) \) fits to \( \text{snd} \) and \( In(t) \) fits to \( \text{rcv} \), and
  (ii) \( \delta \) contains all such transitions.

- \( Out(t) \) fits to \( \text{snd} \) if
  - if \( \text{snd} = [\text{min}, \text{max}] \) then \( \text{min} \leq |Out(t)| \leq \text{max} \),
  - if \( \text{snd} = a_i \) then \( Out(t) = \{ A_i \in S \mid a \in \Sigma_{i,\text{out}} \} \),
  - if \( \text{snd} = s_i \) 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 \( \text{Out}(t) \subseteq S \),
  - the group of components participating in \( t \) by inputting \( a \) is denoted by \( \text{In}(t) \subseteq S \).

(!) A team automaton \( \mathcal{T} \) with synchronization policy \( \delta \) is of type \((\text{snd}, \text{rcv})\) if
  (i) for all transitions \( t \in \delta \) labeled with \( a \in \Sigma_{\text{com}} \),
    \( \text{Out}(t) \) fits to \( \text{snd} \) and \( \text{In}(t) \) fits to \( \text{rcv} \), and
  (ii) \( \delta \) contains all such transitions.

• \( \text{Out}(t) \) fits to \( \text{snd} \) if
  - if \( \text{snd} = [\text{min}, \text{max}] \) then \( \text{min} \leq |\text{Out}(t)| \leq \text{max} \),
  - if \( \text{snd} = a_i \) then \( \text{Out}(t) = \{ A_i \in S \mid a \in \Sigma_{i,\text{out}} \} \),
  - if \( \text{snd} = s_i \) then “as above” + \( a \) is enabled in \( q_i \).

• \( \text{In}(t) \) fits to \( \text{rcv} \) is defined analogously.
Teams with familiar synchronization types

\[(1, 1), (1, 1)\] binary, peer-to-peer communication
Teams with familiar synchronization types

\[ ([1, 1], [0, 1]) \quad \text{binary, peer-to-peer communication, lossy} \]
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

\[
[[1, 1], [0, 1]] \quad \text{binary, peer-to-peer communication, lossy}
\]

\[
[[0, 1], [0, 1]] \quad \text{non-blocking peer-to-peer (CCS)}
\]

\[
[[1, 1], [0, \ast]] \quad \text{multicast}
\]
Teams with familiar synchronization types

\[ ([1, 1], [0, 1]) \] \quad \text{binary, peer-to-peer communication, lossy}

\[ ([0, 1], [0, 1]) \] \quad \text{non-blocking peer-to-peer (CCS)}

\[ ([1, 1], [1, \ast]) \] \quad \text{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)
- $([1, 1], [1, \ast])$: multicast, strong
- $([1, 1], \text{si})$: 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*

\[ ([1, 1], ai) \] 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)
- $([1, 1], [1, \ast])$: multicast, strong
- $([1, 1], ai)$: broadcast, strong
- $([1, \ast], [0, \ast])$: master-slave communication, ‘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)
- $([1, 1], [1, \ast])$: multicast, strong
- $([1, 1], \text{ai})$: broadcast, strong
- $([1, \ast], [1, \ast])$: master-slave communication, strong, ‘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

\[(1, 1), ai\] broadcast, strong

\[(1, *), [1, *]\] master-slave communication, strong, ‘slaves must obey (at least one)’

\[(ai, ai)\] 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)
- $([1, 1], [1, \ast])$: multicast, strong
- $([1, 1], \text{ai})$: broadcast, strong
- $([1, \ast], [1, \ast])$: master-slave communication, strong, ‘slaves must obey (at least one)’
- $(\text{ai, ai})$: full synchronization (FSP)
- $([1, \ast], [1, 1])$: 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)
- $([1, 1], [1, \ast])$: multicast, strong
- $([1, 1], ai)$: broadcast, strong
- $([1, \ast], [1, \ast])$: master-slave communication, strong, ‘slaves must obey (at least one)’
- $(ai, ai)$: full synchronization (FSP)
- $([1, \ast], [1, 1])$: gathering
- $([0, \ast], [0, \ast])$: 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 \([1, 1], ai\):
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 $([1, 1], ai)$:

Runner_1

Runner_2

Controller
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 $([1, 1], ai)$:
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 \([1, 1], ai\):
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 \([1, 1, ai]\):

Runner_1

Runner_2

Controller
**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 \([1, 1], ai\):

![Diagram](image)
Receptiveness requirements: definition

Let \( S = \{ A_i \mid i \in I \} \) be a component system and \( T \) a team automaton with synchronization type \((\text{snd}, \text{rcv})\).

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

If there is a group of components \( G = \{ A_j \mid j \in J \subseteq I \} \) having \( a \) as an output action such that

- \( a \) is enabled in each local state \( q_j \) with \( j \in J \) and
- \( G \) fits to \( \text{snd} (\!\!) \),

then there exists a group of components \( H = \{ A_k \mid k \in K \subseteq I \} \) having \( a \) as an input action such that

- \( a \) is enabled in each local state \( q_k \) with \( k \in K \) and
- \( H \) fits to \( \text{rcv} (\!\!) \).
Receptiveness requirements: definition

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

Receptiveness requirement:

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

If there is a group of components $G = \{ A_j \mid j \in J \subseteq I \}$ having $a$ as an output action such that

- $a$ is enabled in each local state $q_j$ with $j \in J$ and
- $G$ fits to $\text{snd} (!)$,

then there exists a group of components $H = \{ A_k \mid k \in K \subseteq I \}$ having $a$ as an input action such that

- $a$ is enabled in each local state $q_k$ with $k \in K$ and
- $H$ fits to $\text{rcv} (!)$.

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 \(([1, 1], a_i)\):

![Diagram](image-url)
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 $([1, 1], a_i)$:
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 $([1, 1], a_i)$:

Runner₁

Runner₂

Controller'}
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 $([1, 1], a_i)$:

Runner$_1$

Runner$'_2$

Controller$'$
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 $([1, 1], a_i)$:
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 \([1, 1], a_i\):
Responsiveness requirements: definition

Let \( S = \{ A_i \mid i \in I \} \) be a component system and \( T \) a team automaton with synchronization type \((\text{snd}, \text{rcv})\).

**Responsiveness requirement:**

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

If there is a group of components \( G = \{ A_j \mid j \in J \subseteq I \} \) 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
- \( G \) fits to \( \text{rcv} (!) \),

then there exists a group of components \( H = \{ A_k \mid k \in K \subseteq I \} \) 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
- \( H \) fits to \( \text{snd} (!) \).
Responsiveness requirements: definition

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

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

If there is a group of components $G = \{ A_j \mid j \in J \subseteq I \}$ 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
- $G$ fits to $\text{rcv} (!)$,

then there exists a group of components $H = \{ A_k \mid k \in K \subseteq I \}$ 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
- $H$ fits to $\text{snd} (!)$.

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], ai) instead of ([1, 1], ai) in the system:

Runner_1

Runner_2

Controller

start? → 1 → run_1 → 2 → finish!

start? → 1 → run_2 → 2 → finish!

start! → 2 → finish? → 1
Example: non-responsive system

Use synchronization type \((1, 2, a)\) instead of \((1, 1, a)\) in the system:
Example: non-responsive system

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

Runner\(_1\)

Runner\(_2\)

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

*Runner_1*

*Runner_2*

*Controller*
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

Automata \textit{Arbiter}, \textit{Client}_i, \textit{Server}_j, with \(1 \leq i \leq m\) and \(1 \leq j \leq n\)

Messaging protocol: clients communicate messages (\textit{msg}) to the servers (several for robustness) which, upon approval by the arbiter, broadcast the messages (\textit{fwdmsg}) to all clients in the chat.
Teaser: a distributed chat system

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

1. Server (after communication with client and arbiter) in state 5 not ready to receive $\text{join?}$ 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

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

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 $msg$)