Topological Self-Stabilization with Name-Passing Process Calculi

Christina Rickmann
Christoph Wagner, Uwe Nestmann, Stefan Schmid

"'We call the system “self-stabilizing” if and only if, regardless of the initial state [...] , the system is guaranteed to find itself in a legitimate state after a finite number of moves.’’"
- Dijkstra 1974
Definition

Self-stabilization

A system is self-stabilizing if and only if (provided no fault occurs)

Convergence: started in any arbitrary state, the system reaches a desired state after a finite number of steps and
Definition

A system is self-stabilizing if and only if (provided no fault occurs)

**Convergence:** started in any arbitrary state, the system reaches a desired state after a finite number of steps and

**Closure:** if the system is in a desired state, it remains in a desired state.
Characteristics

Self-stabilization:

• specialization of nonmasking fault tolerance
• tolerate arbitrary transient faults
• no initialization
• no fault detection
• must not terminate
• no local knowledge whether stabilized
• adapt to dynamic changes
Linearization

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 \\
\end{array}
\]

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 \\
\end{array}
\]

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 \\
\end{array}
\]
Original Shared-Memory-Algorithm

Gall et al. (2014):
for every node $u$, there are the following rules for every pair of neighbors $v$ and $w$

linearize right($v, w$) :
\[(v, w \in u.R \wedge u < v < w)\]
\[\rightarrow e(u, w) := 0, \ e(v, w) := 1\]

linearize left($v, w$) :
\[(v, w \in u.L \wedge w < v < u)\]
\[\rightarrow e(u, w) := 0, \ e(v, w) := 1\]
Asynchronous Message-Passing-Algorithm

asynchronous messages: non blocking

0  1  2  3

⇒ localized

π-calculus

Rickmann

Self-Stabilization
Asynchronous Message-Passing-Algorithm

asynchronous messages: non blocking

graph diagram
Asynchronous Message-Passing-Algorithm

asynchronous messages: non blocking

\[0 \rightarrow 1 \rightarrow 2 \rightarrow 3\]

\[\downarrow\]

\[0 \rightarrow 1 \rightarrow 2 \rightarrow 3\]

asynchronous message passing + changing communication structure
+ distributed
⇒ localized \(\pi\)-calculus
Asynchronous Message-Passing-Algorithm

\[
\text{Alg} (p, nb) = (\nu nb_p) \left( \text{nb}_p\langle nb \rangle | \text{Alg}_{\text{rec}} (p) | \text{Alg}_{\text{match}} (p) \right)
\]

\[
\text{Alg}' (p, nb, x) = (\nu nb_p) \left( \text{nb}_p\langle nb \rangle | \text{Alg}_{\text{add}} (p, x) | \text{Alg}_{\text{match}} (p) \right)
\]

\[
\text{Alg}_{\text{rec}} (p) = p(x) \cdot \text{Alg}_{\text{add}} (p, x)
\]

\[
\text{Alg}_{\text{add}} (p, x) = nb_p(y) \cdot \left( \text{nb}_p\langle y \cup \{x\} \rangle | \text{Alg}_{\text{rec}} (p) \right)
\]

\[
\text{Alg}_{\text{match}} (p) = nb_p(y) \cdot (\text{let } x = \text{select} (\text{findLin} (p, y)) \text{ in }
\]

\[
\text{if } x = \perp \text{ then } \prod_{j \in y} \bar{j}\langle p \rangle | \text{nb}_p\langle y \rangle
\]

\[
\text{else if } x = (j, k) \text{ then }
\]

\[
\text{if } j < k \land k < p \text{ then } \bar{j}\langle k \rangle | \text{nb}_p\langle y \setminus \{j\} \rangle
\]

\[
\text{else if } j < k \land p < j \text{ then } \bar{k}\langle j \rangle | \text{nb}_p\langle y \setminus \{k\} \rangle
\]

\[
\cdots
\]

\[
| \text{Alg}_{\text{match}} (p) \rangle
\]
Asynchronous Message-Passing-Algorithm

0 1 2 3
Asynchronous Message-Passing-Algorithm
Asynchronous Message-Passing-Algorithm
Asynchronous Message-Passing-Algorithm
Asynchronous Message-Passing-Algorithm
Asynchronous Message-Passing-Algorithm
Asynchronous Message-Passing-Algorithm
Asynchronous Message-Passing-Algorithm
Asynchronous Message-Passing-Algorithm
Asynchronous Message-Passing-Algorithm
Asynchronous Message-Passing-Algorithm
Asynchronous Message-Passing-Algorithm

keep-alive-messages

0 \to 1 \to 2 \to 3
Asynchronous Message-Passing-Algorithm

keep-alive-messages

0 → 1
2⟨1⟩
2 → 3
2⟨3⟩
3 → 2
3 → 0
Let \( Z \) be the set of all states and \( L \subseteq Z \) the set of legal/desired states.

**Convergence**

Starting from a state in \( Z \setminus L \), after a limited number of steps a state in \( L \) is reached.

\[ \Rightarrow \] construct a function \( t : Z \to \mathbb{N} \) (potential function) that decreases with every step and for every state \( x \in L \) holds \( t(x) = 0 \).

**Closure**

Starting in a state in \( L \), each following state again is in \( L \).

\[ \Rightarrow \] usually through an invariant.
Closure

correct configuration is unique

\[
\begin{align*}
0 & \rightarrow 1 \\
1 & \rightarrow 2 \\
2 & \rightarrow 3 \\
3 & \rightarrow 0
\end{align*}
\]
Closure

correct configuration is unique (up to)
Closure

correct configuration is unique (up to)

![Diagram of a network with nodes labeled 0 to 3 and edges indicating connections between configurations. The diagram illustrates the proof closure with arrows showing the transitions between configurations.](image-url)
correct configuration is unique (up to)

\[ \langle 0 \rangle \to \langle 1 \rangle \to \langle 0 \rangle \to \langle 1 \rangle \to \langle 3 \rangle \]

Idea closure proof:
after every step again a correct configuration

- no linearization steps
- actions involve only desired neighbors

⇒ topology remains unchanged
Convergence

Problem: *keep-alive-messages*
Convergence

Problem: *keep-alive-messages*

Starting from any arbitrary state, there is always a way to reach a desired state after a limited number of steps.

Weak Convergence

Strong convergence for restricted cases
Weak convergence in general
(Strong) convergence in case:
- no undesired connections anymore

\[
\begin{align*}
\langle 0 \rangle_1 & \quad \langle 1 \rangle_0 \\
1 & \quad 0
\end{align*}
\]

\[
\begin{align*}
\langle 1 \rangle_0 & \quad \langle 0 \rangle_1 \\
2 & \quad 3
\end{align*}
\]

\[
\begin{align*}
\langle 1 \rangle_0 & \quad \langle 2 \rangle_1 \\
2 & \quad 3
\end{align*}
\]

\[
\begin{align*}
\langle 1 \rangle_0 & \quad \langle 2 \rangle_1 \\
3 & \quad 2
\end{align*}
\]
(Strong) convergence in case:
  • no undesired connections anymore

everyone sending *keep-alive*-messages, all received and processed eventually
Convergence

(Strong) convergence in case:
- no undesired connections anymore
- all desired connections established

\[
\begin{align*}
\overline{0}\langle 1 \rangle &\quad \overline{1}\langle 0 \rangle &\quad \overline{2}\langle 1 \rangle &\quad \overline{2}\langle 3 \rangle \\
1 &\quad 1 &\quad 2 &\quad 2 \\
0 &\quad 1 &\quad 2 &\quad 3
\end{align*}
\]
(Strong) convergence in case:
- no undesired connections anymore
- all desired connections established

Eventually all desired edges
no unnecessary *keep-alive*-messages
convergence by potential function
Weak convergence in general:

- subset of executions without unnecessary *keep-alive*-messages
  non-empty for each initial configuration
  converges strongly
Our work

Conclusion

Our contributions:

• redesigning algorithm shared memory → asynchronous message-passing
• proving:
  • closure,
  • weak convergence in general,
  • strong convergence for restricted cases
• discussing strong convergence

Future work:

• proving strong convergence

Thanks! Questions?
Fault Tolerance

- Faults occur, we have to deal with them!
- masking fault tolerance:
  - aim: avoid system failure if possible
  - fault model
  - describes all faults that can be tolerated
  - never takes all possible faults into account
  - other faults may lead to system failure
  - needs redundancy in space or time
- nonmasking fault tolerance:
  - system may fail partly or temporarily
  - better than complete and/or permanent failure
Extended Localized Pi

**Data Values** $\mathbf{V}$

$v ::= \bot \mid 0 \mid 1 \mid c \mid (v,v) \mid$

$\{v,\ldots,v\} \mid \{\{v,\ldots,v\}\}$, with $c \in \mathbf{A}$

**Variable Pattern**

$\mathbf{X} ::= x \mid (X,X)$, with $x \in \mathbf{A}$

**Expressions**

$e ::= v \mid X \mid (e,e) \mid f(e)$, with $f \in \mathbf{A}$

**Processes** $\mathbf{P}$

$P ::= 0 \mid P \mid P \mid c(X).P \mid \bar{c}(v) \mid (\nu c)P \mid$

$\text{if } e \text{ then } P \text{ else } P \mid \text{let } X = e \text{ in } P \mid K(e)$

**Process Equations**

$D = \{K_j(X) = P_j\}_{j \in J}$ a finite set of process definitions

where in $c(X).P$ variable $x$ as part of $X$ may not occur free in $P$ in input position.
Structural Congruence

\[ P \equiv Q \text{ if } P \equiv_\alpha Q \quad P \mid 0 \equiv P \quad P \mid Q \equiv Q \mid P \]

\[ P \mid (Q \mid R) \equiv (P \mid Q) \mid R \quad (\nu n) 0 \equiv 0 \]

\[ P \mid (\nu n) Q \equiv (\nu n) (P \mid Q), \text{ if } n \notin \text{fn}(P) \]

\[ (\nu n) (\nu m) P \equiv (\nu m) (\nu n) P \]

\[ \text{if } e \text{ then } P \text{ else } Q \equiv P, \text{ if } \llbracket e \rrbracket = 1 \]

\[ \text{if } e \text{ then } P \text{ else } Q \equiv Q, \text{ if } \llbracket e \rrbracket = 0 \]

\[ \text{let } X = e \text{ in } P \equiv \{[e]/X\}P \]

\[ K(e) \equiv \{[e]/X\}P, \text{ if } (K(X) = P) \in D \]
Reduction Semantics

\[
\text{comm: } \quad \frac{c(X).P \mid \overline{c}\langle \nu \rangle \rightarrow \nu / x P}{P \rightarrow P'}
\]

\[
\text{res: } \quad \frac{P \rightarrow P'}{(\nu c)P \rightarrow (\nu c)P'}
\]

\[
\text{par: } \quad \frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q}
\]

\[
\text{struct: } \quad \frac{P \equiv Q \quad Q \rightarrow Q' \quad Q \equiv Q'}{P \rightarrow P'}
\]
System Assumptions

Process Ids
Every process has a unique constant id and every value in the system can be interpreted as the id of an existing process.

No Message Loss
Every message is received after a finite but arbitrary number of steps.

Fairness
Every continuously enabled subprocess will eventually (after an arbitrary but finite number of steps) execute a step.

Weakly Connected
The topology with messages is initially weakly connected.
Algorithm I

\[
\text{Alg}(p, \text{initNb}) = (\nu nb_p) \left( \overline{nb_p} \langle \text{initNb} \rangle \mid \right. \\
\text{Alg}_{\text{rec}}(p) \mid \text{Alg}_{\text{match}}(p) \left. \right)
\]

\[
\text{Alg}'(p, \text{initNb}, x) = (\nu nb_p) \left( \overline{nb_p} \langle \text{initNb} \rangle \mid \right. \\
\text{Alg}_{\text{add}}(p, x) \mid \text{Alg}_{\text{match}}(p) \left. \right)
\]

\[
\text{Alg}_{\text{rec}}(p) = p(x) . \text{Alg}_{\text{add}}(p, x)
\]

\[
\text{Alg}_{\text{add}}(p, x) = nb_p(y) . \left( \overline{nb_p} \langle y \cup \{x\} \rangle \mid \text{Alg}_{\text{rec}}(p) \right)
\]
Algorithm II

\[
Alg_{match}(p) = nb_p(y). (\text{let } x = select(findLin(p, y)) \text{ in }
\]

\[
\text{if } x = \bot \text{ then } \prod_{j \in y} j\langle p \rangle | \overline{nb}_p \langle y \rangle
\]

\[
\text{else if } x = (j, k) \text{ then }
\]

\[
\text{if } j < k \land k < p \text{ then } j\langle k \rangle | \overline{nb}_p \langle y \setminus \{j\} \rangle
\]

\[
\text{else if } j < k \land p < j \text{ then } k\langle j \rangle | \overline{nb}_p \langle y \setminus \{k\} \rangle
\]

\[
\text{else } \overline{nb}_p \langle y \rangle
\]

\[
\text{else } \overline{nb}_p \langle y \rangle
\]

| Alg_{match}(p) |
Algorithm III

LeftN : \( \mathcal{P} \times 2^\mathcal{P} \rightarrow 2^\mathcal{P} \) calculates the *left neighborhood* of a process and corresponding RightN : \( \mathcal{P} \times 2^\mathcal{P} \rightarrow 2^\mathcal{P} \) the *right neighborhood* of a process.

\[
\text{LeftN}(p, y) = \{ q \in \mathcal{P} | q \in y \land q < p \}
\]

\[
\text{RightN}(p, y) = \{ q \in \mathcal{P} | q \in y \land q > p \}
\]
Algorithm III

\[ \text{LeftN} : \mathcal{P} \times 2^\mathcal{P} \rightarrow 2^\mathcal{P} \] calculates the left neighborhood of a process and corresponding \[ \text{RightN} : \mathcal{P} \times 2^\mathcal{P} \rightarrow 2^\mathcal{P} \] the right neighborhood of a process.

\[
\text{LeftN}(p, y) = \{ q \in \mathcal{P} | q \in y \land q < p \}
\]

\[
\text{RightN}(p, y) = \{ q \in \mathcal{P} | q \in y \land q > p \}
\]

The function \( \text{findLin} : \mathcal{P} \times 2^\mathcal{P} \times \mathcal{P} \rightarrow 2^\mathcal{P} \times \mathcal{P} \) calculates all possible linearization steps in the neighborhood of a process.

\[
\text{findLin}(p, y) = \{(q, r) | q, r \in y \land q < r \land (q, r \in \text{LeftN}(p, y) \lor \text{RightN}(p, y))\}
\]
Algorithm III

LeftN : $\mathcal{P} \times 2^\mathcal{P} \rightarrow 2^\mathcal{P}$ calculates the left neighborhood of a process and corresponding RightN : $\mathcal{P} \times 2^\mathcal{P} \rightarrow 2^\mathcal{P}$ the right neighborhood of a process.

$$LeftN(p, y) = \{ q \in \mathcal{P} | q \in y \land q < p \}$$
$$RightN(p, y) = \{ q \in \mathcal{P} | q \in y \land q > p \}$$

The function findLin : $\mathcal{P} \times 2^\mathcal{P} \times \mathcal{P} \rightarrow 2^\mathcal{P} \times \mathcal{P}$ calculates all possible linearization steps in the neighborhood of a process.

$$findLin(p, y) = \{(q, r) | q, r \in y \land q < r \land (q, r \in LeftN(p, y) \lor RightN(p, y))\}$$

The function select : $2^\mathcal{P} \times \mathcal{P} \rightarrow (\mathcal{P} \times \mathcal{P})$ returns one of these linearization steps:

$$select(y) = \begin{cases} \bot & \text{if } y = \emptyset \\ \epsilon x. x \in y & \text{if } y \neq \emptyset \end{cases}$$
Algorithm Configuration (Standardform)

Configuration in Standardform

\[ \text{Alg}_{\text{all}}(P, P', nb, Msgs, add) = \prod_{j \in P} \text{Alg}(j, nb(j)) \mid \prod_{j \in P'} \text{Alg}'(j, nb(j), add(j)) \mid \prod_{(j,k) \in \text{Msgs}} j\langle k \rangle \]

\( \mathcal{P} \) be the set of unique identifiers,
\( P, P' \subseteq \mathcal{P} \) with \( P \cup P' = \mathcal{P} \) and \( P \cap P' = \emptyset \),
\( \text{nb} : \mathcal{P} \to 2^{\mathcal{P}} \) a neighborhood-function
\( \text{Msgs} \in \mathbb{N}^{\mathcal{P} \times \mathcal{P}} \) a multiset of the messages in transit and
\( \text{add} : \mathcal{P} \to \mathcal{P} \) a partial function with \( \forall p \in P'. \exists q \in \mathcal{P}. (p, q) \in \text{add} \) and
\( \forall p \in P. \forall q \in \mathcal{P}. (p, q) \notin \text{add} \) that describes the adding in progress
Topologies

Network Topology Graph

Let \( A \equiv \text{Alg}_{\text{all}}(P, P', nb, \text{Msgs}, add) \) be an arbitrary configuration. Then the (directed) network topology graph \( T(A) = (V, E) \) is defined as follows:

\[
V = P \cup P' = \mathcal{P} \quad \text{and} \quad E = \{(p, q) | p, q \in V \land q \in nb(p)\}
\]

Network Topology Graph with Messages

Let \( A \equiv \text{Alg}_{\text{all}}(P, P', nb, \text{Msgs}, add) \) be an arbitrary configuration. Then the (directed) network topology graph with messages \( T^M(A) = (V, E) \) is defined as follows:

\[
V = P \cup P' = \mathcal{P} \quad \text{and} \quad E = \{(p, q) | p, q \in V \land (q \in nb(p) \lor (p, q) \in \text{Msgs} \lor \text{add}(p) = q)\}
\]
Undirected Topologies

Undirected Topology Graph

Let \( A \equiv \text{Alg}_{\text{all}}(P, P', \text{nb}, \text{Msgs}, \text{add}) \) be an arbitrary configuration. Then the undirected network topology graph \( U(A) = (V, E) \) is defined as follows:

\[
V = P \cup P' = \mathcal{P} \quad \text{and} \quad E = \{\{p, q\} | p, q \in V \land q \in \text{nb}(p)\}
\]

Undirected Topology Graph with Messages

Let \( A \equiv \text{Alg}_{\text{all}}(P, P', \text{nb}, \text{Msgs}, \text{add}) \) be an arbitrary configuration. Then the undirected network topology graph with messages \( U^M(A) = (V, E) \) is defined as follows:

\[
V = P \cup P' = \mathcal{P} \quad \text{and} \quad E = \{\{p, q\} | p, q \in V \land (q \in \text{nb}(p) \lor (p, q) \in \text{Msgs} \lor \text{add}(p) = q)\}
\]
Potential Functions I

Potential Function $\Psi_p$ for Processes

Let $p \in \mathcal{P}$ be an arbitrary process and $A$ be a configuration. Let $Rec : (T \times \mathcal{P}) \rightarrow \mathbb{N}^{\mathcal{P}}$ with

$$Rec(A, p) = \{q \in \mathcal{P} | (p, q) \in Msgs_A \land q \notin \{\text{succ}(p), \text{pred}(p)\}\}$$

multiset of all process ids to $p$ still in transit and not a desired neighbor and $adding : (T \times \mathcal{P}) \rightarrow \mathbb{N}$ with

$$adding(A, p) = \begin{cases} \text{dist}(p, q), & \text{if } add_A(p) = q \in \mathcal{P} \land q \notin \{\text{succ}(p), \text{pred}(p)\} \\ 0, & \text{otherwise} \end{cases}$$

The potential function $\Psi_p : (T \times \mathcal{P}) \rightarrow \mathbb{N}$ sums up the distances of all outgoing connections of the process $p$ while ignoring desired connections:

$$\Psi_p(A, p) = \sum_{q \in (nb_A(p) \setminus \{\text{succ}(p), \text{pred}(p)\})} \text{dist}(p, q) + \sum_{q \in \text{Rec}(A, p)} \text{dist}(p, q) + adding(A, p)$$
Potential Function $\Psi$ for Configurations

Let $A$ be a configuration. The potential function for configurations $\Psi : \mathcal{T} \rightarrow \mathbb{N}$ sums up the distances between all the connections of processes while ignoring desired connections:

$$\Psi(A) = \sum_{p \in \mathcal{P}} \Psi_p(A, p)$$