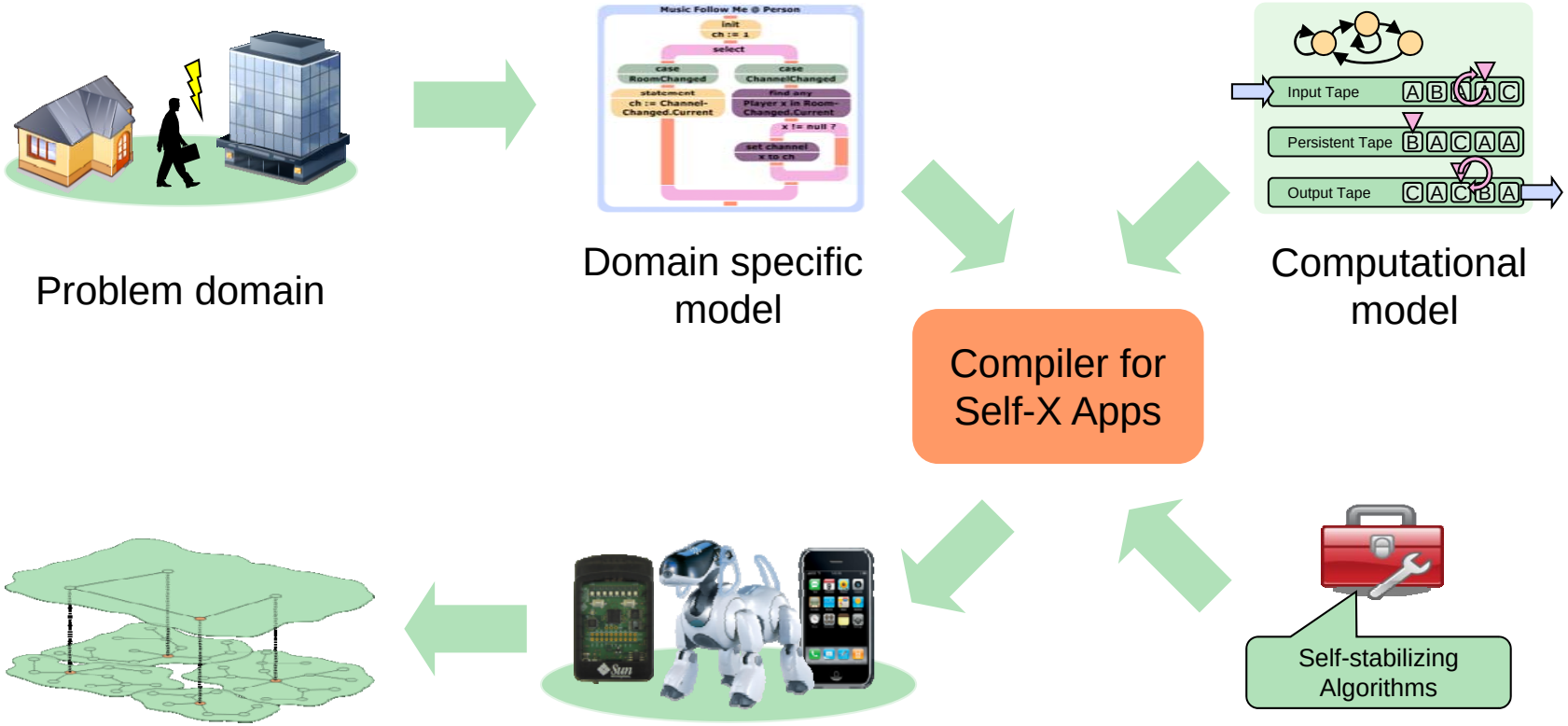
Model-driven Development of Self-organizing Control Applications (MODOC)

Prof. Dr.-Ing. Torben Weis
Dr. Arno Wacker
Dipl.-Inform. Sebastian Holzapfel
Universität Duisburg-Essen

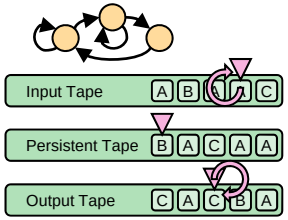
Prof. Dr. Hans-Ulrich Heiß
Dr.-Ing. Gero Mühl
Dr.-Ing. Jan Richling
Dipl.-Inform. Helge Parzyjegla
Dipl.-Inform. Jan Schönherr
Technische Universität Berlin

Software Development Methodology for OC



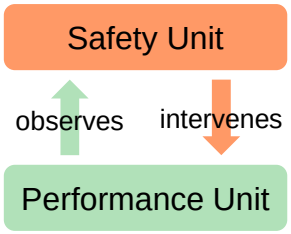
Overview

- Project results (2nd phase)
  - Self-stabilizing Turing machine
  - Clustered publish/subscribe
  - Adaptive routing



Self-stabilizing Turing machine

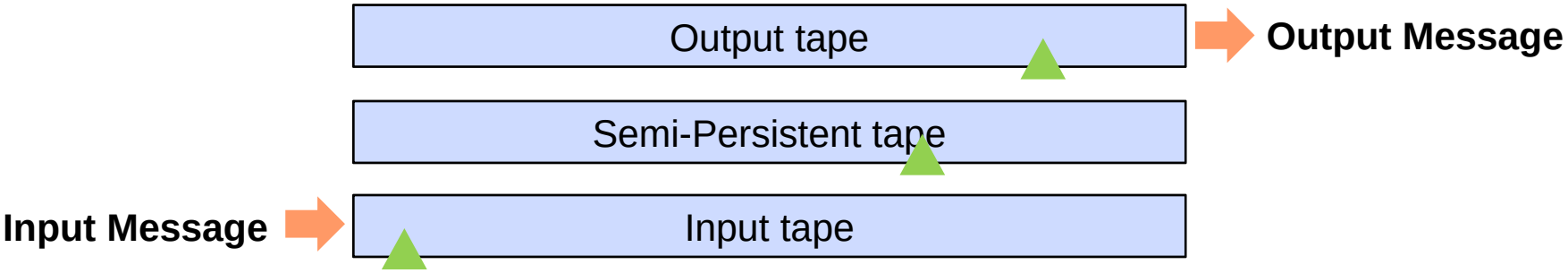
- Project goals (3rd phase)
  - Safety constraints
  - Roundtrip modeling
  - Self-stabilizing virtual machine



Safe Applications via Analytic Redundancy

- Conclusions

Computational Model



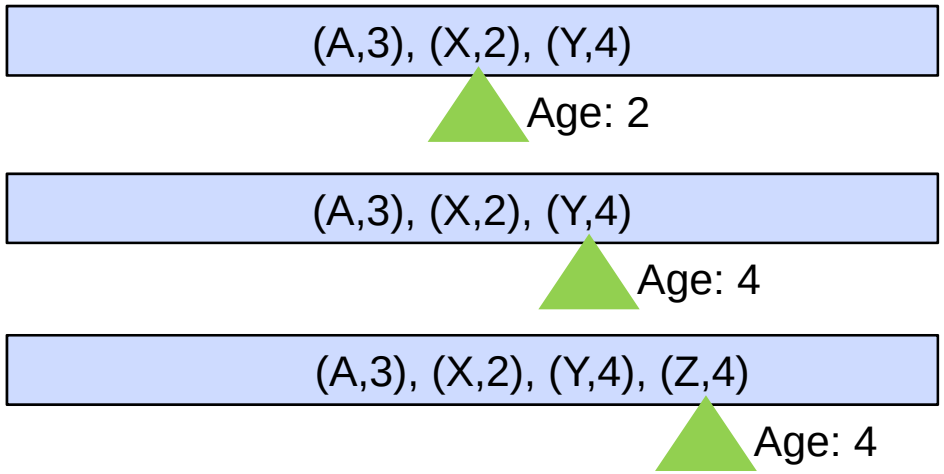
3-Tape Turing Machine with semi-persistent tape

1. Sensor input is written on the input tape
2. The TM writes on the persistent and output tape
3. The output tape data is sent to actuators
4. Input & Output tape are erased
5. Repeat

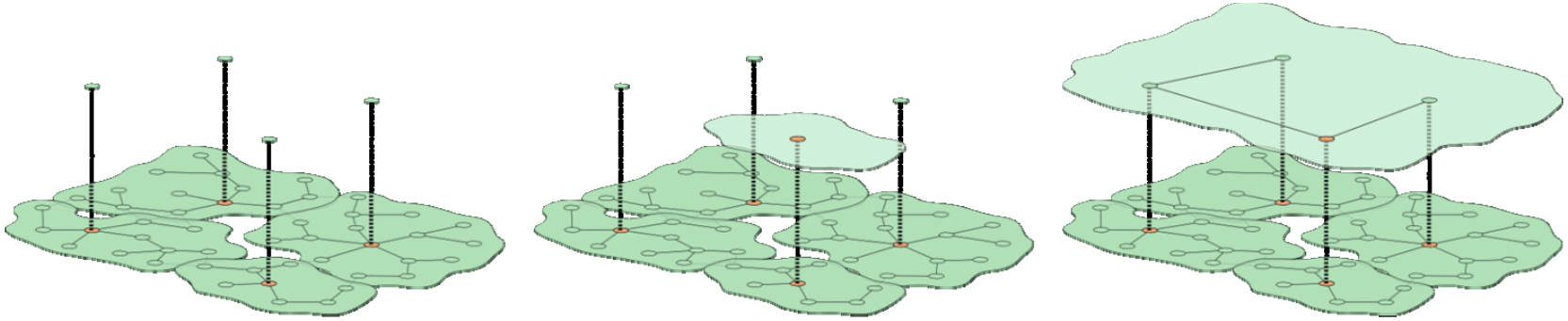
T. Weis, A. Wacker: “Self-stabilizing Automata” in proceedings of BICC 2008

Computational Model

- Self-stabilizing TM
  - Idea: Make the TM forget about old symbols
  - Do not allow to derive fresh symbols from old symbols
- Technical approach
  - All tape symbols have an age, TM heads have an age
  - If a TM head reads old symbols, it becomes old, too
  - Old TM heads can only write old symbols



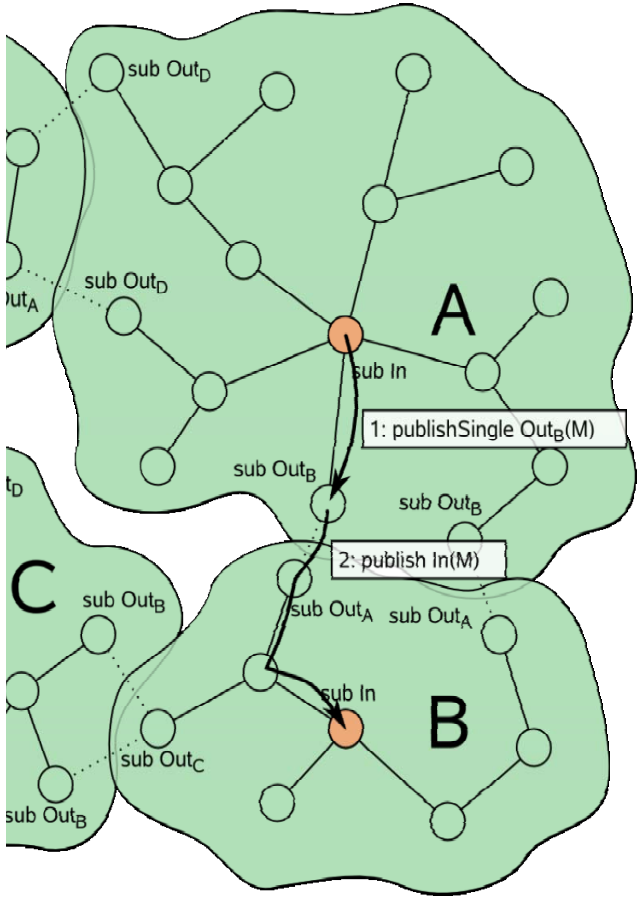
Clustered Publish/Subscribe Algorithm



- Basic clustering mechanism forms k-hop clusters with spanning tree as byproduct
- Publish/subscribe event dissemination follows tree edges
- Neighboring clusters communicate via gateway nodes using local publish/subscribe infrastructure
- Recursive application of clustering algorithm forms higher level clusters until one cluster covers all nodes

Benefits

- Clustering
  - Enables scalability for large scenarios
  - Avoids large reconfigurations
  - Redundant routes between clusters
  - Realizes fault-containment
- Publish/Subscribe as a routing layer
  - Abstracts from concrete routes
  - Allows transparent switching between redundant routes
  - Minimizes incorrect information and message loss during faults



Robustness

- Self-stabilization
  - Clustering and publish/subscribe are self-stabilizing
  - (Recursively) layering self-stabilizing algorithms results in a new self-stabilizing algorithm → fair composition
  - Worst case overall stabilization time is determined by the sum of the stabilization times of the individual layers
- Fault containment
  - Clusters limit effects of different network faults
  - New links are added to the topology, but do not alter it
  - Reconfigurations are kept within a cluster and level if possible
    - Only if the last connection between two clusters fails or a cluster is separated into at least two disjoint parts the parent level is affected
    - Clusters of the same level are never affected, keeping them and their children fully functional

Hybrid Routing Schemes

- Enable adaptation to changing notification traffic by changing the applied publish/subscribe routing algorithm
- Each algorithm has inherent advantages and drawbacks

Flooding-based algorithm

- No subscription overhead
- Forwarding overhead for unwanted notifications
- Filtering based on local interests only → usually smaller filter tables

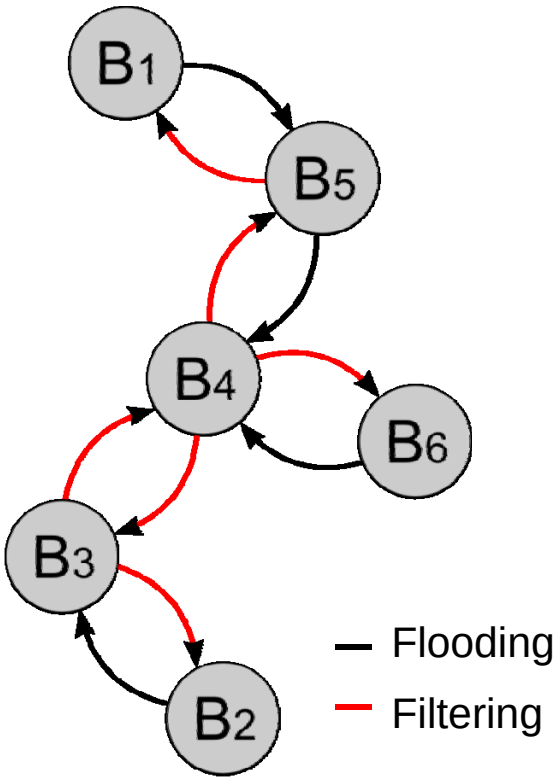
Filtering-based algorithm

- Subscription overhead (forwarding and processing)
- No unwanted notifications
- Filtering based on local and remote interests → usually larger filter tables

- Different routing algorithms can coexist in the same network → hybrid routing configuration
- Switching routing algorithms can reduce network traffic → optimize network performance

Adaptive Routing

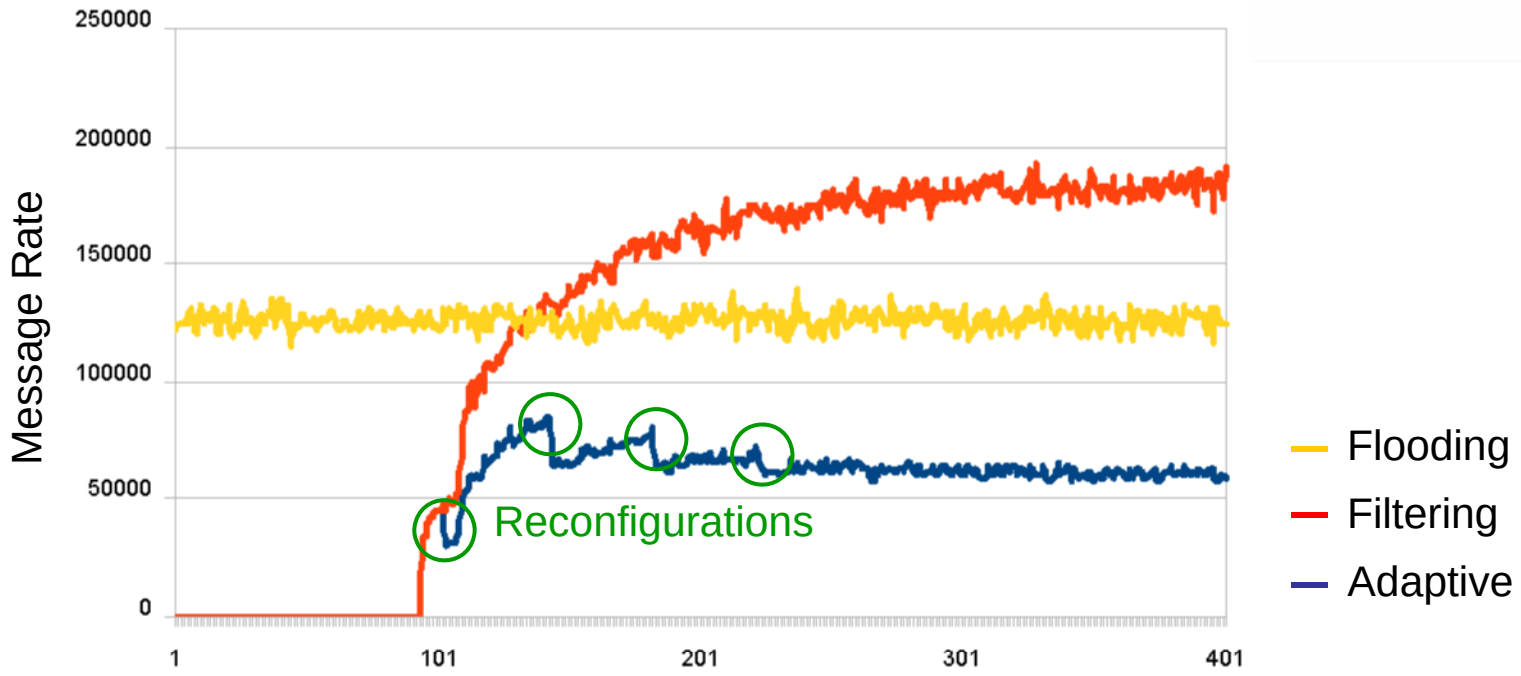
- Adaptivity
  - Fine grained configuration on link basis
  - Links are directed and asymmetric
- Optimization criterion
  - Based on local knowledge only
  - Evaluated by each node individually
- $c_{ij}\omega^{N}_{Flood} > c_{ij}\omega^{N}_{Filter} + c_{ji}\omega^{S}_{Filter}$
  - Switch Flooding → Filtering
- $c_{ij}\omega^{N}_{Flood} < c_{ij}\omega^{N}_{Filter} + c_{ji}\omega^{S}_{Filter}$
  - Switch Filtering → Flooding



c_{ij} : forwarding overhead of a message $B_i \rightarrow B_j$

$\omega^{N/S}_{Algo}$: notification / subscription rate for applied algorithm

Evaluation



- Discrete event simulation with over 100 brokering nodes
- Abruptly increased sub-scription rate at time 100
- Hybrid routing configuration more efficient than pure Flooding or Filtering
- No complex interaction scheme → marginal overhead

MODOC Phase III

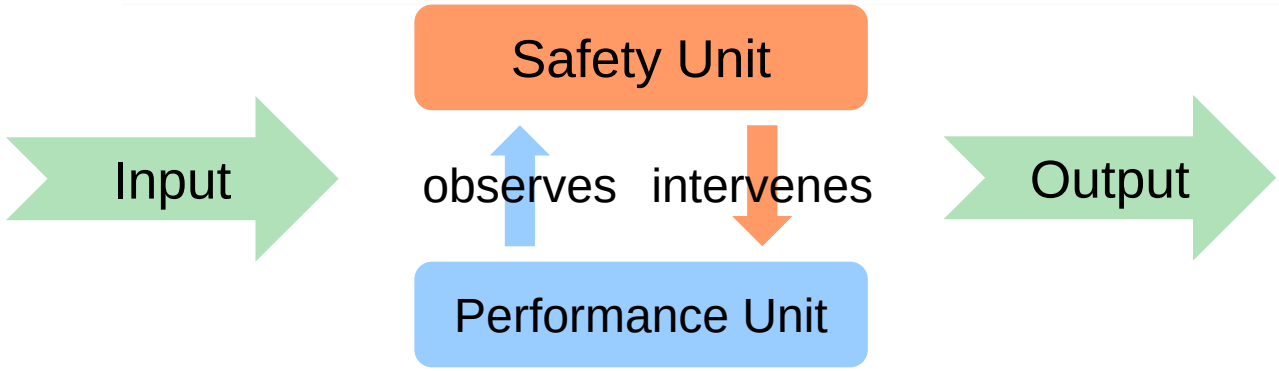
- Safety constraints
  - Enables developers to create safe OC applications
  - Guarantees that generated application code is safe
- Round-trip modeling
  - Provides a comprehensive tool chain for creating OC applications
  - Supports modeling, code generation, debugging, and visualization
- Self-stabilizing Virtual Machine
  - Provides an efficient execution platform
  - Offers data-flow oriented programming abstractions
- Event composition
  - Develops an event composition algebra
  - Optimizes network performance by event aggregation

Safety Considerations

- Safety
  - Freedom from unacceptable risk of physical injury, illness, damage to or loss of equipment or property, or environmental harm
  - BUT: Actuators within MODOC may potentially cause damage!
- Self-stabilization
  - Guarantees that system recovers from transient faults
  - Reaching a valid state needs a fixed time → stabilization time
  - BUT: What happens during stabilization?
- Fault containment
  - Bounds effects of faults within affected components
    → remaining components proceed working correctly
  - BUT: What happens inside affected components?

Self-stabilization and fault containment are not sufficient!

Analytic Redundancy



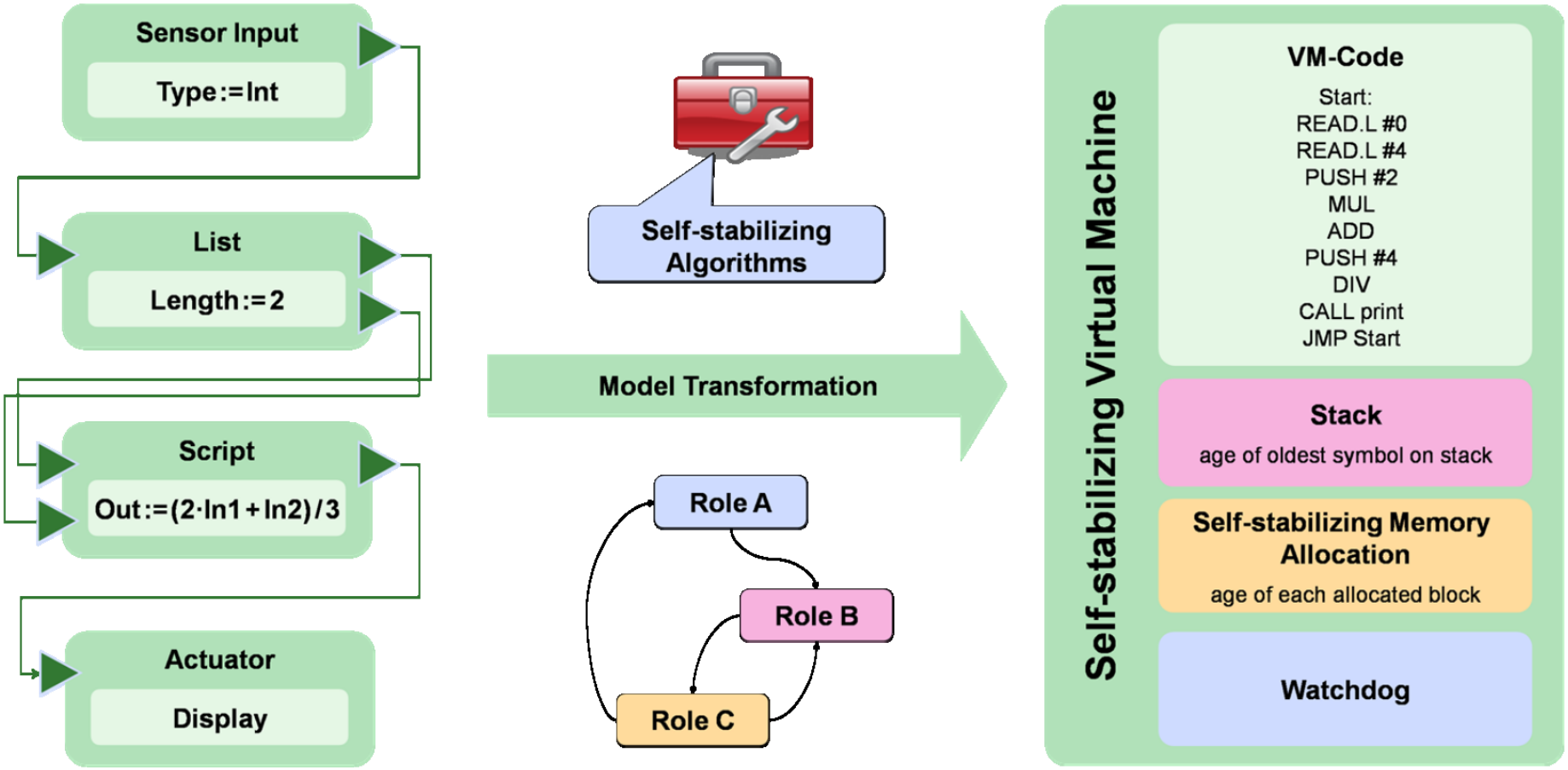
- Fault-tolerant approach between fail-stop systems and complete replication (e.g., TMR)
- Performance unit is optimized for best functionality
- Safety unit delivers minimal functionality needed to fulfill safety requirements → intervenes in critical states
- Instance of observer/controller pattern

Safety @ MODOC

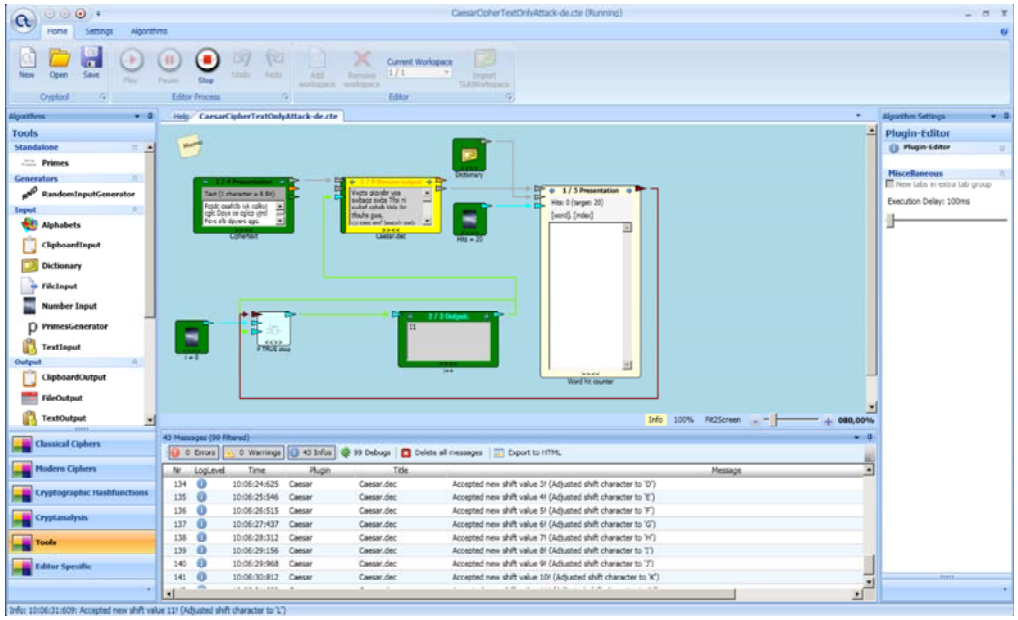
- Modeling safe OC applications
  - Allows the annotation of safety constraints that always hold
  - Requires the definition of safe default values
  - Provides the possibility to specify Event/Condition/Action (ECA) rules to reach a safe state → safety rules

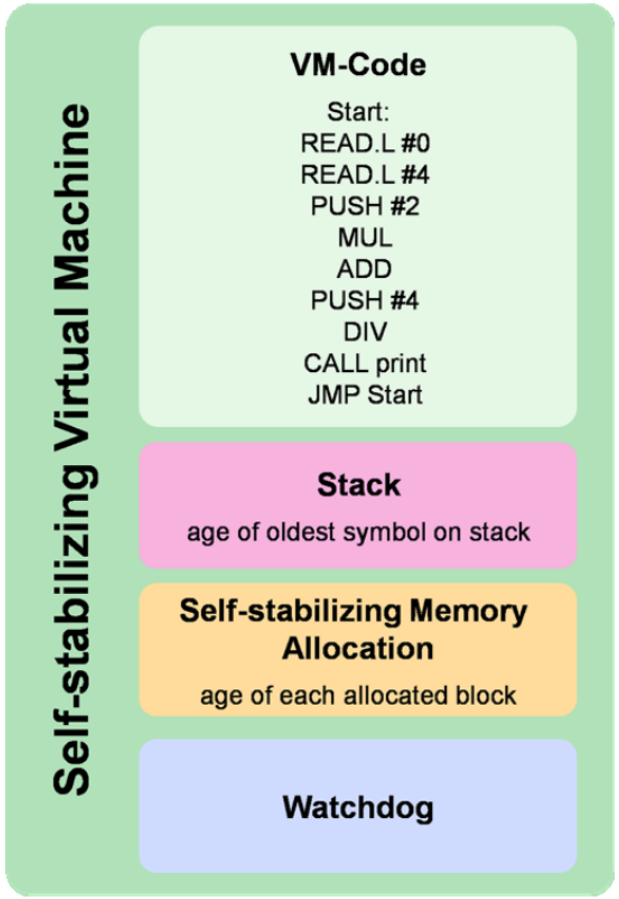
- Generation of safe OC applications
  - Safety constraints, safe default values, and safety rules are encapsulated in safety units
  - Event-triggered safety units take over control in critical states
  - Safety units require composite events

Self-stabilizing VM



Round-Trip Modeling





Back-annotation

Composite Events

- Event-driven applications
  - MODOC application roles communicate by exchanging event notifications via publish/subscribe
  - Actions are often triggered only if several conditions are met
    → signaled by specific event patterns
  - Detecting event patterns is currently done at application level

- Event composition at middleware level
  - Replaces/complements event patterns by composite events
  - Composition algebra makes composition rules explicit
    → increases reusability
  - Requirement to realize system level safety constraints
  - Provides optimization potential
    → event aggregation reduces network traffic

Conclusions

- Model-driven development of OC applications

- Phase II
  - Defined computational model
  - Guaranteed self-stabilization
  - Self-stabilizing publish/subscribe algorithms

- Phase III
  - Efficient execution platform for self-stabilizing algorithms
  - Safety by introducing analytic redundancy
  - Fault-containment during the stabilization period

Discussion

Thanks for your kind attention.

Prof. Torben Weis
Distributed Systems Group
torben.weis@uni-due.de
http://www.uni-due.de/vs