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



Prof. Dr.-Ing. Torben Weis
Universität Duisburg-Essen

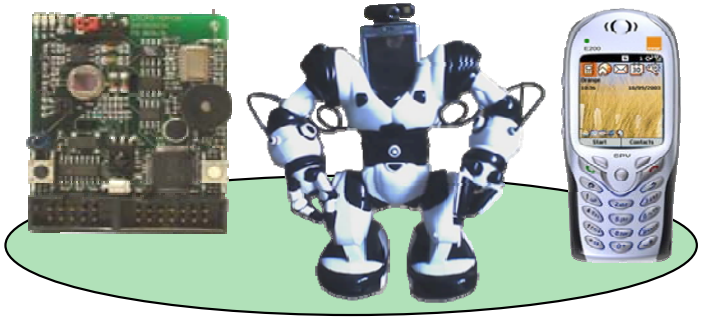

Prof. Dr. Hans-Ulrich Heiß
PD Dr.-Ing. Gero Mühl
Dipl.-Inform. Helge Parzyjegla
TU Berlin

# MODOC Project

- Problem Domain
  - Described in domain-specific terms
- Model-Transformation
  - Encapsulates expert knowledge
  - Transforms model into an executable system
- Realization Domain
  - Described in platform-specific terms

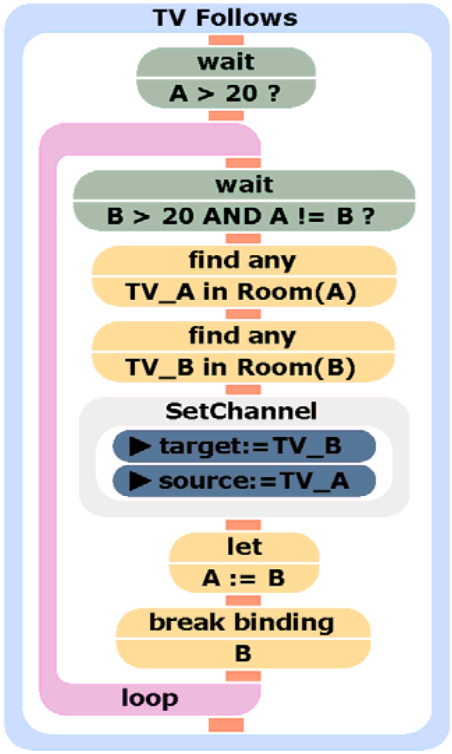
Problem Domain

Model-
Transformation



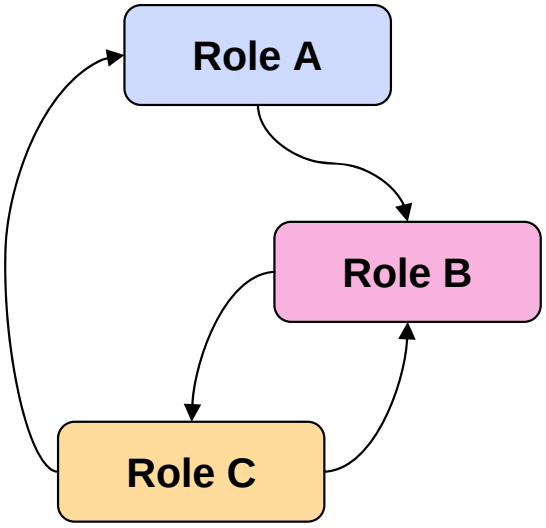
Realization Domain

# Model Transformation




Domain-specific Model

Role Model




Algorithm Toolbox

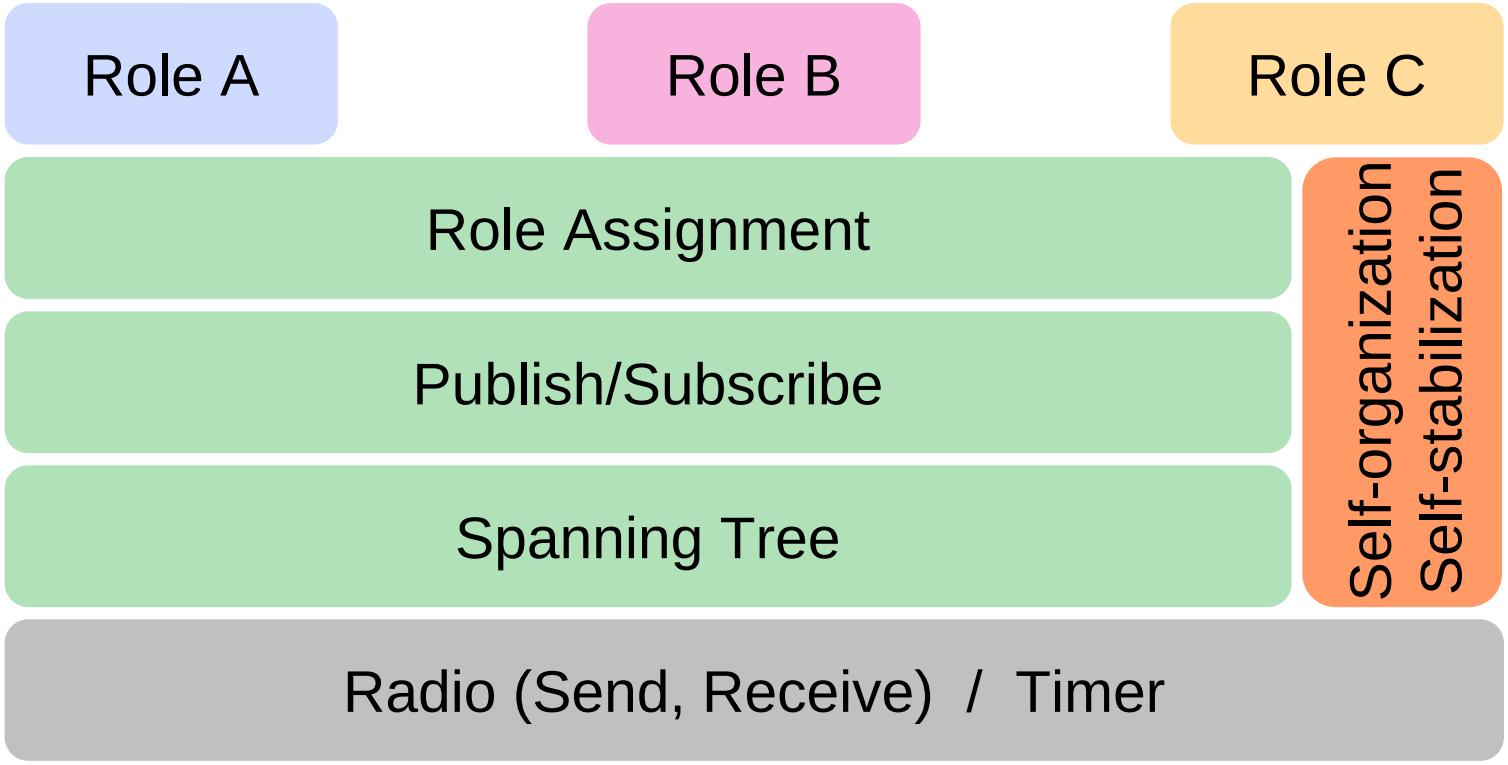
Self-stabilizing Algorithms

Computational Model
(State machine, Turing Machine)

Platform-specific Model

**Model Transformation**

# Role Assignment Algorithm Stack




Spanning Tree

- Structures the network
- Determines the role coordinator

Publish/Subscribe

- Provides communication infrastructure
- Enables addressing of roles

Role Assignment

- Assigns roles to capable nodes
- Monitors roles and reassigns them if necessary

# Optimization Goals

## Accelerate self-stabilization

- Idea: Send more messages to resolve faults faster
- Instantaneous forwarding of subscriptions and notifications
- Introduction of explicit unsubscriptions to remove invalid routing entries

## Save energy

- Idea: Send less messages to decrease energy consumption
- Delay forwarding of messages in favor for piggybacking
- Decouple the forwarding of heartbeats from their receipt

→ Trade-off between stabilization time and energy consumption.

# Role Placement

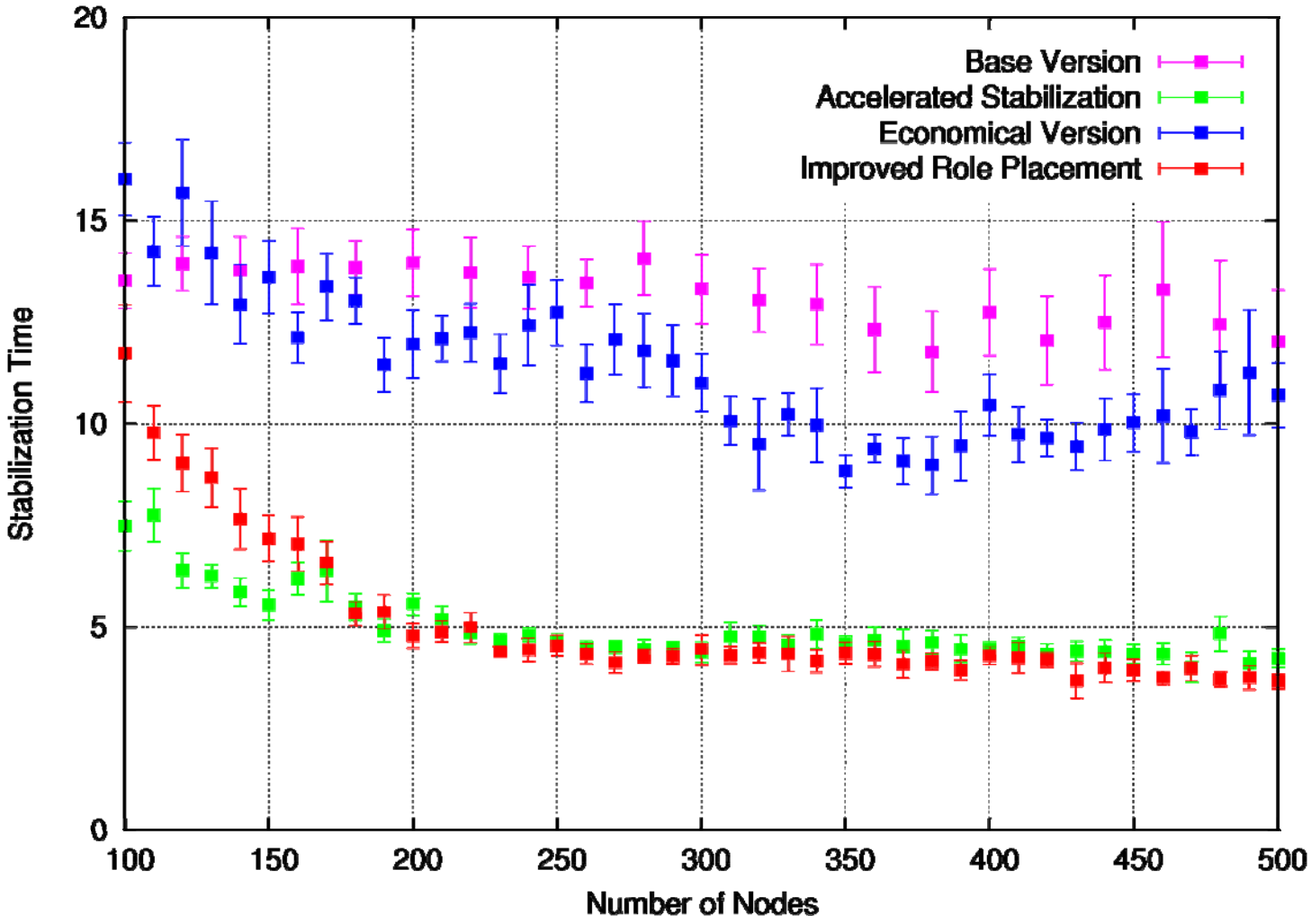
- Basic role placement
  - Assign the role to any node that can execute it
- Improved role placement
  - Collaborating roles should be close to each other

- One network, multiple applications
  - Roles of one application interact heavily
  - Roles of different applications interact less frequently

- Idea: Exploit locality effects
  - Place roles of an application close to each other
  - Reduces the number of hops a message must be forwarded
  - Saves energy and reduces communication delays

# Evaluation

> Average stabilization time

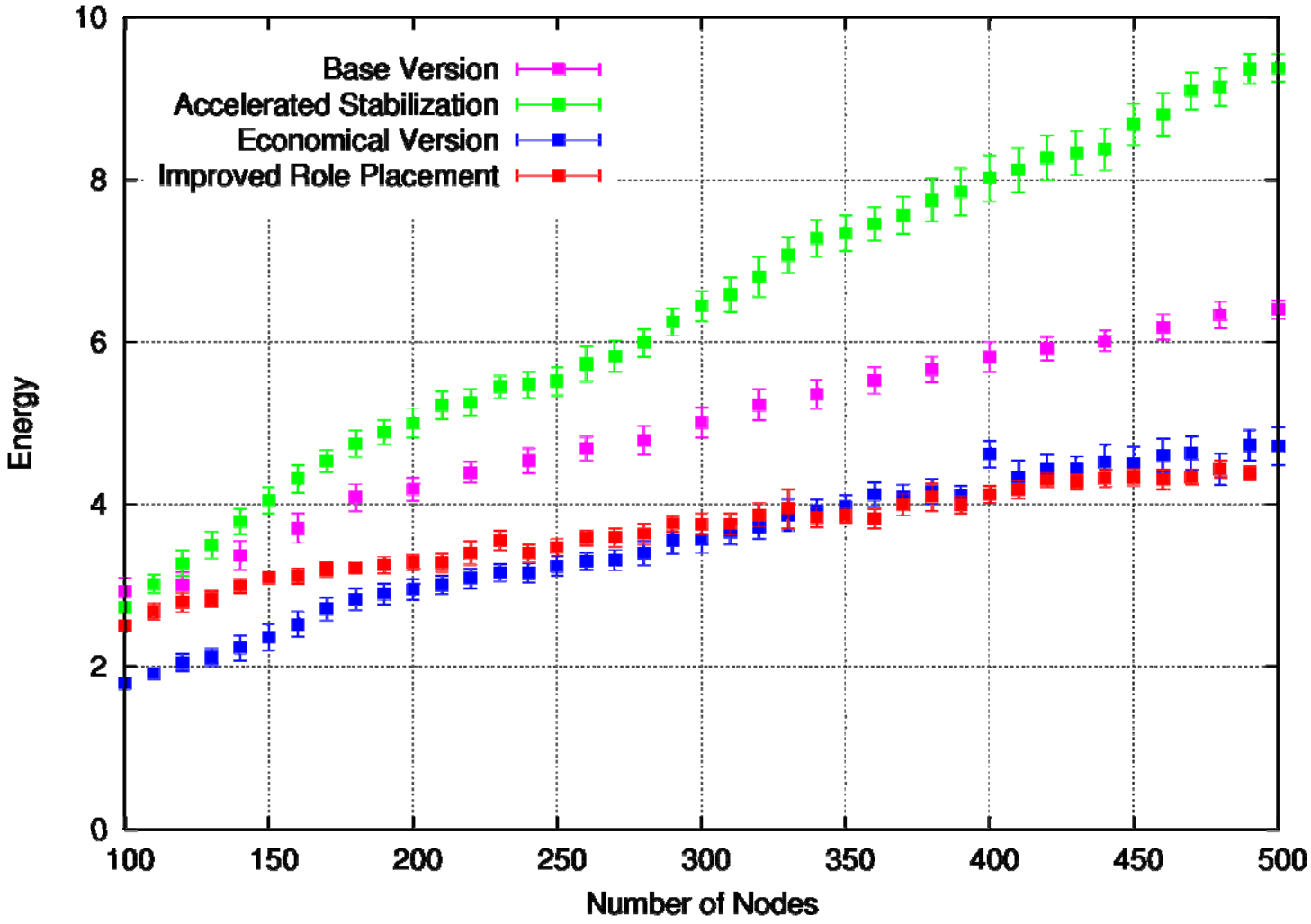



Parameters

> Nodes: 100 – 500
> Applications: 20
> Roles: 80 (total)

# Evaluation

> Average energy consumption

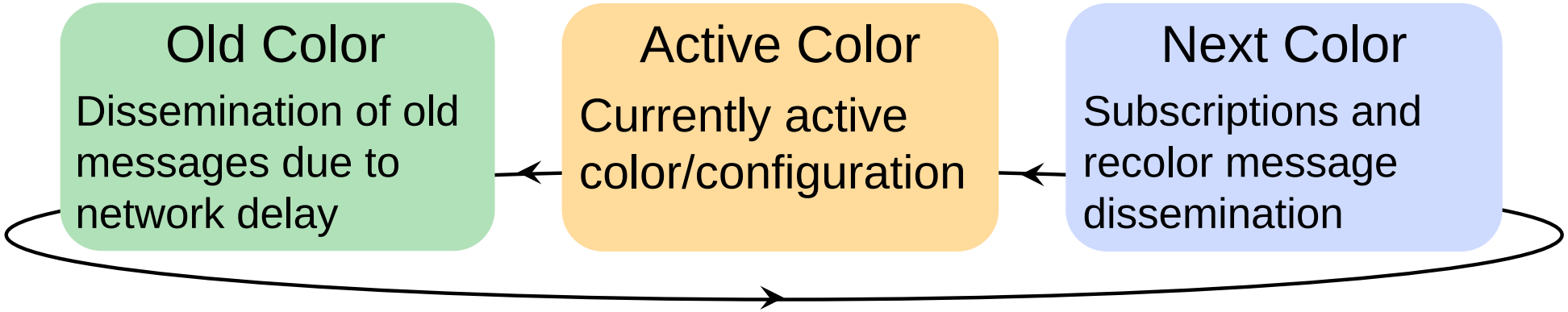



Parameters

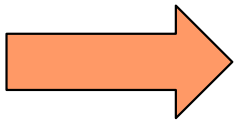
> Nodes: 100 – 500
> Applications: 20
> Roles: 80 (total)

# Reconfiguration

- Reasons for reconfiguration
  - Adaptation to structural changes (e.g., addition/removal of devices)
  - Optimization of system's performance
  - Manual intervention (e.g., maintenance of devices)
- Reconfiguration with layered self-stabilization
  - Keep multiple configurations in parallel
  - Build up next configuration while another is still active
  - Consistently switch between configurations on each layer



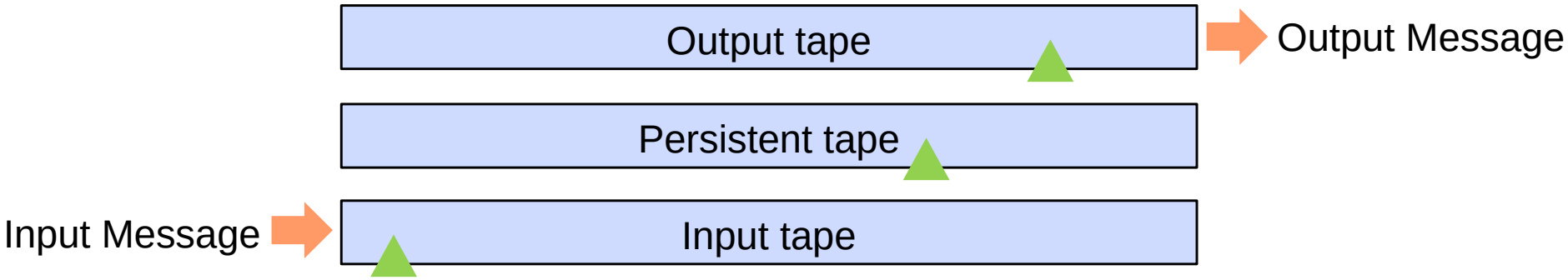
# MODOC Phase II

E-Home    E-Office

- Computational Model: Support more complex applications
  → Specialized Turing Machine instead of State Machine
- Super-stabilization: Give guarantees even during stabilization for certain classes of transient faults → safety constraints
- Fault containment: Locally bound the effects of faults within the affected system‘s component
- Quality of Service: Optimize the role placement to fulfill application specific QoS requirements

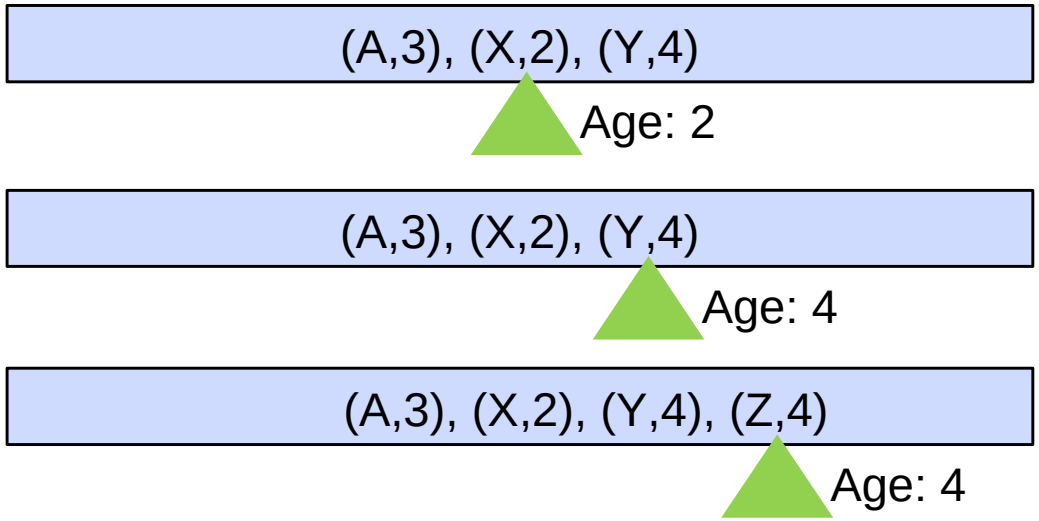
# Computational Model

- Turing Machines & Self-stabilization
  - Ideal case: Show that every Turing Machine can be transformed in a self-stabilizing Turing Machine
  - Theoretically possible (Dolev et. al.), but not practicable
- Turing Machines & Actuator/Sensor Networks (ASNets)
  - Turing Machines model batch operation, i.e. they halt
  - ASNets do never halt
- Solution: Self-stabilizing Persistent Turing Machine

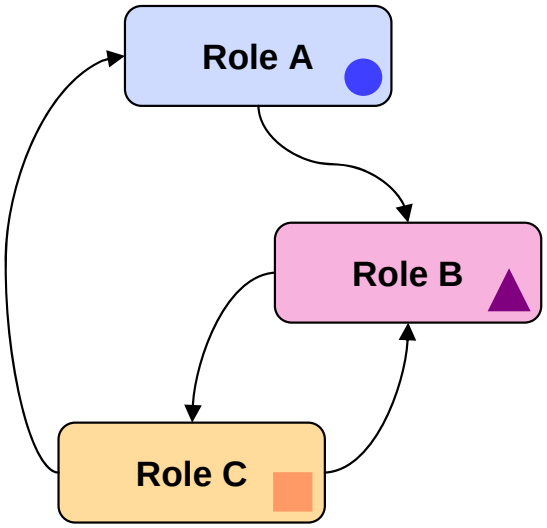


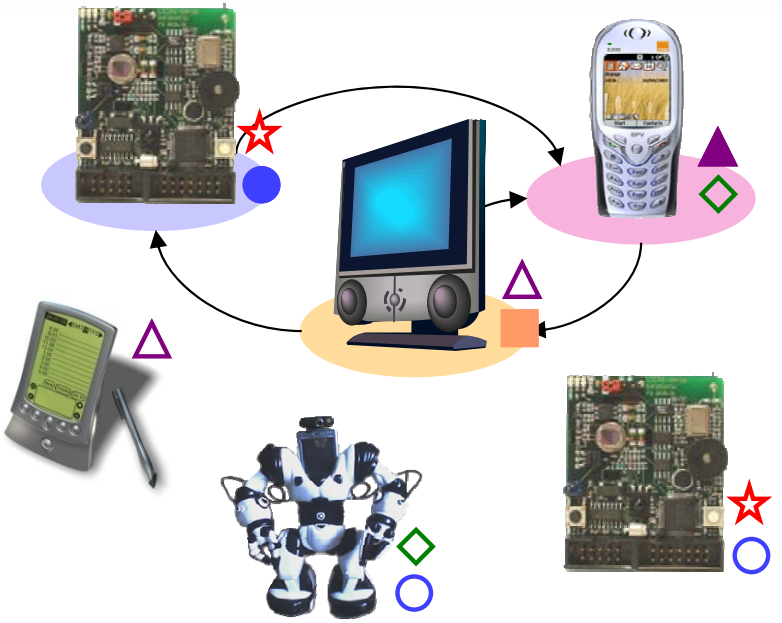
# Computational Model

- 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



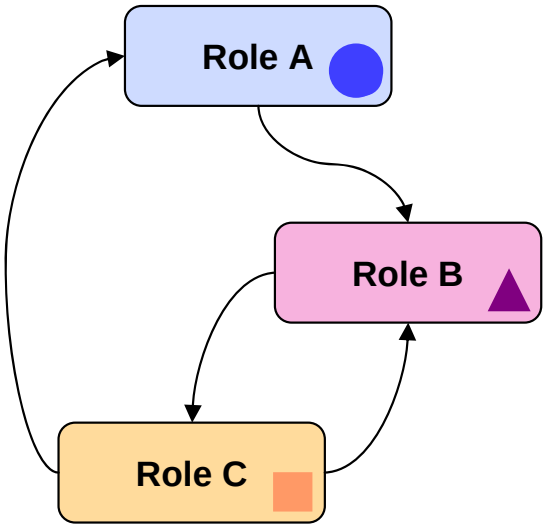
# Revisiting the dynamic Role Assignment

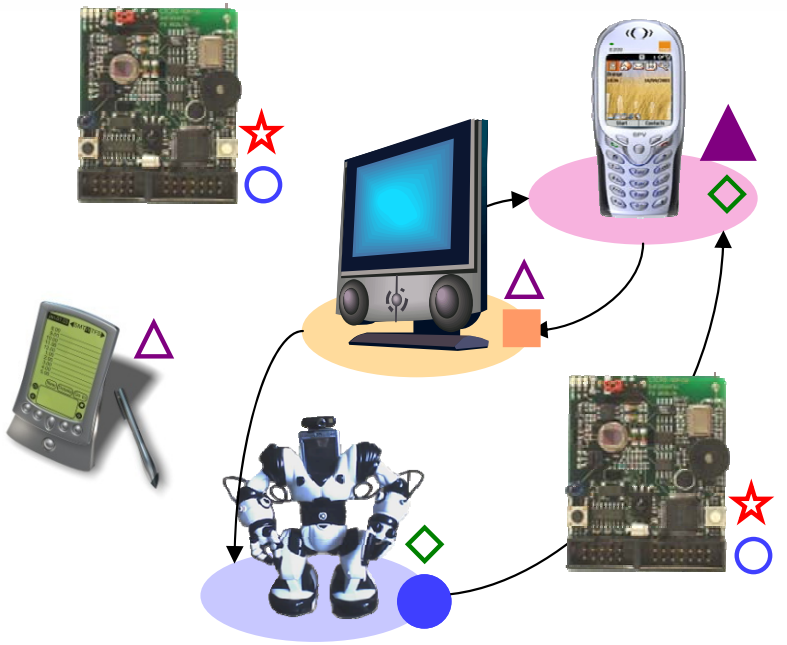


- Dynamic role assignment at runtime
  - Each role poses requirements to the node hosting it
  - Nodes announce their capabilities to serve certain roles
  - A previously elected role coordinator assigns roles to nodes capable of serving them

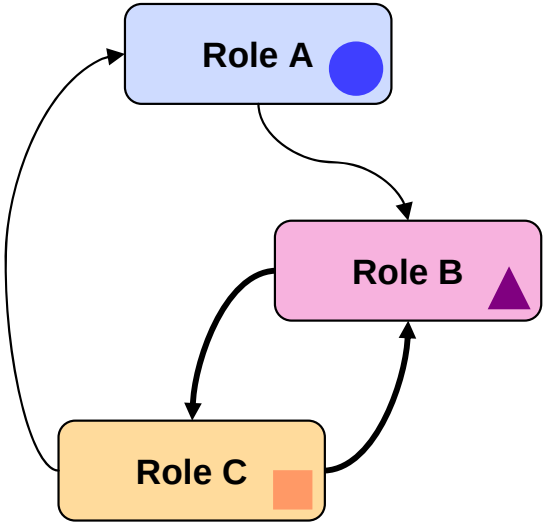
# Quality of Service (QoS)

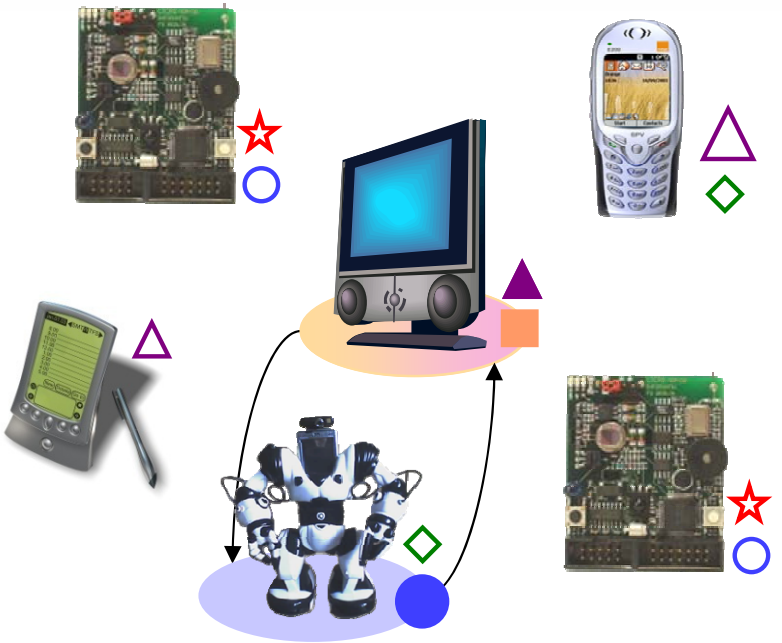


- QoS requirements can be derived from the application model and stored in meta-data during the transformation
- Nodes may use this meta-data and additionally announce how good they are in performing a given role
- Role coordinator chooses a suitable candidate among them

# Optimized Role Placement



- Communication demands of related roles may also be derived from the application model or given annotations
- Role coordinator tries to initially place related roles close to each other that share high communication demands
- Fine-grained optimization may still be necessary during runtime since communication patterns may change over time

# Discussion

Thanks for your kind attention.

Torben Weis

Distributed Systems Group

torben.weis@uni-due.de

http://www.uni-due.de/vs