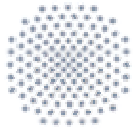
Model-driven Development of Self-organizing Control Applications (MODOC)

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



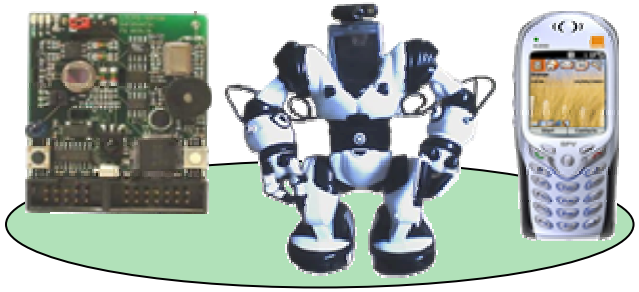
Dr.-Ing. Torben Weis
Universität Stuttgart

MODOC Project

- Target Domain
  - Model-driven development
  - Control-applications
  - Sensor- and actuator networks (SANets)
- Target Objectives
  - Simplify the development process
  - Hide distribution, self-organization, and self-stabilization
  - Encapsulate expert-knowledge in model transformation

Problem Domain

Model-
Transformation



Realization Domain

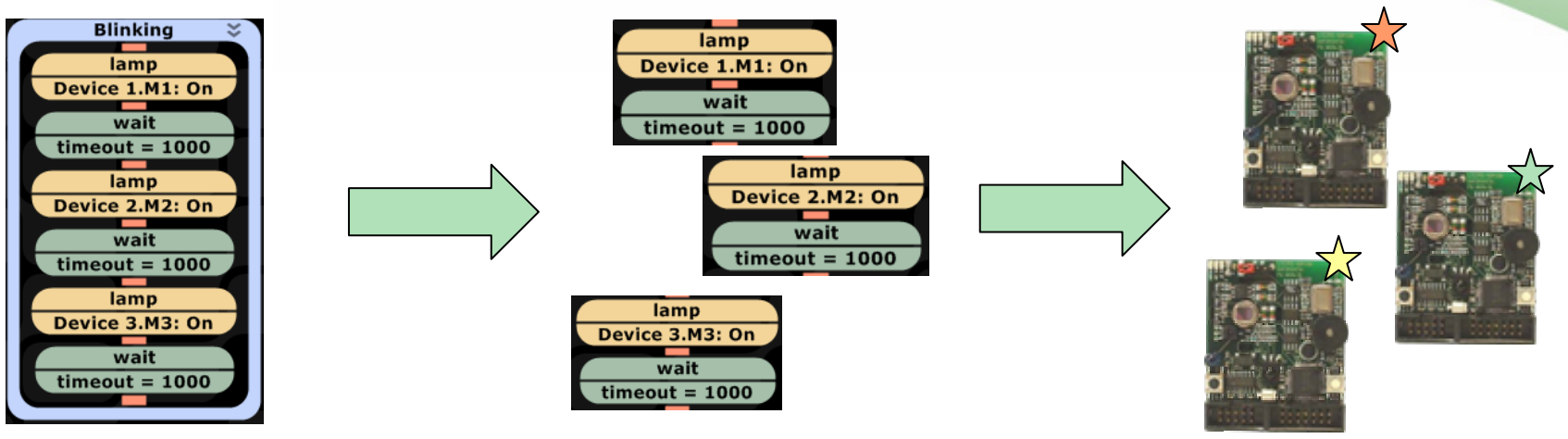
Overview

- Role concept
- Role assignment algorithm stack
  - Spanning tree
  - Publish / subscribe
  - Role activation
- Analysis and simulation
- Project status and outlook

Roles

- Abstraction concept introduced in data modeling (Bachmann and Daya, 1977)
- Roles specify a certain behavior as well as certain properties
- Entities (e.g., nodes, components, objects) can adopt / abandon roles
- *m*:*n* mapping between entities and roles
- Roles support self-organization of flexible, evolving networks
- Example: sensor networks featuring clusterheads, gateway nodes, backbone nodes

Role Concept



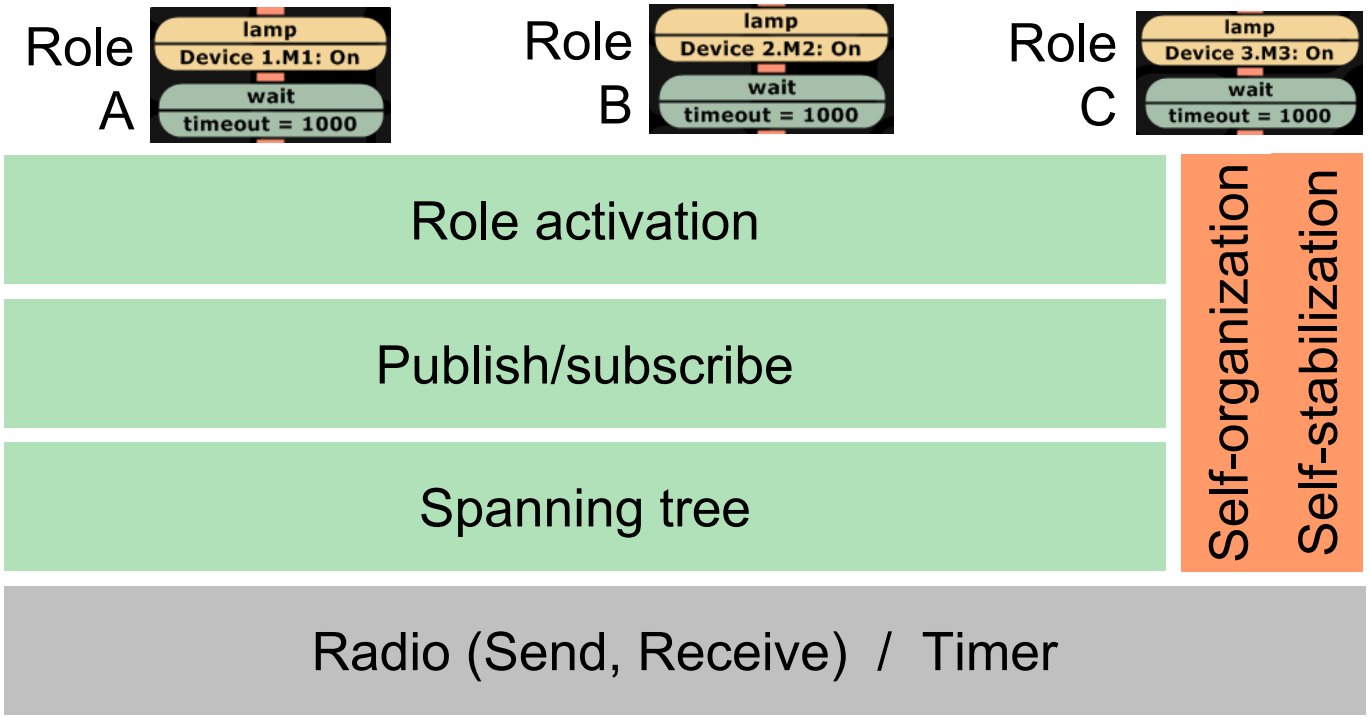
Application model

Roles / code snippets

Dynamic role activation

- Model transformation
  - Inspects process-based application model
  - Splits application in several cooperating roles
  - Generates code snippets for each role
- Role assignment
  - Assigns roles to nodes capable of serving them
  - Monitors roles and reassigns them if necessary
  - Provides means for addressing of roles

Role Assignment Algorithm Stack

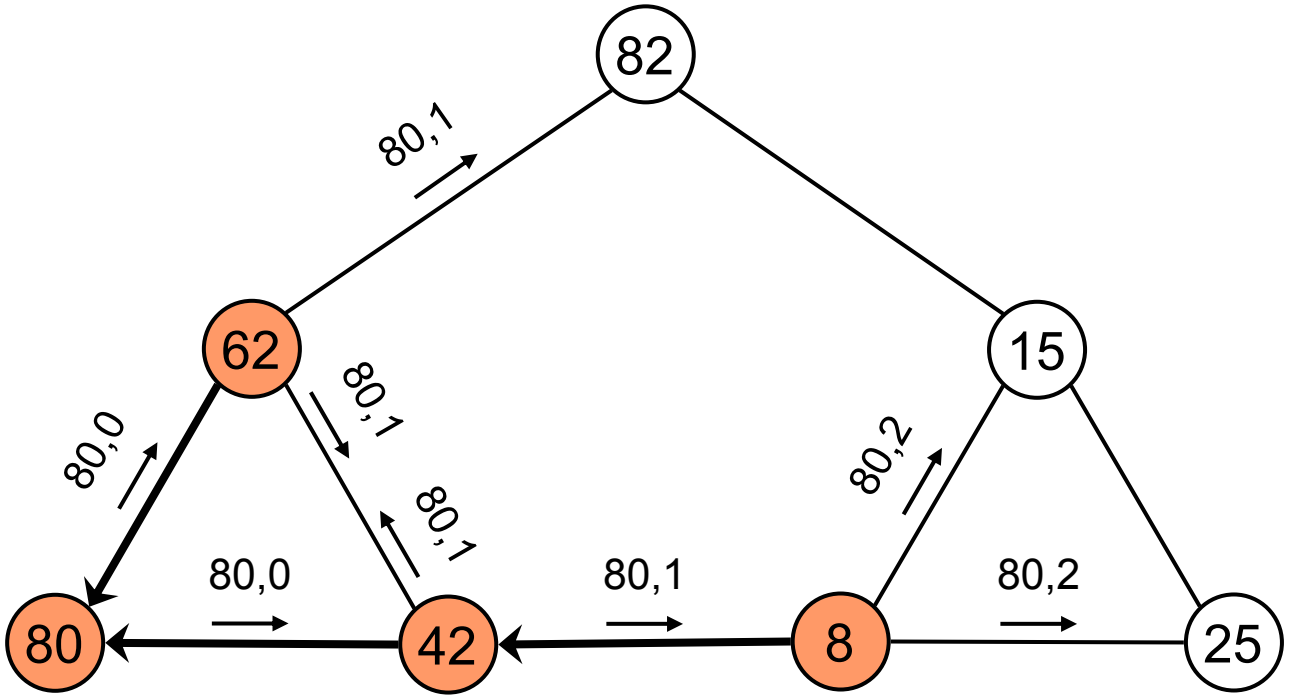


- Self-stabilization
  - Addresses transient faults
  - Guarantees recovery in a bounded time (supposed no other fault occurs meanwhile)
- Self-organization
  - Addresses permanent faults
  - Structural adaptation to environmental changes

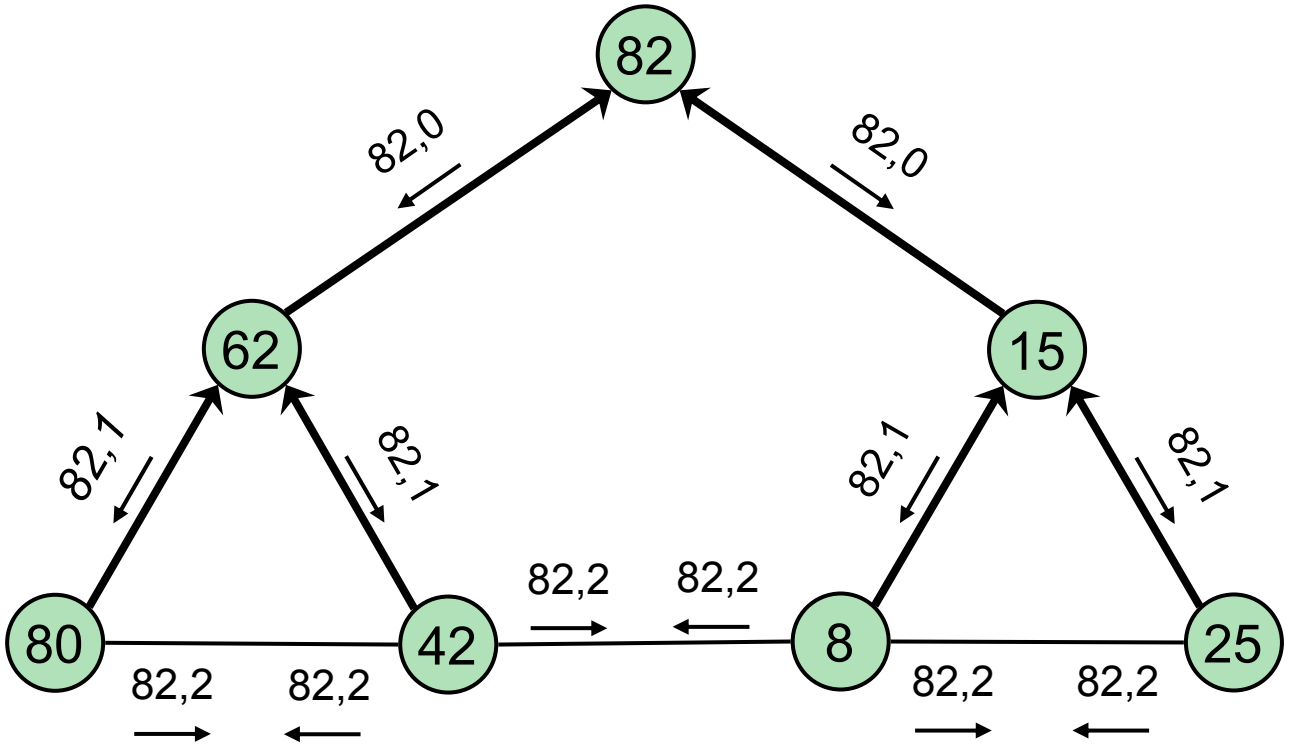
Spanning Tree Algorithm

- Adaptation of a self-stabilizing spanning tree algorithm (Afek, Kutten, and Yung, 1991)
- Based on periodic heartbeat broadcast messages
- Purpose
  - Structuring the network
  - Providing a tree for information dissemination
  - Determining the root node used as coordinator
- Algorithm principles
  - Choose node with largest ID as root
  - Construction of a shortest path tree
  - Tree gets periodically refreshed

Tree Construction



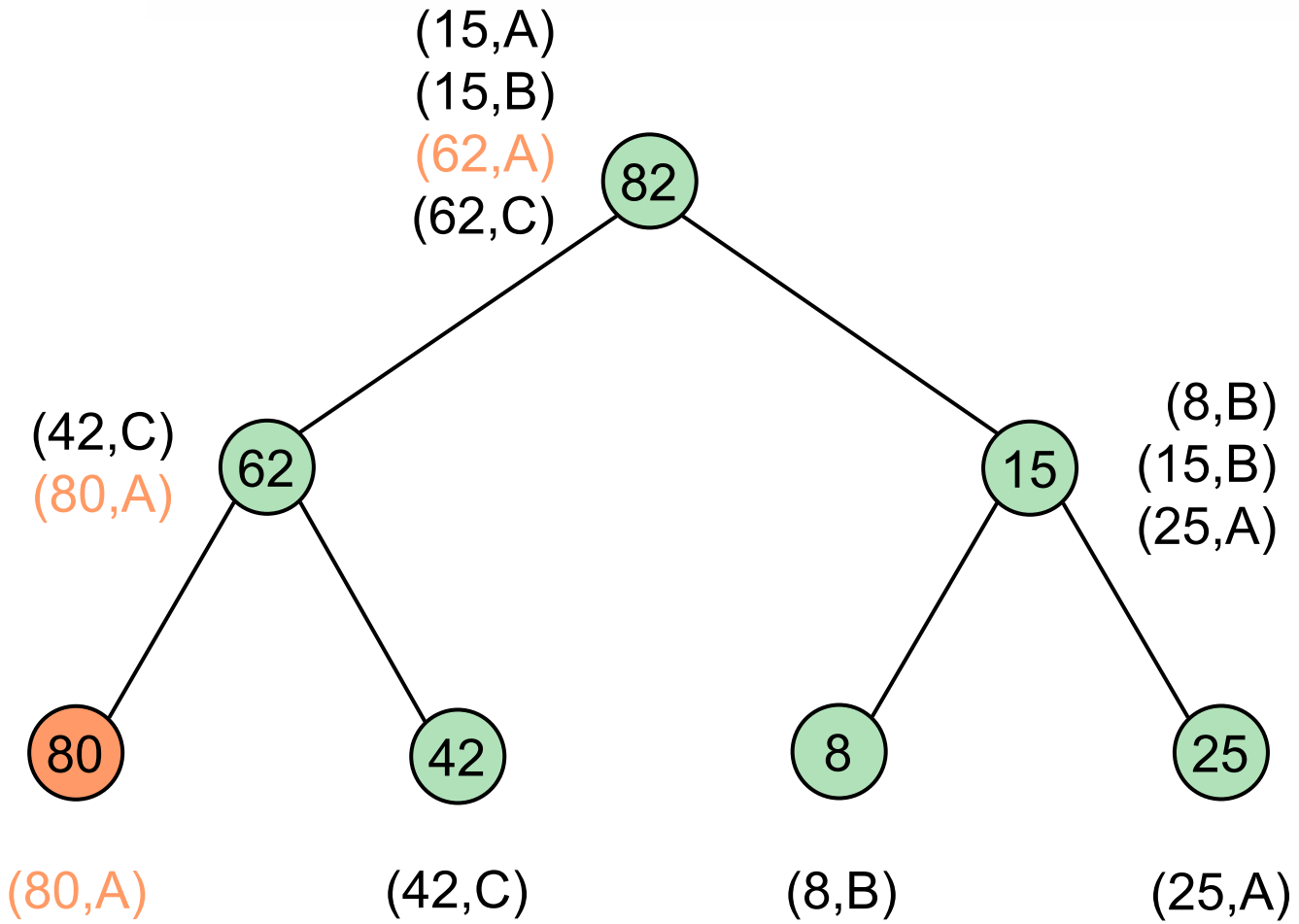
Tree Construction



Publish/Subscribe Algorithm

- Self-stabilizing hierarchical publish/subscribe algorithm (Mühl et al., 2005)
- Self-stabilization achieved by leasing subscriptions
- Enables role-based addressing
- Provides 3 functional primitives
  1. Subscribe(role)
     subscribes for messages to a given role
  2. Publish(role, message)
     publishes a message to all matching subscribers
  3. PublishSingle(role, message)
     sends a message to just one subscriber

Publish/Subscribe Routing

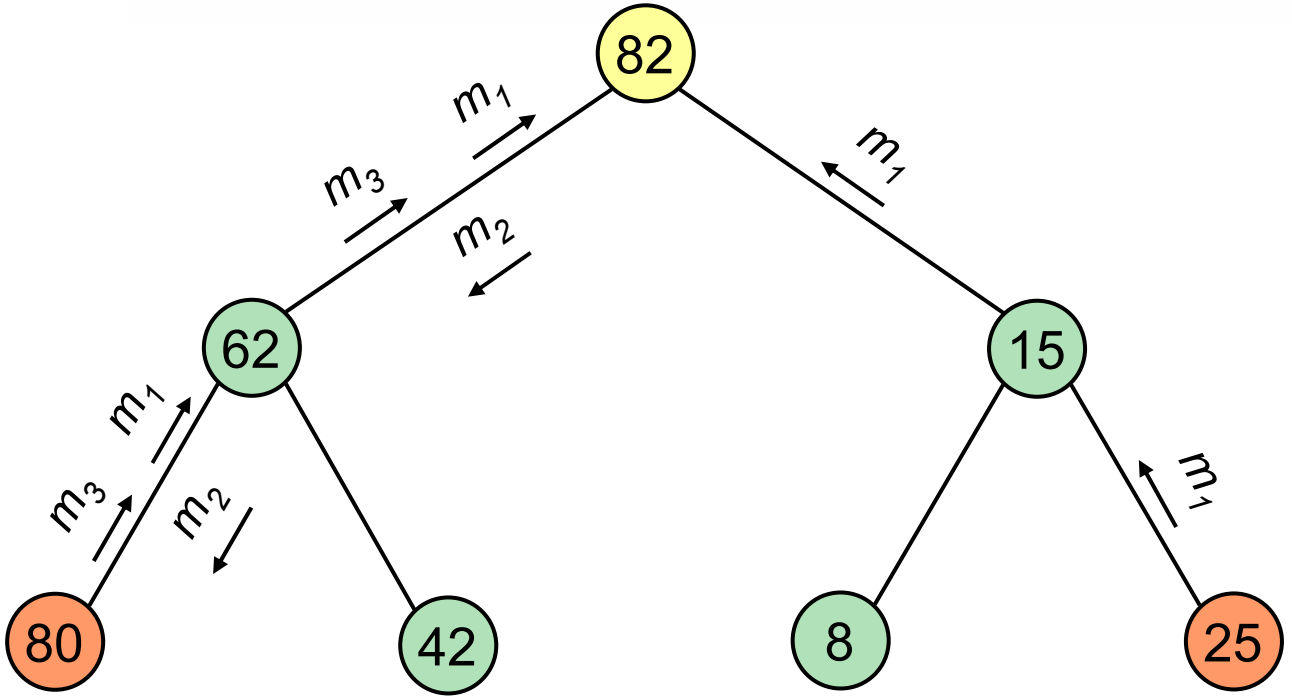


Role Activation Algorithm

Phases

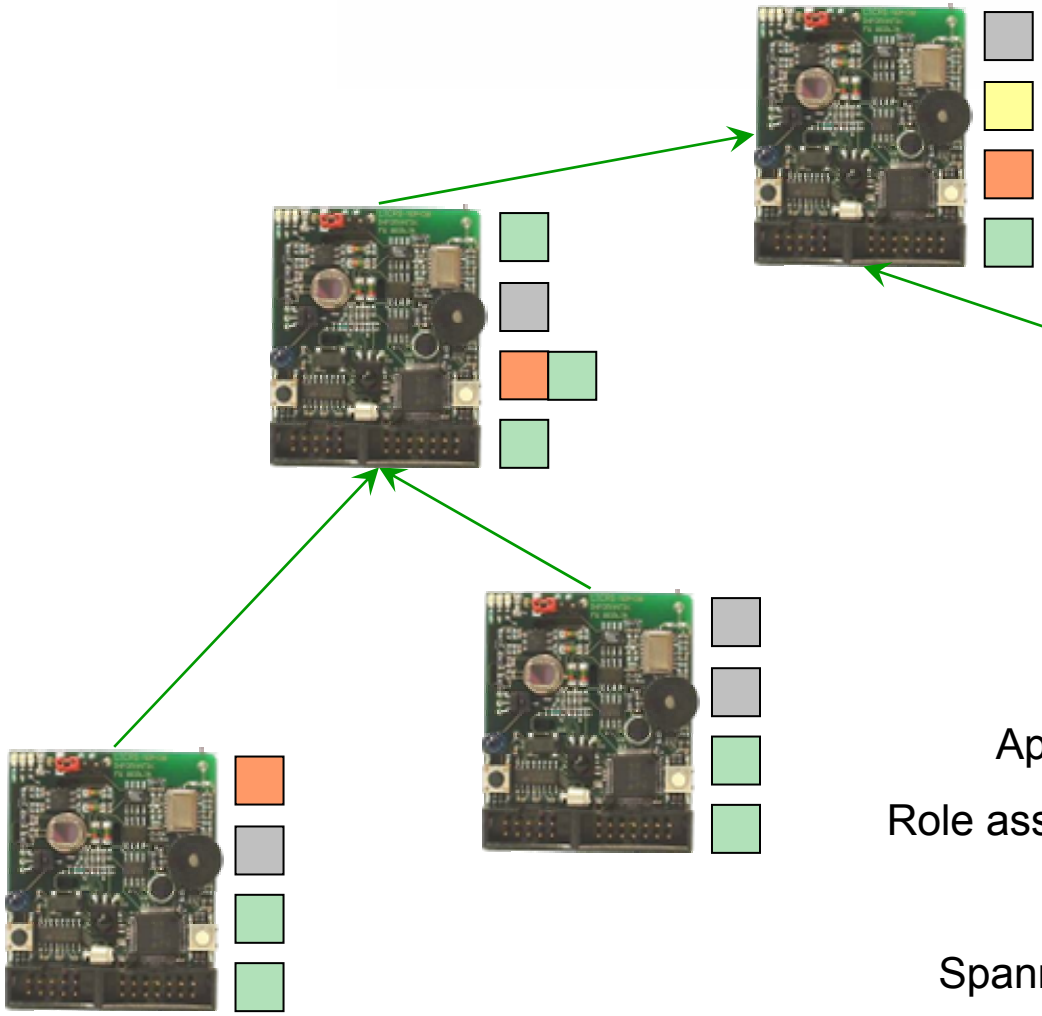
1. Nodes compare their capabilities to role requirements
2. Nodes subscribe to possible roles they may play using message type of (Role | possible)
3. Root node waits until all roles of an application are available
4. Root node assigns all roles of a given application by sending an activation message to single subscribers
5. Receiving node activates role and subscribes to message type (Role | active)
6. Root node monitors role assignment during the runtime loop of the application

Message Exchange



1. m_1: Subscribe((A | possible))
2. m_2: PublishSingle((A | possible), true)
3. m_3: Subscribe((A | active))

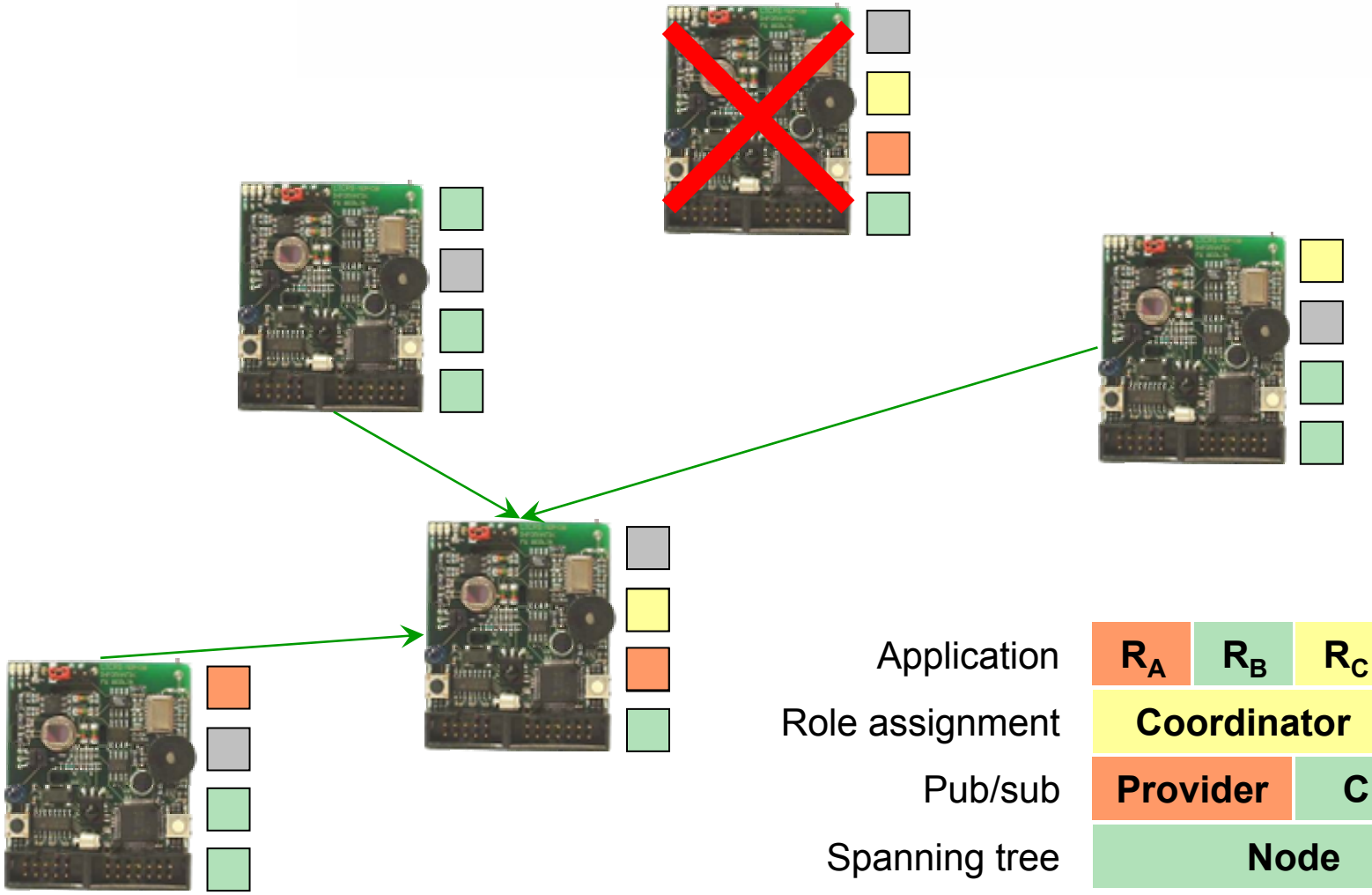
Combined Algorithm Stack



Application	R_A	R_B	R_C	
Role assignment	Coordinator			
Pub/sub	Provider	Client		
Spanning tree	Node			

Role assignment algorithm stack

Combined Algorithm Stack



Role assignment algorithm stack

Algorithm Analysis

> Stabilization time

$$\underbrace{\mathcal{O}(\tau)}_{\text{Tree}} + \underbrace{\mathcal{O}(h \cdot \tau)}_{\text{Pub/Sub}} + \underbrace{\mathcal{O}(h \cdot \pi)}_{\text{Roles}} = \underbrace{\mathcal{O}(h \cdot \tau)}_{\text{Stack}} \Rightarrow \mathcal{O}(\sqrt{n} \cdot \tau)$$

> Message complexity

$$\underbrace{\mathcal{O}(n^2)}_{\text{Tree}} \qquad \underbrace{\mathcal{O}(n)}_{\text{Refresh}}$$

> Memory consumption

$$\underbrace{\mathcal{O}(D \cdot R)}_{\text{Roles}}$$

n : Number of nodes

h : Tree's height

τ : Timeout

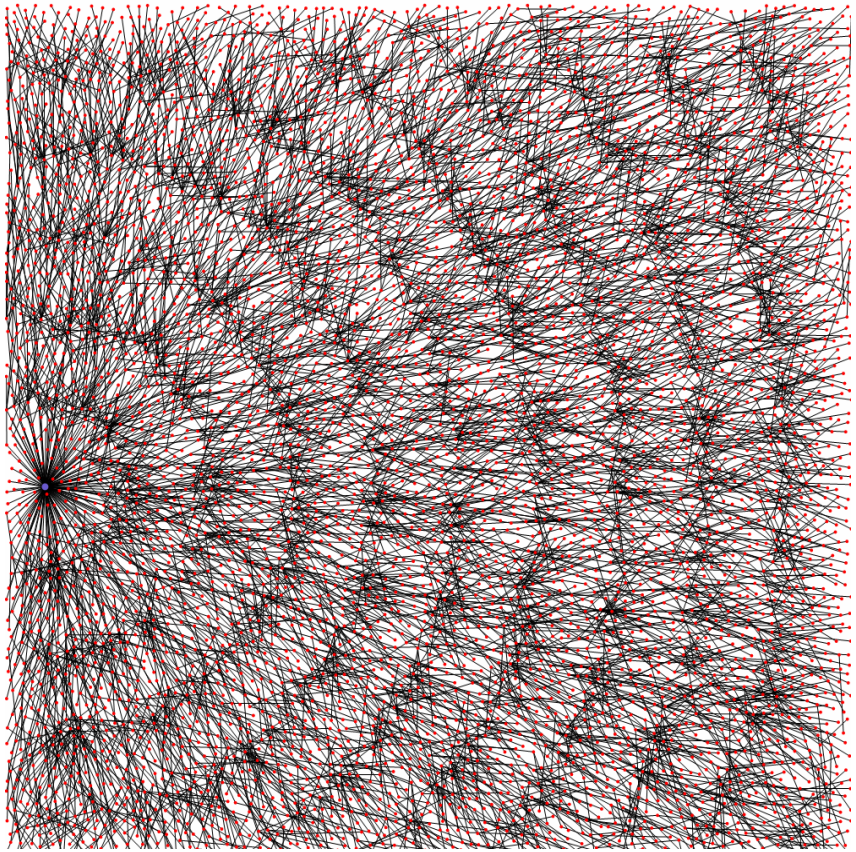
π : Refresh period

D : Max degree

R : Number of roles

Simulation

- Problem
  - Non-uniform load distribution
  - Non-uniform energy consumption leads to network partition
- Reason
  - Rings have distance equal to broadcast range r_{max}
  - Higher probability to be chosen as parent at distance $i \cdot r_{max}$
- Solution
  - Incorporate the energy left for a node in distance metric
  - Construction of more trees



Simulated spanning tree algorithm for a large network

Project Status

- Role-based development process for self-organizing applications
  - Allows for decomposition of the application into roles
- MODOC algorithm toolbox
  - Role assignment and monitoring
  - Role-based communication infrastructure via publish/subscribe
  - All algorithms are self-stabilizing
  - Algorithms use cross-layer optimization
- Self-organization
  - Dynamic assignment of roles to nodes capable of serving them
  - Applications automatically adapt to changing network topologies (e.g., addition and removal of devices)
- Benefit for developers
  - They get self-organization and self-stabilization almost “for free”

Outlook

- Development tools
  - Modeling language for role-based applications
  - Transform models into source code which utilizes the MODOC algorithm toolbox
- Debugging tools
  - How to debug a distributed self-organizing application?
- Optimizations
  - Distribution of roles in a network, i.e. locality
  - Energy consumption
  - Avoidance of instable nodes
- Implementation
  - Currently running in a simulator
  - Demonstrator using ESBs is planned

Discussion

Thanks for your kind attention.

Helge Parzyjegla
parzyjegla@acm.org
http://kbs.cs.tu-berlin.de/~parzy