SMART MICRO-SCALE ENERGY MANAGEMENT AND ENERGY DISTRIBUTION

IN DECENTRALIZED SELF-POWERED NETWORKS USING MULTI-AGENT SYSTEMS

Stefan Bosse

University of Koblenz-Landau, Fac. Computer Science, Germany

10.9.2018

sbosse@uni-bremen.de

OVERVIEW

INTRODUCTION

Goal?

Self-organizing energy management and distribution in autonomous and self-powered **Sensor Networks**, the **IoT**, and **Smart Energy Grids**

Methodology?

Mobile Multi-agent Systems that are capable to transfer (virtually carry) energy between networks nodes with **Divide-and-Conquer** approach

Technology?

Sensor and computer nodes connected by shared data-energy links (i.e., transferring data and energy over a wired or wireless link)

Investigation?

Global **emergence** behaviour of self-organizing energy management agents

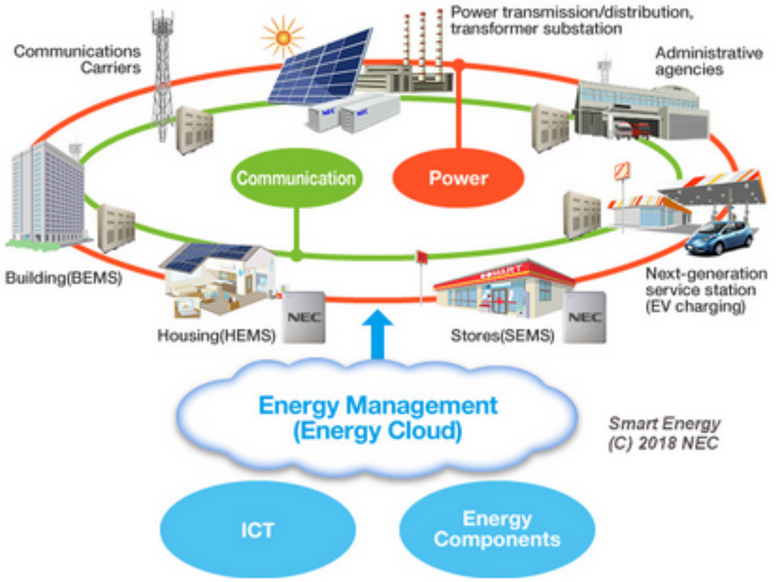
Evaluation?

Simulation of a mesh-grid Sensor Network using the *SEJAM2* simulator

MOTIVATION AND STATE OF THE ART

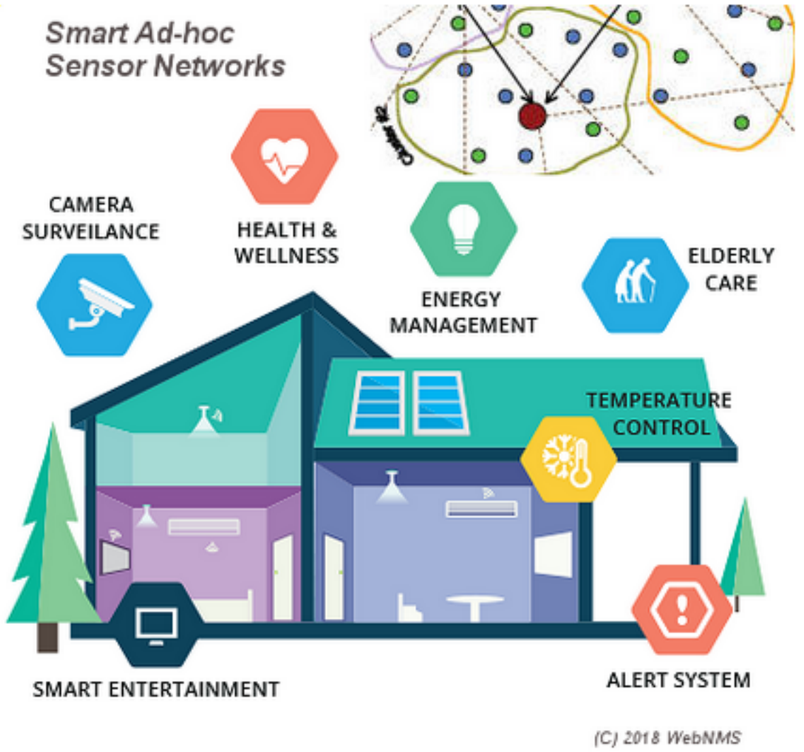
Smart Energy Grid

» Tight coupling of Communication and Power Control



Smart Sensor Networks

» Autonomous sensor nodes, self-powered, mobile, ad-hoc IoT connected



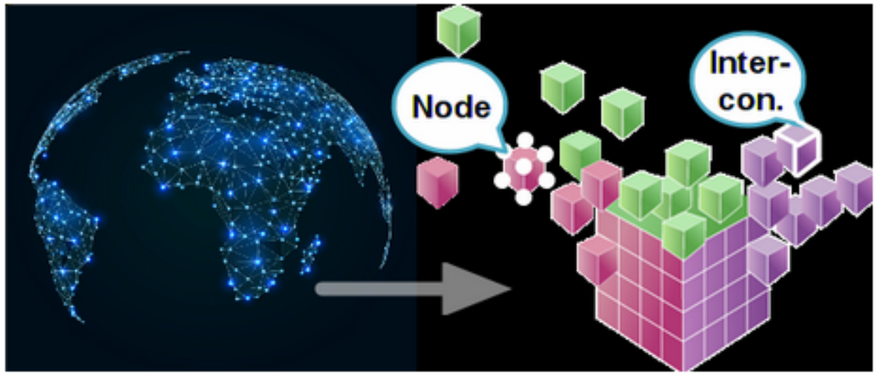
REFERENCE ARCHITECTURE

NETWORK LEVEL

- This work made no technology or architectural specific assumptions.

Definition 1. (*Network*)
A network W is a graph $G(N, C)$ consisting of autonomous nodes $n_i \in N$ and edges $c_i \in C$ connecting nodes (communication channels).

- *Simplification*: Although arbitrary network topologies are supported, this works assumes **three-dimensional mesh-grid networks**
  - » The network can be irregular (missing nodes) and incomplete (missing links) → **Unicast Peer-to-Peer links**
- Each node can be connected with up to six neighbouring nodes:



NETWORK LEVEL

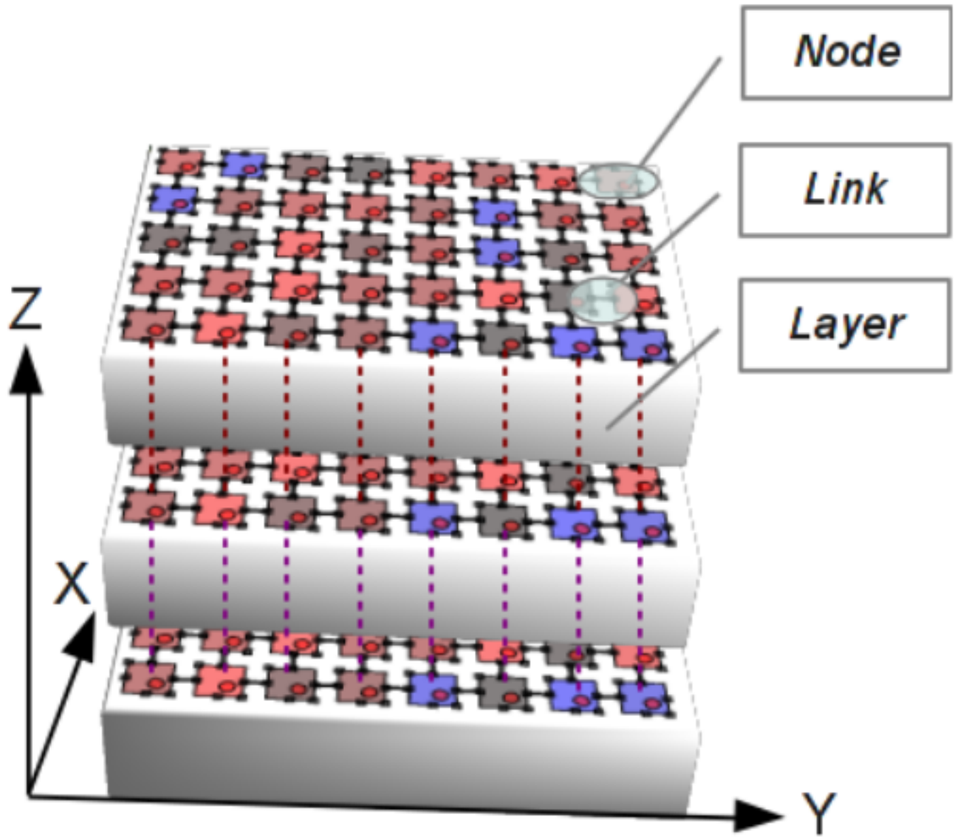


Fig. 1. Three-dimensional mesh grid network

NODE LEVEL

- Each node is some kind of computer (at least a microcontroller) that performs:
  - Data Processing
  - Sensor Processing
  - Energy Management
- Each node provides the following modules and components:
  - Data Processor *P*
  - Communication *C*
  - Set of sensors (Power, Voltage, Light, ..) *S* and Signal Processor *SP*
  - Energy Storage *ES*
  - Energy Harvesting *EH*
  - Energy Transformer (Sender & Receiver) *ET*
  - Agent Processing Platform *APP*
  - A common data base (tuple space) used for agent interaction

COMMUNICATION LEVEL

Channels. Nodes can exchange data with their neighbours by using some kind of serial communication links.

- A communication link can be used for the:
  1. Transfer of **data** and messages;
  2. Transfer of **energy**.
- There are different kinds of physical transmission technologies differing in data bandwidth B, latency τ, and energy capability and transmission efficiency ε:
  - » Electrical → high efficiency ε_1;
  - » Optical → efficiency $\varepsilon_2 < \varepsilon_1$; and
  - » Radio-wave based → $\varepsilon_3 < \varepsilon_2 < \varepsilon_1$.

Issue: Each time a network node transfers energy along a path (n_a,n_b) the total transferred energy decreases exponentially ($\Pi\ \varepsilon$)

TECHNOLOGICAL ARCHITECTURE

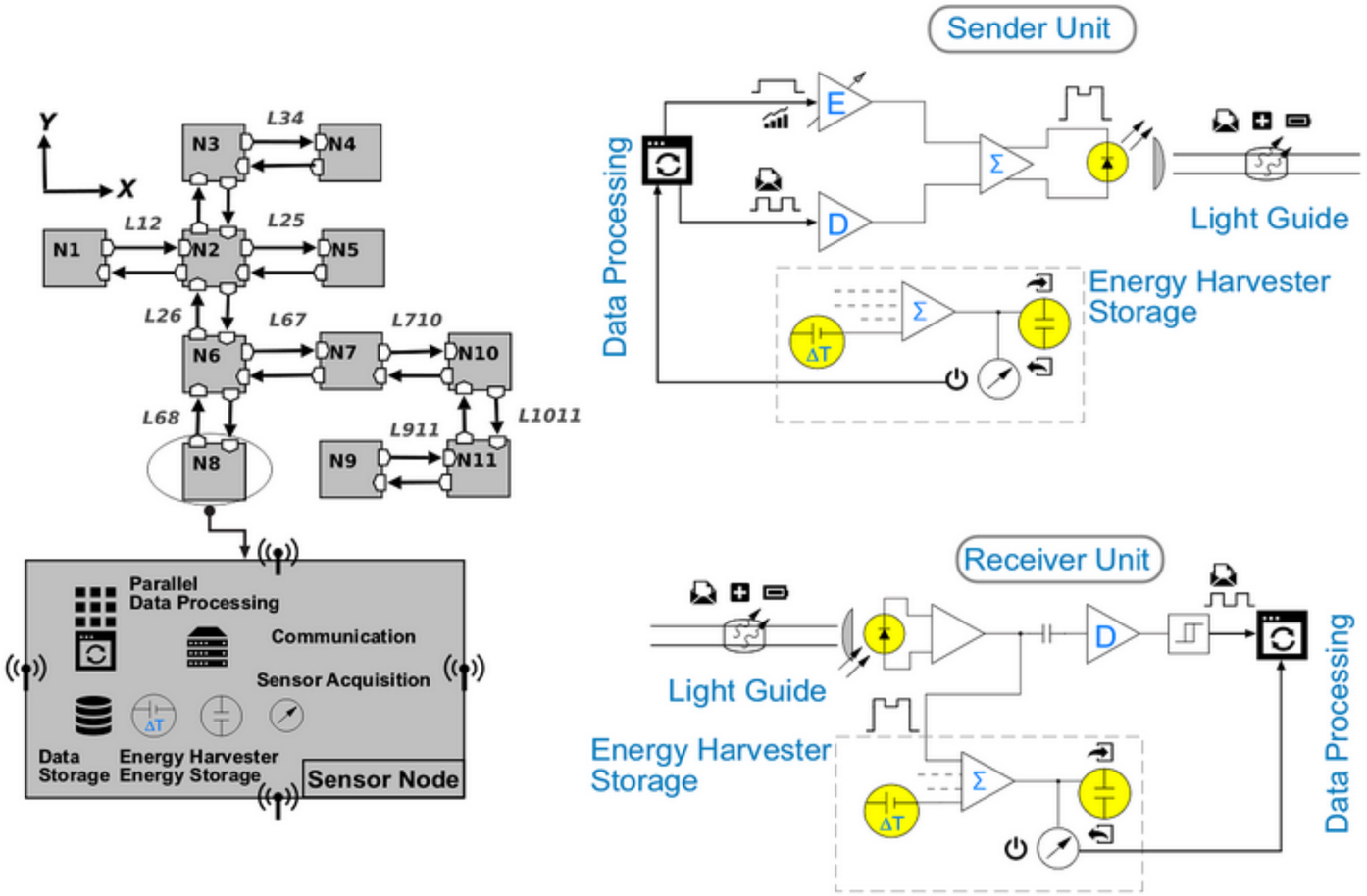


Fig. 2. Network topology (left) composed of nodes with sender and receiver blocks (right) used for data and energy transmission between neighbouring node.

ENERGY MODEL

*The system energy model is introduced for a common understanding and is **not** used by the agents!*

ENERGY TRANSFER

- **Nodes can interchange energy via messages.**
  - » The data processing system can use the communication unit to transfer data (D) and superposed energy (E).
- Each message sent from node *A* to node *B* is an energy transfer E_{msg} consisting of tokens (bits)
- Sender unit can mix additional energy tokens with message data *D* (*N* serial bits) resulting in increased transmitted energy:

$$E_{msg} = N I_D \tau_1 + I_T \tau_2, \text{ with } I_T \gg I_D$$

- Receiver unit stores received energy $E_{msg} \epsilon_{in}$ extracted from each message in local energy storage.

ENERGY SUPPLY TOPOLOGIES

A. Star, Centralized → External Supply
   - » Each network node is connected point-to-point with an energy source.
   - » Low loss of energy, but not suitable for extended/large networks.

B. Grid, Centralized → External Supply
   - » Each node is supplied with energy from neighbouring nodes.
   - » There is one or there are few external supply point(s).
   - » High loss of energy: energy is passed along several nodes (total eff. Ω) from source to destination points:

$$\Omega = \prod_{\forall i \in \{\mathrm{PATH}\}} \varepsilon_i$$

C. Grid, Decentralized → Self-powered
   - » Each node is supplied by local energy source, and
   - » additionally? by neighbouring nodes.

ENERGY SUPPLY TOPOLOGIES

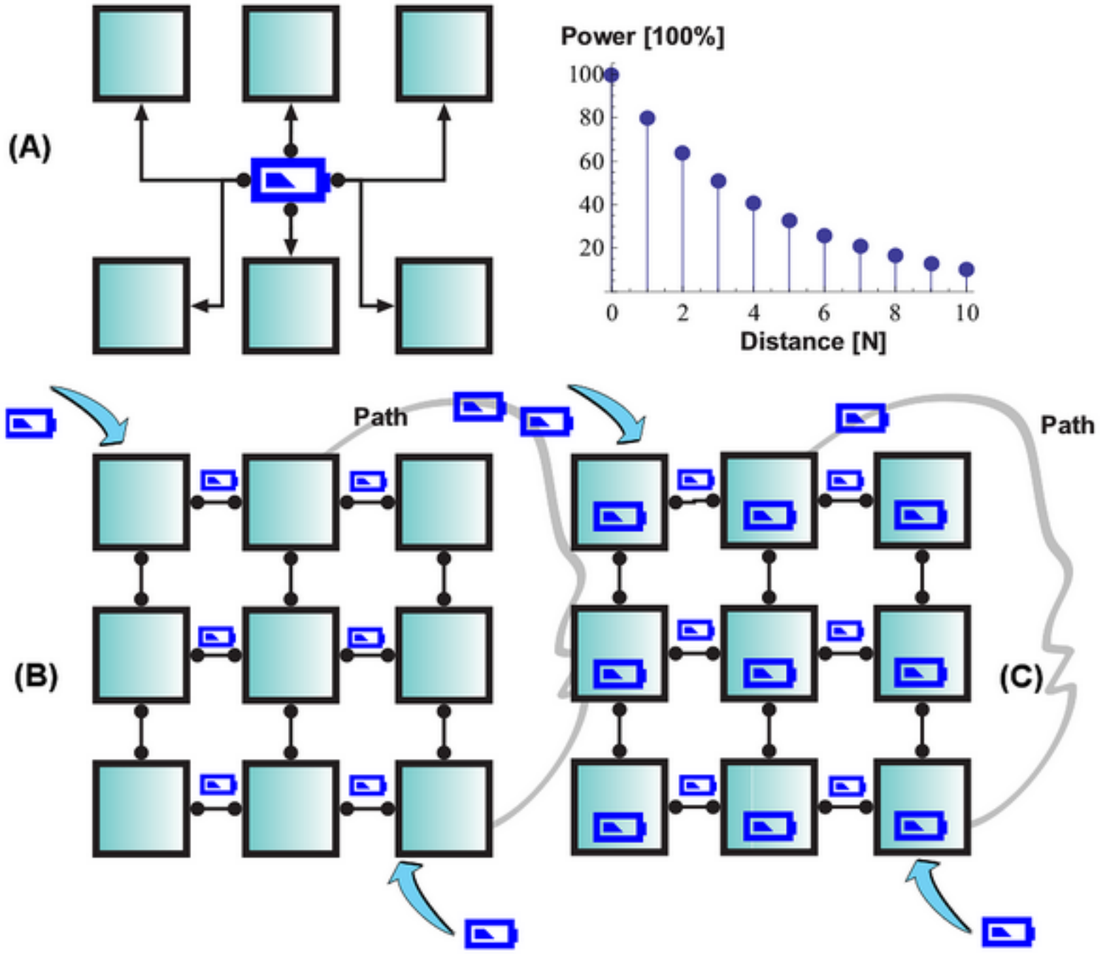


Fig. 3. Different energy supply approaches and efficiency

SYSTEM ENERGY

$$
\begin{aligned}
E_{node}(t) &= e_0(t_0) - \sum k_{decay}(t) \\
&\quad - \sum k_{comp}\tau(Ac_i) \\
&\quad - \sum k_{create} Ag_i(AC) \\
&\quad - \sum k_{comm} k_{link} \sigma(msg_i) \\
&\quad + \sum l_{conv} l_{link} e_{i,deliver} + \sum l_{conv} ha_i \\
E_{net}(t) &= \sum E_{node,i}(t)
\end{aligned}
$$

- e_0: initial node energy (deposit)
- k_{decay}: time-dependent energy decay parameter (energy losses in storage)
- $\tau(Ac_i)$: energy reduction by agent activity Ac_i
- $\sigma(msg)$: size of a communication message
- k_{comm}, k_{link}, k_{comp}, k_{create}: energy consumption parameters for communication and computation
- l_{conv}, l_{link}: energy conversion parameters (efficiency)
- e_i, ha_i: energy delivery and harvesting

ENERGY MANAGEMENT AND DISTRIBUTION

NODE CLASSIFICATION

Very bad node → $E < E_{Alarm}$

A node with very low energy, with restricted node operation (only agents arriving with energy are processed, only *help* emergency agents are sent out).

Bad node → $E_{Alarm} < E < E_{Thres1}$

A node with low energy, resulting in a basically normal node operation but with execution limits (number of agents, agent processing time, agent creation, agent migration, agent class restrictions, increased barriers of processing negotiation).

Good node → $E_{Thres1} < E < E_{Thres2}$

A node with normal energy deposit and a normal node operation state with some resource limitations. All agents are processed and the node agent can create any type of energy agents.

Very good node $E > E_{Thres2}$,

A node with very high energy and normal node operation; only a few or no resource limitations. All agents are processed and the node agent will only create *distribute* energy agents (if behaviour was enabled).

EXAMPLE OF ENERGY DISTRIBUTION

- Two-dimensional mesh grid
- Energy distribution between nodes under the control of mobile agents performing negotiation with node agents
- Goal: Decrease number of bad and very bad (non-operational) nodes

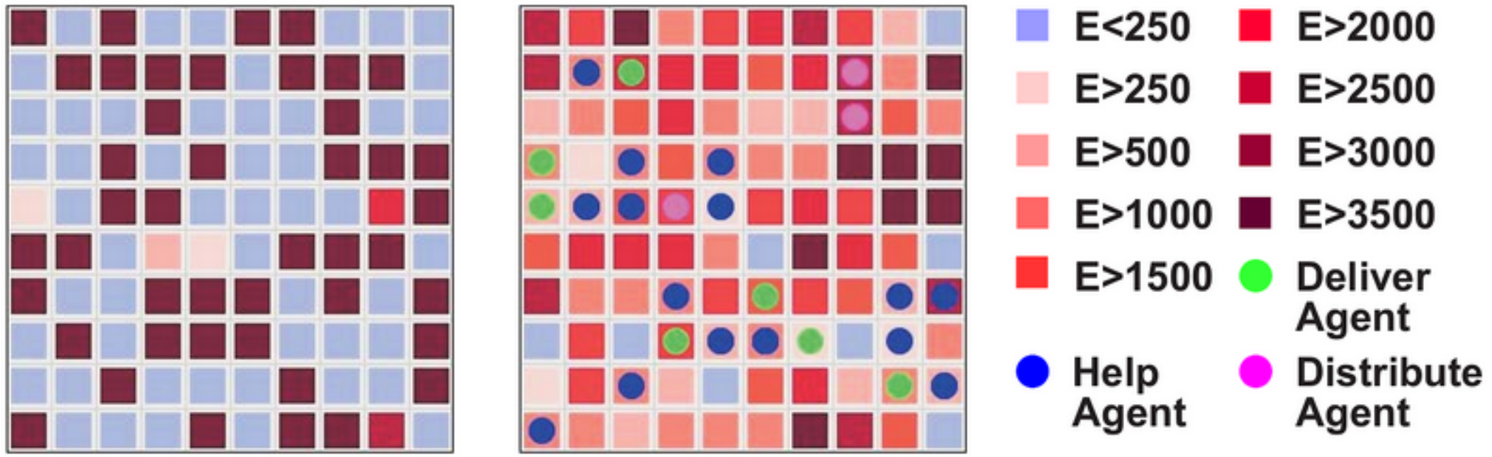


Fig. 4. Left: Energy distribution population map without SEM, Right: with SEM (after 10000 simulation steps)

MULTI-AGENT SYSTEM

Agents interact with each other by exchanging data base tuples or by creating agents

DIVIDE-AND-CONQUER

- There is **no centralized** instance monitoring energy and performing energy management (distribution)
- Each network node performs energy management (EM) within a spatially limited region → multiple instances
- **Global EM goals:**
  - Stability of system operation
  - No bad nodes
  - High availability
- **Emergence behaviour:** Efficient and equalized energy distribution
- **Basic principles** to achieve decentralized local EM with a global emergence:
  - Negotiation, Diffusion, Replication, and Inihibition
  - Self-organization
  - Self-connectivity
  - Self-adaptation
  - Self-healing

ENERGY TRANSPORT BY AGENTS

- Each time an agent that carry energy tokens arrives at a new node, the energy is stored in the node deposit and a virtual energy tuple is **produced** in the tuple space.
- If the agent continues traveling it has to **consume** the energy token again removing energy from the deposit (on sending).

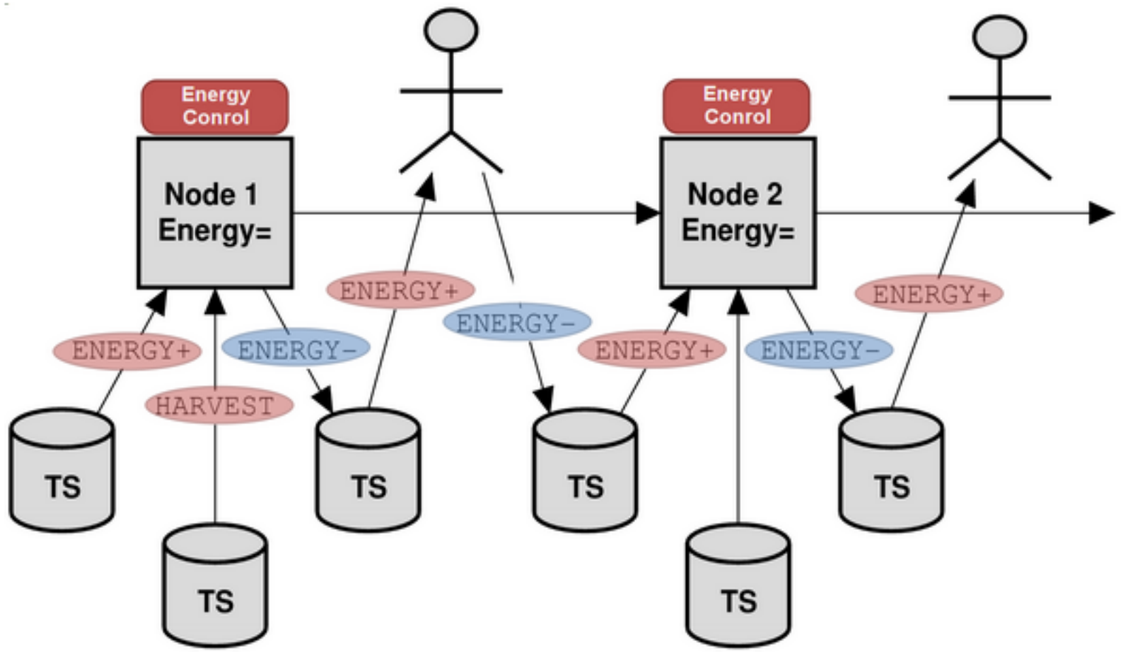


Fig. 5. Agent interaction with tuple spaces exchanging virtual energy tokens via tuples

AGENT BEHAVIOUR AND CLASSES

Node → Stationary Agent

This agent monitors the node state and power history. It has to initiate appropriate actions, i.e., creation of energy request, help, or distribute agents. The node agents has to asses the quality of the SEM locally and can change SEM strategies.

Request → Mobile Point-to-point Agent

This agent requests energy from a specific destination node, returned with a Reply agent. If the destination node cannot deliver energy (bad node), the request agent dies without a reply.

Reply → Mobile Point-to-point Agent

This agent is created by a Request agent, which has reached its destination node. This agent carries energy from one node to another.

AGENT BEHAVIOUR AND CLASSES

Help → Mobile Region Agent → Reactive

This agent explores a path starting with an initial direction and searches a good node having enough energy to satisfy the energy request from a bad node. This agent resides on the final good node (found by random walk within a region) for a couple of times and creates multiple deliver agents periodically in dependence of the energy state of the current node. If the current node is not suitable anymore, it travels to another good node.

Deliver → Mobile Path Agent → Reactive

This agent carries energy from a good node to a bad node (response to Help agent). Depending on selected sub-behaviour (HELPONWAY), this agent can supply bad nodes first, found on the back path to the original requesting node.

Distribute → Mobile Region Agent → Pro-active!

This agent carries energy from the source node to the neighbourhood and is instantiated on a good node. It explores a path starting with an initial direction and searches a bad nodes to supply them with the energy from the agent virtual energy deposit.

NEGOTIATION & MARKING

- To avoid high density of help and request agents on a specific neighbour node each help and request agent places **temporary markings** on the node indicating energy demand on this (good or very good) node (aka. synthetic pheromones) by other nodes.
- These markings are placed in the **tuple space** of the node and removed after a time-out automatically.
- If a new help or request agent arrives on a node and this node has a **strong marking** it will **continue traveling** (help behaviour) or **dies** (request behaviour).
- Each help and request agent **negotiates energy demand** with the node agent via the tuple space.

PARAMETER SET

» Behaviour of energy agent is parameterized:

$$
\begin{aligned}
&parameter : \{ \\
&\quad energy_1 : 50,\ energy_2 : 300,\ //\ operational\ energy\ range \\
&\quad energy_{Alarm} : 50,\ \ //\ e < energy_{Alarm} :\ very\ bad\ Node \\
&\quad energy_{Thres_0} : 100,\ //\ e < energy_{Thres_0} :\ bad\ node \\
&\quad energy_{Thres_1} : 200,\ //\ e > energy_{Thre_1} :\ very\ good\ Node \\
&\quad energy_{Deposit} : 50,\ //\ reservoir \\
&\quad energy_{Request} : 50,\ \ //\ def.\ energy\ to\ be\ requested \\
&\quad energy_{Distribute} : 20,\ //\ def.\ energy\ to\ be\ distributed \\
&\quad explorationRange : 4, \\
&\quad Life_{max} : 4,\ Hops_{max} : 8, \\
&\quad inhibitTime : 20,\ //\ inhibit\ request/help\ agents \\
&\quad energy_K : 0.95,\ \ //\ energy\ conversing\ efficiency \\
&\quad energy_{CommK} : 0.8,\ //\ energy\ transfer\ efficiency \\
&\quad sem : ['help'],\ //\ energy\ management\ strategy \\
&\}
\end{aligned}
$$

ALGORITHM

Algorithm 1. (*Energy Management*)

agent *energy* (*parameter*) =

if $energy_{node} < energy_{Thres_0}$ **then**
A : **choose** *neighbour node randomly* **goto** *node and* **request** *energy token*
if *neighbour node cannot deliver energy* **then**
choose *next node randomly by directed diffusion*
repeat (A) *until a region boundary is reached, or*
a good or very good node was found.
else
B : **send** *and* **deliver** *energy tokens repeatedly back to source node*
sleep ; **repeat** (B) *until this node lack of energy or end of life*
end
elseif $energy_{node} > energy_{Thres_1}$ **then**
C : **send** *and* **deliver** *energy tokens to randomly chosen neighboring nodes within a region*
sleep ; **repeat** (C) *until* $energy_{node} < energy_{Thres_1}$ *or end of life*
end

if *help on way* **then** *charge each bad node along path* **end**
end

ALGORITHM

Utility Function

- A utility function evaluates the effect of actions of an agent or a set of agents performed in the past on the environment and with respect to the goals of the MAS
- The utility function $u(\mathbf{run}) \in [0,1]$ used by the energy agents evaluates the efficiency of the energy distribution (action sequence $\mathbf{run}=\{a_i\}$) and influence the sub-behaviour selection and parameter settings

$$u(run) = \frac{\sum e_{delivered} - \sum e_{consumed}}{\sum e_{collected}}$$

AGENT PLATFORM(S)

All Agent Processing Platforms (APP) used in this work base on the same Activity-based Agent Programming Language Model (AAPL)

AAPL

- **Agent Behaviour:** Activity-Transition Graph (ATG)
  - ATG can be modified by agents
- **Activities:** Set of actions representing sub-goals
- **Actions:** *Computation, Communication, Replication, Mobility, Reconfiguration*
- **Communication:** *Tuple spaces, Signals*

OVERVIEW

- All AAPL-based platforms support mobile agents (representing mobile processes), agent reconfiguration (core morphing), and networking

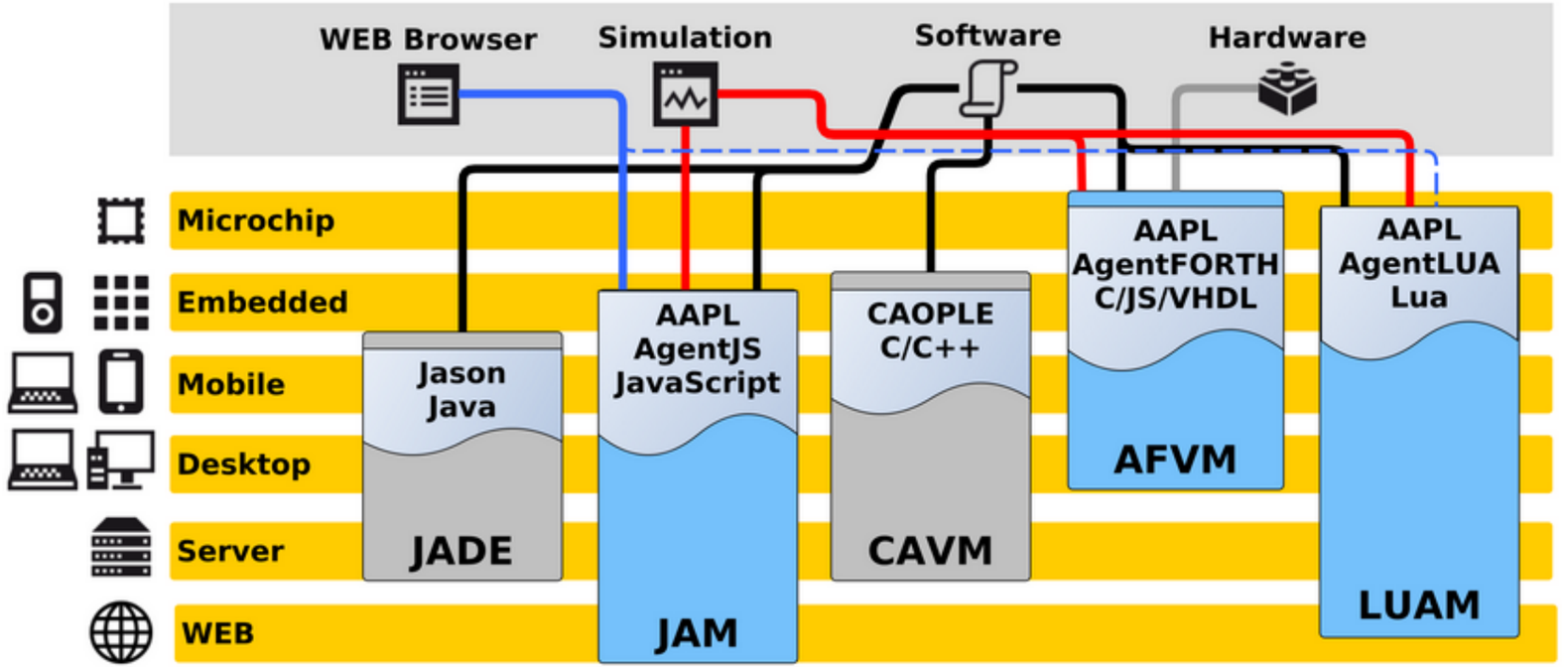
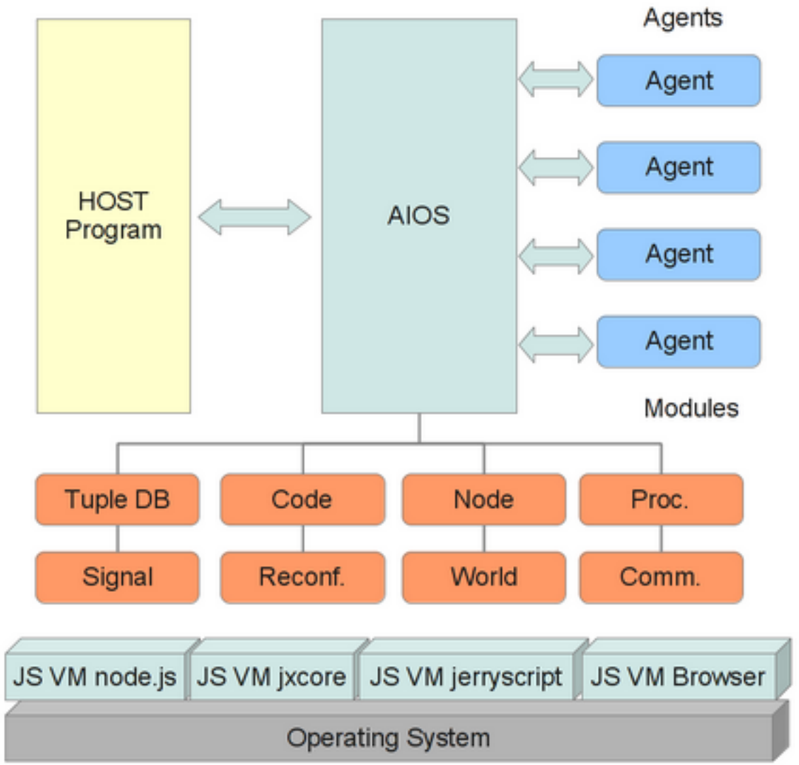


Fig. 6. Different agent processing platforms matching different layers: (Horizontal) Implementation (Vertical) Network domain and host platform

JAM

JAM: JavaScript Agent Machine

- JAM is implemented entirely in JavaScript
- Agents are implemented entirely in JavaScript (AgentJS)
- JAM can be executed on any JS engine (node.js, jxcore, jerryscript, spidermonkey, Browser,..)



- Tuple space is hierarchically organized (*arity* → *type signature* → *key hashtag*)
- Pattern matching of tuples in log_n time
- Agent execution in sandbox environment
- Security: Different agent roles (privileges); capability protection

JAM

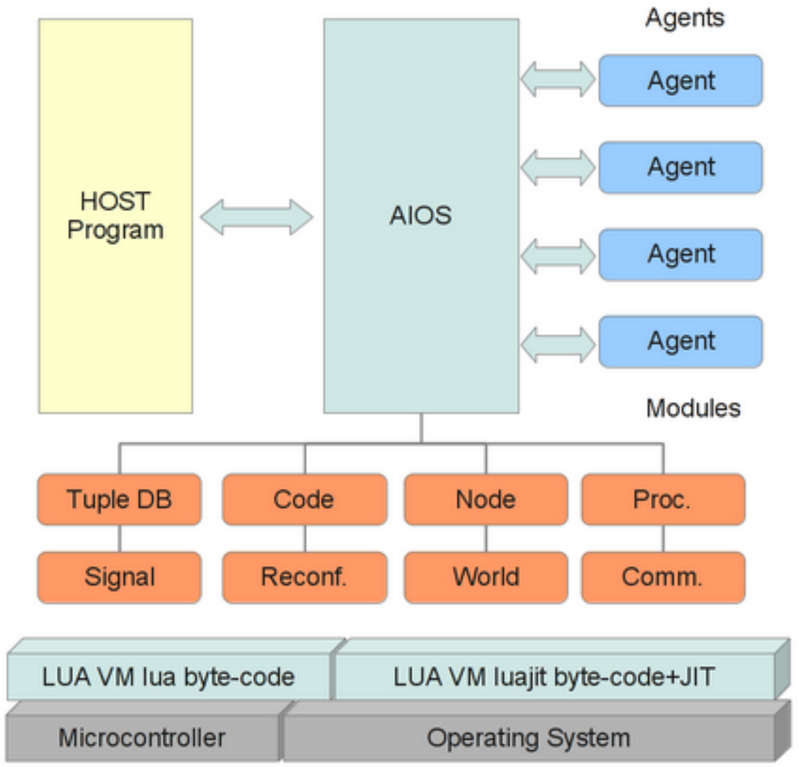
Definition 2. (*AgentJS Class Template*)

$$
\begin{aligned}
&\textbf{function}\ ac\ (p_1, p_2, ..)\ \{\\
&\quad this.\textbf{v}\ =\ \epsilon\\
&\quad this.\textbf{act}\ =\ \{\\
&\qquad a_1\ :\ function\ ()\ \{\ ..\ \},\\
&\qquad a_2\ :\ function\ ()\ \{\ ..\ \},\ ..\\
&\quad \}\\
&\quad this.\textbf{trans}\ =\ \{\\
&\qquad a_1\ :\ a_2,\\
&\qquad a_2\ :\ function\ ()\ \{\ return\ \epsilon\},\ ..\\
&\quad \}\\
&\quad this.\textbf{on}\ =\ \{\ ..\ \}\\
&\quad this.\textbf{next}\ =\ a_1;\\
&\}
\end{aligned}
$$

LUAM

LUAM: Lua Agent Machine

- LUAM is implemented entirely in Lua programming language
- Agents are implemented entirely in Lua (AgentLua)
- LUAM can be executed on *lua* (pure byte-code machine) and *luajit* (hybrid byte-code and just-in-time native code compilation)



- » Tuple space is hierarchically organized (*arity* → *type signature* → *key hashtag*)
- » Pattern matching of tuples in log_n time
- » Agent execution in sandbox environment
- » Security: Different agent roles (privileges); capability protecttion

LUAM

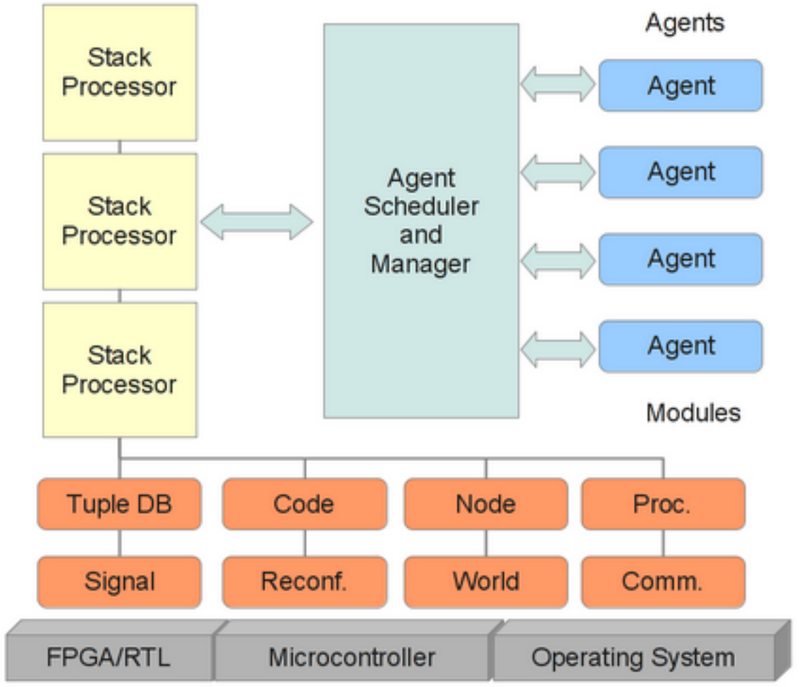
Definition 3. (*AgentLua Class Template*)

$$
\begin{aligned}
&ac \;=\; class()\\
&\textbf{function}\; ac : init\,(p_1, p_2, ..)\\
&\quad self.\textbf{v} \;=\; \epsilon\\
&\quad self.\textbf{act} \;=\; \{\\
&\qquad a_1 \;=\; function\,()\;..\;\; end,\\
&\qquad a_2 \;=\; function\,()\;..\;\; end,\;..\\
&\quad \}\\
&\quad self.\textbf{trans} \;=\; \{\\
&\qquad a_1 \;=\; a_2,\\
&\qquad a_2 \;=\; function\,()\; return\; \epsilon end,\;..\\
&\quad \}\\
&\quad self.\textbf{on} \;=\; \{\,..\,\}\\
&\quad self.\textbf{next} \;=\; a_1;\\
&end
\end{aligned}
$$

AFVM

AFVM: Agent FORTH Virtual Machine

- AFVM is implemented either in Software (C,JS,Java,..) or in **Hardware** (FPGA/RTL)
- AFVM is a multi-stack multi-processor executing FORTH code
- Agents are implemented entirely in FORTH and organized in code frames (AgentFORTH)



- » Token-based pipelined Multiprocessing
- » Code frames holding entire agent code and state (data)
- » Agent execution in strict sandbox environment controlled by processor
- » Agent processors share tuple spaces and agent management data base

AFVM

Definition 4. (*AgentFORTH Code Frame Template*)

par p_1 *integer*
par p_2 *integer*
..
var v *integer*
..
: * a_1 .. ;
: * a_2 .. ;
..
: f .. ;
: %**trans**
| a_1 .. .
| a_2 .. .
..
;
trans

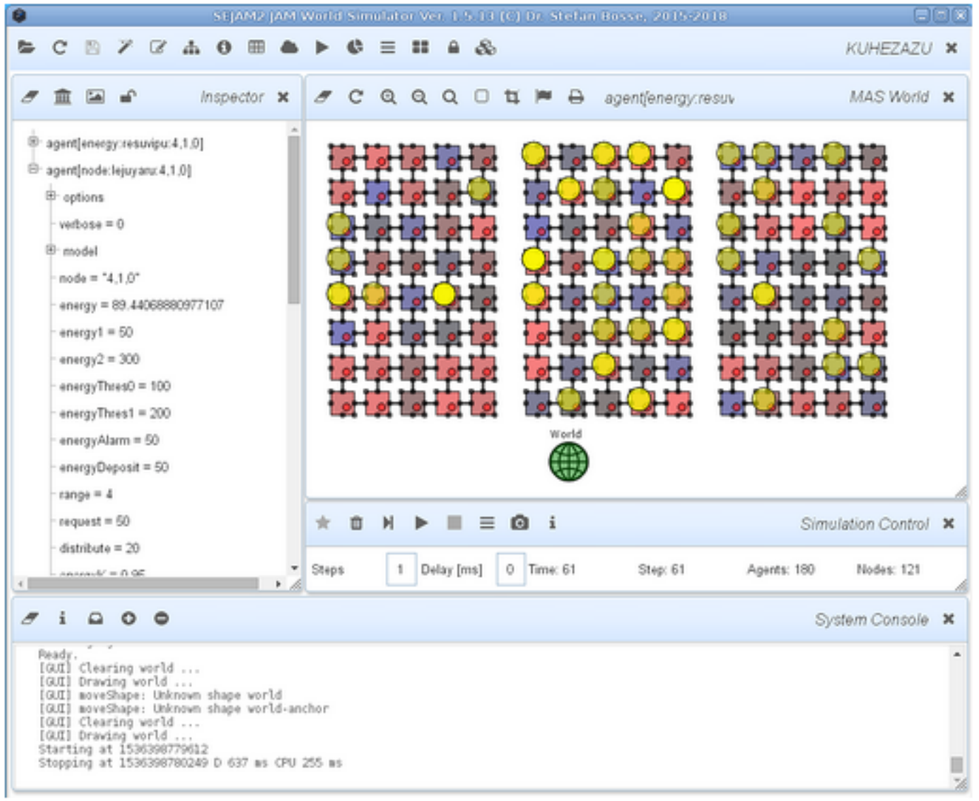
SIMULATION AND EVALUATION

SEJAM2

SEJAM: Simulation Environment for JAM

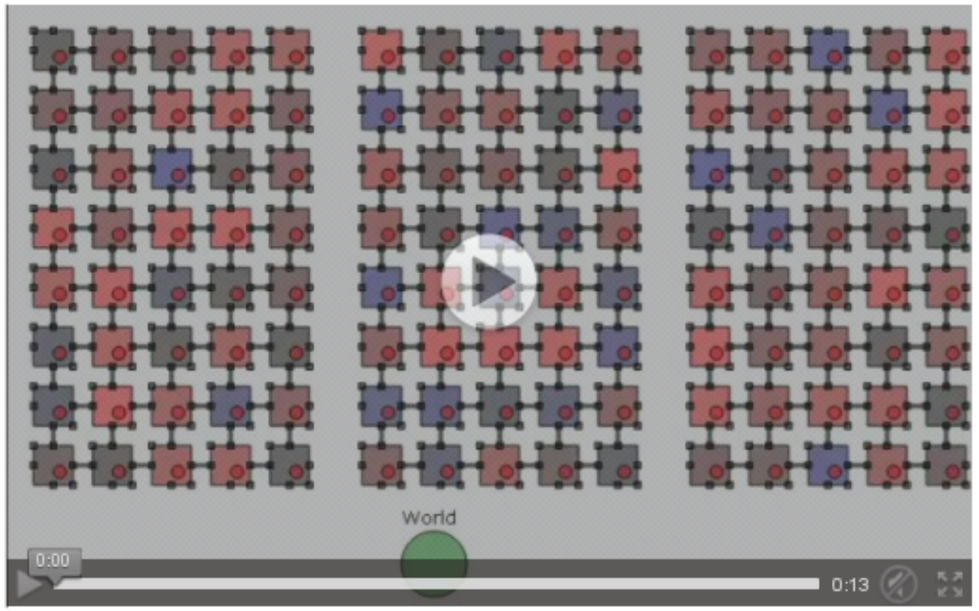
- SEJAM is a MAS simulation environment on top of JAM

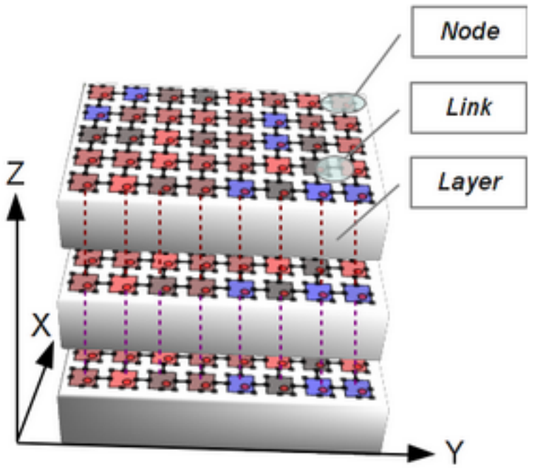
» SEJAM provides a GUI **simulation control** and a **graphical simulation world**
» JAM agents are extended by a **visual property**
» Simulations can be modeled with a JSON+ (extended code version) file
» SEJAM can be connected to real JAM networks (**hardware-in-the-loop** simulation)



LIVE

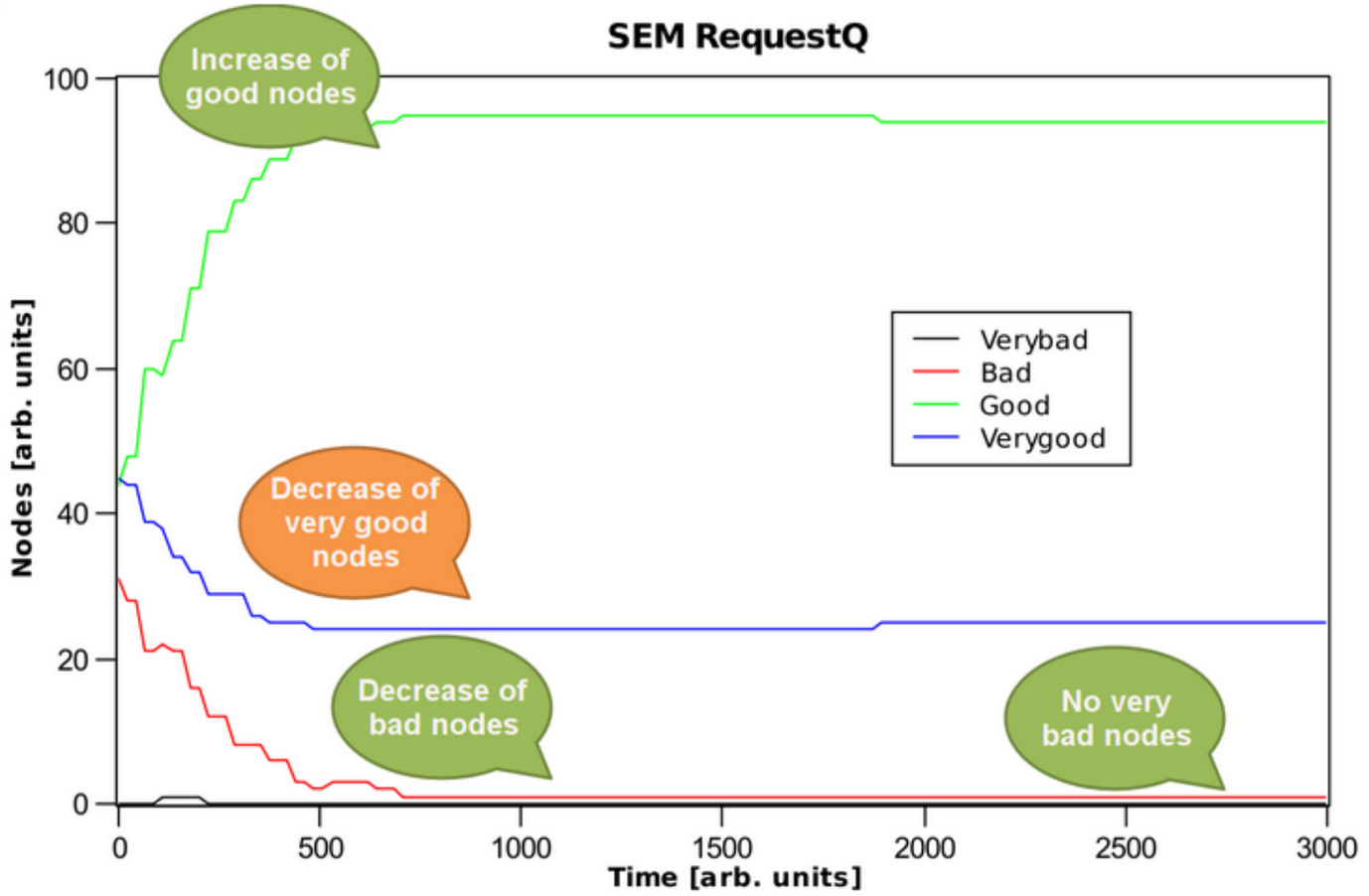
- Network: Three layers of 8x5 network nodes (3D mesh grid)
- Initial situation: 30% (very) bad nodes
- Energy agents are distributing energy between nodes
- Finally no bad nodes remain





REQUEST BEHAVIOUR

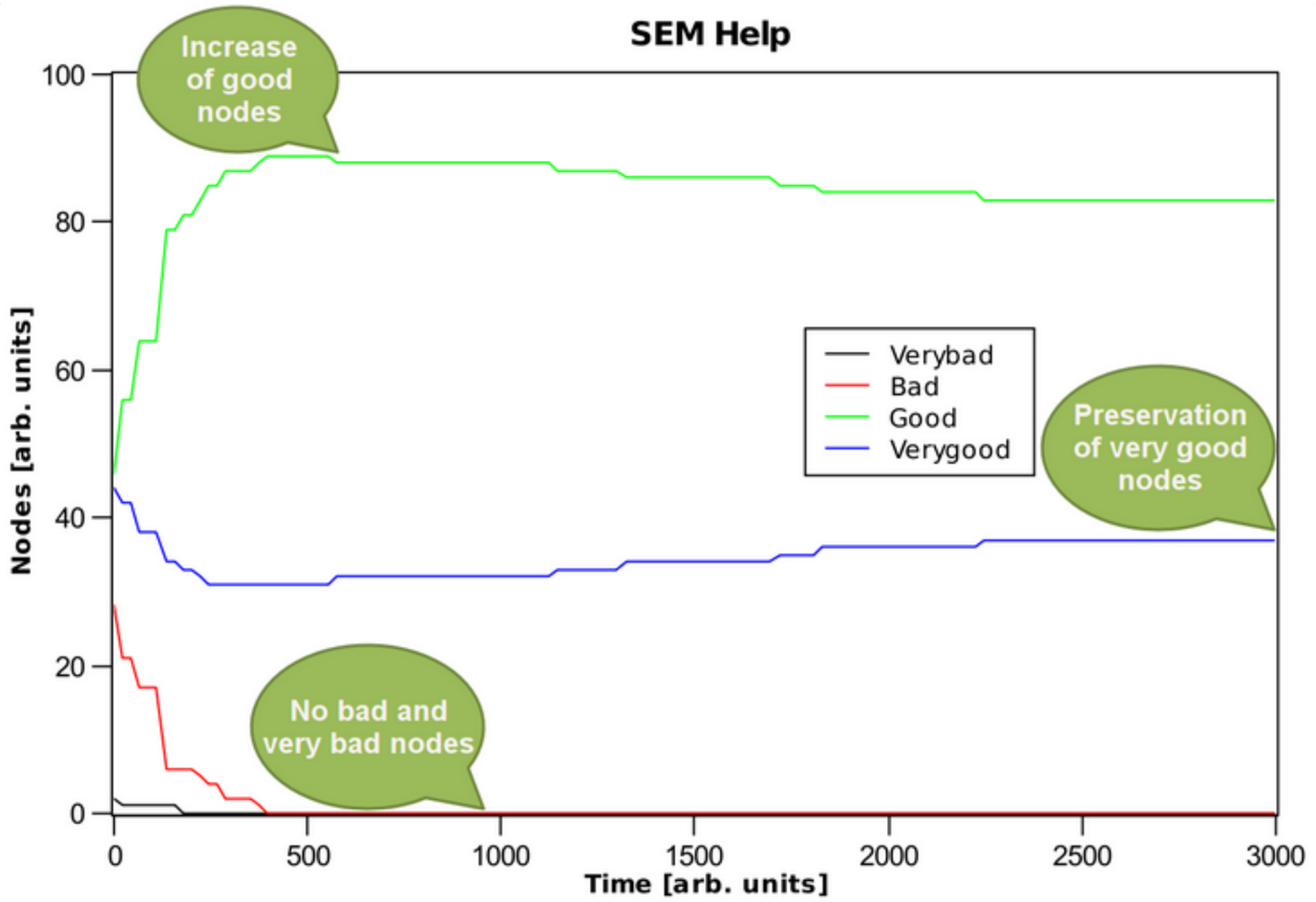
- Reactive request behaviour delivers overall good results



[120 nodes total]

HELP BEHAVIOUR

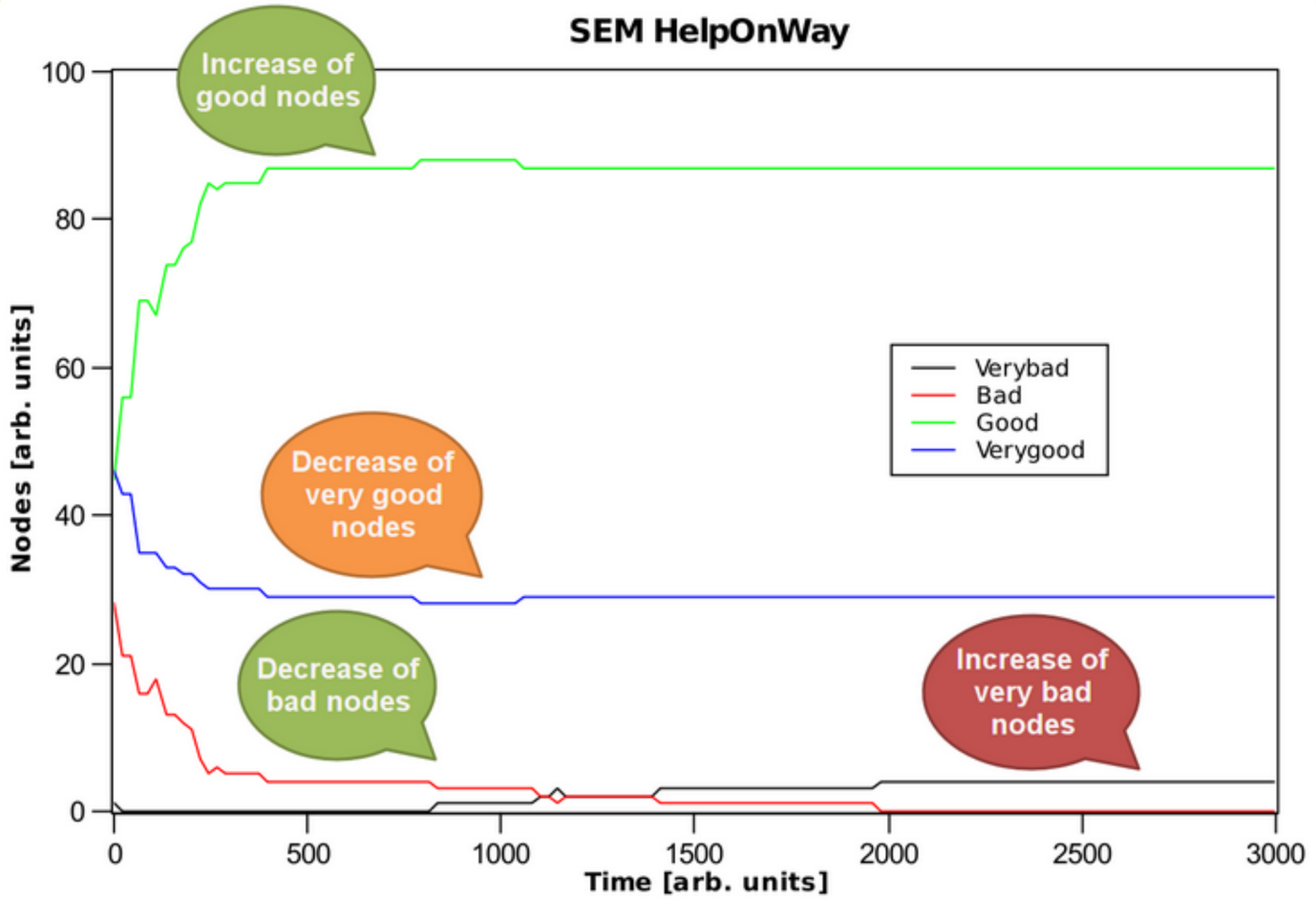
- Help behaviour delivers overall best results (fast convergence)



[120 nodes total]

HELP-ON-WAY BEHAVIOUR

- Help-on-way behaviour can create very bad nodes (non-operational)

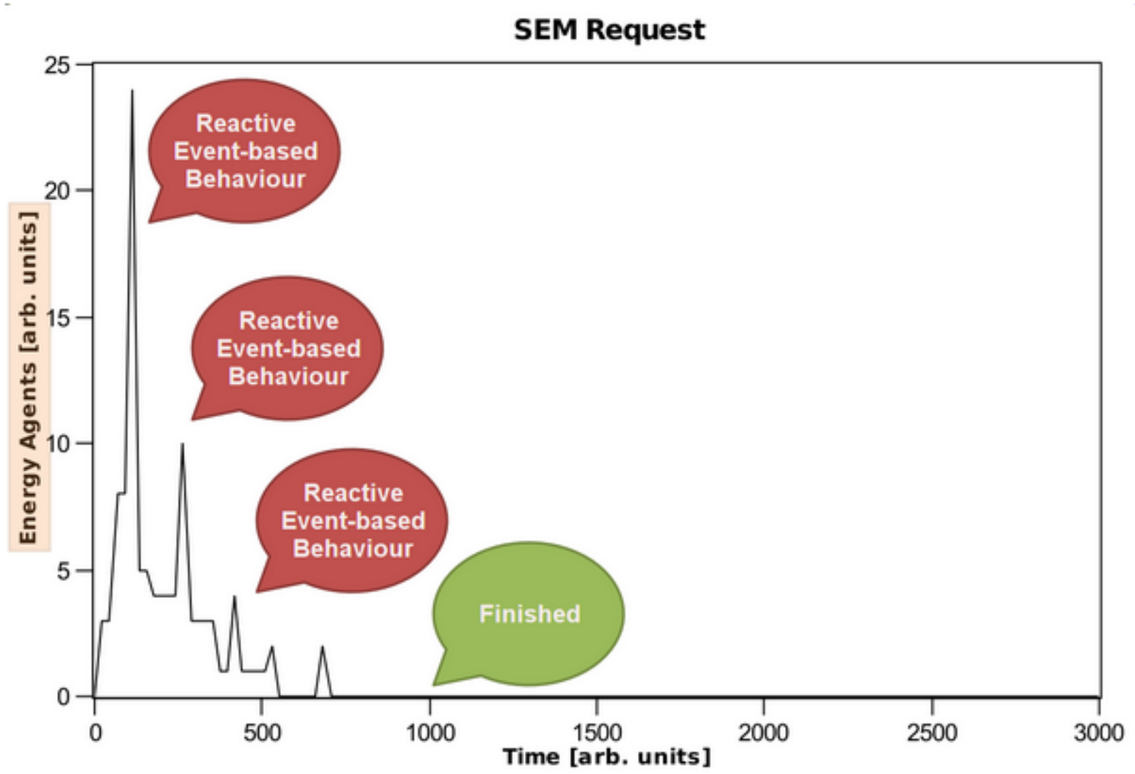


[120 nodes total]

EVENT-BASED BEHAVIOUR

The entire MAS is **self-organized and self-managing** based on:

- energy perception,
- reactivity → event-based,
- inhibition,
- replication,
- diffusion, and
- negotiation.



CONCLUSION

1. Decentralized goal-driven energy distribution was performed with mobile Multi-agent Systems
   - » Agent are able to carry (virtual) energy tokens
   - » Energy can be exchanged between nodes based on agent negotiations
   - » Local behaviour ⇒ Global Emergence (E.g., goal to minimize bad nodes)
2. No particular system, energy, or network models are required → **model-free approach**
3. Agents were implemented in this work in JavaScript and executed by the JavaScript Agent Machine (JAM)
   - » JAM can be executed on a wide range of devices and host platforms (embedded computer .. server)
4. One major field of application (but not limited to): Dynamic Sensor Networks with ad-hoc connected and self-powered autonomous nodes