DOI: 10.15439/2018F282
ISSN 2300-5963 ACSIS, Vol. 15

Smart Micro-scale Energy Management and Energy Distribution in Decentralized Self-Powered Networks Using Multi-Agent Systems

Stefan Bosse
University of Koblenz-Landau
Faculty Computer Science, Institute of Software Technology,
Koblenz, Germany
Email: sbosse@uni-bremen.de

***Abstract*—Energy distribution as a main part of energy management in self-powered micro-scale networks like sensor networks is a challenge with the goal to satisfy a safe and reliable operational state on system and node level. Under the assumption that nodes are arranged in mesh-like networks with links posing the capability to transfer data and energy between nodes a self-organizing Mulit-agent System based on divide-and-conquer is deployed in this work successfully to distribute energy without a system/world level model and knowledge of single nodes about the system state. Different agent behaviour were investigated and the emergence evaluated. An exploring help strategy with energy deliver child agents showed the best and efficient overall behaviour. Mobile agents were programmed in JavaScript using the JavaScript Agent Platform that can be deployed in strong heterogeneous environments.**

I. Introduction

AMONG energy supply and consumer networks on a macro-scale level there are sensor networks with self-powered sensor nodes consisting of an energy storage and energy supply. Both classes of networks require distributed, adaptive and self-organizing energy management to satisfy (1) A balance of energy supply and energy consumption, and (2) Operational stability on system level [1]. The energy management addressing the control of consumption and production is basically a distributed resource sharing and scheduling problem [2]. Sensor networks consists of nodes optionally equipped with an energy harvester collecting energy from the environment from different sources posing varying availability that cannot be controlled, in contrast to macro-scale energy sources (power engines, ..). Power management and energy harvesting are central issues in sensor networks [3]. Additionally, self-powered nodes can be supplied by external energy sources not directly attached to the node using other nodes to transfer energy. Nodes in a sensor network can use communication links to transfer energy, for example, optical links are capable of transferring energy using Laser or LE diodes in conjunction with photo diodes on the destination side, with a data signal modulated on an energy supply signal.

Typically, energy management is performed by a central controller on software level [1], with limited fault robustness and the requirement of a well-known environment world model for energy sources, sinks, and storage. In a centralized approach, energy is only transferred on node level. With a distributed approach, energy management in a network involves the transfer of energy between node instances, too. In [4], multi-agent systems (MAS) are used to perform energy management (between sinks and sources) in a renewable energy grid by solving a (global) optimization problem by the MAS. In [1], a MAS performs energy distribution by a token-based approach solving an optimization problem, too. In [5], MAS based on the Belief-Desire-Intention architecture (BDI) are deployed in distributed sensor networks to perform goal and knowledge orientated energy management (but without considering energy distribution). These examples pose the benefit of agent-based systems solving energy management and distribution in large-scale networks. Agents are already deployed in industrial applications and the Industrial IoT [6].

We propose a smart energy management and distribution approach for a broad range of applications ranging from micro-scale sensor network architectures to large-scale energy networks with nodes supplied by 1. energy collected from a local source (energy harvesting, [3]), and 2. by energy collected from neighbour nodes using smart energy management (SEM) and self-organization, based on early work investigating primarily technological aspects in sensorial materials and agent-on-chip hardware architectures [7]. For the sake of simplicity, nodes are arranged in a n-dimensional grid with connections to their direct neighbours, i.e., in a three-dimensional network there exist up to six connections in directions North, South, East, West, Up, and Down. It is assumed that the network is irregular (missing nodes) and incomplete (missing links). Each node can store collected energy and distribute energy to neighbour nodes via communication links.

Each autonomous node provides communication, data processing, and energy management. There is a focus on single System-On-Chip (SoC) design satisfying low-power and high miniaturization requirements addressing the micro-level as well as macro-scale networks with power supplies and consumers.

Energy management is performed 1. For the control of local energy consumption, and 2. For collection and distribution of energy by using the data links to transfer energy.

Considering strong heterogeneous networks and host platforms with a loosely coupling of nodes arranged in grids the distributed data processing is a challenge. Multi-agent systems (MAS) poses a distributed computation and communication model providing autonomy, self-organization, and group behaviour. MAS are already deployed in energy management and energy distribution systems**[8][9][1]**. Most published MAS perform negotiation between energy producer and consumer under varying environmental, node and system level states. The negotiation results effect and control energy production and consumption. In contrast to public energy markets, self-powered sensor networks do not provide such a control as energy harvesting is strongly influenced by not controllable environmental conditions (e.g., sun shine duration influencing solar cell harvesting efficiency).

This work investigates and evaluate the emergence behaviour of self-organizing energy management agents that are capable to transfer (virtually carry) energy between nodes of networks and that are deployed in large-scale decentralized energy supply and consumer networks under resources and reliability constraints, e.g., self-powered sensor networks**[10]**. The desired emergence is the efficient improvement of the energy distribution in such networks to satisfy operational stability of the entire network on system level and in bounded regions, i.e., avoiding nodes with too low energy being operational. Agents perform decision making based on actual and past node energy, reward, and interaction with other agents.

The next sections introduce the underlying energy model on node level, energy management and distribution, the MAS and the agent processing platform (APP), finally used in an evaluation of a large-scale network simulation.

II. Energy Model

There is no system level model in this work considering only the node level energy. The total energy of a network node is a balance of energy loss due to computation, communication, and agent creation, and energy harvesting via energy delivery by agents and local energy harvesters (power supplies). The energy balance equation is shown in Eq. **1.** and used throughout this work.

$$\begin{aligned} E_{node}(t) \quad = \quad & e_0(t_0) - \sum k_{decay}(t) \\ & - \sum k_{comp}\tau(Ac_i) \\ & - \sum k_{create}Ag_i(AC) \\ & - \sum k_{comm}k_{link}\sigma(msg_i) \\ & + \sum l_{conv}l_{link}e_{i,deliver} + \sum l_{conv}ha_i \end{aligned} \tag{1}$$

It is assumed that the time variable of the energy E is a discrete variable with a time resolution between milliseconds and seconds. The initial energy is e_0 with a time-dependent decay k_{decay} due to losses in the energy storage of a node. The energy is reduced by agent activity Ac_i(with $\tau(Ac)$ as the execution time of an agent activity scaled by a parameter k_{comp}), agent creation $Ag(AC)$ from class AC (can be expressed by a computational time, too). Communication further requires energy and depends on the size of the message$\sigma(msg)$ scaled with the parameter k_{comm} and the link specific energy consumption given by the parameter k_{link}. Finally, energy is increased by agents delivering energy via the communication links (amount can vary, and energy conversion losses are covered by the parameter l_{conv} and the loss of the link by l_{link}) and from local power sources (ha, again covering conversion losses by the parameter l_{conv}).

Nodes are classified by their *energy deposit* and operational state:

1. A node with very low energy $E{<}E_{Alarm}$, with restricted node operation (only agents arriving with energy are processed, only help emergency agents are sent out).
2. A node with low energy $E{<}E_{Thres1}$, 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).
3. A node with normal energy deposit $E{<}E_{Thres2}$ 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.
4. A node with very high energy $E{>}E_{Alarm}$, with node operation being normal with 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).

For the sake of simplicity a three-dimensional grid network connecting spatial neighbour nodes is assumed, shown in Fig. **1**. Connected nodes can exchange data and energy. The shown example network consists of three layers (z-axis, levels) and each layer consists of 5x8 nodes. Each node has any time t an energy deposit $0{>}E(t){>}E_{max}$, illustrated in Fig. **1** by different colors (blue: low energy, red: high energy).

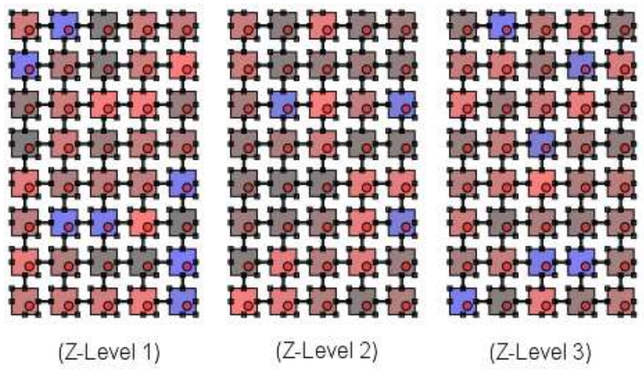

Figure 1: Example of an energy distribution in a three-dimensional network. Squares: Nodes (red color indicates enough node energy - good node - to be operational, blue color indicates critical low energy capacity - bad node), Circles: Agents, Lines: Communication and Energy links. The different z-levels are connected by up- and down links.

III. Energy Management and Distribution

In this work mobile agents perform energy management and distribution, discussed in detail in Sec. IV. The energy management relies on three main principles:

A. Adaptive Routing

Agents are responsible to find an appropriate path between a source and a distinct destination node by using adaptive routing or by using data centric routing linking an information (energy) supplier and an information (energy) sink.

B. Energy Transport

Mobile agents can carry energy tokens being able to be transferred between nodes. An energy token is requested on a node (granted or negotiated by the node agent) and can be migrated to other nodes. Each time an agent arrives on a new node it delivers its energy to the node energy deposit. If the agent has to transport the energy token to another node the agent has to collect the energy token again reduced by some technical conversion losses, shown in Fig. **2**. This approach is reasonable with respect to a technical representation of energy transfer in networks via communication links.

C. Negotiation

To avoid a high density of help and request agent populations 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 agents arrives on this nodes 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 (by accessing active tuples using the evaluate operation and placed by the node agent using a listen operation).

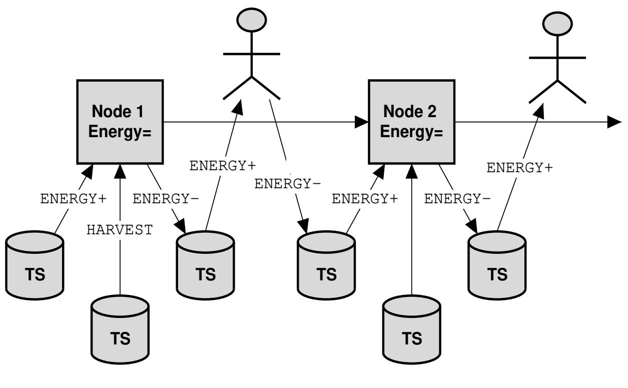

Figure 2: Energy transport by agents: Each time an agent carrying energy tokens arrives at a new node the energy is stored in the node deposit and a virtual energy tuple is stored in the tuple space. If the agent continues travelling it has to collect the energy token again.

D. Energy Management

The following parameters are used for the dynamic energy management (which can be changed at run-time) performed by the MAS:

Energy Thresholds: $E_{Alarm}<E_{Thres0}<E_{Thres1}$, $E_{Deposit}$
Energy Transfer Tokens: E_{Req},E_{Dist}
Efficiencies: k_{Conv},k_{Comm}
Exploration Radius: r_{expl}
Agent Lifetime: τ_{max}
Maximal hop-count: h_{max}

IV. MAS

The Multi-agent system (MAS) used for distributed energy management and energy distribution consists of different agent classes posing different behaviour and goals. Each network node part of the distributed energy management system provides an agent processing platform (APP). At least one stationary (non-mobile) node agent is created on each node to initialize the sensor processing and energy management. Based on the current state of the node and the power history of the node the node agent will create further energy management agents, summarized in Tab. **I**. All other energy management agents are mobile and can migrate (travel) along a path in the network crossing multiple nodes.

The node management agent can choose different energy management strategies: *Help*, *request*, and *distribute*. Depending on the operational and energy state of the node one or multiple strategies are applied to improve the energy deposit.

The different behaviour of the energy management agent is shown in the activity-transition diagram in Fig. **3**. Different parameters have an impact on each behaviour at run-time. Each agent owns a set of body variables, e.g., a delta vector indicating the position in the network, a charge and energy variable for distributed energy management. Most central parameters are the current positionΔ(relative to a source node) and the energy deposit of the current node *E* that is retrieved via the tuple space and provided by the node agent that interacts with power components via a HAL of the physical node. Energy agents are either created by the node agent (*help*, *request*, *distribute*) or by already created energy agents (*reply*, *deliver*).

Each agent can carry virtual energy as outlined in Sec. III. Since agents are mobile they can transfer energy from a node *A* to a new node *B* via the communication link (or any other power link between these nodes) just by migration. The energy transfer is handled by the APP on migration if the*charge* agent body variable holding a mobile energy token is greater than zero. Before travelling the *charge* value is transferred to the *energy* body variable of the agent to store the energy virtually. After the arrival on a new node the value of the*energy* variable is transferred back to the *charge* variable. There are now two possible cases: The agent delivers the energy finally on the current node (technically the energy was already transferred to the local energy storage), or it continues travelling and transfers the energy again to the next node via links.

Agent Class	Behaviour
`Node`	This stationary 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`	*Point-to-point agent*: This mobile 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`	*Point-to-point agent*: Mobile reply agent created by a `Request` agent, which has reached its destination node. This agent carries energy from one node to another.
`Help`	*ROI agent*: This mobile 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`	*Path agent*: This mobile 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`	*ROI agent*: This mobile 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.

Table I: SEM agents with different behaviour used to manage and distribute energy in bounded regions (ROI: region of interest) based on negotiation.

The movement of mobile agents are constrained by three parameters: The maximal hop count, the maximal mobility radius relative to the source node, and a maximal lifetime. The constrained mobility ensures a relaxation and limitation of the population of the MAS after a stimulus occurred, i.e., energy decrease or increase that can trigger the creation of energy agents.

The request and distribute strategies are the most simple ones that can be performed by bad and good nodes, respectively. The help strategy is more advanced usually performed by bad and very bad nodes. A variation of the help strategy adds help-on-way behaviour performed by the deliver agents to charge bad nodes on the way back to original requesting bad node, too.

Agents perform decision making based primarily on the node energy class (bad/good), but also on actual and past node energy recording, reward and utility feedback for charging their home node, and interaction with other agents.

V. Agent Platform

In this work agents are programmed and implemented in *JavaScript* using the *JavaScript* Agent Machine platform (*JAM*) and the *AgentJS* programming language used for the implementation of the state-based reactive agents.

In the considered use-case scenario the MAS is deployed in a large-scale strongly heterogeneous network environment, which can be additionally extended with mobile devices. This heterogeneous network requires a unified agent processing platform (APP), which can be deployed on a wide variety of host platforms, ranging from embedded devices, mobile devices, to desktop and server computers. E.g., some measuring stations are attached to buoy or installed on small islands, equipped only with low-power low-resource computers. To enable seamless integration of mobile MAS in Web and Cloud environments, agents are implemented in *JavaScript*(*JS*), executed by the *JS* Agent Machine (*JAM*), implemented entirely in *JS*, too. *JAM* can be executed on any *JavaScript* engine, including browser engines (Mozilla's *SpiderMonkey*), or from command line using*node.js*(based on *V8*) or *jxcore*(*V8* or *SpiderMonkey*), or a low-resource engine *JVM*. The last three extend the *JS* engine with an event-based (asynchronous using callback functions) IO system, providing access of the local file system and providing Internet access. But these*JS* engines have high resource requirements (memory), preventing the deployment of *JAM* on low-power and low-resources embedded devices. For this reason,*JVM* was invented. This engine is based on *jerryscript* and *iot.js* from Samsung, discussed in **[11]**. *JVM* is a Bytecode engine that compiles*JS* directly to Bytecode from a parsed AST. This Bytecode can be stored in a file and loaded at run-time. *JVM* is well suited for embedded and mobile systems, e.g., the Raspberry PI Zero equipped with an ARM processor.*JVM* has approximately 10 times lower memory requirement and start-up time compared with *nodes.js*.

JAM consists of a set of modules, with the Agent Input Output System (*AIOS*) module as the central agent API and execution level. The deployment of agents in the Internet requires an additional Distributed Organization Layer (*DOS* with capability-based security). *JAM* is available as an embeddable library (*JAMLIB*). The entire *JAM* and *DOS* application requires about 600kB of compacted text code (500kB Bytecode), and the *JAMLIB* requires about 400kB (300kB Bytecode), which is small compared to other application programs and commonly used Java-based platforms like *JADE*/*AgentSpeak*. *JVM+JAMLIB* requires only 3 MB total RAM memory on start-up.

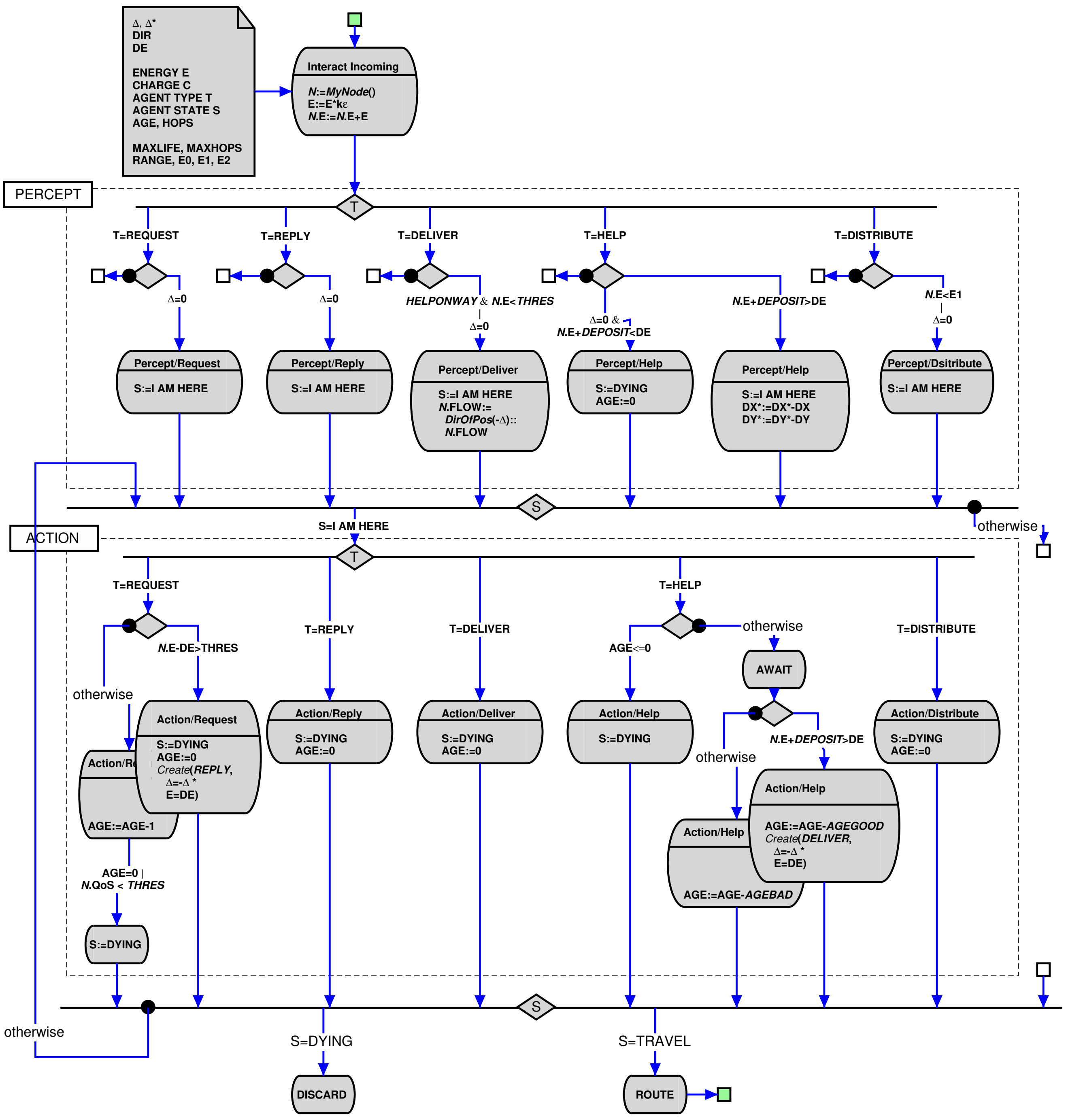

Figure 3: Principle activity diagram of the Energy Agent behaviour. There are five different agent classes (or sub-classes of the energy management agent) differing in their behaviour: Request, Reply, Help, Deliver, and Distribute.

JAM is capable of handling thousands of agents per node, supporting virtualization and resource management. JAM agents can migrate between different (physical) node APPs supporting true agent mobility with process snapshots preserving and embedding the entire agent state with low-resource overhead. Depending on the used*JS* VM, agent processes can be executed with nearly native code speed. *JAM* provides Machine Learning as a service that can be used by agents. The agent only saves a learned model, but not the learner code. Agent interaction and synchronization is provided by exchanging data tuples stored in a tuple space on each*JAM* node.

The agent behaviour is modelled according to an Activity-Transition Graph (ATG) model. The behaviour is composed of different activities representing sub-goals of the agent, and activities perform perception, computation, and inter-action with the environment (other agents).

Agent interaction (communication) is performed by using tuple spaces and mobile signals (point-to-point or point-to-N messages). Using tuple spaces is a common approach for agent communication, as proposed by**[12]**, much simpler than **[13]** proposed with *AgentSpeak*. The transition to another activity depends on internal agent data (body variables). The ATG is entirely programmed in *JavaScript* (*AgentJS*, see **[14]** for details). The ATG can be modified by agents at run-time enabling code morphing and optimization (behaviour adaptation or sub-classing).

JAM agents are mobile, i.e., a snapshot of an agent process containing the entire data an control state including the behaviour program, can migrate to another *JAM* platform. *JAM* provides a broad variety of connectivity, available on a broad range of host platforms. Although *JAM* is used in this work as a simulation platform in the *SeJAM* simulator only, it is ready to use in real-world networks and is capable to execute thousands of agents. The *SeJAM* simulator is built on top of a *JAM* node adding simulation control and visualization, and can be included in a real-world closed-loop simulation with real devices. Since *JAM* can be embedded in any host application and its capability to be easily extended enables the binding of *JAM* to low-level power management and technical energy components (conversion, storage, transfer). A hardware abstraction layer (HAL) enables the access of the power components by agents completely via the tuple space or by extended *AIOS* functions.

VI. Simulation and Evaluation

The simulation was carried out with the *SeJAM* simulator and a three-dimensional grid network consisting of three z-levels (layers), 8 rows, and 5 columns, as already shown in Fig.**1**. Each node of the network is a virtual *JAM* instance connected with up to six neighbouring nodes via serial links. Each node is associated with a virtual energy storage and energy harvester. The links are capable to transfer data and energy as introduced. Each node is populated with at least one node agent. An artificial world agent controls the simulation, perform monitoring, and reforms Monte Carlo simulation of the energy harvesting, and the initial start condition with respect to the initial energy deposit of each node. The randomized energy distribution assign nodes with an initial energy storage in the range [e_1=50,e_2=300](arb. energy units). Depending on the energy threshold settings there is initially a fraction of bad and very bad nodes about 20% of the total number of 120 nodes. In periodic intervals the nodes are charged with randomized energy amounts in the interval [0,e_Δ=0.3].

Each simulation run consists of 3000 simulation steps (in all considered cases the SEM converged either during this simulation range or never).

A typical parameter set used by the MAS is shown below.

```
parameter:{
 energy1:50, Energy range of nodes
 energy2:300,
 energyAlarm:50,   e < eAlarm: Very Bad Node
 energyThres0:100,  e < eThres0: Bad Node
 energyThres1:200,  e > eThre1: Very Good Node
 energyDeposit:50,  Reservoir
 energyRequest:50, Def. en. to requeste
 energyDistribute:20, Def. en. to distribute
 explorationRange:4,
 maxLife: 4,
 maxHops: 8,
 Inhibit request/help agent send out
 inhibitTime: 20,
 Internal energy conversing efficiency
 energyK: 0.95,
 energyCommK: 0.8, Energy transfer efficiency
 sem: ['help'],  Energy management strategy
 doHarvest:true,
 harvest:0.3,
 cpuK:0.3,
 createK:3,
}
```

One important outcome of the simulation was the observation that the emergent behaviour on system level depends on the starting condition of the network, i.e., the initial energy distribution. That means the result of the MAS operation can vary significantly under different situations discussed below.

Typical examples of the run and progress of different energy management strategies with respect to the node classification population (very bad, bad, good, very good) are shown in Fig. **5**. Without SEM (not shown), there is commonly no change in the network situation, i.e., the number of bad nodes (typically about 20%) remains unchanged. Using SEM, the MAS is capable to decrease the fraction of all bad nodes below 1%. All three SEM strategies request, help, and help-on-way, show a fast convergence and reaching of the goal to minimize bad nodes but still preserving a high amount of good and very good nodes. The help behaviour has the fastest convergence time, usually eliminating all bad and very bad node states, whereas the request behaviour has a slower convergence time. The help-on-way behaviour can create a remaining fraction of bad and very bad nodes and seems not be appropriate to satisfy the system level goal.

Typical variations of the run and progress of different energy management strategies are shown in Fig. **6**. The request behaviour poses a lower stability in the final outcome of the MAS than the help and help-on-way behaviour. Although help-on-way seems to be more reliable, it tends to create very bad node cluster as shown in Fig. **4**.

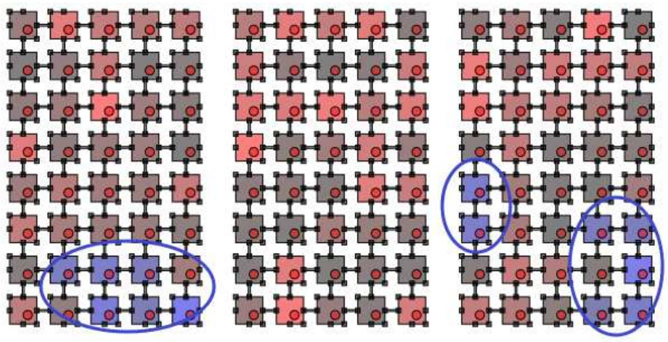
Figure 4: Formation of isolated very bad node clusters

One important measure is the temporal development of energy agents during a SEM run and the total number of created energy agents (help, request, distribute, deliver, reply), shown in Fig. **7**. The help behaviour performs optimally (both concerning the convergence time and the number of agents required). The help-on-way behaviour shows the aforementioned instability and missing convergence.

Fig. **7** summarizes the evaluation of the impact of different SEM agent parameters (parameter sets *B*-*F*). The parameter sets are explained in App. A. Parameter sets *B*-*E* are used to investigate the help MAS behaviour with different maximal help agent life times (staying on a good node and sending out deliver agents). Increasing the lifetime increases the total number of created energy agents without decreasing the fraction of bad nodes significantly. A fraction about 4% still remains (with large variations). But increasing the exploration range and the maximal number of agent hops results always (regardless of the initial start condition) in 0% bad nodes!

Finally, the energy efficiency of the MAS (defined as the fraction of start+harvest energy/final energy) was analyzed and is shown in the center plot of Fig. **8**. All parameter sets show a high efficiency between 87-92%.

The energy equation Eq. **1.** is updated during simulation about every 50 simulation step providing a sufficient smooth change of *E* based on updated perception and energy harvesting activities.

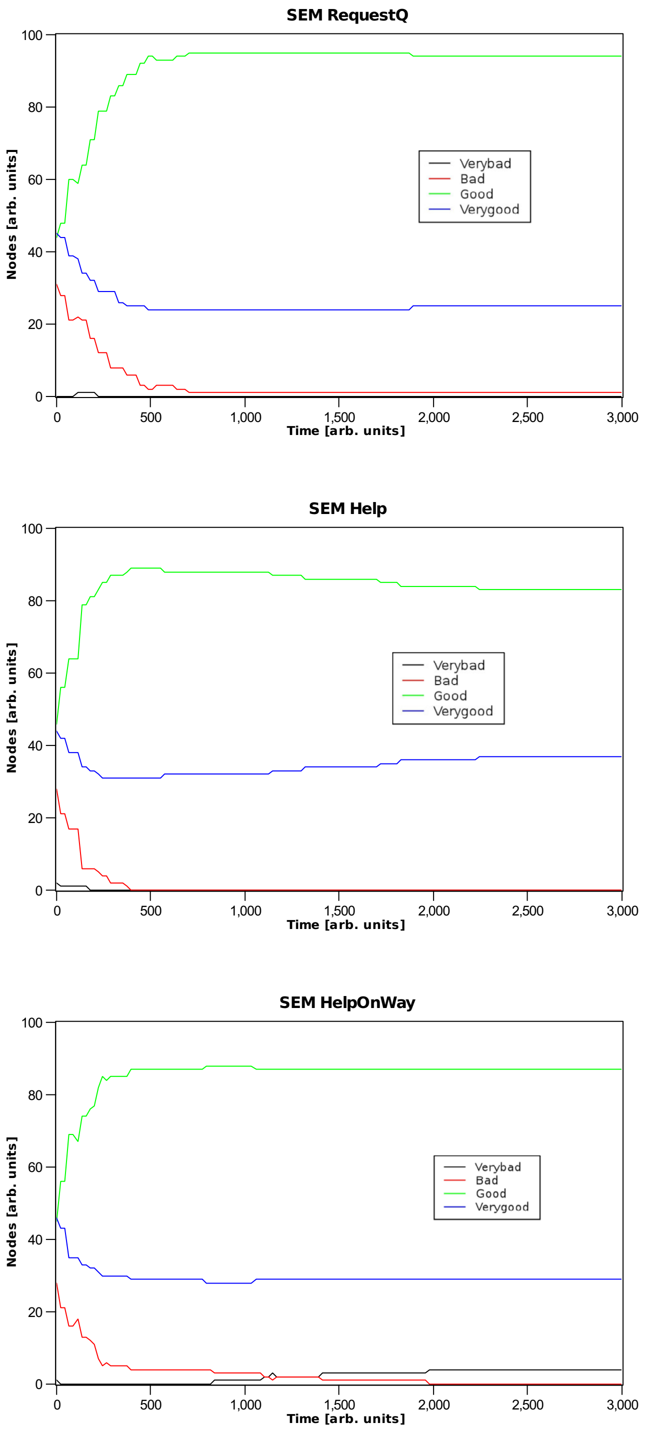

Figure 5: Typical examples of the run and progress of different energy management strategies with respect to the node classification population. (Top) SEM with RequestQ behaviour (Middle) Help behaviour (Bottom) Help On Way behaviour [x-axis: simulation time in arbitrary units, y-axis: node number]

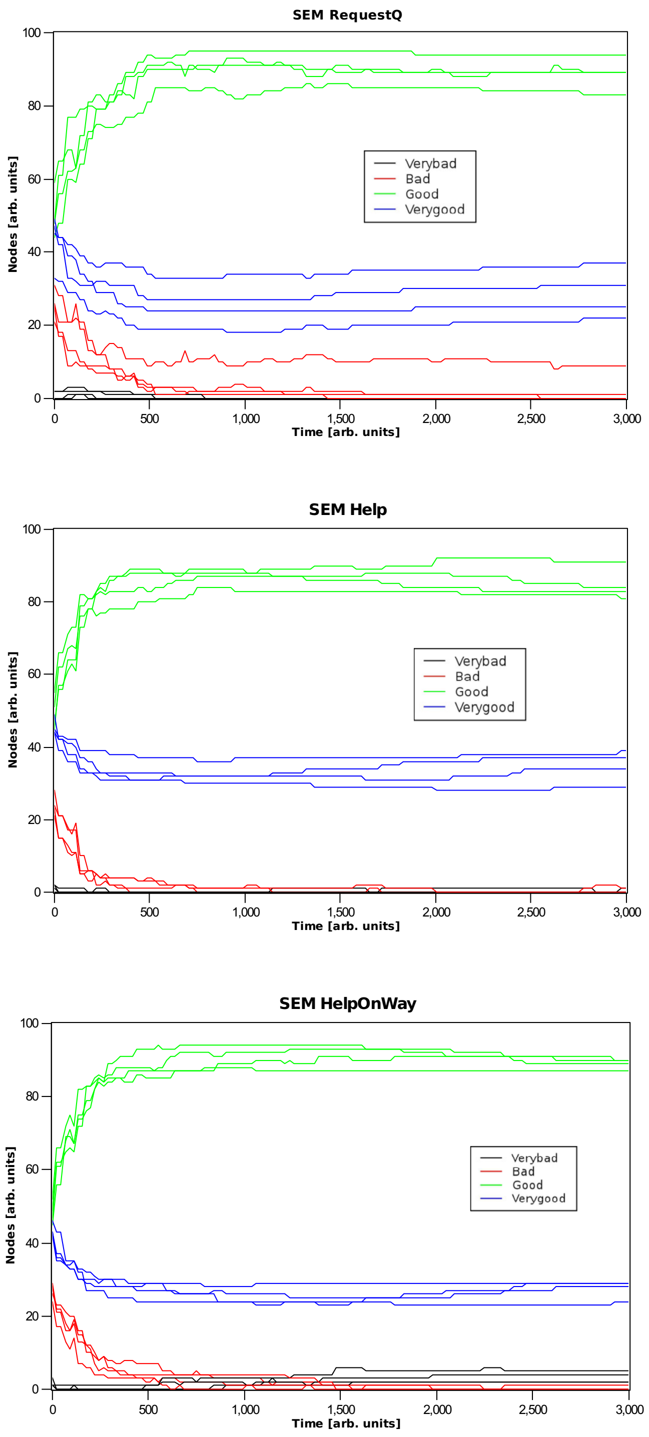

Figure 6: Typical variations of the run and progress of different energy management strategies with respect to the node classification population. (Top) SEM with RequestQ behaviour (Middle) Help behaviour (Bottom) Help On Way behaviour [x-axis: simulation time in arbitrary units, y-axis: node number]

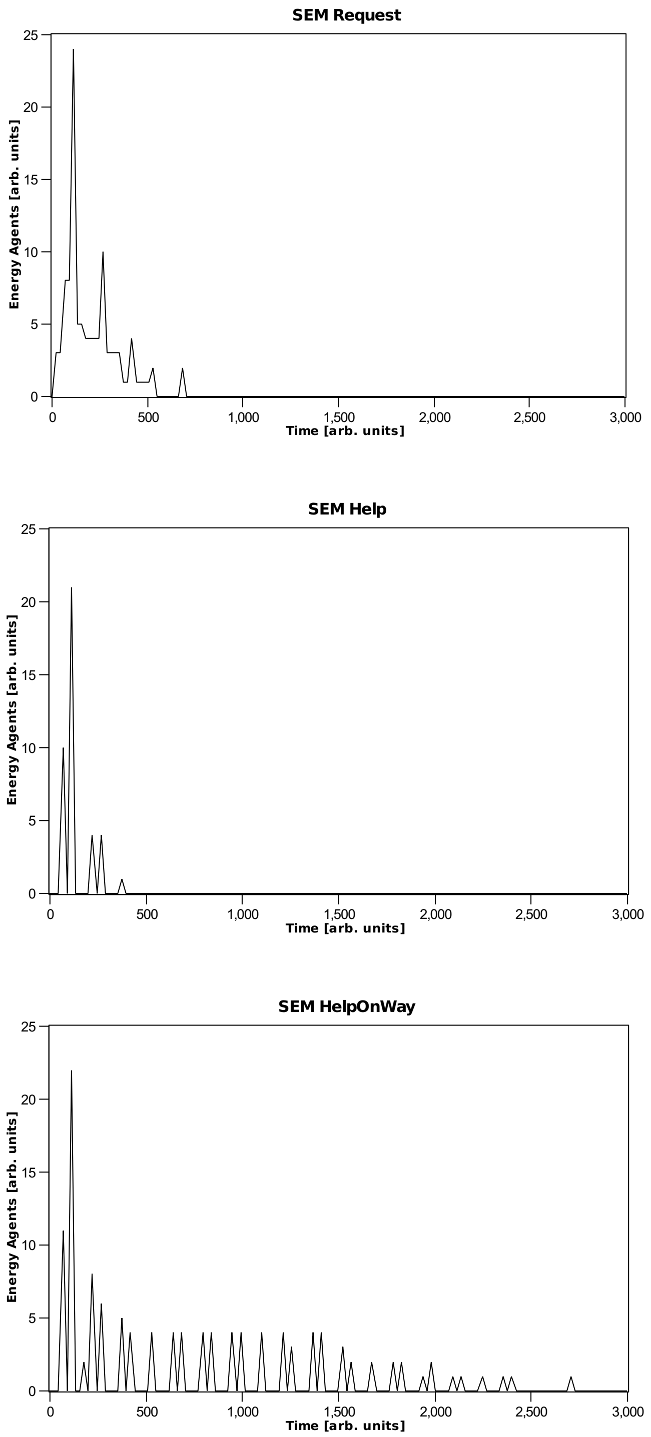

Figure 7: Typical temporal energy agent populations of runs with different energy management strategies. (Left) SEM with RequestQ behaviour (Center) Help behaviour (Right) Help On Way behaviour [x-axis: simulation time in arbitrary units, y-axis: agent number]

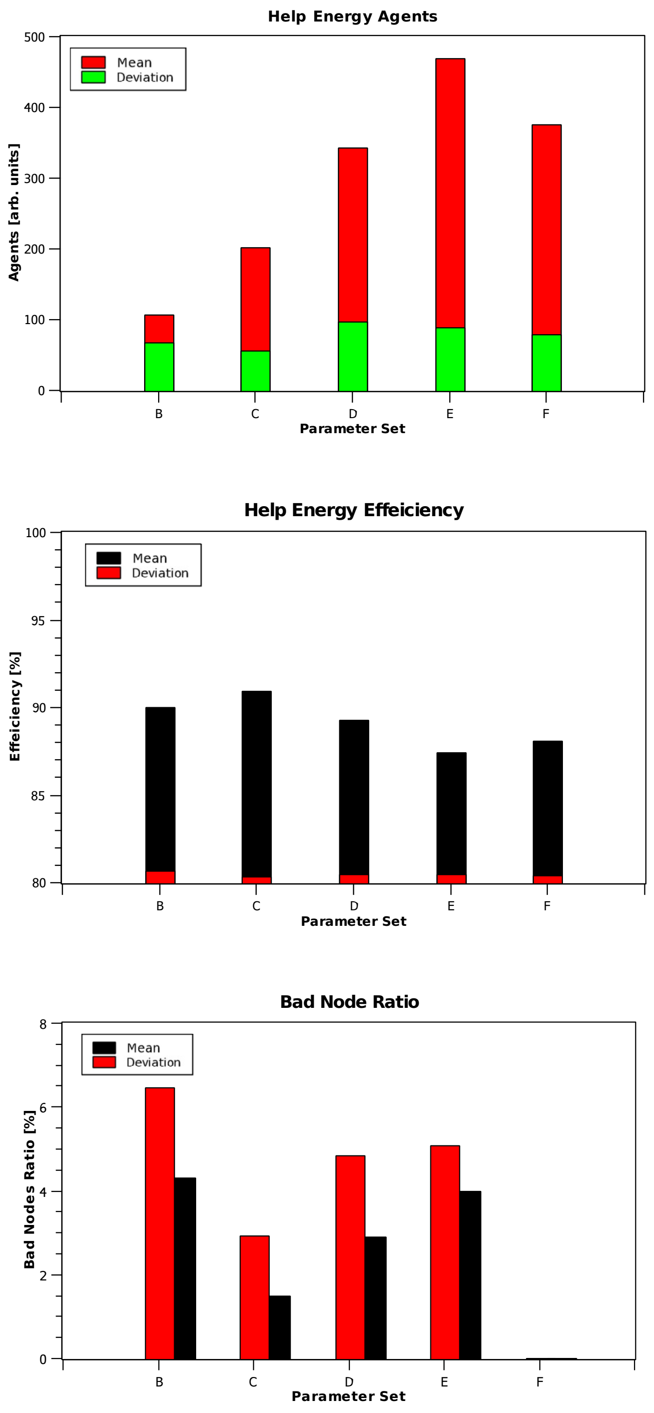

Figure 8: Analysis of the impact of different parameter sets of the SEM help behaviour. (Left) Total number of energy agents created (Center) Energy Efficiency comparing start, harvested, and final energy sum on system level (Right) Bad node ratio (before SEM/after SEM) [x-axis: parameter set, y-axis: agent number, efficiency and bad node ratio in %]

VII. Conclusion and Outlook

Smart energy distribution in self-powered micro-scale networks like sensor networks is a challenge with the goal to satisfy a safe and reliable operational state on system and node level. Under the assumption that nodes are arranged in mesh-like networks with links posing the capability to transfer energy and data between nodes a self-organizing MAS was deployed in this work successfully to distribute energy without a system/world level model and knowledge of the single nodes about the system state. Different agent behaviour were investigated and evaluated. The exploratory help strategy with deliver child agents showed the best and efficient overall behaviour.

The agents were programmed in *JavaScript* using the *JavaScript* Agent Machine Platform (*JAM*) that can be deployed in strong heterogeneous environments on a wide range of devices.

Among the agent behaviour already presented in this work, the issue of rising very bad (non operable) node clusters observed in the current MAS framework must be prevented. One possible solution is directed diffusion propagation around nodes giving energy away, i.e., an agent consuming and transferring energy from a node should trigger energy transfer in the opposite delivery direction (away from the energy valley up to energy hills). Furthermore, distributed supervised learning can be used to improve the emergence of the entire network and MAS on system level and on local region level. The used *JAM* platform already supports agents with an extensive set of learning algorithms posing mobile models that can migrate with agents.

Parameter Sets

```
B={explorationRange:2, maxLife:1, maxHops:4,
 inhibitTime: 20}
C={explorationRange:2, maxLife:2, maxHops:4,
 inhibitTime: 20}
D={explorationRange:2, maxLife:4, maxHops:4,
 inhibitTime: 20}
E={explorationRange:2, maxLife:8, maxHops:4,
 inhibitTime: 20}
F={explorationRange:4, maxLife:4, maxHops:8,
 inhibitTime: 20}
```

Agent Behaviour

The following algorithms describe parts of the agent behaviour in *AAPL* short notation (details [15]) of node and energy agents and their interaction using the tuple space. The stationary node agent controls energy storage, harvesting, and transfer on network nodes (with direct access to energy devices). The mobile energy agent performs energy negotiation and transport.

Notation: Ψ: Agent class, φ: Subclass, Σ: Agent body variables, α: Agent activity, Θ: Agent creation/destruction (+:create, $\times$:destroy, $\rightarrow$:fork), ∇: Tuple space access (+:out, -:inp, %:rd, $\pm$:listen, $\mp$:evaluate), $\Leftrightarrow$: Agent migration, π: activity transitions. Tuple listener receive tuples (with actual and formal parameters) from a corresponing evaluate operation and pass modifieded tuples to the evaluation request.

```
$\Psi$node : $options \rightarrow \{$
  $\Sigma = \{\ energy,\ energy_{thres*},\ energy_{Deposit},\ ..\}$
  $\alpha init\ :\ \{$
    • Agent asks for energy demand $(+)$ or grant $(-)$
    $\nabla^{\pm}(ENERGY?, ?) \rightarrow (*, ask)\ \{$
      $energy < energy_{Thres_0} + energy_{Deposit}$ ?
       • energy demand
       $ask \leftarrow energy_{Thres_1} - energy$ :
       • energy grant
       $ask \leftarrow -(energy - energy_{Thres_0} - energy_{Deposit})$
    $\}$
    • Agent requests energy token
    $\nabla^{\pm}(ENERGY-, ?) \rightarrow (*, con)\ \{$
      $energy > con - energy_{Deposit}$ ?
        $energy \leftarrow energy - con,$
        $consumed \leftarrow consumend + con,$
        • conversion loss
        $con \leftarrow con * energy_K$ :
        $con \leftarrow 0$
    $\}$
    • Agent delivers energy token
    $\nabla^{\pm}(ENERGY+, ?) \rightarrow (*, del)\ \{$
      • conversion loss
      $del \leftarrow del * energy_K$
      $energy \leftarrow energy + del$
    $\}$
    • Agent delivers energy token after migration
    $\nabla^{\pm}(ENERGYC+, ?) \rightarrow (*, del)\ \{$
      • conversion loss
      $del \leftarrow del * energy_{CommK}$
      $energy \leftarrow energy + del$
    $\}$
    • Generic energy request
    $\nabla^{\pm}(REQUEST, ?) \rightarrow (*, req)\ \{$
      • Is this node good, very good, or bad?
      $energy < energy_{Thres_0}$ ?
        • bad node, needs energy
        $req \leftarrow 0$ :
        $energy < energy_{Thres_1}$?
         • goog node : grant only half energy request!
         $req \leftarrow req\ /\ 2$ :
         • very good node : grant full request
    $\}$
  $\}$
$\}$
```

Energy negotiation is peformed by different tuples (energy tokens): *ENERGY?*, *ENERGY-*, *ENERGY+*, *ENERGYC+*, *REQUEST*. Energy agents can replicate based on behaviour.

```
$\Psi$energy : $options \rightarrow \{$
  $\Sigma = \{\ state,\ charge,\ energy,\ age,\ hops,\ \Delta,\ age,\ ..\}$
  $\alpha init\ \ :\ \{$
   $state \leftarrow SEARCHING$
   $charge?\ \nabla^{\mp}(ENERGY-, charge) \rightarrow$
     $(*, *)\{charge \leftarrow 0\}$
  $\}$

$route\ () \rightarrow \{$
  $range?\ \{$
    • Random walk behaviour
    $dir \leftarrow random([NORTH, SOUTH, WEST, ..])$
    ..
  $\}\ :\ \{$

    • Simple $\Delta$routing
    $\Delta_x > 0 \wedge ?\Lambda(EAST)\ ?\ \Delta_x$ --, $\rightarrow EAST$
    $\Delta_x < 0 \wedge ?\Lambda(WEST)\ ?\ \Delta_x$ ++, $\rightarrow WEST$
    $\Delta_y > 0 \wedge ?\Lambda(SOUTH)\ ?\ \Delta_y$ --, $\rightarrow SOUTH$
    ..
  $\}$
$\}$

$\alpha travel\ \ :\ \{$
 $nextdir \leftarrow route()$
 $\neg\ nextdir?\ \Theta^{\times}(self)$
 $charge?\ \nabla^{\mp}(ENERGY-, charge) \rightarrow$
   $(*, req)\ \{\ energy = req, charge = 0\ \}$
 $hops$ ++, $lastdir \leftarrow nextdir, \Leftrightarrow (nextdir)$
$\}$

$\alpha arrived\ \ :\ \{$
 $energy > 0?\ \nabla^{\mp}(ENERGYC+, energy) \rightarrow$
   $(*, *)\ \{charge \leftarrow energy, energy \leftarrow 0\}$
 $\Delta = 0?\ state \rightarrow IAMHERE$
$\}$

$\alpha terminate\ \ :\ \{\ ..\}$
$\alpha wait\ \ \ \ :\ \{\ ..\}$

$\varphi$request : $\{$
  $\alpha percept\ \ :\ \{$
    $\Delta = 0\ \wedge\ request?\ \nabla^{\mp}(REQUEST, request) \rightarrow$
      $(*, x)\ \{charge \leftarrow x\}$
  $\}$
  $\alpha action\ \ \ :\ \{$
    $charge?\ \Theta^{\rightarrow}(type : REPLY,\ \Delta : -\Delta,$
                       $charge : charge,\ state : SEARCHING),$
      • Stay only if the requested charge was granted
      $charge * 1.1 > request\ ?\ age$ -- : $age \leftarrow 0$
      $charge \leftarrow 0$
  $\}$

  $\pi\ :\ \{$
    $percept \rightarrow \Delta = 0\ ?\ action\ :\ travel$
    $action \rightarrow age > 0\ ?\ wait\ :\ terminate$
  $\}$
$\}$

$\varphi$reply : $\{$
  $\alpha percept\ \ :\ \{\ \Delta = 0?\ state \rightarrow IMAHERE\ \}$
  $\alpha action\ \ \ :\ \{\ \}$
  $\pi\ :\ \{$
    $percept \rightarrow state = SEARCHING\ ?\ travel\ :\ terminate$
    $action \rightarrow percept$
  $\}$
$\}$
$\varphi$deliver : {
  $\alpha percept$ : {
    • $Help - on - way\ behaviour?$
    $helponway ?\ \nabla^{\mp}(ENERGY?, *) \rightarrow$
     $(*, req)\{\ req > 0\ ?\ charge \leftarrow$
              $charge - charge/(range + 2)\ \}$
    $\Delta = 0?\ state\ \rightarrow IAMHERE$
  }
  $\alpha action$ : { .. }
  $\pi$ : {
    $percept\ \rightarrow state = SEARCHING\ ?\ travel\ :\ terminate$
    $action\ \rightarrow percept$
  }
}

$\varphi$help : {
  $\alpha percept$ : {
    $charge\ \rightarrow 0,\ state\ \rightarrow SEARCHING$
    $\Delta \neq 0\ ?\ \nabla^{\mp}(ENERGY?, *)\ \rightarrow (*, req)$
     $\{\ req < 0 \wedge -req > request?state\ \rightarrow IAMHERE\ \}$
  }
  $\alpha action$ : {
    $state = IAMHERE?$
     $\Theta^{\rightarrow}($
       $type : DELIVER,\ state : SEARCHING,$
       $charge : request,\ energy : 0,$
       $delta : -\Delta range : 0$
     $)$
    $age$--
  }

  $\pi$ : {
    $percept\ \rightarrow hops > maxhops\ ?\ terminate\ :$
                 $(state = SEARCHING\ ?\ travel\ :\ action)$
    $action\ \rightarrow age > 0\ ?\ wait\ :\ terminate$
  }
}

$\varphi$distribute : {
  $\alpha percept$ : {
    $charge > 0\ ?\ \nabla^{\mp}(ENERGY?, *) \rightarrow$
     $(*, req)\ \{$
       $req\ >\ charge\ ?$
         • $Deliver\ all\ charge\ on\ this\ node$
         $charge\ \rightarrow 0,\ state\ \rightarrow IAMHERE\ :$
         • $Deliver\ charge\ requested\ from\ node$
         $req\ >\ 0\ ?\ charge\ \rightarrow charge\ -\ req$
       }
  }

  $\alpha action$ : { }
  $\pi$ : {
    $percept\ \rightarrow action$
    $action\ \rightarrow state = SEARCHING\ \wedge hops < maxhops\ ?$
        $travel\ :\ terminate$
  }
}
}
```

References

[1] J. Lagorse, D. Paire, A. Miraoui, *A multi-agent system for energy management of distributed power sources* , J. of Renewable Energy, Vol. 35, Issue 1, 2010

[2] S. Ghani, M. Mousavi, and A. Movaghar, *Distributed Algorithms for Power Saving Optimization in Sensor Network* , Proceedings of the 8th WSEAS international conference on Data networks communications computers, pp. 109 −115, 2009.

[3] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, *Power management in energy harvesting sensor networks* , ACM Transactions on Embedded Computing Systems, vol. 6, no. 4, p. 32-es, 2007.

[4] Z. Jun, L. Junfeng, W. Jie, and H. W. Ngan, *A multi-agent solution to energy management in hybrid renewable energy generation system* , Renewable Energy 36, vol. 36, pp. 1352-1363, 2011.

[5] G.M. P. O'Hare, D. Marsh, A. Ruzzelli, and R. Tynan, Agents forWireless Sensor Network Power Management, in Parallel Processing, 2005. ICPP 2005 Workshops. International Conference Workshops on, 2005.

[6] P. Leito and S. Karnouskos, *Industrial Agents Emerging Applications of Software Agents in Industry*. Elsevier, 2015.

[7] S. Bosse, F. Kirchner, *Smart Energy Management and Energy Distribution in Decentralized Self-Powered Sensor Networks Using Artificial Intelligence Concepts* , Proceedings of the Smart Systems Integration Conference 2012, Session 4, Zürich, Schweiz, 21 − 22 Mar. 2012, 2012, ISBN: 978-3-8007-3423-8.

[8] E. Rokrok, M. Shaekhah, P. Siano, and J. P. S. Catalao, *A Decentralized Multi-Agent-Based Approach for Low Voltage Microgrid Restoration* , Energies, vol. 10, no. 1491, 2017.

[9] B. Zhao, M. Xue, X. Zhang, C. Wang, and J. Zhao, *An MAS based energy management system for a stand-alone microgrid at high altitude* , Applied Energy, vol. 143, 2015.

[10] S. Bosse, A. Lechleiter,*Structural Health and Load Monitoring with Material-embedded Sensor Networks and Self-organizing Multi-agent Systems* , Procedia Technology, 2014, http://dx.doi.org/10.1016/j.protcy.2014.09.039

[11] E. Gavrin, S.J. Lee, R. Ayrapetyan, R., A.Shitov, (2015) *Ultra lightweight JavaScript engine for internet of things* , in SPLASH Companion 2015 Companion Proceedings of the 2015 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity, 2015, pp. 19-20

[12] L. Chunlina. , L. Zhengdinga., L. Layuanb. , Z. Shuzhia,. (2002) *A mobile agent platform based on tuple space coordination,* Advances in Engineering Software, vol. 33, no. 4, pp. 215−225.

[13] R.H. Bordini., J.F: Hübner. (2006) *BDI agent programming in AgentSpeak using Jason* , Computational Logic in Multi-Agent Systems, Volume 3900 of the series Lecture Notes in Computer Science, Springer,, pp. 143-164.

[14] S. Bosse, *Mobile Multi-Agent Systems for the Internet-of-Things and Clouds using the JavaScript Agent Machine Platform and Machine Learning as a Service*, in The IEEE 4th International Conference on Future Internet of Things and Cloud, 22-24 August 2016, Vienna, Austria, 2016, http://dx.doi.org/10.1109/FiCloud.2016.43.

[15] S. Bosse, A. Lechleiter, *A hybrid approach for Structural Monitoring with self-organizing multi-agent systems and inverse numerical methods in material-embedded sensor networks*, Mechatronics, (2016), http://dx.doi.org/10.1016/j.mechatronics.2015.08.005