Chapter 13

Energy Management

Low-power Design and Smart Energy Management

epubli, ISBN 9783746752228 (2018)

This chapter introduces different techniques for the optimization and the low-power design of application-specific data processing and agent processing platforms. Furthermore, agent-based energy management is introduced, in addition to the energy management SoMAS introduced in Chapter 9.

13.1 Power Analysis and Algorithmic Selection

In contrast to various other energy management approaches targeting algorithms and architectures with high computational effort, *Smart Energy Management* (SEM) can be performed spatially at run-time by applying a dynamic selection from a set of different (implemented) algorithms classified by their demand of computational power, and temporally by varying data processing rates. The smart energy management can be implemented with decision trees, based on Quality-of-Service (QoS) and energy constraints. It can be shown that the power and energy consumption of an application-specific SoC design strongly depends on the computational complexity of the used algorithms.

For example, a classical Proportional-Integral-Differential (PID) controller used for the feedback position control of an actuator requires basically only the *P*-part; the *I*- and *D*-parts only increase position accuracy and response dynamics, which are selectable. Depending on the actual state of the system and the actual and estimated future energy deposit, suitable algorithms can be selected and executed optimizing the QoS and the trade-off between accuracy and economy.

Signal and control processing can be modelled on abstract algorithmic level using signal flow diagrams. These signal flow graphs are mapped to Petri Nets to enable direct high-level synthesis of digital SoC circuits using a multiprocess architecture with the Communicating-Sequential-Process model on execution level and the high-level synthesis framework *ConPro*.

Power analysis using simulation techniques on digital gate-level provides input for the algorithmic selection at run-time of the system leading to a closed-loop design flow. Additionally, the signal-flow approach enables power management by varying the signal flow rate, which will be discussed later.

The smart energy management using algorithmic selection relates to the entire design flow for low-power SoC data processing and control systems, introduced in Chapter 12, and which should be demonstrated in the following using a concrete example. The system is modelled on an abstract level using signal flow diagrams [BOS10D].

Figure 13.1 shows a composition of a complete feedback-controlled system consisting of sensor signal acquisition (ADC), filtering, an error controller with a proportional, integral, and differential sub-controller [ISE89], and finally a signal generator (DAC) driving an actuator.

epubli, ISBN 9783746752228 (2018)

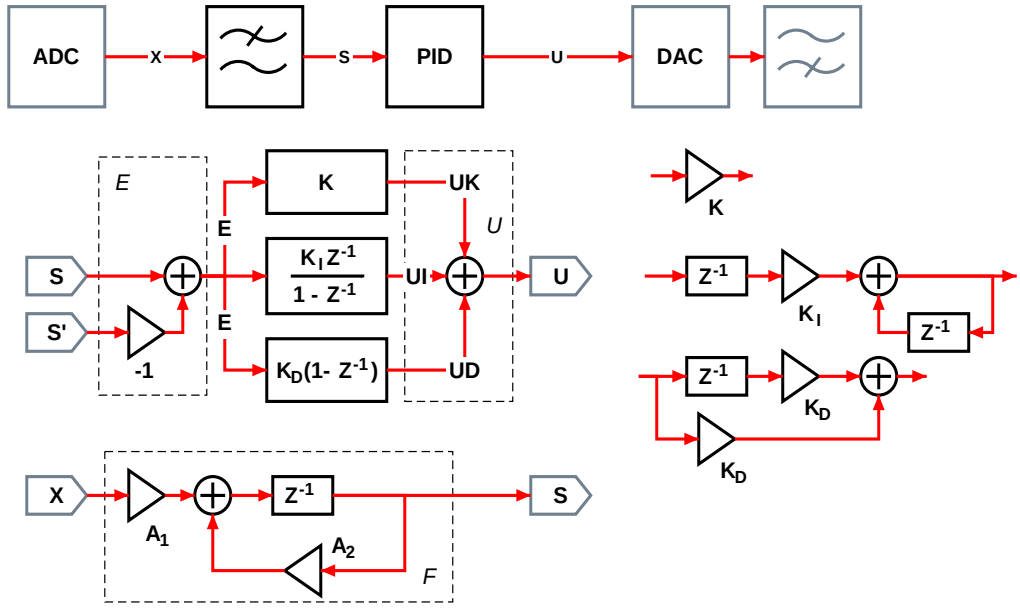

Fig. 13.1 *Composition and modelling of a digital control system with signal flow diagram* [BOS11B]

The controller is used to control the position of an actuator. The control error is defined by the difference of the acquired position signal *X*(*S*) and the desired position parameter *S'*.

This initial specification (see also Section *5.5*) is used to derive 1. A multi-process programming model; and 2. A hardware model for an SoC design on Register-Transfer level. Furthermore, the signal flow diagram provides input for the energy optimization at synthesis- and run-time.

The signal flow diagram is first transformed into an S/T Petri Net representation, which is shown in Figure *13.2*. Functional blocks are mapped to transitions, and states represent data, which is exchanged between those functional blocks. The partitioning of functional blocks to transitions of the net can be performed at different composition and complexity levels. The signal flow diagram from Figure *13.1* was partitioned using complex blocks (composed of low-level blocks like multipliers and adders) to reduce communication complexity (and data processing latency).

Sensor data (*X*) is acquired periodically and passed to the data processing system. A token of the net is equal to a data set of one computation processed by the functional blocks. The functional blocks ***P***, ***I***, and ***D*** are placed in concurrent paths of the net.

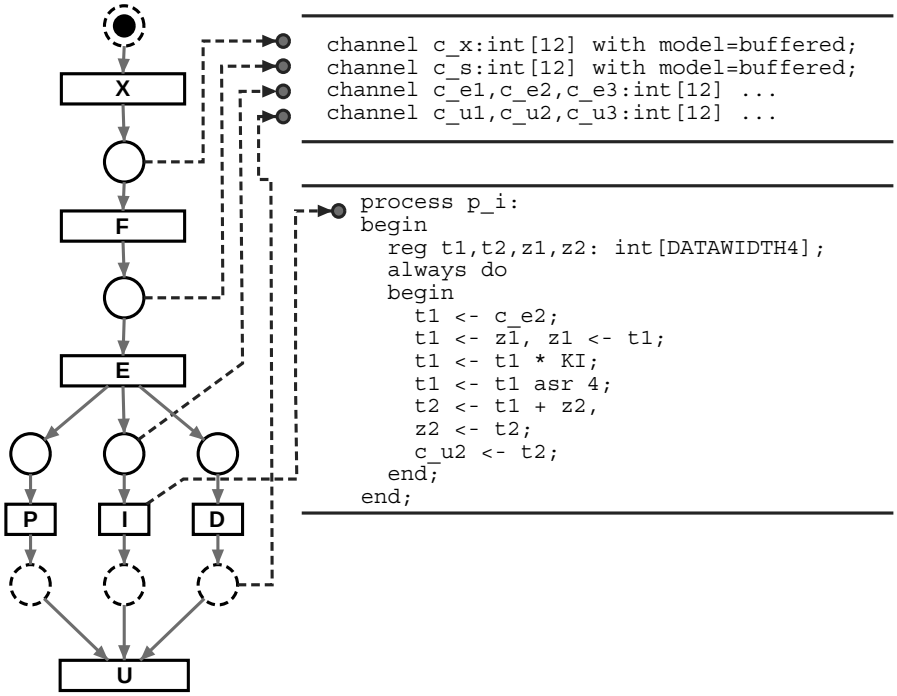

Fig. 13.2 *Mapping of the signal flow diagram to a Petri Net and mapping of Petri Net to communication channels and sequential processes using the ConPro programming language* [BOS11B].

The Petri Net is then used 1. To derive the communication architecture, and 2. To determine an initial configuration for the communication network. Functional blocks with a feedback path require the injection of initial tokens in the appropriate states (not required in the example).

States of the net are mapped to buffered communication channels and transitions are mapped on concurrently executing processes - each with sequential instruction processing - using the *ConPro* programming language, shown in Figure *13.2*, too. Details are discussed in Section *5.5*.

Energy Analysis

The derived multiprocess programming model was synthesized to a digital logic SoC using high-level synthesis. For simulation, gate-level synthesis was performed with a standard logic cell technology library. The resulting net-list was analysed with an event-driven simulator, calculating the overall cell activity for each time unit, defined by terms of cell output changes. Synthesis and analysis were performed using the Concurrent Programming (*ConPro*) compiler and the Silicium-Compiler-and-Analyser framework (*SiCA*).

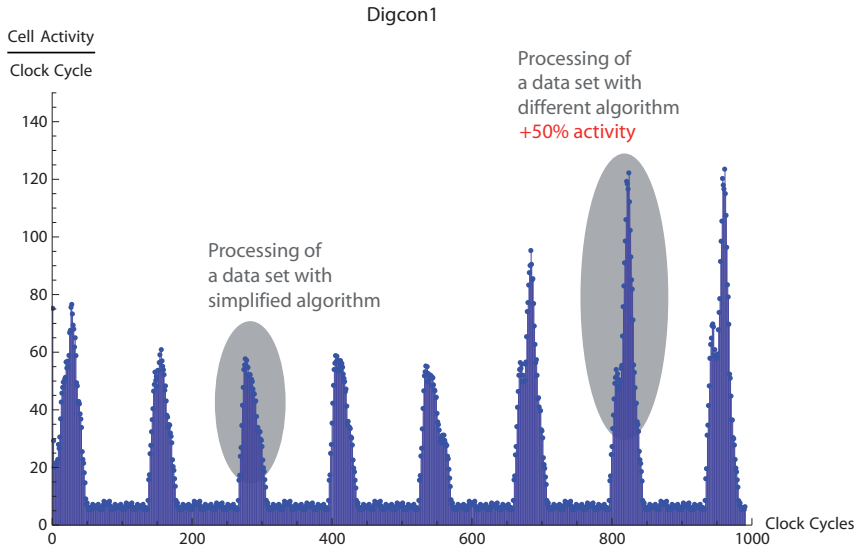

Fig. 13.3 *Averaged SoC cell activity correlates strongly with computation and signal/data flow. After obtaining the fifth result value U, the I and D computational blocks are enabled.* [BOS11B]

The SoC circuit activity correlates strongly with a computation of a new data set (with sensor data input sampled periodically) and the computation complexity, shown in Figure *13.3*.

The logic cell activity of the circuit has strong peaks around the computation of a new output value *U*. About every 140 clock cycles a new input value *X* is generated, triggering the calculation of a new output value *U*.

The first five data sets are computed with an enabled P-part of the controller only. After the fifth computation, the *I* and *D* parts were enabled, too. This results in an increase of circuit activity of about 50%.

Unfortunately, the power dissipation cannot be estimated directly from this cell activity. Logic cells consist of a network of (paired) transistors. Power dissipation of a CMOS circuit depends proportionally from the transistor switching activity. Simulation results for the controller are shown in Figure *13.4* (using *SiCA*, too). There is only weak correlation between data processing activity (and computation complexity) and power dissipation due to clocking activity of registers.

Power dissipation can only be estimated from the above circuit cell activity if clock-gated registers are assumed [XIA02]. The principle-architecture of gated registers is shown in Figure *13.5*. The clock gating prevents switching activity (and hence power dissipation) inside the register cell in the case of no change of input data (D=Q).

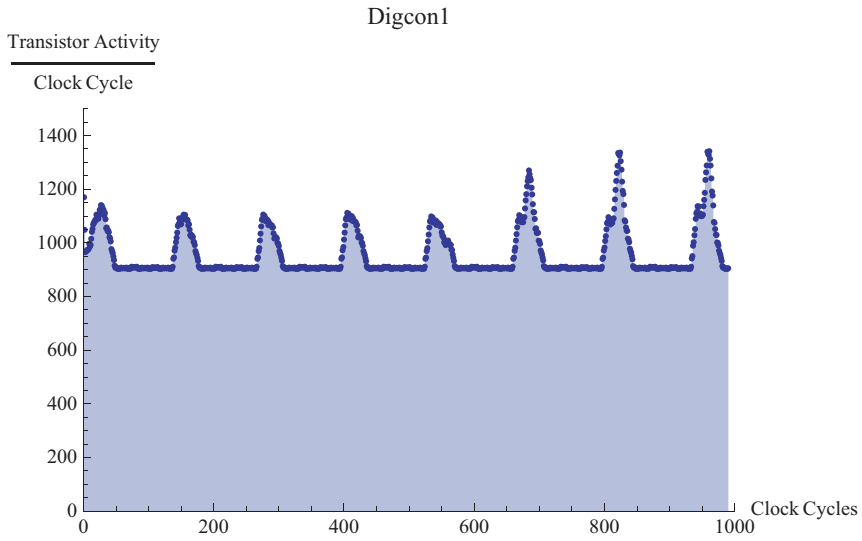

Fig. 13.4 *Averaged SoC transistor switching activity of the circuit retrieved from simulation. Power dissipation is proportional to transistor activity.* [BOS11B]

Fine-grained gate-level clock gating is a requirement for the proposed low-power design method and enables again a strong correlation between computation activity (and hence algorithmic complexity) with the power dissipation of the data processing system.

Results of such a modified control system with clock-gated registers are shown in Figure *13.6*. There is again a significant increase of transistor switching activity of about 30% if the two different computation levels (*P*, *PID*) are compared.

The demonstrated correlation of the algorithmic and computational complexity with the energy and power consumption assumed an optimized ASIC design technology process.

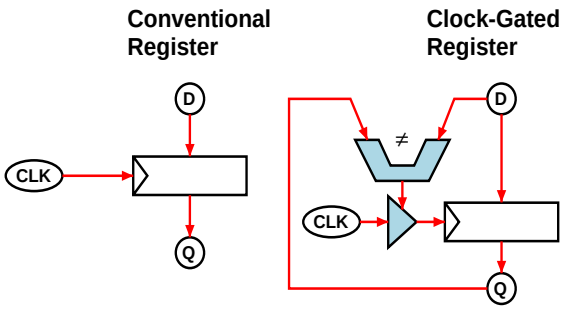

Fig. 13.5 *Register Clock Gating (adapted from* [XIA02]*).*

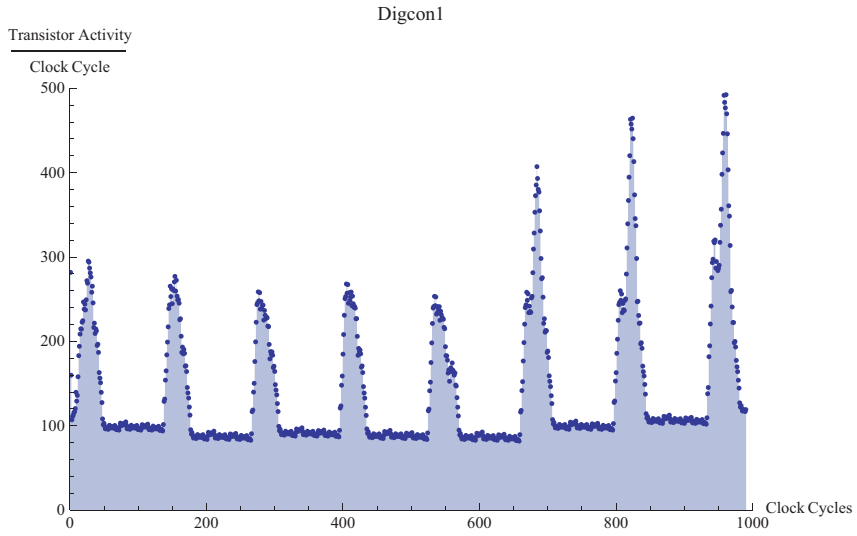

Fig. 13.6 *Averaged SoC transistor switching activity of the circuit retrieved from simulation. Using clock-gated registers results again in strong correlation between data processing activity (and computation complexity) and power dissipation.* [BOS11B]

In [TAS13] the algorithmic correlation was investigated for a SRAM-LUT-based FPGA technology, showing again a significant algorithmic correlation suitable for run-time energy management based on algorithmic selection.

Smart Energy Management at Run-time

Self-powered systems must deal with a limited amount of energy during run-time. The energy charge in future is uncertain. Smart energy management should handle the conjunction of energy demand and energy conversation.

Methods from Artificial Intelligence (AI) can be used to manage energy at runtime with dynamic parameter adaptation and algorithmic selection. AI methods differ in complexity, thus only few are suitable to be embedded in microchips. Suitable methods are for example constraints nets and decision trees in conjunction with machine learning approaches.

A simulation of a complex sensor-actuator system implementing the PID controller from Figure *13.1* should demonstrate the benefits of using a decision tree method for dynamic parameter adaptation, which can be retrieved from machine learning. Parameters to be controlled are data processing rate *R* and the algorithmic level *L* (1: only *P*, 2: *P*+*I*+*D* controller). The controller performs minimization of the position error of the actuator, that means the difference between a desired and a measured (angular) position.

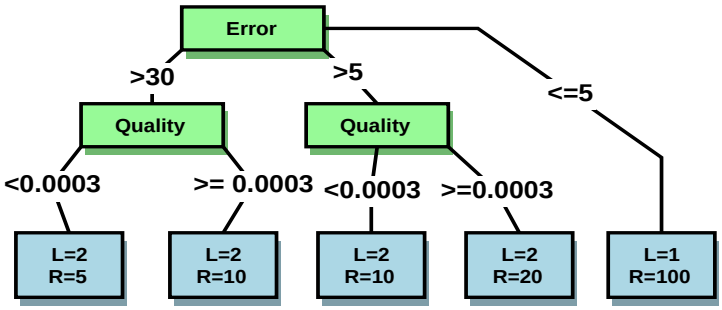

Fig. 13.7 *Decision tree used for energy optimization at runtime based on parameters. Input parameters: Error, Quality=Run-time/Error. Output parameters: Data processing rate R={1,5,10,20,100}, Algorithmic level L={P:1,PID:2}*

The system is charged with a stochastic energy source and discharged by computation and actuator activity. The power and energy required for the computation can be calculated from the results of the power analysis, the power and energy required for actuator activity can be calculated by a simplified physical simulation.

The decision tree used in the simulated system is shown in Figure *13.7*. A decision tree is system- and environment-specific and must be derived for each different system.

Parameters are modelled with a discrete set of values related to a discrete set of cost values, shown in Definition *13.1*.The cost values are used to calculate the overall runtime costs of the system. The goal is to minimize the overall costs and to maximize the energy conversation, but still serving the quality of the service to be provided (in this case the accurate position control of the actuator).

Def. 13.1 *System parameter value sets and related cost values.*

```
Data processing rate      R={1,5,10,50,100} [milliseconds]
                          R_C={100,50,10,5,1}
Algorithmic level         L={1,2}
                          L_C={100,150}  derived from power analysis
Actuator position error   E={0,5,10,100} [arb. units]
                          E_C={0,250,500,5000}
```

The smart energy algorithms used for the system simulation are shown in Algorithm *13.1*. The *estimate* procedure calculates actual computation and quality-of-service costs for the system, and the *choose* procedure calculates optimized values for the data processing rate and algorithmic level based on

actual system state. The costs are derived from the previous power analysis results.

The *costs* function returns a linear interpolated cost value of the particular parameter.

The basic concepts of the simulation are shown in Algorithm *13.2*. The *charge* procedure stores energy in the system from a random source. The *controller* procedure implements the selectable *P*/*PI*(*D*) controller, and finally the *stimuli* procedure simulates the simplified mechanical actor behaviour due to a drive signal calculated by the controller.

Alg. 13.1 *Smart energy management algorithm based on algorithmic selection* [BOS11B]

```
VAR
  level,rate: actual algorithmic level and data processing rate
  error: actual position control error
  runtime: passed runtime in time units
  energy: energy storage in arb. units
  cost: quality of service costs

PROCEDURE Estimate:
  for each time unit do
    delta := Costs(level)*Costs(rate)+Costs(error);
    energy := energy - delta;
    quality := Average(runtime)/Average(cost);
    runtime := runtime + 1;
    cost := cost + Costs(error);
    Choose(level,rate);

PROCEDURE Choose:
  Use decision tree to choose optimal {level,rate} values based on
  averaged quality and actual error
```

Alg. 13.2 *System simulation algorithms* [BOS11B]

```
VAR
  pos_act,pos_set: actual and desired actuator position

PROCEDURE Stimuli:
  each 500th time unit do: pos_set := -pos_set;
  pos_act := pos_act + drive/4;
  delta := pos_act - pos_set;
  error := min 100 (abs delta);

PROCEDURE Charge:
  energy := energy + Random(CHARGE_MAX)
```

```
PROCEDURE Controller:
  case l is
    when 1:  P-controller
      for each rate time unit do:
        delta := pos_act-pos_set;
        S := delta*KD/100;
        drive := S;
    when 2:  PI controller
      for each rate time unit do:
        delta := pos_act-pos_set;
        S := delta*KD/100;
        Z1' := Z1, Z2' := Z2, Z1 := delta;
        T := (Z1'*KI)/100+Z2';
        Z2 := T;
        drive := S+T;
```

Figure *13.8* shows simulation results of the complex sensor-actuator system implementing the *PID* controller. The system runs always out of energy if a fixed parameter setting {*Rate*, *Level*} is used, regardless of the parameter values. In contrast the system can reach a stable state balancing energy charging and discharging if dynamic parameter adaptation based on the decision tree method and system feedback is used.

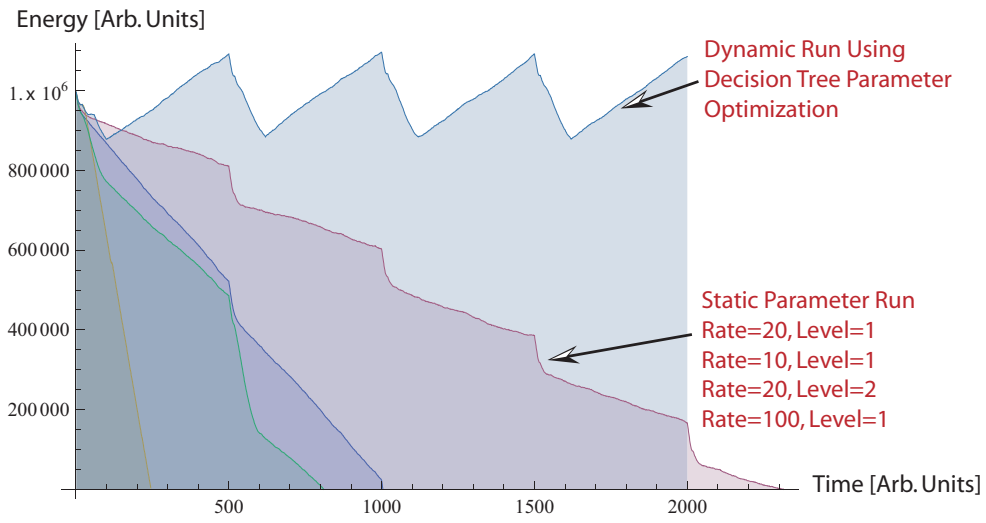

Fig. 13.8 *System simulation with different runtime behaviours using a decision tree, which can be retrieved by machine learning methods. Parameters: Data processing rate={1,5,10,20,100}, Algorithmic level={P:1,PID:2}* [BOS11B]

13.1.1 Low-power Agent Processing

Dynamic algorithmic selection can be used to optimise the power consumption of the agent processing at run-time. The power consumption for each activity of an ATG of a specific agent class AC can be analysed. This analysis marks all activities with a power weight, which can be used at run-time to estimate the energy required for execution of this specific activity. Agents are capable to adapt their behaviour based on these power weights and energy information provided by the host, i.e., by reading an energy and energy management tuple from the his tuple-space, which was stored by a stationary energy management agent.

Application-Specific PCSP Platform

In the case of the hardware implementation of the non-programmable *PCSP* Agent Processing Platform the power analysis is performed with gate-level simulation techniques, which provides input for the calculation of the power weights. Usually the simulation of the entire application specific APP is too complex. Therefore, each activity is decomposed in elementary statements, which can be simulated separately.

In the case of the software implementation, the machine code is analysed and profiled for some representative test patterns.

The Agent-based Platform simulation techniques can be used to estimate the power weights of activities roughly for some representative test patterns. This method gives the best balance between power analysis accuracy and analysis complexity, which can be significantly large in the case of the hardware simulation.

Programmable PAVM Platform

The programmable platform approach cannot be optimized in this way with respect to a specific algorithm due to its generic nature. But with the analysis of the energy requirement of specific machine instructions it is possible to optimise the overall VM design with a back propagation of the energy information to the software *AFM*/*AML* compiler energy optimized programs can be created (design time). Furthermore, at run-time the energy demand can be estimated based on the energy profiling, enabling the power efficient scheduling of different agents. An agent token can be associated with an energy parameter that can be used to provide input for decision making process regarding the execution of an agent.

epubli, ISBN 9783746752228 (2018)

13.2 Smart Energy Management with Artificial Intelligence

13.2.1 Introduction

Sensorial materials equipped with embedded miniaturized smart sensors provide environmental information required for advanced machine and robotics applications. With increasing miniaturization and sensor-actuator density, decentralized self-supplied energy concepts and energy distribution architectures are preferred and required.

Self-powered sensor nodes collect energy from local sources, but can be supplied additionally by external energy sources. 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.

A decentralized sensor network architecture is assumed with nodes supplied by 1. energy collected from a local source, and 2. by energy collected from neighbour nodes using smart energy management (SEM). Nodes are arranged in a two-dimensional grid with connections to their four direct neighbours. Each node can store collected energy and distribute energy to neighbour nodes.

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.

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.

Typically, energy management is performed by a central controller in that a program is implemented [LAG10], with limited fault robustness and the requirement of a well-known environment world model for energy sources, sinks, and storage. Energy management in a network involves the transfer of energy.

The loss of energy ε (in the range between 0 and 1) at each node occurring each time when “energy” is routed along different nodes from a source to a destination node (assuming N intermediate nodes) reduces overall efficiency dramatically in the order of $\eta=\varepsilon^N$.

By using electrical connections, only negligible loss of energy can be expected in a distributed network, in contrast to optical and radio wave connections, which have significant loss in the order of $\varepsilon \cong$10-30% per node. Additionally, in the latter case there is no physical interaction between a source and a sink node requesting energy, thus requiring active management (routing).

To overcome these limitations and to increase operational robustness, smart energy management is performed by using concepts from artificial

intelligence. Initially, the sensor network is a distributed group of independent computing nodes. Interaction between nodes is required to manage and distribute information and energy. One common interaction model is the mobile agent.

Different kinds of agents with different behaviours are used to negotiate energy demands and energy distribution and to implement group communication. A multi-agent system is a decentralized and self-organizing approach for data processing in a distributed system like a sensor network.

Recent work shows the benefit and suitability of multi-agent systems used for energy management [LAG10].

13.2.2 Communication, Data and Energy Transport

Network nodes, arranged for example in a two-dimensional mesh-network (see Figure *13.9*), can exchange data with their neighbours by using serial communication links. There are different kinds of physical transmission technologies available: electrical, optical, and radio-wave based.

In contrast to electrical interconnect technologies, optical and radio-wave technologies have the disadvantage of lower efficiency ε. This is negligible for the exchange of information, but significant for the distribution and exchange of energy required for the supply of nodes. Optical communication has clear advantages such as extremely small and light-weight hardware, ultra-low power consumption, and the ability to optimally focus and match the beam to the transmission medium (optical fibre) [KED06].

Sharing of one interconnect medium for both data communication and energy transfer significantly reduces node and network resources and complexity, a prerequisite for a high degree of miniaturization required in high-density sensor networks embedded in sensorial materials. Point-to-point connections and mesh-network topologies are preferred in high-density networks because they allow good scalability (and maximal path length) in the order of $\Theta(\log N)$, with N as the number of nodes.

Figure *13.9* shows the main building blocks of a sensor node, the proposed technical implementation of the optical serial interconnect modules, and the local energy management module collecting energy from a local source, for example a thermo-electric generator, and energy retrieved from the optical communication receiver modules. The data processing system can use the communication unit to transfer data (D) and superposed energy (E) pulses using a light-emitting or laser diode. The diode current, driven by a differential-output sum amplifier, and the pulse duration time determine the amount of energy to be transferred. The data pulses have a fixed intensity several orders lower than the adjustable energy pulses. On the receiver side, the incoming light is converted into an electrical current by using a photo diode. The data part is separated by a high-pass filter, the electrical energy is stored by the harvester module.

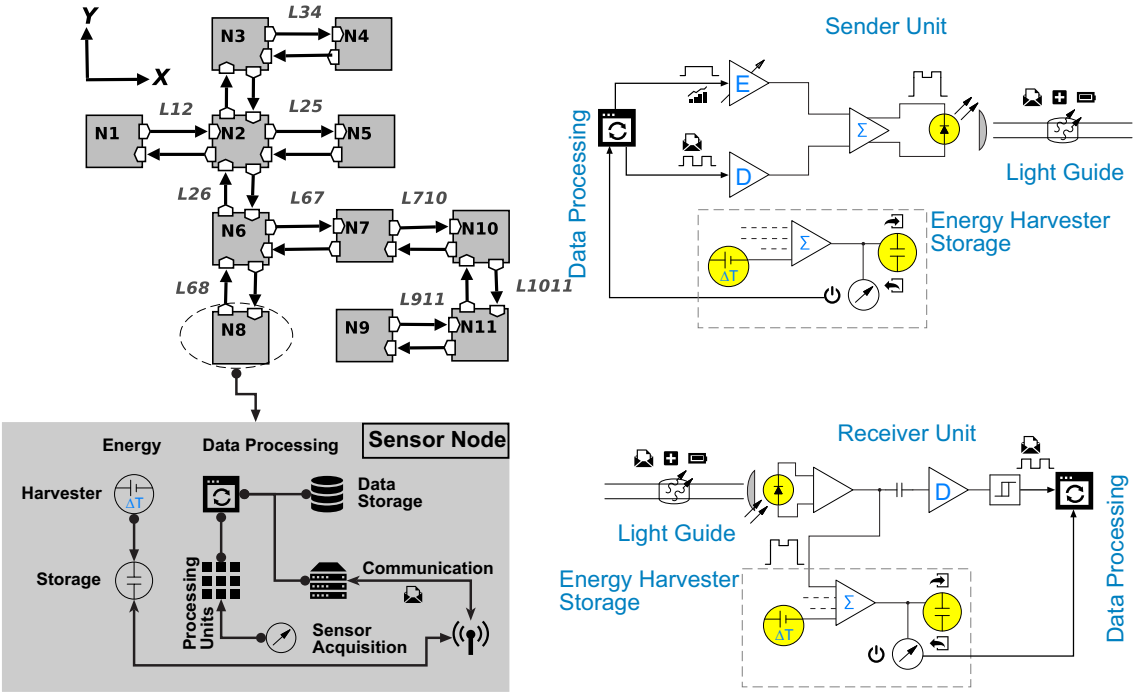

Fig. 13.9 *Network topology (left) and sender and receiver blocks (right) used for data and energy transmission between neighbour nodes that are connected in the network.*

Information and energy is carried with agents that are encapsulated in messages routed in the network from a source to a destination node using, for example, a simple delta-routing path protocol.

13.2.3 Multi-Agent Interaction Model and Implementation

Having the technical abilities explained in the previous section, it is possible to use active messaging to transfer energy from good nodes having enough energy towards bad nodes, requiring energy. An agent can be sent by a bad node to explore and exploit the near neighbourhood. The agent examines sensor nodes during path travel or passing a region of interest (perception) and decides to send agents holding additional energy back to the original requesting node (action). Additionally, a sensor node is represented by a node agent, too. The node and the energy management agents must negotiate the energy request.

The agents have a data state consisting of data variables and the control state, and a reasoning engine, implementing behaviours and actions. Table

13.1 explains different agent behaviour required for the smart energy management (see page 485).

A node having an energy level below a threshold can send a help agent with a delta-distance vector specifying the region of interest in a randomly chosen direction. The help agent hops from one sensor node to the next until the actual delta-vector is zero. If there is a good node found, with local energy above a specified threshold, the help agent persists on this node and tries to send periodically deliver agents transferring additional energy to the originator node. An additional behaviour, help-on-way, changes the deliver agent into an exploration agent, too. Such a modified agent examines the energy level of nodes along the path to the destination. If a bad node was found, the energy carried with the agent is delivered to this node, instead to the original destination node. A more general *AAPL* agent model of this SoS is discussed in detail Section *9.3*.

This approach is independent of the agent processing platform and the mobile agents can be processed on both proposed platforms (*PAVM*, *PCSP*). Originally, this SEM approach was implemented on simplified application-specific processing platforms, shown in Figure *13.10*.

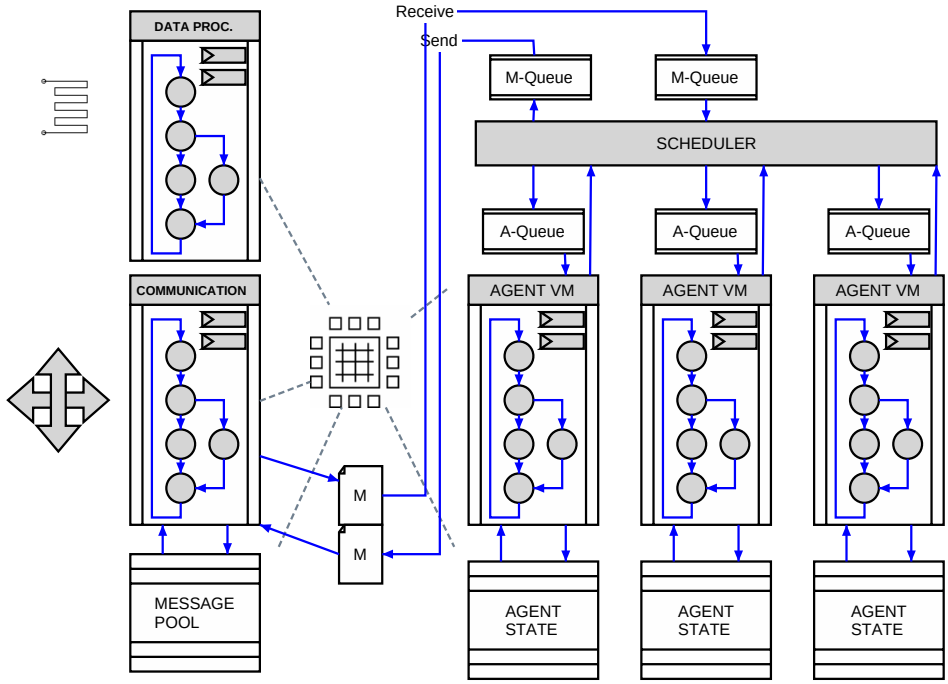

Fig. 13.10 *Sensor node building blocks providing mobility and processing of multi-agent systems: Parallel agent virtual machines, agent-processing scheduler, communication, and data processing.*

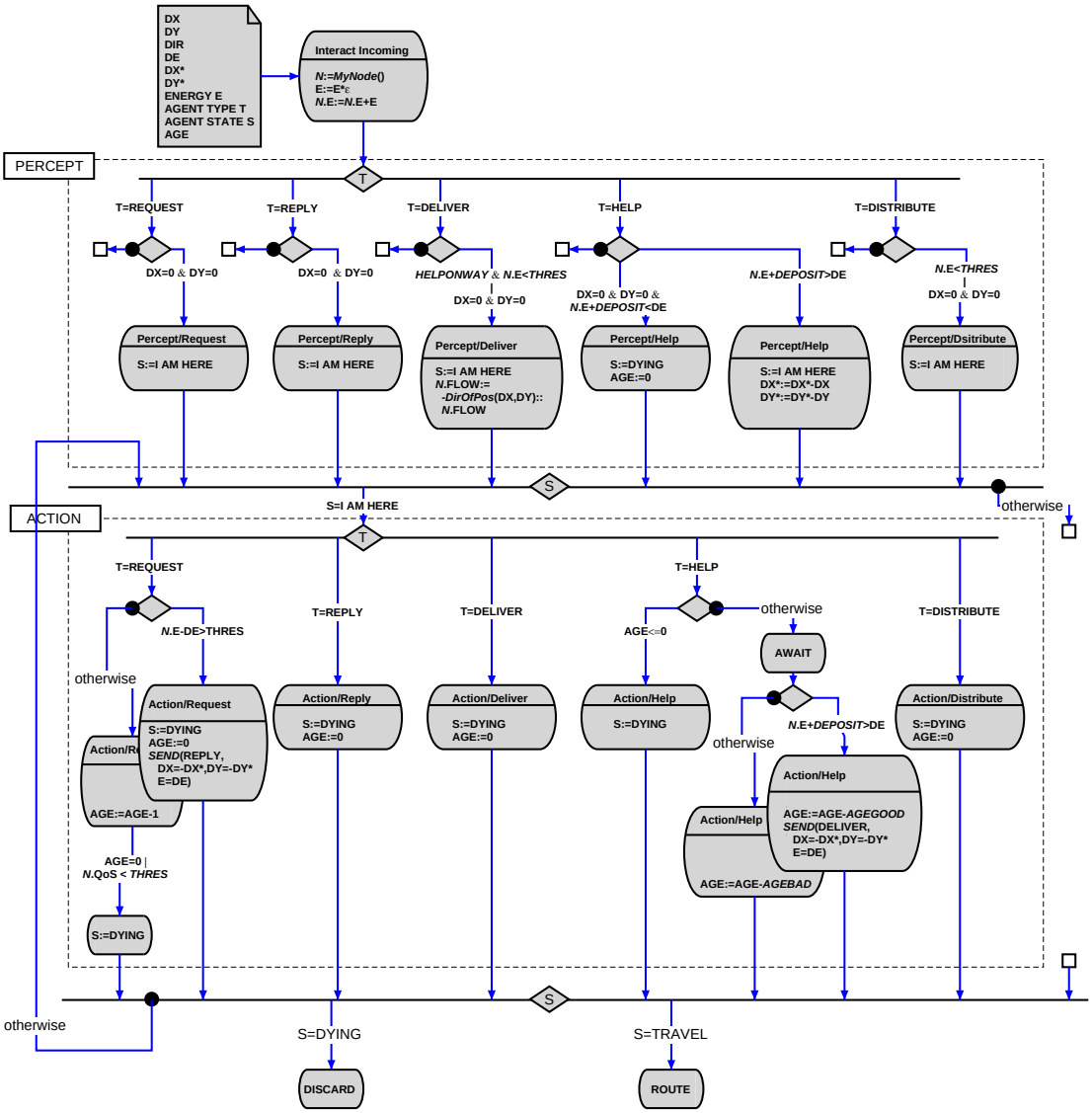

Fig. 13.11 *UML activity diagram for the Smart Energy Management Agent behaviour.*

In this early approach the state of an agent was completely kept in the message transferred in the network, but not the functional behaviour. Figure *13.10* shows the execution environment for the energy management agents. There is a message module implementing delta-distance routing (see Section *4.3.2* for a discussion), and several finite-state-machines implementing the agent behaviour that provide virtual machines capable to process incoming

agents. All parts are mappable on digital logic with RTL and SoC system architectures.

Figure 13.11 shows the *UML* activity diagram of the reasoning engine, implementing all types of agents, forming a super agent. The delta-distance or region of interest vector (*DX*, *DY*) is modified on each message routing, the (*DX**, *DY**) vector holds an unmodified copy, which is used for replies, the *DE* entry specifies the requested energy, and the *ENERGY* entry reflects the actual energy carried with the message (without the contribution of the data part itself). This entry is altered each time a message is routed, respecting the transmission efficiency ε.

Agent Type	*Behaviour*
Request	*Point-to-point agent*: this agent requests energy from a specific destination node, returned with a `Reply` agent.
Reply	*Point-to-point agent*: 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 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 for a couple of times and creates multiple deliver agents periodically in dependence of the energy state of the current node.
Deliver	*Path agent*: 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	*ROI agent*: this agent carries energy from and is instantiated on a good node and explores a path starting with an initial direction and searches a bad node supplying it with the energy.

Tab. 13.1 *Agents with different behaviour used to manage and distribute energy (ROI: region of interest).*

13.2.4 Analysis and Experimental Results of the SEM

A first proof of concept and experimental results were achieved by using a multi-agent simulation framework (*SeSAM*, see Section 11.1). The simulation test-bed consists of a network with N=100 nodes arranged in a 10 by 10

matrix. Each node can communicate with up to four neighbours in the directions *DIR*= {`North, South, West, East`}.

Each node N_i periodically collects (store) energy from a local source having a stochastically distributed energy spectrum in the range $[0,E_{i,max}]$. Monte Carlo simulation is used to specify each $E_{i,max}$ before a simulation run. Data processing, interaction with the environment (e.g. sensor acquisition), and agents consume energy, which reduces the energy deposit E_i.

Each sensor node is modelled with an agent, too. Energy management agents and sensor node agents negotiate energy demands and communicate by using globally shared variables. If the energy deposit of a node is below a threshold $E_i<E_{low}$ (called bad node), help agents are sent out, if $E_i>E_{high}>E_{good}$ then distribute agents are used to distribute energy to surrounding bad nodes. If $E_i>E_{good}>E_{low}$ then the node is fully functional (called good node).

Assuming a specific stochastic spatial configuration $\{E_{i,max}\}$, simulation results in Figures *13.12* and *13.13* show the benefit of energy management using help- and distribute agents. Without energy management, there are about 50% bad nodes (blue rectangles) never reaching an energy level above a critical threshold. With agents, the spatial energy distribution is more regular, and the fraction of bad nodes is below 10% all the time. The Figure *13.14* compares the number of bad nodes resulting from different combinations of agents and agent behaviours.

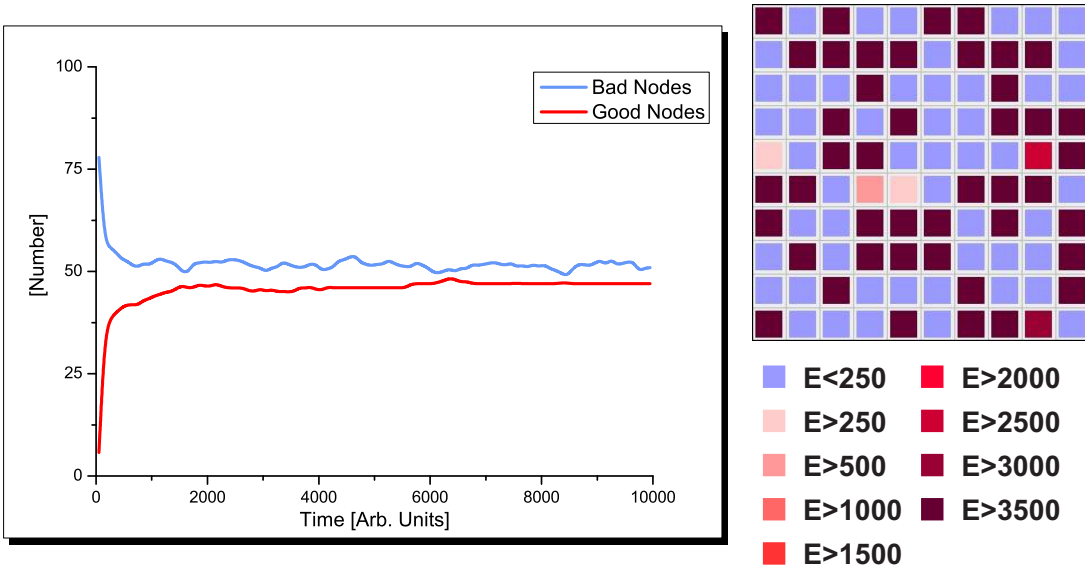

Fig. 13.12 *Simulation results for a network of sensor nodes arranged in 10 by 10 matrix topology without any distributed energy management. Shown are the spatial energy distribution after 10000 time steps (right) and the temporal population for bad and good nodes (left).*

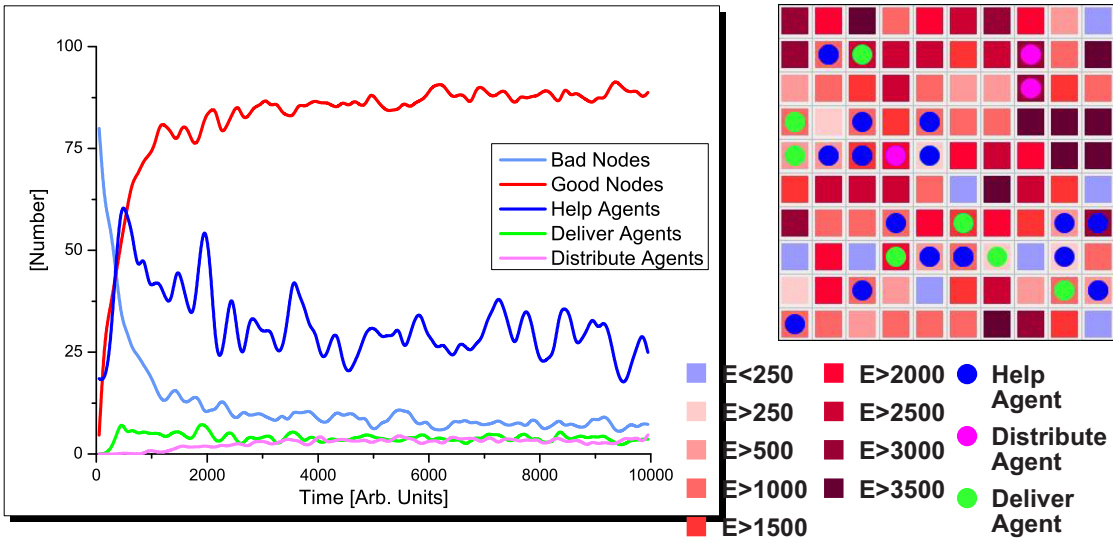

Fig. 13.13 *(Left) Simulation results with energy management. Help, deliver, and distribute agents are used to compensate low-energy nodes. (Right) Agent deployment in network*

Using additional distribute agents results in a decrease of 30% relative to the case using only help agents, but absolutely the benefit is below 5% and is therefore negligible. Moreover, the fraction of all nodes with mean up-time below a critical threshold (10%) is always below 5%.

The Figure *13.14* shows the temporal progress of total system energy in dependency on different energy management agents, too. Due to the high loss of energy transfer between nodes (here 20%), the total energy efficiency is dramatically decreased compared with the case without management, and distribute agents reduce the total system efficiency again about 50%.

The multi-agent implementation offers a distributed management service rather than a centralized approach commonly used. The simple agent behaviours can be easily implemented in digital logic hardware (application-specific platform approach).

To summarize, help agents with simple exploration and exploitation behaviours are suitable to meet the goal of a regular energy distribution and a significant reduction of bad nodes unable to contribute sensor information, but additional distribute agents create no significant benefit.

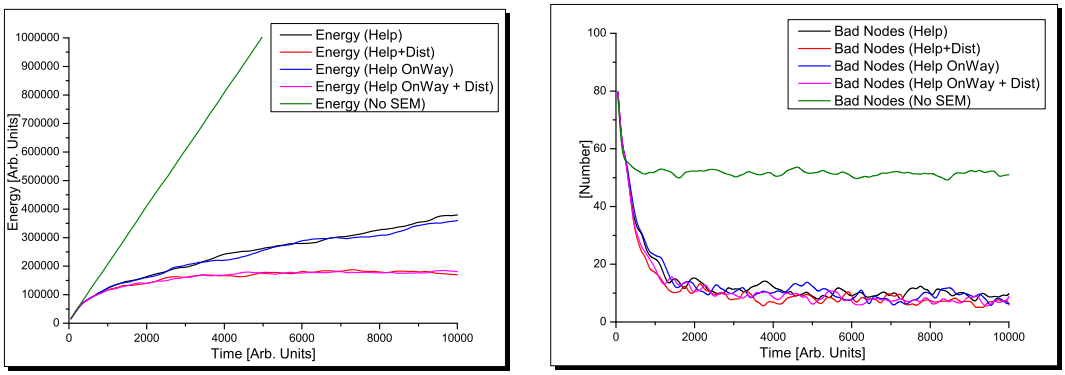

Fig. 13.14 *Comparison of total sensor network energy (left) and bad node population (right) for different energy management agent systems*

13.2.5 Algorithmic Selection and SEM

In Section *9.4* a self-organizing MAS was introduced that is capable to distribute energy between nodes of a sensor network. The algorithmic analysis and selection approach introduced in the previous section can be a prerequisite for the specification of the various parameters of the mobile smart energy management (SEM) and node energy management (SEN) agents. The SEM and SEN agents make assumption about the cost of computation. The methods from the previous section can be used to select different SEM and SEN behaviour and to compute the reward for energy distribution and help requests. The algorithmic selection approach can be embedded in the agent transition process to add energy saving goals that can be related to the original activity and mobility goals.

13.3 Further Reading

1. D. Kedar and S. Arnon, *Optical wireless communication in distributed sensor networks*, SPIE Newsroom, 2006.
2. S. Bosse, D. Lehmhus, W. Lang, M. Busse (Ed.), *Material-Integrated Intelligent Systems: Technology and Applications*, Wiley, ISBN: 978-3-527-33606-7 (2018)
3. Clemens Moser, *Power Management in Energy Harvesting Embedded Systems*, PhD Thesis, ETH Zürich, 2009, Shaker, ISBN 978-3832282059
4. S. Borlase, Ed., *Smart Grids - Infrastructure, Technology, and Solutions*, CRC Press, 2013, ISBN 9781439829103