Material-integrated Cluster Computing in Self-Adaptive Robotic Materials using Mobile Multi-Agent Systems

Stefan Bosse[1], Dirk Lehmhus[2]

[1]University of Koblenz-Landau, Faculty Computer Science, Koblenz, Germany
[2]Fraunhofer IFAM, Bremen, Germany

Abstract Recent trends like Internet-of-Things (IoT) and Inernet-of-Everything (IoE) require new distributed computing and communication approaches as size of interconnected devices moves from a cm^3- to the sub-mm^3-scale. Technological advance behind size reduction will facilitate integration of networked computing on material rather than structural level, requiring algorithmic and architectural scaling towards distributed computing. Associated challenges are linked to use of low reliability, large scale computer networks operating on low to very low resources in robotic materials capable of performing cluster computing on micro-scale. Networks of this type need superior robustness to cope with harsh conditions of operation. These can be provided by self-organization and –adaptivity. On macro scale, robotic materials afford unified distributed data processing models to allow their connection to smart environments like IoT/IoE. The present study addresses these challenges by applying mobile Multi-agent systems (MAS) and an advanced JavaScript agent processing platform (JAM), realizing self-adaptivity as feature of both data processing and the mechanical system itself. The MAS' task is to solve a distributed optimization problem using a mechanically adaptive robotic material in which stiffness is increased via minimization of elastic energy. A practical realization of this example necessitates environmental interaction and perception, demonstrated here via a reference architecture employing a decentralized approach to control local property change in service based on identification of the loading situation. In robotic materials, such capabilities can support actuation and/or lightweight design, and thus sustainability.

Keywords: Pervasive Computing, Ubiquitous Computing, Agents, Optimization, Material Informatics, Self-organizing and self-adaptive systems

1 Introduction

Cluster computing is commonly related to large-scale networks composed of powerful computers and servers connected by reliable high-bandwidth interconnections and the Internet. There is an ongoing trend to create big machines providing high storage, memory, and CPU capacities towards Cloud computing and data centers. This is the macro-scale level of computing. But there is also an emerging micro-scale level of computing, using low-resource and miniaturized computers towards the mm^3 scale deployed in material-integrated computer networks and the Internet-of-Things **[1]**. The term material is related to the micro-size-scale level (element size about mm^3), and structures to the meso-size-scale level (element size about cm^3).

In the last decades there was a shift from passive single sensors and centralized sensor data processing towards large networks **[2]** of smart sensors equipped with Information-Communication-Technologies (ICT) forming Sensorial Materials as one class of Material-Integrated Intelligent Systems (MIIS) **[3]**. Furthermore, there was a significant increase of the sensor density in sensor networks integrating sensors and ICT in materials **[3]**, used, e.g., for Structural Health Monitoring (SHM) **[4]** or in smart textiles. Finally, micro system technologies enable the integration of actuation in materials creating Robotic Materials **[5]**, e.g., using thermoplastic actuators **[6]** to modify structures.

Robotic Materials are characterized by a tight coupling of sensing, actuation, computation, and communication. Data processing in Robotic Materials can be considered as cluster computing on micro-scale level. Sensorial and Robotic Materials are operating under harsh environmental conditions requiring resilient behaviour, captured by the entire design of the ICT architecture and distributed algorithms.

Load-bearing structures are typically designed towards relevant load cases known at design-time. New technologies enabling the design of structures that change material properties in service in response to load change (i.e., using robotic materials) could raise additional weight saving potentials, extending the lifetime, and increasing operational safety. Thus, self-adaptive Robotic Materials support lightweight design and sustainability inspired by bionic design, e.g., by a top-down approach based on the structure, load, and function with similarity classification **[7]**. But still bionic design has to consider specific load cases and perform classification in advance, e.g., performed by the ELiSE design framework **[8]**. Examples of cross-sectorial and material-independent structural lightweight constructions propose weight savings about 20% - 50% **[9]**. Our approach aims to overcome this limitation.

Structures composed of simple and unreliable actuators are difficult to control due to non-linearity and a lack of physical models. One concern regarding active smart cellular structures is the correlated and self-organizing control of cells' responses, and the underlying informational organization providing robustness.

Robustness and real-time capabilities require some kind of self-organizing and self-adapting computational model instead of the functional models commonly used (e.g., in control theory).

The adaptation of mechanical actuated structures to varying load situations based on external perception or proprioception that is considered in this work can be treated as an optimization problem that will be controlled in this work by Multi-agent Systems.

The next section 2 outlines the general problem description of distributed computing in material-integrated ICT systems and the novelty of this work.

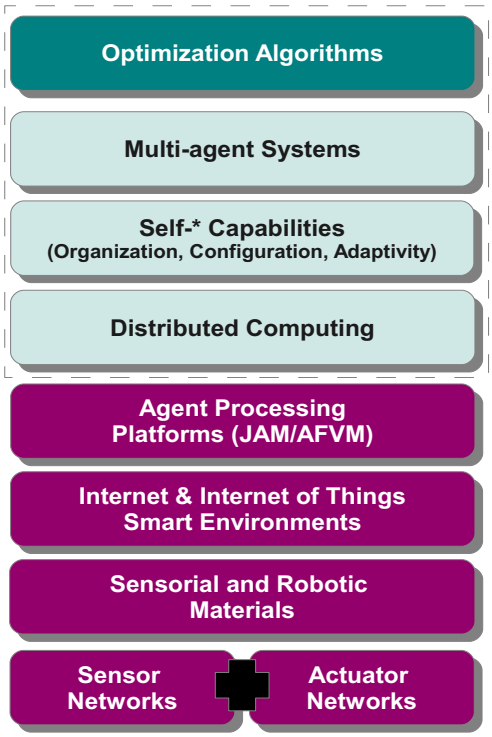

Figure 1. Overview of the concepts fusioned in this work (lower three levels are the deployment fields)

Section 3 discusses data processing integrated in materials and structures from a technical point of view. Robotic materials and the reference architecture used in this work is described in Section 4. The optimization problem addressed in this work is introduced in Section 5, and the MAS implementation is described in Section 6.Sections 7, 8, and 9 addresses the agent and simulation platform. Finally, sections 10 and 11show and evaluate a use-case study.

2 Problem description and contribution

An overview of the concepts addressed in this work is shown in Fig. **1**. Assuming large-scale sensor- and actuator networks (i.e., sensorial and robotic materials) that are integrated in materials (or microscopic structures), distributed computing problems are solved by self-* Multi-agent Systems, finally performing distributed optimization (adaptation of mechanical properties of a robotic material). The four upper layers (green boxes) are addressed in this work.

The lower four layers are the platform and deployment fields of the proposed distributed computing approach using mobile agents, not addressed in this work.

Optimization problems are commonly solved mathematically and numerically **[10][11]**or treated as global problems **[6][12]** that can only be solved by processing the entire sensor data set in each solver iteration. But the transition to distributed approaches with local processing is necessary to reach scaling of large systems in the future consisting of Thousands and Millions of small perceptive and computational nodes (embedded computers). These tiny computational units are characterized by their low computational power and data storage capacity.

This work addresses distributed information processing integrated in materials and structures posing self-adaptivity of the material using a MAS approach with algorithms based on Multi-Phase Topology Optimization (MPTO) **[13]** and simulated annealing, which can be considered as a bionic structural optimization approach..

On the macro-scale level, agents and distributed agent-based systems are already deployed successfully in heterogeneous large environments, e.g., production and manufacturing processes **[14][15]**, facing adaptive manufacturing, maintenance, evolvable assembly systems, quality control, and energy management aspects, and in sensing applications, e.g., monitoring of mechanical structures and devices **[16]**. Finally, the paradigm of industrial agents meeting the requirements of modern industrial applications by integrating sensor networks was introduced in **[33]**. In the present study, we show that agents can be deployed successfully on the micro-scale level, too.

The central approach in this work focuses on mobile agents solving an optimization problem by a divide-and-conquer approach. They pose the ability to support mobile reconfigurable code embedding the agent behaviour, the agent data, the agent configuration, and the current agent control state, finally encapsulated in a portable textual representation.

In this work JavaScript code (*AgentJS*) is used and executed by the *JAM* platform **[17]**. The code is capable of migration between nodes in the network required for autonomous distributed data processing. This approach requires only a minimal Agent Processing Platform Service (APPS). The AgentJS code can be directly executed by the underlying *JS* VM (e.g., *node.js*, *jxcore*, *JVM*, *webview* for mobile App. development, or *spidermonkey* used in browsers).

On the one hand, Robotic Materials provide internal and external perception that can be used in a wide range of sensing applications on the Internet (e.g., product life-cycle management). Robotic Materials will be part of larger networks, i.e., the Internet-of-Things. On the other hand, Robotic Materials can profit from environmental information, e.g., collected by mobile devices and crowd sensing. Therefore, these material-integrated computational networks should be connected to a local Intranet or to the global Internet (see Fig. **2**, left side).

One of the major challenges in distributed sensing and control systems is the derivation of meaningful information from sensor data. Often the sensors of mobile consumer devices (such as accelerometer, humidity, light, battery, temperature, and location) suffer from a poor quality. Distributed sensor fusion can be applied to improve the statistical significance of such sensor signals by collecting sensor data in a region of interest from multiple devices. Fusion can profit from Machine Learning (ML), which usually bases on classification algorithms derived from supervised machine learning or pattern recognition using, e.g., self-organizing **[2]** and distributed multi-agent systems with less or no a-priori knowledge of the environment.

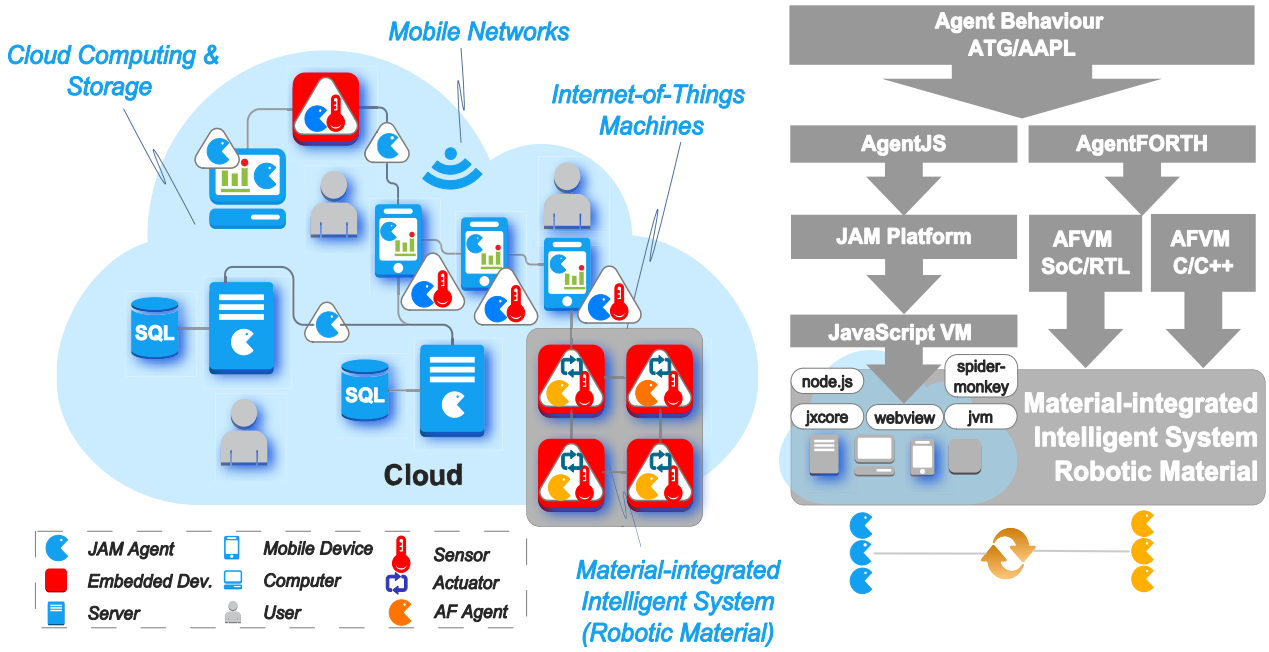

Figure 2. Unified distributed information processing in heterogeneous MIIS / IoT / Cloud environments with mobile agents using a hybrid platform framework consisting of the JavaScript Agent Machine Platform (JAM) and the low-resource *AgentFORTH* machine (AFVM)

The IoT, material-integrated ICT networks, mobile, and Cloud environments differ significantly in terms of resources (computational power and data storage). The IoT and mobile networks consist of a large number of low-resource devices interacting with the real world and having strictly limited storage capacities, energy, and computing power, and the Cloud consists of large-scale computers with arbitrary and extensible computing power and storage capacities in a basically virtual world.

A unified and common data processing and communication methodology is required to merge the IoT with Cloud environments seamlessly, which can be fulfilled by mobile agent-based computing proposed in this work.

The scalability of complex ubiquitous applications using such large-scale cloud-based and wide area distributed networks deals with systems deploying thousands up to a million agents.

Considering such strong heterogeneous environments with computers ranging from 1 MIPS computational power and 1 MB RAM to 1 GIPS and 1GB RAM a hybrid dual-platform approach have to be used for agent processing (Fig. 2, right side).

Addressing the Internet, mobile networks and the IoT, the *JAM* platform **[17]** is used offering agents directly implemented in JavaScript (*AgentJS*) program code holding the entire control and data state of an agent

Agent processing on very-low resource platforms is performed with a stack-based processor approach (executing *AgentFORTH* code) featuring hardware implementation and advanced token-based agent process scheduling using the *AFVM* platform **[18]**.

Both program code models base on the same behaviour and agent behaviour model (ATG/AAPL) and can be transformed to each other. The agents themselves conform to the mobile processes model introduced by Milner **[19]** ensuring seamless agent mobility. The code can be modified by the agent itself using code morphing techniques required for behaviour adaptation (directly supported by JavaScript Just-in-time and Bytecode Compiler VM platforms).

This work adds to earlier work **[20]** the following extensions and novelties:

- Distributed and decentralized solving of mechanical optimization problems using different algorithms;
- Self-* Multi-agent System performing distributed optimization in a robotic material providing self-adaptation based on varying load situations and mechanical defects;
- Reference architecture of a Robotic Material composed of a mesh-grid network of agent processing platforms;
- Different enhanced distributed mechanical optimization algorithms and their evaluation applied to robotic materials that are used to meet mechanical constraints under varying load and damage situations (e.g. holes);
- Multi-domain simulation with tight coupling of physical and computational agent models.

The next sections offer an introduction to Material-Integrated Intelligent Systems and the Robotic Material reference architecture, followed by an introduction of the proposed distributed optimization algorithms and their implementation with MAS. Finally, an evaluation of the distributed MAS optimization is performed by a multi-domain simulation (coupling physical and computational models) of a *JAM* network embedded in a robotic material.

3 Material-integrated Intelligent Systems

Material-integrated intelligent systems feature materials with embedded data processing, communication, sensors, actuators, and energy supply. These materials can be classified in:

1. Sensorial Materials providing intrinsic or environmental perception capabilities by coupling sensing, computation, and communication.
2. Robotic Materials providing intelligent perception and mechanical adaptation capabilities by coupling sensing, actuation, computation, and communication.
3. Self-adaptive Robotic Materials providing a closed control loop using perception to adapt the material/ structure to changing load situations or defects.

All three classes use large-scale autonomous distributed networks with embedded systems characterized by low power consumption, low data storage resources, and low processing power. Commonly, the network nodes are self-powered by using local energy storage and harvesting.

The main fields of application of Sensorial Materials are Load and Structural Health Monitoring, the field of application of robotic materials are mechanical adaptive structures.

Intelligence is provided on software level featuring self-* capabilities. Adaptivity, reconfiguration, and healing in the presence of technical failures (data processing, networking, communication, low energy, sensor faults) are elementary features of resilient systems in safety-critical environments.

Load-bearing structures are typically designed towards relevant load cases assuming static shape and fixed sets of materials properties decided upon during design and materials selection. New technologies enabling the design of structures that could change local properties in service in response to load change could raise additional weight saving potentials, thus supporting lightweight design and sustainability. Materials with such capabilities must necessarily be composite in the sense of a heterogeneous build-up, exhibiting, e.g., an architecture consisting of networks with numerous active cells providing sensing, signal and data processing, communication, and actuation/stimulation capability **[4]** forming Robotic Materials and Structures **[21]**. One example for such a material is a special class of polymers being capable to change their elasticity based on the influence of optical, thermal, or electrical fields [5].

Although no specific technology assumptions were made in this work, there are already micro-scale computers suitable for such material-integrated computing. An example is the Micro MOTE M3 **[37]** providing about 1MIPS computing power and 4kB data memory with 0.1mm^2 chip area consuming less 100µW power. The FreeScale KL03 provides 50MIPS/2kB with 4mm^2 chip area and 3mW power consumption. Both microcontrollers base on ARM architecture.

4 Robotic Material and Reference Architecture

The reference architecture in this work consists of a three-dimensional grid network of controller nodes with physical and communication connectivity between nodes, shown in Fig. **3**. The physical links are actuators that can change mechanical properties of the material (e.g., stiffness). No technology-specific assumptions were made regarding communication links and actuators.

In our understanding, a self-adaptive Robotic Material provides the following major features:

1. Perception using various kinds of sensors, e.g., measuring of strain, displacement, temperature, pressure, forces;
2. Capability of changing local material and structure properties by actuators, e.g., stiffness or damping variation;
3. Integrated networks of information processing and communication technologies (ICT);
4. Distributed approach: Local sensor processing, aggregation, and actuator control; Global cooperation and coordination solving optimization.

The general model of such a Robotic Material is shown in Fig. **3**. It is assumed that the material (on micro-scale level) or structure (on meso-scale level) consists of volume elements (bounded regions of the material) that are connected with neighbourhood elements via links. The links should provide some kind of sensing (e.g., measuring the strain or displacement along the link main axis) and some kind of material or structure control providing a controllable actuator (e.g., modifying the stiffness of the link). A link can be a discrete part or a continuous region of the material. A set of actuators is connected to each node. Two nodes share an actuator (e.g., a damped spring of variable stiffness and damping).

Each element contains an embedded computer that is considered as a node in a mesh-grid communication network. Each node provides some kind of data processing (processor or digital logic RTL), data and program memory, communication, energy supply and energy management. Since the distributed sensing and control of the material should be performed by an agent-based approach each node has to provide an agent processing platform (APP).

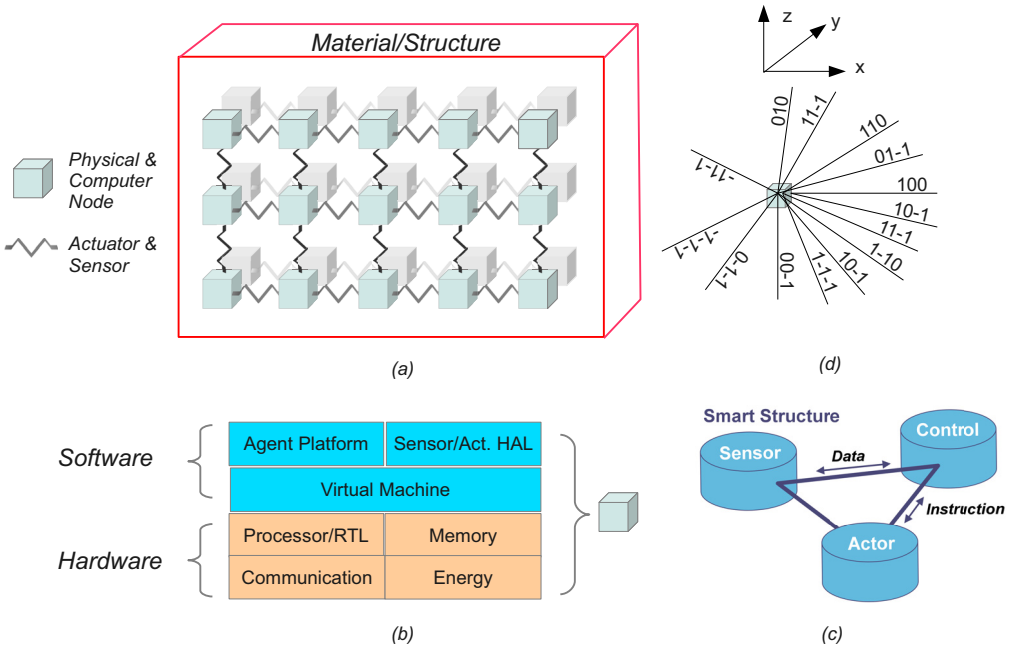

Figure 3. (a) General network architecture of a Robotic Material = Sensorial + Actuated Material (b) Hardware + Software architecture (c) Functional Decomposition: Sensing, Acting, Processing ↔ Data + Instruction Streams (d) Node connectivity (physical) of controlled actuators attached to neighbour nodes (shown are the delta vectors relative to a node position)

The APP is executed typically on a virtual machine, discussed in Section 7. Agents have to be able to access sensors and actuators by a hardware abstraction layer (HAL), optimally provided by the APP.

It is assumed that the nodes are organized in an ad-hoc way. Technical failures of single nodes is considered as the common and may not affect the operation of the Robotic Material and its capability to satisfy some global objective function. This implies a self-organizing and self-adaptive approach with respect to the connectivity structure has to be used to create some kind of holonic system and is the first level of intelligence in a smart system.

Each node controls a number of actuated springs and dampers connected to neighbouring nodes as shown in Fig. **3** (d). Each spring has a virtual strain (displacement) sensor attached, e.g., a strain gauge sensor. Together with the current stiffness parameter value, the spring energy can be computed. It is assumed that the stiffness of each spring can be varied between a lower and an upper boundary $[s_0,s_1]$. A technically reasonable range is ±50% around a nominal stiffness value. The missing sensor values (spring displacement and current stiffness) must be retrieved from neighbour nodes.

The models of the actuator (relation of relative displacement l-l_0 and force $f_{i,j}$ between two nodes) and the sensor (strain gauge signal $S_{i,j}$) are given in Eq. **1** and **2**, respectively.

$$f_{i,j}(l,s,d) = -k_0 s(l - l_0) - du \qquad (1)$$

$$S_{i,j}(f_{i,j}) = S_0 f_{i,j} GF, \text{ with } GF = \Delta R / (\varepsilon R_G) \qquad (2)$$

with s: stiffness parameter, k: a spring constant, d: damping parameter, u: relative velocity between nodes i and j, R: electrical resistance, GF: gauge factor, ε:strain.

The adaptivity of the robotic material requires perception based on sensors, interpretation of current and past load situation (locally or globally or both) and the control of the actuators. Besides the energy required to perform computation and communication, the most prominent part of the energy consumption during operation is caused by the actuators. The change (increase or decrease) of the stiffness of the springs commonly requires or releases energy (e.g., thermoplastic materials require controlled heating), or requiring energy in both directions, depending on the actuator technology (e.g., optically excited bi-stable materials)!

The entire computation and the entire agent processing implementing the structure optimization discussed in the next Section is performed by the material-integrated ICT network. There is no off-loading of parts of the computation to external computers.

5 Distributed Self-Optimization

The concept of self-adaptive robotic materials draws much of its appeal from the fact that engineering design has to decide which are the load cases or service conditions to base design on, while reality typically knows no such limitations: Conventional techniques like shape, topology or multi-material optimization all need to single out few or even one set of boundary conditions to adapt to. Any real

world load-bearing structure may see different scenarios including misuse, which might be impossible to capture and define in the design stage. Such unforeseen loads may in the worst case lead to immediate failure - in other cases, they might just wear out a structure, causing it to fail prematurely and/or in places determined by unexpected local load histories. A smart adaptive material or structure that is capable of adjusting its mechanical characteristics - like stiffness - to external loads could actively manage this local load history. It could protect areas already worn out and distribute loads to others instead.

There are three different optimization algorithms that are applied to materials of this kind and evaluated in this work:

1. Global optimization;
2. Segmented optimization;
3. Neighbourhood optimization (originally proposed in **[20]**),

summarized in Tab. **1**. They are originally based on the concept of Multi-phase Topology Optimization (MPTO) and simulated annealing used to optimize structures at design time by reorganizing material parameter distributions **[13]**. The algorithms are reactive and iterative, in contrast to classical mathematical minimization being commonly functional, applying small changes step-wise to satisfy the global goal and to minimize an error function. The global algorithm is presented here for reference only as it requires a central instance, although the action is performed locally but based on a global observable. The neighbourhood algorithm operates continuously without a defined termination, while the two other algorithms can terminate if they satisfy a bounded error condition.

The segment and neighbourhood algorithms are suitable for distributed processing performing region exploration and negotiation, characterized by self-* features (self-adaptivity and robustness in case of sensor or node failures and changes in the network connectivity). That means, there are multiple instances operating on spatially bounded data (local data) satisfying global objectives, e.g., minimizing the mechanical strain energy of a structure. One major issue in distributed systems without a central instance is the efficient collection and computation of observation variables (observables) like the accumulated strain energy of a region of nodes. This is discussed in the next section. The optimization of the structure is performed by modifying a target variable (control parameter), e.g., the stiffness of the actuators, based on a comparison of a local observable (e.g., the local node strain energy) with a global or glocal (region) observable, e.g., the strain energy of nodes in a region around the particular node or just a global accumulated value.

The total strain energy U of a mechanical structure is the integral of the strain tensor ε over the volume V of the structure. The discretized strain energy for each node u_i is computed from all attached springs $j=\{1,..,n\}$ with their displacement d_j and current stiffness s_j. The total discretized strain energy of a structure is the sum over all node strain energies given by Eq. **3**.

Table 1. Different optimization algorithms (simplified) for material/structure adaptation, with $\mathbb{N}$: Set of nodes, $\mathbb{S}$: Set of segments $\boldsymbol{\mathcal{S}}$, U: Total strain energy; ε: Strain; σ: Stress; x_i: observation variable of i-th node (element), x: Some observation variable; X: Some accumulated observation variable; r_i: Optimization ratio parameter for i-th node, $|X|$: Cardinality of a set X, R: Some radius, D: Some discretization function; s_i: Stiffness of i-th node/element, $[s_0,s_1]$: Limits of stiffness, $[r_0,r_1]$: Limits of optimization parameter, k_x: Some problem specific mapping and scaling function with weight factor w; $|pos_1\text{-}pos_2|=1$: Neighbourhood relationship of elements or segments.

Global Algorithm

do with $x \in \{\varepsilon,\sigma,U\}$
 $X:=0$; $\forall n\in\mathbb{N}$ do $X := X + x_n$
 $X := X/|\mathbb{N}|$
 $\forall n\in\mathbb{N}$ do
  $r_n := k_x(x_n/X, s_n)$
  $s_n := s_n * r_n$
until $|Err| < Err_0$
$k_U^1 :(q,s) \rightarrow$ if $q*w\in[r_0,r_1] \vee$
 $s\notin[s_0,s_1]$ then 1 else
 elsif $r*w<r_0$ then r_0 else r_1
$k_U^2 :(q,s) \rightarrow$ if $s\notin[s_0,s_1]$ then
 1 elsif $q*w\in[r_0,r_1]$ then $q*w$
 elsif $r*w<r_0$ then r_0 else r_1

Segment Algorithm

do with $x \in \{\varepsilon,\sigma,U\}$
 $\mathbb{S} = \{\boldsymbol{\mathcal{S}}_i\}$
 $\boldsymbol{\mathcal{S}}_i=\{n_i,\{n_j\} \in \mathbb{N} \mid i\neq j \wedge$
  $|pos(n_i)\text{-}pos(n_j)|\leq R\}$
 $\forall\boldsymbol{\mathcal{S}}_i\in\mathbb{S}$ do
  $X_s:=0$;
  $\forall n\in\boldsymbol{\mathcal{S}}_i$ do $X_s := X_s + x_n$
  $X_{s,i} := X_{s,i} / |\boldsymbol{\mathcal{S}}_i|$
  $\forall n\in\boldsymbol{\mathcal{S}}_i$ do
   $r_n := k_x(x_n/X_{s,i}, s_n)$
   $s_n := s_n * r_n$
until $|Err| < Err_0$

Neighbour Algorithm

do
 $\forall\{n_i,n_j \in \mathbb{N} \mid i\neq j \wedge$
  $|pos(n_i)\text{-}pos(n_j)|=1\}$ do
  if $u_i < u_j \wedge$
   $s_i\text{-}\Delta s > s_0 \wedge$
   $s_j+\Delta s < s_1$ then
   $s_i := s_i - \Delta s$
   $s_j := s_j + \Delta s$
  end if
always

$$U = \int_V \varepsilon^T \sigma \varepsilon dV$$
$$u_{node} = \sum_{i=1}^{n} d_i{}^2 s_i \quad (3)$$
$$U = \sum_{i=1}^{N} u_i$$

There is a set of observation variables: Strain ε, Stress σ, and the strain energy U finally computed, which is an objective variable, too. The optimization variable is the stiffness of elements ε, which is used to minimize the strain energy, stress, or strain. Thus, the optimization (objective) function is: min$U$(ε,σ) with ε(e) and σ(e) being a function of the e module (stiffness).

The basic principle of optimization uses a ratio parameter r based on the relation of observation variables (either ε, σ, or U) to a spatially extended ensemble value of this variable (mean). The ratio parameter is applied to the target variables, i.e., the stiffness of elements of the material/structure. In Tab.**1** there are two different example functions k_U^1 and k_U^2 shown that compute the actual ratio parameter using the strain energy $U(=x)$.

There is a set of all nodes: $\mathbb{N}$, a set of all segments grouping nodes (uniquely or overlapping): $\mathbb{S}$, and segments with a sub-set of nodes: $\boldsymbol{\mathcal{S}} \subseteq \mathbb{N}$. Each segment is centered around a node of the network, i.e., neighbour segments overlap that corresponds to a moving window. The segmented algorithm performs modification of elements partitioned in segments based on locally bounded data. The neighbourhood algorithm performs element modification between two neighbouring nodes only (point-to-point).

6 Self-* MAS

Self-* capabilities are addressed in this work by (1) Distributed data processing and (2) Mechanical optimization. The MAS in combination with used distributed algorithms pose self-capabilities on different levels (compare **[22]**):

1. Self-configuration
2. Self-organization
3. Self-adaptivity
4. Self-healing

There is no a-priori knowledge of the network and cluster structure or any world model. The global system behaviour is a result of self-configuration and self-organization of the agents on a local domain scope and by neighbourhood relations. Self-adaptivity is provided by means of (1) Data processing in the presence of technical failures of nodes or communication and changing network connectivity, and (2) Algorithmic adaptation of mechanical properties of the structure based on perception and negotiation as part of the optimization problem. Self-healing is provided by the MAS (1) By locating and isolating faults (technical failures) **[23]**, and (2) By compensating defects or holes in the mechanical structure by adapting mechanical properties.

Node controller agents haveto control a set of springs pointing to neighbouring nodes and sensors, shown in Fig. **3**d. Each spring has a virtual sensor attached delivering the strain (displacement between two nodes). Together with the current stiffness parameter value of a spring, the spring energy can be computed. It is assumed that the stiffness of each spring can be varied between a lower and an upper boundary $[s_0,s_1]$. A technical reasonable range is ±50% around a nominal stiffness value. The missing sensor values (spring displacement and current stiffness) from other neighbour nodes are delivered by remote tuple operations (see Fig. **4**b).

The MAS has the goal to minimize a global or local mechanical variable, e.g., the total strain energy U and the strain by using an observation variable. The control is performed by modifying a control variable, e.g., the spring stiffness. The three different algorithms introduced in Sec. 5 require different MAS behavioural models. The neighbourhood algorithm bases on negotiation and uses a simple stiffness swap algorithm, discussed below. The segment algorithm bases on a region mean of the observable. i.e., the mean strain energy in the region. The mean value is used to modify the control variable based on a simple decision tree.

The MAS is composed of different agent classes. All approaches (Tab. **1**) uses node agents but apply different behavioural models that can be united in one agent enabling the selective change of the optimization approach and objectives. The negotiation approach uses an additional broker agent for negotiation, having a distinct behaviour described below and illustrated in Fig. **4**. Additional explorer and notification agents can be used for segmented or regional action (e.g., to compute a global observable by random walk and directed diffusion).

A. Neighbour Negotiation

Node Agent

The negotiation approach considers only observables of two neighbour nodes. The node agent is a stationary agent. Its behaviour consists of the following main activities.

init

The initialization activity mainly creates a broker agent, starts the adaptation handler timer, and set up the sensor acquisition and spring control.

perceive

Getting notifications via the tuple-space data base. Listening for tuples: {`SENSOR`, `STIFFNESS`, `ADAPT`}. The `SENSOR` tuple is either sent by the world agent or by surrounding neighbour node agents. The `ADAPT` tuple is sent by the broker agent if there is a change in the spring stiffness that was negotiated (either an increase or decrease). All controlled springs of a node are always set to the same stiffness.

process

New sensor data is processed, and the current strain of each spring and the total node strain energy is computed. Relevant sensor distribution and negotiation events are started. In the case of an energy event (i.e., the current strain energy is above a threshold), an alarm tuple `ALARM(energy, stiffness, ..)` is created that is consumed by a waiting and listening broker agent.

notify

The notification activity distributes sensor and spring data to neighbour nodes by sending remote `SENSOR` tuples. The attached broker agents is notified by `ENERGY` and `ALARM` tuples. Based on sensor data of all known springs attached to this node a `NEIGHBOURS` tuple is sent to the broker to inform about connectivity.

adapt

This handler changes the stiffness parameters of all controlled springs. To avoid high stiffness changes, a low-pass filter is applied. This activity is executed periodically by a timer. Again, all springs are set always to the same stiffness.

Broker Agent

The negotiation and broker agent is a mobile agent and is responsible for the neighbourhood self-adaptation of the material based on the current load situation in the neighbourhood and self-regulation. It interacts with the node agents and other broker agents via tuple exchange.

Its behaviour consists of the following main activities:

perceive

Getting notifications via the tuple-space data base. Listening for tuples: {`ENERGY`, `ALARM`, `NEIGHBOURS`, `VOTE`}. The `ENERGY` tuple informs about the current strain energy and stiffness setting of the node, whereas the `ALARM` tuple triggers a voting cycle (transition to vote activity). A `VOTE` tuple containing a `SWAP?` request initiates an election.

vote

A voting cycle is initiated by sending a `VOTE(SWAP?)` request tuple to a randomly chosen neighbourhood node. During the voting or election cycle further `ALARM` and `SWAP?` requests from other nodes are blocked (ignored).

notify

If this broker started a voting cycle and got a `SWAP+` response, it notifies the node agent about the successful stiffness swap negotiation by sending an `ADAPT` tuple to commit the election result.

election

This is the election handler managing a swap election initiated by the first `SWAP?` vote after a time-out. The election determines the majority vote among all collected votes. If the major vote with the highest energy can be granted, the voter gets a `SWAP+` declaration, otherwise it is declined with a `SWAP-` declaration, and also all voters losing the competition get a `SWAP-` declaration. The decision for the stiffness swapping bases on the local strain energy and stiffness, compared with the major requesting node's energy and stiffness, i.e., *swapit=this.energy < major.energy && this.stiffness-major.request > this.stiffness.low*.

B. Segment and Global Control

Node Agent

The non-negotiated distributed control requires only one node agent per node. The original numerical segment approach (Tab. **1**) partitioned the network in rectangular segments. This requires a central instance. To avoid such external instance, each node creates a segment region around its position posing a set of overlaid moving window segments. Each segment implements a virtual sensor of the observable **[24][25]**. In contrast to the negotiation approach an extended region observable is collected in the segmented approach, i.e., the accumulated strain energy of nodes in the neighbourhood within a distance n (e.g., 2). The actuator modification (i.e., stiffness of springs) bases on the ratio of this accumulated observable and the local observable value. Node agents communicate with each other directly via remote tuples or signals. The modification of the target variable (spring stiffness *s*) can be step-wise and discretized, i.e., applying a small fixed delta change based on the observable ratio (u/U) resulting finally in a minimum/maximum stiffness distribution, or linear using the observable ratio directly for the computation of a new correction value for the target variable:

```
Step-wise:
  if u > U+ΔU then s=s+Δs
  else if u < U-ΔU then s=s-Δs
Linear:
  s=s0*u/U*k
```

init

The initialization activity explores the neighbourhood connectivity and starts the adaptation handler timer, and set up the sensor acquisition and spring control.

perceive

Getting notifications via the tuple-space data base. Listening for tuples: {`SENSOR`, `OBSERVABLE`}. The `SENSOR` tuple is either send by the world agent or by surrounding neighbour node agents. The `OBSERVABLE` tuple is sent by neighbouring nodes (alternatively using remote signals, see below)

process

New sensor data is processed, and the current strain of each spring and the total node strain energy is computed.

notify

The notification activity distributes sensor and computed observable data to neighbour nodes by sending remote `SENSOR` and `OBSERVABLE` tuples or signals. To prevent

high notification rates, the notify activity is inhibited for some time if there was recently a notification event.

adapt

All actuators (springs) of a node are updated (setting new stiffness values based on computation and perception).

Signals and Handlers

Depending on the data distribution approach, signal handlers are used to receive neighbour and to trigger actuator updates (signals `SENSOR`, `OBSERVABLE`). Instead, using an actuator update activity (`adapt`) that has to be activated on an event, a signal handler can be used that is activated by a periodical timer.

C. Agent Behaviour: Explorer and Notify Agent

The explorer and notify agents explore a larger radius of neighbourhood (not limited to direct node neighbours) to collect and distribute sensor data and physical state information (e.g., a mean strain energy of a segment region or the global observable mean). Commonly a central instance is required for global action. Random walk and directed diffusion behaviour can be used to approximately compute and distribute a global observable without a central instance. The evaluation of the proposed distributed processing from Sec. 9. shows that there is no strict requirement for any global observable. An observable mean in a large region is sufficient.

D. Distributed Computation of Observables using Chains

Another approach without a central instance is neighbourhood exploration and chaining. Each node propagates its current local observable value (e.g., the node strain energy u_0) to all its direct neighbour nodes via remote signals if the observable has changed. After the first delivery phase, each node has neighbourhood information of distance 1. Each next delivery can deliver distance 2 values by mean computation u_1 of orthogonal node observables with respect to the current delivery direction and the over next node observable in the opposite direction (u_2), shown in Fig. **4**a. Finally, each node can compute a region value of the observable. The propagation of observable values to neighbour nodes is given by the following algorithm:

```
∀ dir ∈ {NORTH,SOUTH,WEST,EAST,UP,DOWN} do
 ne=neighbours,u0=this.u
 dir=WEST|EAST?: u1=ne[NORTH].u0+ne[SOUTH].u0
 dir=NORTH|SOUTH?: u1=ne[WEST].u0+ne[EAST].u0
 dir=NORTH|SOUTH?:   u2=ne[SOUTH|NORTH].u0
 dir=WEST|EAST?:     u2=ne[EAST|WEST].u0
 dir=UP|DOWN?:       u2=ne[DOWN|UP].u0
 sendto(dir, { u0:u0,u1:u1,u2:u2 })
```

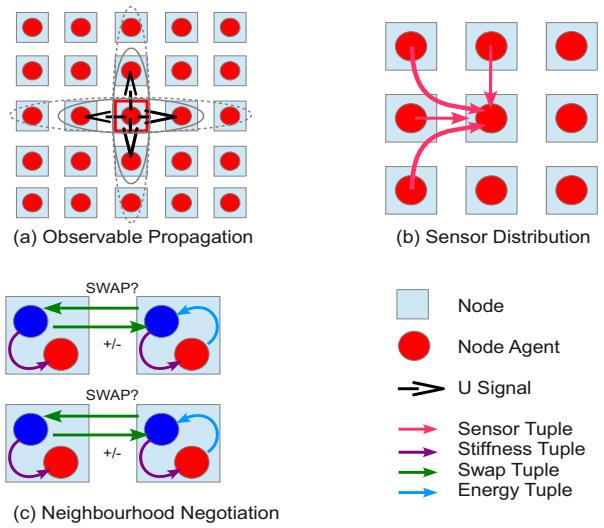

Figure 4. MAS interaction: (a) Observable propagation using remote signals (b) Sensor distribution using remote tuples (c) Negotiation in neighbourhood algorithm using remote tuples

7 Agent Platforms

The selection of an appropriate agent programming model (APM) and agent processing platform (APP) is eminent for the foreseen use case of material-integrated ICT networks that are connected to the Internet and Cloud environments.

The deployment of a single APP (the agent VM, commonly, e.g., the *JAVA*-based *JADE* platform) in such strongly heterogeneous environments with different constraints and organizational structures is not possible. Therefore, a set of APPs have to be deployed. But all APPs should address the same APM to enable cross-over migration.

There are only few agent programming models that cover very low-resource processing platforms. The *AAPL programming model* (a meta model, detailed description in **[17][18]**) relies on the Activity-Transition Graph behaviour model given by an agent class template containing activity and transition sections. *AAPL* is an abstract programming meta model that can be implemented with different programming languages. The agent is composed of activities (graph nodes) with conditional or unconditional transitions based on agent data (perception and state). *AAPL* offers statements for parametrized agent instantiation, like the creation of new agents and the forking of child agents inheriting the parent state. Unified agent interaction is provided by using synchronized Linda-like tuple database spaces and signal propagation (messages carrying simple data delivered to signal handlers). Agent mobility (migration) is provided by transferring the entire state and code of an agent process. The destination can be specified by a geometric relation and a direction (e.g., North, South, ..), a delta-dis-

tance vector usable in mesh grids, a host port, or an URL/IP address.

There are actually two different programmable agent processing platforms supporting agents based on the meta agent programming model *AAPL*: The *Agent FORTH* Virtual Machine (*AFVM*)**[18][26]** as a low-resource platform and the JavaScript Agent Machine (*JAM*)**[25]** as an Internet platform.

A. AFVM Platform

The *AFVM* platform (see Fig. **5**, right) is a low-resource platform suitable for hardware integration (microchip level), requiring about 1000k logic gates (~1mm^2 chip core area with a TSMC 65nm manufacturing process). Among a hardware implementation there are different software implementations fully operational and code compatible with each other that can be connected in heterogeneous networks. The *AFVM* is a multi-processor computer architecture using a stack processor providing multi-process scheduling (of agent processes) using a token and queue-based approach. Agents are implemented with code frames consisting of *FORTH* code with agent specific extensions and embedded data storing the entire agent control and data state. Code morphing enables agent behaviour modification by changing activity or transition functions.

B. JAM Platform

The *JAM* platform (see Fig. **5**, left) addresses the deployment in large-scale networks, i.e., the Internet (of Things), mobile networks, and Clouds. It requires a *JavaScript* execution platform, e.g., *node.js*, *jxcore*, or any JS capable Web browser. The deployment of *JAM* in large-scale world-wide networks in an Earthquake monitoring use-case was shown in **[27]**. Although both platforms support agents with nearly the same behaviour and operational model they are not compatible on code level. *JAM* agents are programmed in *AgentJavaScript* (**[28]**). *JAM* consists of a set of modules providing agent programming, agent control, agent mobility, agent interaction, and machine learning. Basically *JAM* is implemented as a library that can be integrated in any host application. The central Agent Input Output System (AIOS) module is the (secured and sand-boxed) interface between agents and *JAM*. Agents are always created from *JS* text code. An agent *JS* object stores the entire agent code and data. Due to functions being first order values in *JS* the agent behaviour can be changed by modifying activities or transitions directly without recompilation or recoding.

The mobility is granted by converting the agent program and process snapshot in a textual *JSON+* representation that can be sent via various communication technologies, and finally executing the code by parsing the text again. This mobile agent program code can be executed on a variety of different host platforms including mobile devices, embedded devices, macroscopic sensor nodes, and servers, using *JAM* and a generic *JS* VM, bridging the gap between the IoT and Internet/Cloud infrastructures.

C. Hybrid Architecture and the IoT

Although in this work MAS are implemented and processed in *JAM* networks, the technological realization of material-integrated computing requires the execution of MAS on the *AFVM* platform.

To enable the composition of future large-scale applications ranging from very-low-resource nodes (1mm^3 computers integrated in materials and structures) to high-resource devices (generic computers, servers, mobile and embedded devices), both platform architectures have to coexist supporting agent migration seamlessly. This co-existence demands a compatibility layer that can be realized by introducing an on-the-fly cross-compiler enabling agent mobility between different platform technologies seamlessly, shown in Fig. **5** (middle). This cross-compiler translates agents in *AgentFORTH* object code to *AgentJS* text code agents and vice versa by preserving the entire agent state. This compiler is optimally contained in *JAM* as a service.

8 Agent Communication

Communication is central in large-scale distributed systems with a high impact on the overall system performance and stability. Two main issues arise that affect design considerations: 1. Efficiency of the communication with respect to latency, computational complexity, and storage; 2. Addressing and delivery of messages to destination entities. Efficiency is a key factor in self-powered material-integrated low-resource computing networks. Commonly, end-to-end communication is established between processes using IP protocols and computer nodes having unique addresses, which is not suitable for large-scale material-integrated networks and mobile agents. The agent model usually implies autonomy and requires a loosely coupling to the environment, i.e., without a strong binding to a specific node. Moreover, there is usually no global knowledge of the current position of an agent in the network at all. Basically three approaches are available to exchange information between agents: Tuple-space access, signal messages, and agent migration. Tuple-spaces were proposed in **[29]** and **[30]** as a suitable MAS interaction and co-ordination paradigm. They are discussed below and compared in Fig. **6**.

A. Tuple-Spaces

Tuple-space communication exchanges data tuples between entities based on pattern matching, i.e., between processes and agents. The information exchange is data-driven and bases on the data structure and content of the data, and does not require any destination addressing or negotiation between communicating entities. Furthermore, tuple-space communication is generative, i.e., the lifetime of data can exceeds the lifetime of the producer. There are producer and consumer agents. A tuple-space can be characterized by a local bounding posing a limited access region, commonly limiting data exchange to entities executed on the same network node. Distributed tuple-spaces require inter-node synchronization and base on replication and some kind of distributed memory model.

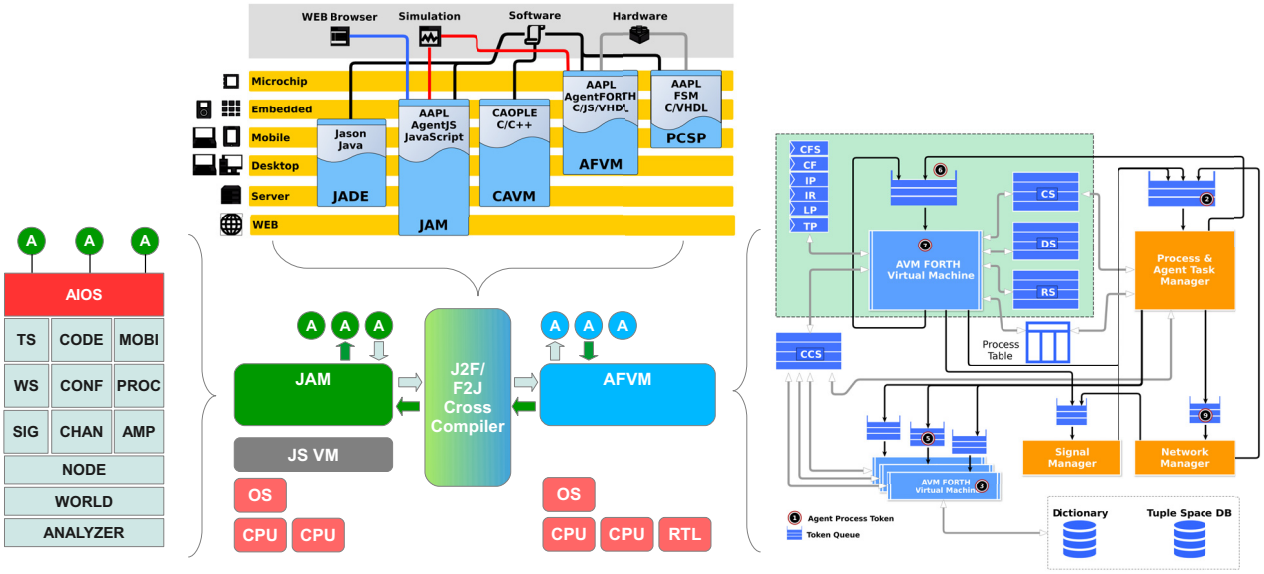

Figure 5. The hybrid architecture merges two Agent Processing Platforms: (Left) JAM & (Right) AFVM (Middle) J2F Bridge compiler (Top) Deployment areas compared with some other popular platforms; Legend: TS: Tuple Space, WS: Watchdog, AMP: Agent Management Port, CHAN: Channel, MOBI: Mobility, CONF: Code reconfiguration, SIG: Signals, AIOS: Agent Input Output System ,CS/DS/RS: Stacks, CCS: Common Code Segment

The *AAPL* agent model originally uses tuple-spaces only for agent communication executed on the same node.

B. Agent-to-Agent (A2A) Signals

Signals are lightweight messages that are delivered to specific agents (Agent-to-Agent, A2A), in contrast to the anonymous tuple exchange. One major issue is facilitating remote agent communication, i.e., agents executed on different network nodes. Although an agent can be addressed by a unique identifier, the path between a source and destination agent is initially unknown. For the sake of simplicity and efficiency, advance routing table management and network exploration is avoided. Instead, the *APPL* platforms (*JAM*/*AFVM*) support signal delivery along paths of mobile agents only. That is, that a signal from a source node *A* can only be delivered to a destination agent actually on node *B* iff the destination agent was executed (or created) on node *A* some time ago.

The concept thus requires that two agents had to be executed on the same node in the past. Agent migration and signal propagation is recorded by the agent platform using look-up table caches with time limited entries and garbage collection. Signals have the advantage of being delivered and proecssed asynchronously by signal handlers (although not pre-emptively) compared with tuples that have to be read or consumed by the agent explicitly.

C. Agent-to-Node (A2N) Signals

Previous agent platform implementations only support signal delivery along migration paths based on the destination agent identifier (private, uni-cast) or the agent class (public, broadcast). The new *JAM* platform 2.0 introduces signal delivery of signals to specific remote platforms (remote signalling) based on paths specified by the signal sender agent. The destination platform node broadcasts the signal to all listening agents executed on this particular node. To simulate private A2A uni-cast (or multi-cast) communication, agents can use a randomly generated signal name only known by the sender and the receiver. This new approach enables agent interaction between agents never executed on the same node. Furthermore, these remote signals are used to implement distributed tuple-spaces, discussed later.

D. Mobile Agents

Mobile agents can be used to distribute information in networks. They can get data/information from the tuple-space of the current node and store them in remote tuple-spaces by migrating to the respective nodes. The advantage of this approach is the ability to find suitable remote nodes and paths to nodes autonomously or based on content negotiation, and to filter, map, or process the collected data, e.g., using data fusion techniques. The disadvantage is a high communication and processing overhead caused by the agent process migration.

Additionally, mobile agents can be used to deliver data in mobile network environments by using a mobile device for spatial migration (piggyback approach, see **[25]** for details), not possible with A2A/A2N signals or remote tuples.

E. Distributed Tuple-Spaces

The new *JAM* platform 2.0 introduces the support for tuple migration using the *collect*, *copyto*, and *store* operations performed by agents. This feature enables the composition of distributed tuple-spaces controlled by agents. The *collect* and *copyto* operations transfer tuples from the local tuple-

space to a remote using pattern matching, similar to the *inp* and *rd* operations. The *store* operation sends a tuple to a remote tuple-space, similar to the *out* operation.

Remote tuple space access is performed by A2N signals.

F. Assessment of Communication

The scaling and performance of distributed and cluster computing is strongly related to the communication architecture. In this work, computation is coupled to a physical system and communication depends on changes of the physical system and the respective sensors (e.g., strain sensors). To evaluate and compare different communication approaches described previously, a simulation of a large scale network under real operation conditions was performed using the reference architecture from Section 4 and the multi-domain simulation from Section 9.

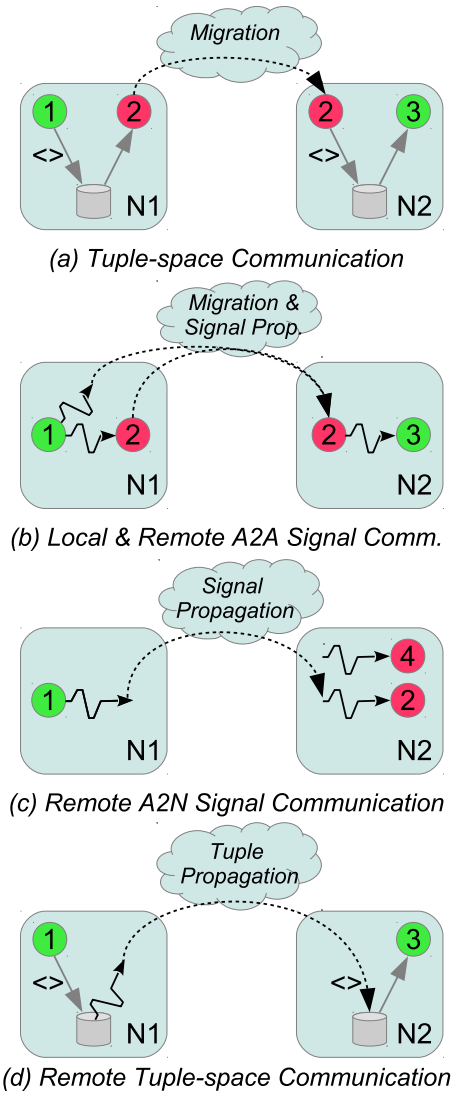

Figure 6. Agent communication in AAPL (a) Tuple-space communication between agents on same node (b) Agent-to-Agent Signals (c) Remote Agent-to-Node Signals (d) Remote Tuple Operation

The computational network consists of 8x5x3 nodes, and the physical Device under Test (DUT) is a plate (modelled with 8x5x3 body nodes). The network can perform event-based sensor distribution as discussed in Sec. 6. Results are shown in Fig. **7**.

The bottom plots show the event-based communication activity in the network as a response to a physical state change. Nodes and their node agents decide individually about distributing new sensor data avoiding the flooding of the network with messages. Three different physical configurations were compared resulting in different network behaviour: Full plate, plate with hole, and with an additional load (Fig. **7**, top). The last case causes the highest network activity due to the highest level of dynamic changes of the physical system.

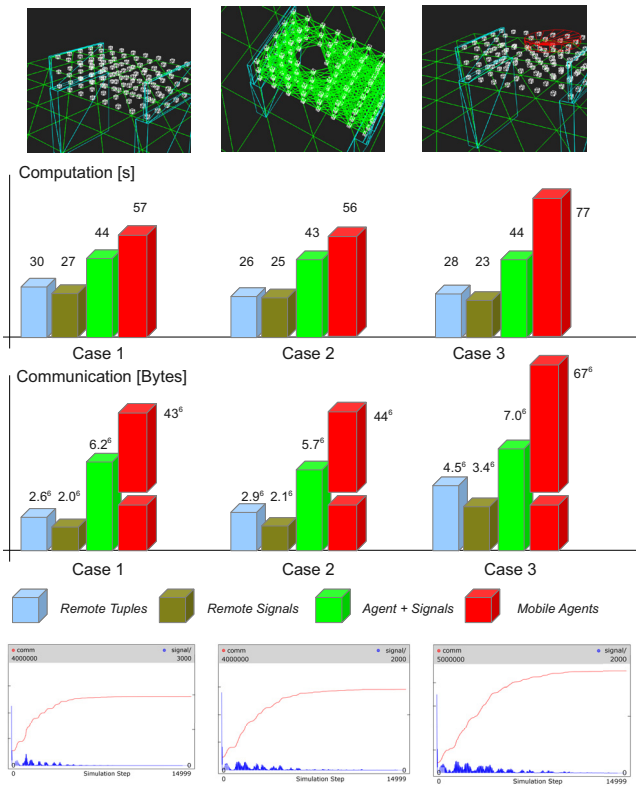

Figure 7. Computation and communication of a MAS: Simulation results and comparison of different communication strategies under different structure and load situations of the reference architecture. Action of the MAS: Event-based sensor distribution. (Top) Case 1: Plate, Case 2: Plate with large hole, Case 3: Plate with hole and an additional circular load on the top. (Middle) Total computation and communication after 15000 simulation steps (Bottom) Selected time plots of communication and signals for remote tuple approach.

9 Multi-domain Simulation

The design and technological implementation of Robotic Materials is a considerable challenge. Fundamental concepts have to be proven before any real system can be de-

veloped. Due to the strong coupling of sensing, reactive control, and information processing a multi-domain and multi-scale simulation was conducted posing a tight coupling of physical and computational models.

There are different mechanical models and solvers that can be used to compute the behaviour of the physical system in response to different load and use case situations and the reaction of a computational control system:

1. Finite Elements Analysis (FEA); and
2. Multi-Body Physics (MBP, based on coupled spring-mass element networks).

The second model is closer to the proposed computational mesh-grid node network architecture and structures on macro level, while the first model is better related to materials on micro level. Based on these two mechanical models two different computational frameworks performing applying structure/material optimization are used.

FEA-NUM: Finite Element Analysis is used to develop and evaluate optimization algorithms, performing, e.g., the minimization of a target function value numerically, e.g., reducing the total strain energy, local strain, or inner force peaks. FEA is stationary, i.e., no swinging or oscillation of the structure under test is considered, and a FEA simulation iteration ends with a stationary state of the structure. Algorithms operating on global (centralized) and local data (distributed) were originally investigated using *Abaqus* FEA simulation and *Matlab* performing the computational part of the algorithms **[31]**.

MBP-MAS: Simulation of Multi Body Physics in combination with MAS is used to investigate and evaluate the computational and coordination model proposed in this work (details in **[20]**). In contrast to static FEA the MBP simulation is dynamic and provides real-time resolution behaviour of structures. The MAS interacts with non-stationary states of the structures (swinging, oscillation, ..), too, which is closer to real-world interaction. The MAS implements the algorithms investigated in prior FEA.

One major difference between the FEA-NUM and the MBP-MAS approach concerns the sensor processing. The nearly real-time resolution MBP-MAS approach performs sensor distribution by MAS communication that introduces different time delays affecting the optimization algorithms and their outcome. Additionally, the MAS operates event-based, leading to varying responses to a sensor stimulus as a result of structure dynamics.

Monte-Carlo Methods: Complex distributed and coupled systems tend to be very sensitive to small parameter changes, i.e., small input changes (actio) result in large output changes (reactio). Therefore, it is essential to repeat simulations with varying randomized disturbances, e.g., by applying small variations to an initial set-up of the spring stiffness.

Using the *SEJAM* simulator, the physical and computational system can be simulated simultaneously studying the coupling and response of both systems to modifications.

The entire monolithic simulation framework is shown in Fig. **8**. It is entirely programmed in JavaScript and executed by the *node webkit* framework (*node.js JS* VM for computation and IO combined with Chromium WEB browser for GUI implementation), and the simulation model consisting of the MAS program code and the physical *CANNON* model is specified in JavaScript and *JSON*, too. The computational and the physical simulation are coupled tightly, i.e., agents can access the physical model and simulation and vice versa. Agents are processed with a *JAM* instance. This architecture enables the integration of the simulator in real world environments (hardware-in-the-loop) deployed with *JAM* networks. Furthermore, the direct *JAM* processing provides advanced and realistic computational and communication analysis.

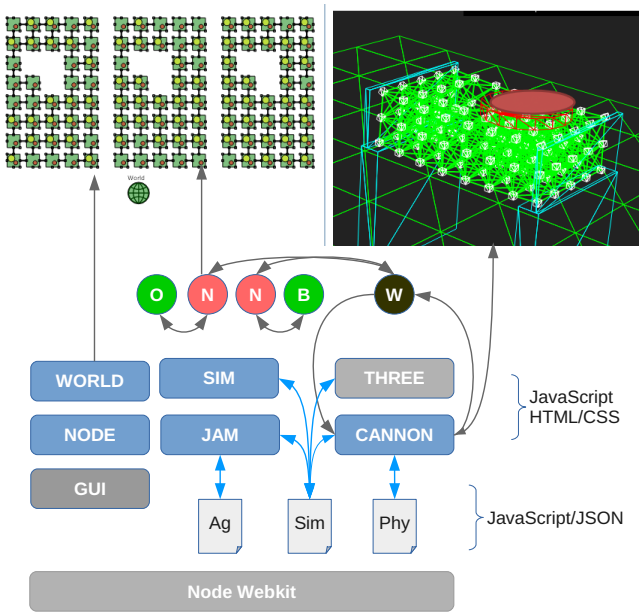

Figure 8. (Bottom) The multi-domain simulation environment coupling physical and computational systems (Top, left) Computational network using JAM for computation and communication (Top, right) Physical body-spring network using CANNON for solving the physical equations

10 Use-case and Evaluation

The multi-domain simulation introduced in the previous section is used to demonstrate and evaluate the proposed optimization algorithms, MAS implementation providing self-organizing and self-adaptivity features, and the *JAM* platform with an advanced use-case. The device under test used in the simulation is a plate consisting of 8x5x3 nodes. Each node is a mass in the MBP model connected up to nine neighbours via springs and a computer in the MAS model connected to up to six neighbour nodes via communication links. The start condition of the physical plate is shown in Fig **8**. Spring and gravity forces have an effect on each mass element of the plate. Therefore, the plate will swing be-

tween a maximal and a minimal bend until it reaches a static state. The MAS will respond in a given time to dynamic changes. The global goal may not be effected by this dynamic response, e.g., the MAS behaviour must not oscillate.

The event-based MAS will process sensor input data differently (spatially and temporal) unlike the numerical algorithms presented in Tab. **1**. The results from the simulation uses as a Device under Test (DUT) a plate basic shape with a large central hole and an additional load on the top of the DUT, which is shown Fig. **9**. The diagrams in Fig. **9** compare the outcome of physical and computational properties of the DUT with different optimization algorithms. All optimization results are compared with the same DUT without performing optimization (no actuation, only perception and sensor processing).

The physical properties give the accumulated total strain energy *U* of the structure (global value), the maximum node strain energy *u* (local value) from all nodes (including boundaries), the maximum node energy of only the core area of the structure, maximum of forces between nodes (taken from the entire optimization run), and the accumulated stiffness of all springs in the structure as a measure of energy required for actuators (in arb. units).

The communication and computational properties return the accumulated communication load (in Bytes), the accumulated number of remote signals (messages), and the total accumulated active computation time of all nodes (in arb. units) as a measure of the electrical energy required for the entire optimization run.

The hole in the DUT introduces interruptions of the communication network and affects the migration of agents or the delivery of signals, but not the global emergence behaviour satisfying the optimization goal of the MAS, shown in the optimization progress of the segment algorithm still achieving nearly the same results as the global algorithm.

The results show the suitability of the event-based MAS approach and the distributed algorithms versa the global centralized algorithm. The achieved strain energy reduction is about 40%. The segmented approach using a floating segment window around each node's center position delivers comparable results without the need for a central instance. However, there is a significant difference between the stepwise and linear correction function. Only the linear stiffness adaptation between the (technical possible) limits achieves a high total (global) strain energy and maximal node strain energy reduction. The step-wise correction (leading basically to a discrete stiffness distribution) achieves only a 10% reduction. Among the total strain energy the total sum of stiffness settings of all nodes is important. Each stiffness increase (of an actuated spring) requires energy, whereas a decrease can release energy (that could be harvested and stored by the system). The linear correction leads to an increase of the stiffness sum about of 40%, whereas the stepwise (discrete distribution) approach decreases the sum about of 35%. This overall stiffness reduction can be considered as a structural relaxation, and is a base for a high dynamic adaptation capability of the structure as there is a better utilization of the actuator ranges. The neighbour negotiation algorithms underperforms compared with the new segment approach but requires the lowest communication and computational resources (about 50% less than the segmented approach) As expected, the global approach creates the highest communication and computational costs.

It should be noted that the results are sensitive to the MAS parameter settings like sensor event and notification thresholds. For example, the communication and computational costs depend on the notification behaviour inhibition delay (update interval setting). The lowest accumulated computation time of the ICT network is required by the neighbour negotiation algorithm due to the lowest sensor event rates and actuator modifications. The three algorithms differ in convergence and stability (oscillation of the structure due to over- and under-optimization) and therefore have a significant influence on the ICT activity and the energy consumption.

The evaluation of a structure configuration with a large hole inside of the structure, interrupting communication and actuation, demonstrates the capability of the used MAS behaviour and optimization algorithm to compensate missing nodes. The proposed approach shows a low sensitivity to damages (both concerning the ICT and the structure optimization).

11 Dynamic Topology Optimization and Lightweight Structures

Topology optimization in its classical form is a technique to improve material utilization: Within a given design space, and for a given load case, blocks of material are removed in areas with limited internal loads. A review of various approaches has recently been provided by Sigmund and Maute **[34]**. The underlying assumption is that these material fractions contribute least to structural stiffness: It may be added here that topology optimization is generally executed in the linear-elastic regime. Since material is only removed, but not relocated, stiffness must effectively be reduced in the process.

However, this necessary loss in stiffness falls short of the reduction in weight, leading to an increase in specific or weight- related stiffness. A designer can thus in principle exploit the result of an optimization run to generate a final structure which may - if permissible - transgress the boundaries of the original part and offer the same stiffness at reduced weight. The weight reductions achievable via this principle are limited, but significant. For example, by rule of thumb, if the original problem is a solid beam, supported at both ends and loaded in bending, a weight reduction of roughly 30% can be achieved at a marginal loss of stiffness. For other, more technical and thus more complex components and associated load cases, weight reductions of approximately 20% have been demonstrated. Naturally, the success of the method in optimizing a given structure depends on the level of sophistication of the original design. In many cases, however, an exact quantification is impossible because the topology-optimized design cannot be matched against a conventional one, as the method is typically applied early in the design phase and using the complete design space as starting point.

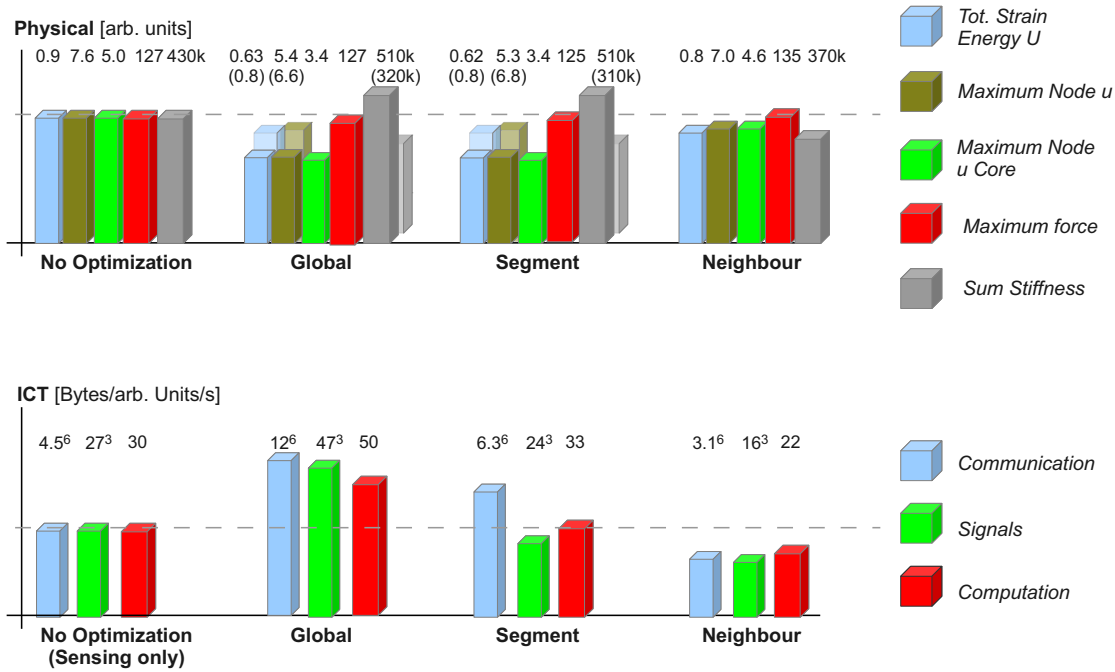

Figure 9. Evaluation of simulation results (DUT: plate with hole and external load) using different optimization algorithms (Top) Outcome of physical properties of the DUT: Total strain energy U and node strain energy u, forces, and stiffness [arbitrary units] (Bottom) Outcome of ICT properties: Total communication [Bytes], signals [arbitrary units], and computation [time, s]

In practice, manufacturing process constraints and limitations have sometimes hampered the implementation of topology-optimized designs. Processes like milling require direct access to the surfaces that are to be machined. Typically, optimum designs thus undergo a reworking based on manufacturability. In contrast, the geometrical flexibility of additive manufacturing has the potential of developing into a high road for pushing topology- optimized design from the virtual to the real world on a larger scale. A recent publication by Zhu et al. has collected several examples that illustrate the application of topology optimization to aerospace structures **[36]**.

Valdez et al. have recently published and overview of benchmark solutions for typical, recurring 2D problems. Their study does not provide clear answers on achievable weight savings, since their starting point is a standard rectangular design space, but conveys a good impression of the type of solution the method generates, and their dependence on - sometimes slight - changes in the boundary conditions **[35]**. Multi-phase topology optimization as developed by Burblies deviates from the above in considering multiple materials having individual properties - specifically, their elastic properties (Young's modulus etc.) are assumed to differ. Depending on the load case, the optimization approach relocates the originally arbitrarily distributed material elements within the set build envelope/space to achieve maximum stiffness. This is done by means of an "element exchange" technique which minimizes total strain energy in the system. The result is a structure which matches the original one in weight (material is not removed, but only redistributed), but shows superior stiffness. The high stiffness solution can then be transformed into a design which takes over the suggested structuring, but reduces the amount of material used and thus the structure's weight while still meeting stiffness requirements **[13]**. The central drawback of the methods as described above is that optimization is typically done for a single load case. As a consequence, the resulting structure may underperform compared to a non-optimized one if the loading situation changes. Besides, the optimized structure may turn out to behave more critical under unexpected load cases: For example, topology-optimized designs typically tend to be more prone to failure modes like buckling.

To avoid these issue, strategies like average design can be adopted which generate a common solution out of the optimization results obtained for several load cases. However, this approach will naturally limit the weight saving potential, since the final solution moves away from the optimum for each single load case considered. This situation is the realm which must seem most promising for the suggested dynamic optimization approach: Instead of having a structure that can cope reasonably well with different load cases, we propose a structure that can dynamically adapt to the load cases it recognises and will thus adopt the optimum configuration for each of them.

Needless to say, this type of structure will also be able to respond to load cases not explicitly considered in the component design phase. This capability may allow for reduc-

tion of safety margins and thus facilitate additional weight savings.

In summary, we consider an average weight saving potential accessible through the suggested adaptive materials of approximately 20% a conservative estimate. Beyond suitable algorithms as discussed in this paper, materials and/or structural solutions that allow implementation of these will be a major issue to be solved in future research.

12 Conclusion and Outlook

This paper showed the suitability of Multi-Agent Systems for reliable distributed computing in large-scale low-resource computer networks by evaluating the use case with a reference architecture of a self-adaptive robotic material composed of nodes connected by actuated spring elements. Each node is a micro-controller providing sensor processing, power electronics, communication (wired or wireless), an agent processing platform, and data storage. Such robotic materials are characterized by a tight coupling of computation, communication, sensing, and actuation. The distributed computing and communication in material-integrated large-scale computer networks can be considered as micro-scale cluster computing.

The goal of the MAS is to optimize the material or structure under varying load situations and damages by collecting observation variables (i.e., strain sensor data) and modifying target variables (actuators, i.e., stiffness of spring parameters) without any central instance (self-organized and event-based). The emergence behaviour of the MAS was robust by adaptivity in the presence of large defects.

Different optimization algorithms were introduced and evaluated using the *JavaScript* Agent Machine platform. A global optimization approach based on Multi-Phase Topology Optimization was used as a reference algorithm for comparison with two distributed segment and neighbour algorithms enabling dynamic topology optimization at run-time. The segment algorithm achieved a 40% reduction of the total mechanical strain energy of the structure under test (with a damage and external load applied), which is a relevant measure of stability by self-adaptation and self-healing behaviour.

Although the *JAM* platform can be used on low-resource embedded computers, another very-low resource platform *AFVM* was introduced that is suitable for material-integrated computing. A just-in-time cross-compiler approach can be used to enable seamless migration of mobile agents between both agent processing platforms, which is mandatory for connecting Robotic Materials to the IoT and Clouds.

Different agent communication approaches were discussed and evaluated under real conditions (sensor processing in the Robotic Material) showing efficient and scalable behaviour. Mobile agents with a unified agent programming and processing platform are the key technology for the creation of smart IT materials, e.g., tangible user interfaces **[32]**, product life-cycle management **[33]**, integration in the Internet of Things environment and providing Cloud Services for and with Robotic Materials in everyday customer devices.

Future work has to investigate the implementation of the proposed MAS using the *AFVM* platform deployed in real robotic structures. Although the *AFVM* bases on the same *AAPL* agent metal model, is forces much higher limitations and constraints with respect to agent code and data complexity, and overall storage capacity. In this work the simulation was performed with ideal actuators and sensors described via linear input-output models. Technical actuators, e.g., thermoplastic materials actuated by temperature variations, show highly non-linear behaviour. The proposed MAS and optimization algorithms have to be evaluated towards such unreliable and non-linear actuators and sensors. Finally, the weight saving potential and energy balances have to be investigated rigorously.

One major field of application and future road map for the proposed MAS and ICT architecture is the Dynamic Topology Optimization (DTO) in complex mechanical structures and components introduced conceptually in Section 11. A proof-of-concept has to demonstrate the weight saving by DTO by the MAS at run-time. Furthermore, transferring this future work to actual physical materials and structure heavily depends on advances in manufacturing technologies, sensors, and hybrid material design.

REFERENCES

[1] T. L. M. Choi, Y. Sui, I. H. Lee, R. Meredith, Y. Ma, G. Kim, D. Blaauw, Y. B. Gianchandani, *Autonomous Microsystems for Downhole Applications: Design Challenges, Current State, and Initial Test Results*, Sensors (Basel, Switzerland), vol. 17, no. 2190, 2017.

[2] V. Di Lecce, M. Calabrese, and C. Martines, *From Sensors to Applications: A Proposal to Fill the Gap*, Sensors & Transducers, vol. 18, no. Special Isse, pp. 5–13, 2013.

[3] S. Bosse, D. Lehmhus, W. Lang, M. Busse (Ed.), *Material-Integrated Intelligent Systems: Technology and Applications*, Wiley, ISBN: 978-3-527-33606-7 (2018)

[4] M A McEvoy, Nikolaus Correll, *Materials science. Materials that couple sensing, actuation, computation, and communication*, Science, 347(6228):1261689 (2015)

[5] M. A. McEvoy and N. Correll, *Materials science.- Materials that couple sensing, actuation, computation, and communication*, Science, vol. 347, no. 6228, 2015.

[6] M. A. McEvoy, N. Correll,.*Thermoplastic variable stiffness composites with embedded, networked sensing, actuation and control.Journal of Composite Materials* 49(2014) DOI: 10.1177/0021998314525982.

[7] L. Zhao, J. Ma, T. Wang, and D. Xing, *Lightweight Design of Mechanical Structures based on Structural Bionic Methodology*, Journal of Bionic Engineering, vol. 7, pp. 224-231, 2010.

[8] C. Hamm, S. Möller, *ELiSE—An Integrated, Holistic Bionic Approach to Develop Optimized Lightweight Solutions for Engineering, Architecture and Design*, in C. Hamm, Ed., Evolution of Lightweight Structures. Springer, 2015.

[9] C. Hamm, *ELiSE: Bionic Lightweight Design*, project flyer, Alfred-Wegener-Institut Helmholtz-Zentrum für Polar-

und Meeresforschung, Bremerhaven, Germany, 2013.

[10] C. Jog, R. Haber, M. Bendsøe, *Topology design with optimized, self-adaptive materials*, International Journal for Numerical Methods in Engineering, vol. 37, issue 8 (1994) pp. 1323-1350

[11] M. A. McEvoy, N. Correll, *Distributed Inverse Kinematics for Shape-Changing Robotic Materials*. Procedia Technology 26 (2016) 4 - 11.

[12] J. J. Joo1, B. Sanders, G. Washington, *Energy based efficiency of adaptive structure systems*, Published 9 January 2006, IOP Publishing Ltd, Smart Materials and Structures, Volume 15, Number 1

[13] A. Burblies and M. Busse, *Computer Based Porosity Design by Multi Phase Topology Optimization*, Multiscale & Functionally Graded Materials Conference (FGM2006), Honolulu, October 15th -18th 2006

[14] M. Caridi and A. Sianesi, *Multi-agent systems in production planning and control: An application to the scheduling of mixed-model assembly lines*, Int. J. Production Economics, vol. 68, pp. 29–42, 2000.

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

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

[17] 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.

[18] S. Bosse, *Unified Distributed Computing and Co-ordination in Pervasive/Ubiquitous Networks with Mobile Multi-Agent Systems using a Modular and Portable Agent Code Processing Platform*, in The 6th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2015), Procedia Computer Science, 2015

[19] R. Milner, *The space and motion of communicating agents*. Cambridge University Press, 2009.

[20] S. Bosse, D. Lehnhus, *Towards Large-scale Material-integrated Computing: Self-Adaptive Materials and Agents*, doi: 10.1109/FAS-W.2017.123 (2017)

[21] H. Janocha, Ed., *Adaptronics and Smart Structures*, 2nd ed. Springer (2007).

[22] S. Poslad, Ubiquitous Computing: Smart Devices, Environments and Interactions. 2009, Wiley

[23] F. Haneef and S.Angalaeswari, *Self-Healing Framework for Distribution Systems*, International Journal of Scientific & Engineering Research, vol. 4, no. 7, 2013.

[24] E. Pournaras, I. Moise, and D. Helbing, *Privacy-preserving Ubiquitous Social Mining via Modular and Compositional Virtual Sensors*, in IEEE 29th International Conference on Advanced Information Networking and Applications, 2015.

[25] S. Bosse, E. Pournaras, *An Ubiquitous Multi-Agent Mobile Platform for Distributed Crowd Sensing and Social Mining*, FiCloud 2017: The 5th International Conference on Future Internet of Things and Cloud, Aug 21, 2017 - Aug 23, 2017, Prague, Czech Republic

[26] S. Bosse, *Design of Material-integrated Distributed Data Processing Platforms with Mobile Multi-Agent Systems in Heterogeneous Networks,* Proc. of the 6'th International Conference on Agents and Artificial Intelligence ICAART 2014, 2014, DOI:10.5220/0004817500690080.

[27] S. Bosse, *Incremental Distributed Learning with JavaScript Agents for Earthquake and Disaster Monitoring*, International Journal of Distributed Systems and Technologies (IJDST), (2017)

[28] S. Bosse, *Distributed Machine Learning with Self-organizing Mobile Agents for Earthquake Monitorin*g, in 2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W) , SASO Conference, DSS, 12 September 2016, Augsburg, Germany, 2016.

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

[30] Z. Qin, J. Xing, and J. Zhang, *A Replication-Based Distribution Approach for Tuple Space-Based Collaboration of Heterogeneous Agents*, Research Journal of Information Technology, vol. 2, no. 4. pp. 201–214, 2010

[31] D. Lehmhus, S. Bosse, A. Gemilang, *A Multi-Agent System based approach for Adaptive Property Control in Smart Load-Bearning Structures*, European Congress and Exhibition on Advanced Materials and Processes, EUROMAT (2017), Symposium E6, Modeling, Simulation and Optimization, 17-22 September, 2017, Thessaloniki, Greek

[32] J. Kang, *Technique of Tangible User Interfaces for Smartphone*, 12 International Conference on Information and Computer Applications (ICICA 2012) IPCSIT vol. 24 (2012)

[33] D. Lehmhus, T. Wuest, S. Wellsandt, S. Bosse, T. Kaihara, K.-D. Thoben, and M. Busse, *Cloud-Based Automated Design and Additive Manufacturing: A Usage Data-Enabled Paradigm Shift*, Sensors MDPI, vol. 15, no. 12, pp. 32079–32122, 2015, DOI 10.3390/s151229905.

[34] O. Sigmund, K. Maute, *Topology optimization approaches - a comparative review, Structural and Multidisciplinary Optimization*, vol. 48 (2013) pp. 1031–1055, DOI: 10.1007/s00158-013-0978-6

[35] S. Ivvan Valdez, S. Botello, M. A. Ochoa, J. L. Marroquin, V. Cardoso, *Topology Optimization Benchmarks in 2D: Results for Minimum Compliance and Minimum Volume in Planar Stress Problems, Archives of Computational Methods in Engineering*, vol. 24 (2017) pp. 803–839, DOI: 10.1007/s11831-016-9190-3.

[36] J.-H. Zhu, W.-H. Zhang, L. Xia, *Topology Optimization in Aircraft and Aerospace Structures Design, Archives of Computational Methods in Engineering*, vol. 23 (2016) pp. 595–622, DOI: 10.1007/s11831-015-9151-2.

[37] M. Choi, Y. Sui, I. H. Lee, R. Meredith, Y. Ma, G. Kim, D. Blaauw, Y. B. Gianchandani, T. Li, *Autonomous Microsystems for Downhole Applications: Design Challenges, Current State, and Initial Test Result*s, doi:10.3390/s17102190 (2017)