From the Internet-of-Things to Sensor Clouds - Unified Distributed Computing in Heterogeneous Environments with Smart and Mobile Multi-Agent Systems

Stefan Bosse, University of Bremen, Dept. of Mathematics & Computer Science, 28359 Bremen, Germany

Abstract: A novel and unified design approach for reliable distributed and parallel data processing in large scale networks consisting of high- and of low-resource nodes using mobile agents is introduced. This approach enables the development of sensor clouds of the future integrated in daily use computing environments and the Internet. Agents can migrate between different hardware and software platforms by migrating the program code of the agent, embedding the state and the data of an agent, too. Agent mobility crossing different execution platforms, agent interaction by using tuple-space databases, and agent code reconfiguration enable the design of reliable distributed sensor processing networks.

1. Introduction

Trends recently emerging in engineering and micro-system applications such as the development of sensorial materials **[1][2]** show a growing demand for distributed autonomous sensor networks of miniaturized low-power smart sensors embedded in technical structures. Multi-agent systems (MAS) can be used for a decentralized and self-organizing approach of data processing in a distributed system like a resource-constrained sensor network (discussed in **[4]** and **[5]**), enabling smart and adaptive distributed information extraction, for example, based on pattern recognition (e.g., referring **[6]** and **[7]**), by decomposing complex tasks in simpler cooperative agents. It can be shown that MAS-based data processing approaches are scalable from generic computer to single microchip level platforms which can aid the material-integration of Structure Monitoring applications. Currently there are only few proposed agent processing platforms which can be scaled to microchip level.

In **[4]** the agent-based architecture considers sensors as devices used by an upper layer of controller agents. Agents are organized according to roles related to the different aspects to integrate, mainly sensor management, communication and data processing. This organization isolates largely and decouples the data management from changing networks, while encouraging reuse of solutions.

The deployment of agents can overcome interface barriers and closes the gap arising between platforms and environments differing considerably in computational and communication capabilities, enabling the integration of sensor networks in large-scale world-wide-web (WWW) applications and providing Internet connectivity, shown in Fig. **1**. This is addressed by using a unified agent-based programming and interaction model, independent of the underlying processing platform. For the proposed advanced agent processing platform architecture there exist suitable hardware, software, and simulation model implementations, which can be interconnected in networks. They are compatible on the operational and execution level, thus, agents can migrate between these different implementation platforms.

Agent mobility crossing different execution platforms and agent interaction by using tuple-space databases and global signal propagation aid solving data distribution and synchronization issues in the design of distributed sensor networks.

Usually sensor processing and information computation require known world models including mechanical models, for example, in load monitoring use cases of technical structures. Self-organizing MAS are useful in unreliable and partially unknown environments, which can overcome world environment and mechanical model limitations successfully.

Adaptation of the agent behaviour, i.e., based on learning, offers a reliable reaction mechanism in the presence of environmental changes, e.g., changes in network connectivity or node failures. Mobility - the ability to migrate a agent processing unit to a different execution platform and node - and autonomy together with a high degree of independency from the processing platform ensure robust data processing in large-scale networks.

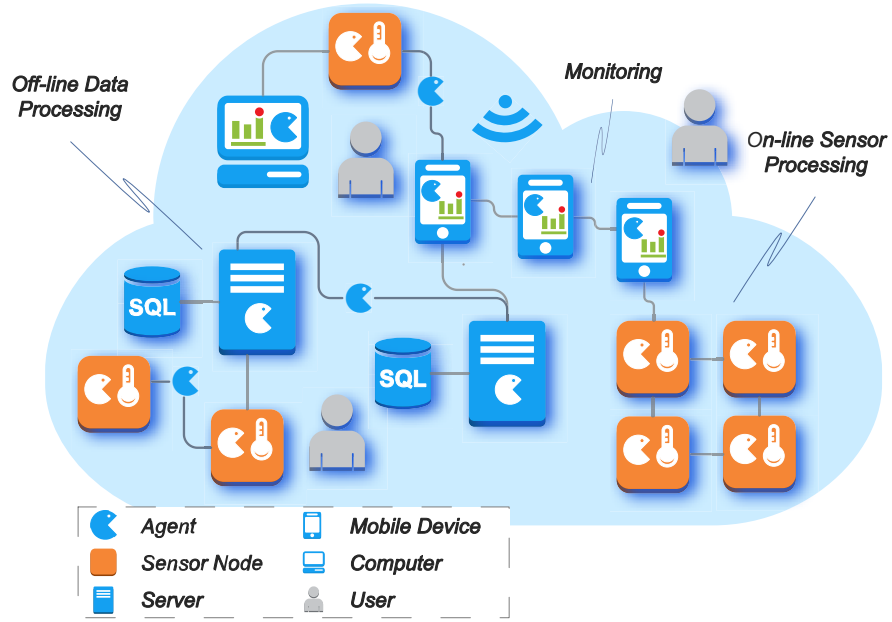

Figure 1. Deployment of Agents in Sensor Clouds and Internet Applications

Traditional LM/SHM/TS algorithms like inverse numerical approaches, supervised machine learning, correlation analysis, and pattern recognition, are characterized by a high computational complexity and high memory requirements. Usually these high-level computations are performed off-line (outside the network and not in real-time). It can be shown that agent-based computing can be used to partition these computations in off-line and on-line (in network and real-time) parts resulting in an increased overall system efficiency (performance and energy demands) and a unified programming interface between off- and on-line parts. The agent model is also capable and used to provide a programming model for distributed heterogeneous systems crossing different network domains. Multi-agent systems are used to enable a paradigm shift from traditional continuos data-stream based to event-driven sensor data processing, resulting in increased robustness, performance, and efficiency. Event-based sensor data processing and self-organizing systems reduces the communication and processing complexity significantly without a loss of Quality-of-Service (QoS).

Autonomy-orientated computation, respectively self-organizing MAS with directed diffusion, replication, exploration, and voting behaviour are used to implement model

free sensor-to-information mapping, suitable for information extraction in sensor networks and LM/SHM/TS applications based on pattern recognition and data-centric algorithms. Smart learning agents based on decision trees are used to distribute and deliver information in unreliable and changing sensor networks. The run-time behaviour and the requirements of computational, communication, and energy resources of different MAS are analyzed using simulation and real-time monitoring techniques in a technical demonstrator.

A case study using self-organizing MAS and event-based sensor data distribution demonstrated the suitability and impressive efficiency of MAS and the advanced agent processing platform for a Structural Health Monitoring use case, which connects low-resource large-scale sensor networks, consisting of hundreds of nodes, with distributed computers solving inverse numeric computations (details in **[3]**).

2. Mobile Multi-Agent Systems

The deployment and programming of MAS means programming of distributed systems. The programming of distributed system is a combination of computation and coordination. In this work, the agent behaviour, perception, reasoning, and the action on the environment are encapsulated in agent classes, with activities representing the control state of the agent reasoning engine, and conditional transitions connecting and enabling activities. Activities provide a procedural agent processing by a sequential execution of imperative data processing and control statements. Agents can be instantiated from a specific class at run-time. Activities are connected by transitions, the edges of an *Activity-Transition Graph* (ATG), shown in Fig. **2**.

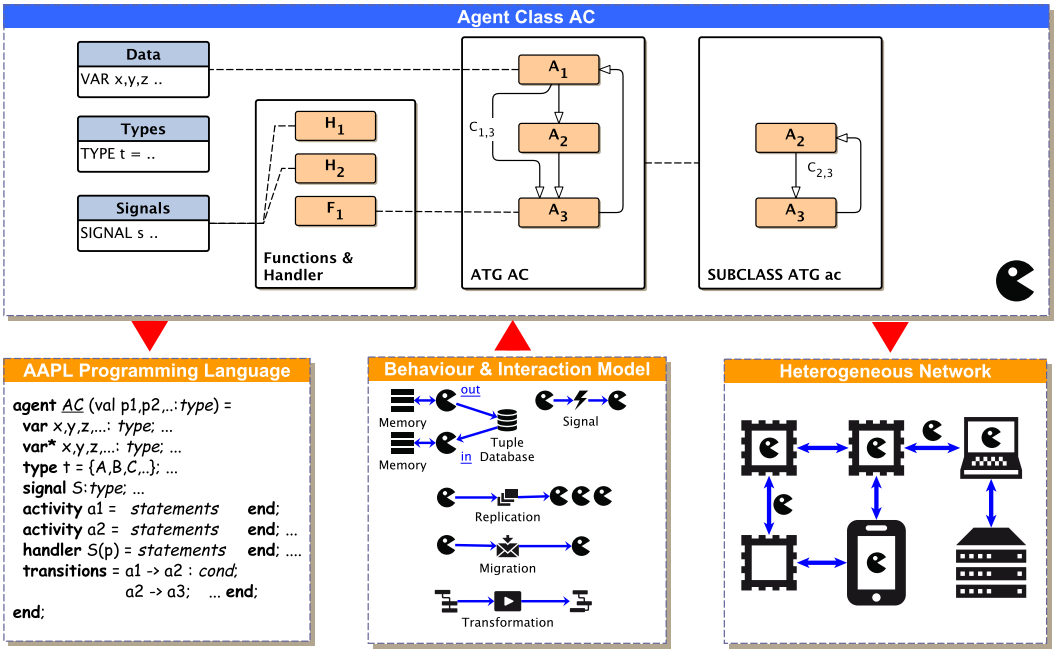

Figure 2. Agent behaviour programming level with activities and transitions (AAPL, left, details in **[1][2]**); agent class model and activity-transition graphs (top); agent instantiation, processing, and agent interaction on the network node level (middle), agent deployment in heterogeneous network (right)

There is a Multi-Agent system (MAS) deployed in heterogeneous networks consisting of a set of individual agents $\{a_1,a_2,..\}$. There is a set of different agent behaviour, called classes $\mathcal{C}=\{AC_1, AC_2,..\}$. An agent belongs to one class. In a specific situation an agent a_i is bound to and processed on a network node $N_{m,n,o,..}$ (e.g. a microchip, a computer, or a virtual simulation node) at a unique spatial location given by the vector $(m,n,o,..)$. There is a set of different nodes $\mathcal{N}=\{N_1, N_2,..\}$ arranged in a mesh-like network with peer-to-peer neighbour connectivity (e.g., a two-dimensional grid). Each node is capable of processing a number of agents $n_i(AC_i)$ belonging to one agent behaviour class AC_i, and supporting at least a subset of $\mathcal{C}' \subseteq \mathcal{C}$. An agent (or at least its state) can migrate to a neighbour node where it continues working. Each agent class is specified by the tuple $AC = \langle A,T,V,F,S,H\rangle$. A is the set of activities (graph nodes), T is a set of transitions connecting activities (relations, graph edges), F is a set of computational functions, S is a set of signals, H is a set of signal handlers, and V is a set of body variables used by agents of the agent class.

This agent model is generic and not limited to traditional program orientated systems. For instance, a node of a network can be represented by a stationary agent, too. This node agent offers node services used by other (mobile) agents, usually part of an operating system.

An ATG describes the complete agent behaviour. Any sub-graph and part of the ATG can be assigned to a sub-class behaviour of an agent. Therefore modifying the set of activities A and transitions T of the original ATG introduces several sub-behaviours implementing algorithms to satisfy a diversity of different goals. The reconfiguration of activities $A'=\{A_1 \subseteq A, A2 \subseteq A, ..\}$ from the original set A and the modification or reconfiguration of transitions $T'=\{T_1,T_2,..\}$ enables *dynamic ATG*s and agent sub-classing at run-time (DATG).

There is an ATG-based Agent Programming Language *AAPL* which can be synthesized to different agent implementation and processing models (details in **[1][2]**). The activity-graph based agent model is attractive due to the proximity to the finite-state machine model, which simplifies the hardware implementation. An activity is started by a transition depending on the evaluation of (private) agent data (conditional transition) related to a part of the agents belief in terms of the Belief-Desire-Intention (BDI) architecture, or started by unconditional transitions (providing sequential composition). The Agent Behaviour model is specified in Def. **1**.

Def. 1. Multi-Agent Behaviour Model and Multi-Agent Processing

---

Communication and Interaction of Agents. Agents can interact with each other by exchanging data using a tuple-space (TS) database as a shared object supporting synchronized and atomic read, test, remove, and write operations. Agents can communicate and synchronize peer-to-peer by using signals, which can be delivered to remote execution nodes, too.

A tuple space is basically a shared memory database used for synchronized data exchange among a collection of individual agents, which is a well known and suitable MAS interaction paradigm. The scope and visibility of a tuple space database can be unlimited and distributed in the whole network, or limited to a local scope, e.g., the network node level. A tuple space provides abstraction from the underlying platform architecture, and offers a high degree of platform independence, vital in a heterogeneous network environment.

A tuple database stores a set of n-ary data tuples, $tp_n=(v_1, v_2,.., v_n)$, a n-dimensional value tuple. The tuple space is organized and partitioned in sets of n-ary tuple sets $\nabla=\{TS_1,TS_2,..,TS_n\}$. A tuple is identified by its dimension and the data type signature. Commonly the first data element of a tuple is treated as a key. Agents can add new tuples (output operation) and read or remove tuples (input oper-

ations) based on tuple pattern and pattern matching, $pat_n=(v_1, x_2?, .., v_j,.., x_j?,., v_n)$, a n-dimensional tuple with actual and formal parameters. Formal parameters are wildcard placeholders, which are replaced with values from a matching tuple. The input operations can suspend the agent processing if there is actually no matching tuple is available. After a matching tuple was stored, blocked agents are resumed and can continue processing. Tuple databases provide inter-agent synchronization, too. This tuple-space approach can be used to build distributed data structures and the atomicity of tuple operations provides data structure locking. The tuple spaces represent the knowledge of agents.

Agent Processing and Agent Processing Platforms (APP). Agents are computational entities with a high degree of autonomy and independence from the underlying processing platform. In this work, the agents are implemented with Agent FORTH program code that is executed on virtual stack machines, which can be implemented alternatively on hardware (System-on-Chip), simulation, and software level, which can be embedded in microcontroller, desktop applications, web applications, or server programs. The agent program code is a self-containing and self-initializing unit embedding the (private) agent data and the current control state of the agent, which simplifies migration significantly. The program is able to modify itself by using code morphing. This approach leads to a low computational dependency from the current execution environment, which is vital to strong heterogeneous environments. There is only a small set of knowledge about the program which is required by the VM to execute the agent program, and vice versa.

3. The Big Thing: Domains, Networks, and Mobile Agent Processing

The previous section introduced a unified agent behaviour and programming model offering computation, instantiation of agents, mobility, and multi-agent interaction, and which can be implemented on a diversity of processing platforms. One major goal of the deployment of MAS is overcoming heterogeneous platform and network barriers arising in large scale hierarchical and nested network structures, consisting and connecting, e.g., the Internet, sensor networks, body networks, production and manufacturing Cyber-Physical System (CPS) networks, shown in Fig. **3** on the left. The large diversity of execution platforms, network topologies, services provided by network nodes, and the programming environments require a unified and abstract behavioural and structural representation model. The Bigraphical model proposed by Robin Milner models the entire "computing" environment with place and link graphs, composing finally bigraphs **[8]**, shown on the right of Fig. **3**. They include agents, and they are offering a unified model and platform for ubiquitous systems and the foundation for an Ubiquitous Abstract Machine, and supporting reconfigurable spaces (dynamic topologies). Bigraphs virtualize communicating processes (agents) and information objects (tuple-spaces), and they originate in process calculi for concurrent systems, especially the pi-calculus **[9]** and the calculus of mobile ambients **[10]** for modeling spatial configurations of networks with a dynamic topology.

The environment consists of places where computation occurs, e.g., computers, agents, rooms, buildings, machines, and so on. The links are abstract, providing the possibility of interaction between different places, i.e., transferring of agents and their mobile processes. Agents are treated as active computational units. Places introduce spatial and logical bindings. Bigraphs allow the nesting of nodes and places, natural for many real-world computing environments, and they can be applied for wide reactive systems. All nodes have a fixed number of ports, providing an endpoint for links. Agents have two ports: a processing port link and an interaction (communication) link. Bigraphs, which represents the system state, can be modified by the application of reaction rules, which changes the linking and place relations. Bigraphs can be composed of other bigraphs matching inner and outer interfaces.

A link is a hyperedge connection which connects nodes, outer, and inner names, where names are open linkings that support additional connectivity, i.e., used for the dynamic composition of bigraphs at "run-time". Connectivity not only provides the platform for agent migration between different places, it provides information exchange, which is provided here by place-bounded tuple-spaces and signals. Migration of mobile processes is just another form of interaction with and the modification of the environment.

To adapt this Bigraphical Reactive System (BRS) model to MAS it is necessary to distinguish subjects (entities which can perform actions, the agents) and objects (here data, tuples, tuple-spaces, signals, and processing platforms themselves).

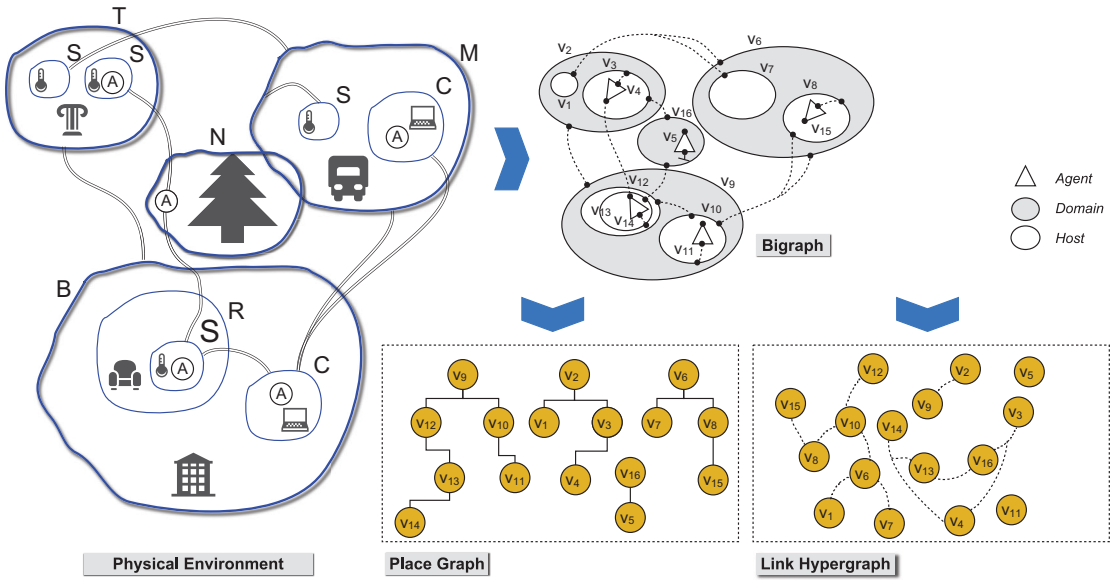

Figure 3. From physical maps (left) to unified logical maps: link (right, bottom) and structure place (middle, bottom) graphs composing bigraphs (right, top) [S: Sensor, T: Technical Structure, M: Mobile Device, N: Net. Router, B: Building, R: Room, C: Computer, A: Agent]

4. Robust Event-based Sensor Data Processing

Assume a distributed sensor network, e.g., equipped with strain-gauge sensors used for load monitoring of technical structures. Traditional sensor data acquisition read every sensor periodically, independent of a change of the sensor value, which leads to high communication load and worse scaling in large networks. Instead, a different sensor data processing and distribution approach is used and implemented with autonomous agents, leading to a significant decrease of network processing and communication activity and a significant increase of reliability and the Quality-of-Service:

1. An event-based sensor distribution behaviour is used to deliver sensor information from source sensor to computation nodes based on sensor region changes.
2. Adaptive path finding (routing) supports agent migration in unreliable networks with missing links or nodes by using a hybrid approach of random and attractive walk behaviour.

3. Self-organizing agent systems with exploration, distribution, replication, and interval voting behaviour based on feature marking are used to identify a region of interest (ROI, a collection of stimulated sensors) and to distinguish sensor failures (noise) from correlated sensor activity within this ROI.

It is assumed that sensor nodes arranged in a two-dimensional grid network (as shown in Fig. **4**) providing spatially resolved and distributed sensing information of the surrounding technical structure. The computational nodes arranged at the outside of the network are further divided in pre-computation and the final computation nodes (the four nodes located at the corners of the network). The pre-computational nodes can be embedded PCs or single micro-chips, and the computational nodes can be workstations or servers physically displaced from the material-embedded sensor network. Only the inner sensor nodes are micro-chip platforms embedded in the technical structure material, for example, using thinned silicon technologies.

The computation of the system response information requires basically the complete sensor signal matrix *S*. In this approach presented here the elements of the sensor matrix are only updated if a significant change of specific sensors occurred. Only the four computational nodes at the corners store the complete sensor matrix and perform the load computations (e.g., using inverse numeric or supervised machine learning). The sensor processing uses both stationary (non-mobile) and mobile agents carrying data, illustrated in Fig. **4** on the left side. There are two different stationary (non-mobile) agents operating on each sensor node: the sampling agent which collects sensor data, and the sensing agent, which pre-processes and interprets the acquired sensor data. If the sensing agent detects a relevant change in the sensor data, it sent out four mobile event agents, each in another direction. The event agent carries the sensor data and delivers it to the pre-computation nodes at the boundary of the sensor network. The Agent behaviour are specified in Def. **2**.

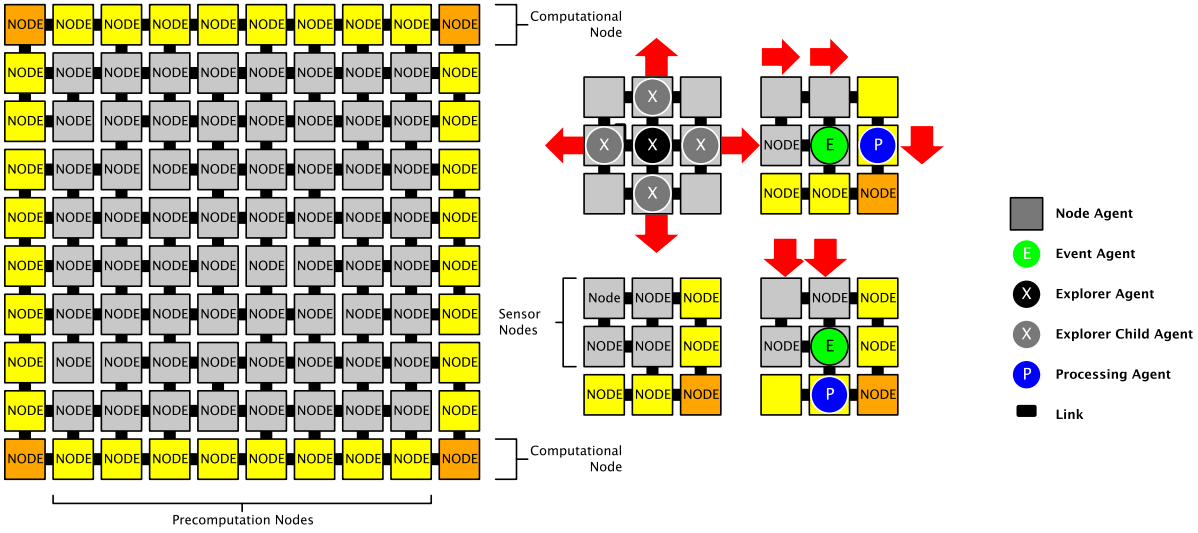

Figure 4. A sensor network deployed with explorer (X), event deliver (E), node (N), and computational processing agents (P). The sensor network can contain missing or broken links between neighbour nodes.

Def. 2. MAS Agent Behaviours for event-based sensor data processing

Event Agents. An event agent has a pre-defined path in the direction *dir* which is followed by the move activity as long as there is connectivity to the next neighbour node in this direction. Normally the agent travels to the outside of the network in the given direction by applying a normal routing strategy successfully (goal: minimizing the distance). If it is not possible to migrate in the pre-defined direction, an alternative path is chosen by using an opposite routing strategy, which chooses a path away from the original destination (random walk) to bypass not connected nodes and missing communication links. Using a relax routing strategy the agent is directed again to the original planned path. Making routing decisions and migration are performed in a move activity of the agent, followed by a check activity which collects sensor data from the current node and checks the destination node goal, and if reached delivering the sensor values in a deliver activity.

Each pre-computation node stores a row or a column of the sensor matrix *S*. If their data changes, the pre-computation nodes will send out two mobile distribution agents in opposite directions, delivering a row or column of *S* to the final computation nodes, located at the edges of the sensor network.

This approach offers robustness in the case of link or node failures by smart and autonomous path finding and redundancy.

Explorer Agents. The goal of the explorer agents is finding the outline of extended correlated regions (ROI) of increased sensor stimuli which can be distinguished from the neighbourhood. The output is used to trigger the event agents. Furthermore, faulty or noisy sensors which can disturb the further data processing algorithms significantly should not delivered to the computational nodes.

An initial root explorer agent is instantiated by the sensing agent with an initial direction argument *ORIGIN*. This explorer agent will read the local sensor values from the tuple database. The root agent will send out explorer child agents to all connected neighbour nodes. These child agents compute a partial term of the local stimuli H calculation by sending out additional explorer child agents until the boundary of the ROI is reached. To avoid multiple visiting of a node by different child agents of the same exploration group, a marking is set on each visited node (a tuple with a limited lifetime removed by a garbage collector). If there is already a marking, an explorer child agent will go back immediately to its parent agent node location and delivers the computed partial term h of H. An explorer or explorer child agent that sent out additional child agents will wait (sleep) until all child agents have returned their computation results or a time-out occurs. Data is exchanged between child and parent agents by using the tuple-space database and synchronization (wake-up) is handled by using signals.

5. Case Study: A Material-integrated Load Monitoring Network

An example use-case should demonstrate the deployment of the introduced unified agent model, agent interaction, and mobility of agent processes in an heterogeneous network environment and technical structures equipped with sensor networks. Initially unknown external forces acting on a mechanical structure lead to a deformation of the material based on the internal forces, shown in Fig. **5**.

A material-integrated active sensor network with strain-gauge sensors, electronics, data processing, and communication, together with mobile agents is used to monitor relevant sensor changes with the event-based information delivery behaviour, finally distributing and pre-computing the sensor data according to the agent behaviour introduced in Sec. **4.** The unknown system response for an externally applied load *L* is measured by the strain sensor stimuli response *S*, finally computing an approximation of the response *L*' using inverse numeric methods.

Inversion problems, in particular those with incomplete and noisy data, are usually extremely ill-conditioned, meaning that small errors in the signals or the model lead to huge errors in any solution gained by such a naïve approach. Therefore, inverse methods try to stabilize the inversion process using regularization techniques (Tik-

honov, CG), performed off-line (details can be found in **[3]**) based on prior FEM simulations of the structure under test finally calculating the inversion matrix, which is required for the inverse computation of the load matrix from the sensor data matrix.

The Agent processing Platform nodes introduced in Sec. **2.**, which are capable of executing Agent FORTH machine code programs, are implemented on SoC microchip, in software, and in a simulation environment using the SeSAm Agent-Simulator, all connected in a multi-domain network, shown in Fig. **5**. together with the unified Bigraph representation.

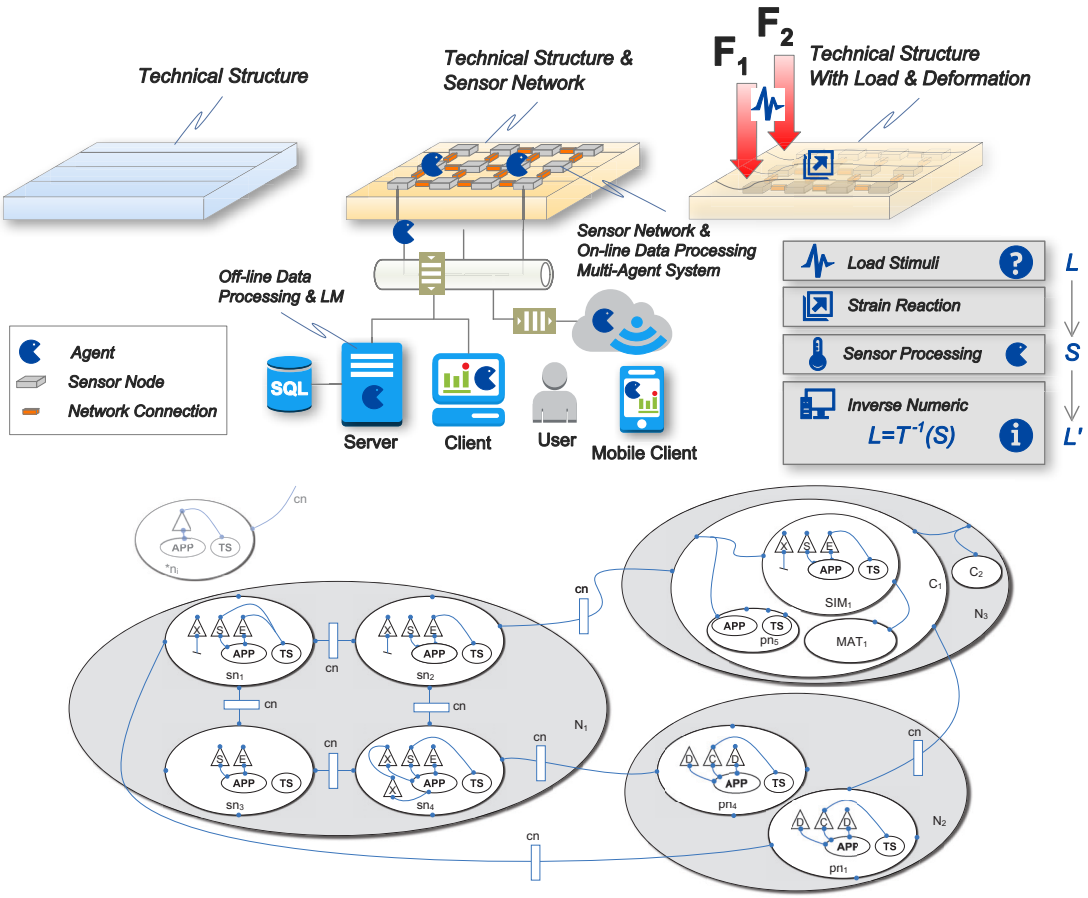

Figure 5. Top: A Sensorial Material with a material-embedded sensor network connected to a computational network, partitioning sensing and computation in on-line and off-line domains. Agents can migrate between different networks and hosts (sensor nodes, computers, servers, mobile devices). Bottom: Bigraph of the environment [sn/pn: Sensor/ Computational Node, cn: Communication Channel, N: Network, C: Computer, SIM: Agent Simulator, MAT: Matlab]

6. Conclusion

A novel and unified design approach using mobile agents for reliable distributed and parallel data processing in large scale networks consisting of high- and of low-resource nodes can enable the development of sensor clouds of the future integrated in daily use computing environments and the Internet. Agents can migrate between different hardware and software platforms (they are compatible on the execution level) by migrating the program code of the agent, embedding the state and the data of an

agent, too. The AAPL programming language and Bigraph models are the main tools for designing large scale and strong heterogeneous networks using MAS.

7. References

1. S. Bosse, *Distributed Agent-based Computing in Material-Embedded Sensor Network Systems with the Agent-on-Chip Architecture*, IEEE Sensors Journal, DOI 10.1109/JSEN.2014.2301938
2. S. Bosse, *Design of Material-integrated Distributed Data Processing Platforms with Mobile Multi-Agent Systems in Heterogeneous Networks*, ICAART 2014, DOI:10.5220/00048175006-90080
3. S. Bosse, A. Lechleiter, S*tructural Health and Load Monitoring with Material-embedded Sensor Networks and Self-organizing Multi-Agent Systems*, Procedia Technology, Elsevier, DOI: 10.1016/j.protcy.2014.09.039
4. M. Guijarro, R. Fuentes-fernández, G. Pajares, *A Multi-Agent System Architecture for Sensor Networks*, Multi-Agent Systems - Modeling, Control, Prog., Simulations and Applications, 2008.
5. A. Rogers, D. D. Corkill, N. R. Jennings, *Agent Technologies for Sensor Networks*, IEEE Intelligent Systems, vol. 24, no. 2, 2009.
6. X. Zhao, S. Yuan, Z. Yu, W. Ye, J. Cao, *Designing strategy for multi-agent system based large structural health monitoring*, Expert Systems with Applications, 2008, 34(2), 1154–1168. doi:10.1016/j.eswa.2006.12.022
7. J. Liu, *Autonomous Agents and Multi-Agent Systems*, World Scientific Publishing, 2001 (ISBN 981-02-4282-4)
8. R. Milner, *The space and motion of communicating agents*. Cambridge University Press, 2009.
9. R. Milner, *Communicating and mobile systems: the π-calculus*, Cambridge University Press, Cambridge (1999)
10. L. Cardelli, A: Gordon, *Mobile Ambients*. Theoretical Computer Science, Special Issue on Coordination 240(1), 177–213 (2000)