Data Processing and Communication in Distributed Low-Power Sensor Networks Using Multi-Agent Systems

S. Bosse[1,3]*, F. Pantke[2,3] and F. Kirchner[1,3]

[1]University of Bremen, Department of Computer Science, Workgroup Robotics
[2]TZI-Center for Computing and Communication Technologies, Univ. of Bremen
[3]ISIS Sensorial Materials Scientific Centre, Bremen
*E-Mail: sbosse@uni-bremen.de

Summary: We propose and compare two different data processing and communication architectures for the implementation of mobile agents in sensor networks consisting of single microchip low-resource nodes.

Keywords: Sensor Networks, Distributed Data Processing, Agent, Computer Architecture, Communication, Single-System-On-Chip, High-Level Synthesis.

1. Introduction

Recently emerging trends in engineering and micro-system applications such as the development of sensorial materials show a growing demand for autonomous networks of miniaturized smart sensors and actuators embedded in technical structures [6]. With increasing miniaturization and sensor-actuator density, decentralized network and data processing architectures are preferred or required. A multi-agent system is used for a decentralized and self-organizing approach of data processing in a distributed system like a sensor network, enabling the mapping of distributed data sets to related information, for example, required for object manipulation with a robot manipulator.

Traditionally, mobile agents are executed on generic computer architectures [7,8], which usually cannot easily be reduced to single-chip systems like they are required, e.g., in sensorial materials with high sensor node densities.

We propose and compare two different data processing and communication architectures for the implementation of mobile agents in sensor networks consisting of single microchip low-resource nodes.

The distributed programming model of mobile agents has the advantage of simplification and reduction of synchronization constraints owing to the autonomy of agents.

2. Distributed data processing with state-based agents

Initially, a sensor network is a collection of independent computing nodes. Interaction between nodes is required to manage and distribute data and computed information. One common interaction model is the mobile agent. An agent is capable of autonomous action in an environment with the goal to meet its delegated objectives. An agent is a data processing system, a program executed on a computer system, that is situated in this environment [1] . A multi-agent system is a collection of loosely coupled autonomous agents migrating through the network. Agents can be used in sensor networks for:

- Sensor data processing and extraction
- Sensor data fusion, filtering, and reduction of sensor data to information in a region of interest
- Sensor data and information distribution and transport
- Global energy management, exploration and negotiation

Agents can operate state-based. An agent consists of a state, holding data variables and the control state, and a reasoning engine, implementing behaviours and actions. In this proposed data processing and communication architecture, the state of an agent is completely kept in messages transferred in the network providing agent mobility. The functional behaviour of an agent is implemented statically with a finite-state machine part of the local data processing system on register-transfer level (RTL).

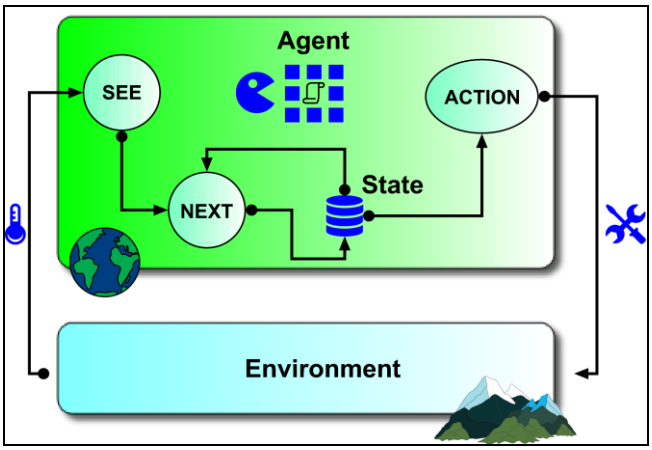

Figure 1. State-based agents and interaction with environment.

Agents record information about the environment state $e \in E$ and history. Let I be the set of all internal states of the agent. An agent's decision-making process is based on this information. The perception function *see* maps environment states to perceptions, function *next* maps an internal state and percept to an internal state, the action-selection function *action* maps internal states to actions (see also Fig. 1):

$$see : E \rightarrow Per$$
$$next : I \times Per \rightarrow I$$
$$action : I \rightarrow Act$$

3. Approach I: message-based/state machine agent implementation

Figure 2 shows the proposed execution environment used for the data processing agents. There is a message module implementing smart delta-distance routing of messages [2], providing some kind of fault-tolerance regarding interconnect failures, and several finite-state machines implementing the agent behaviours and providing virtual machines able to process incoming agents. All parts are mappable to digital logic on RTL and single-SoC system architecture, a prerequisite for miniaturized sensor nodes embedded in structures and sensorial materials.

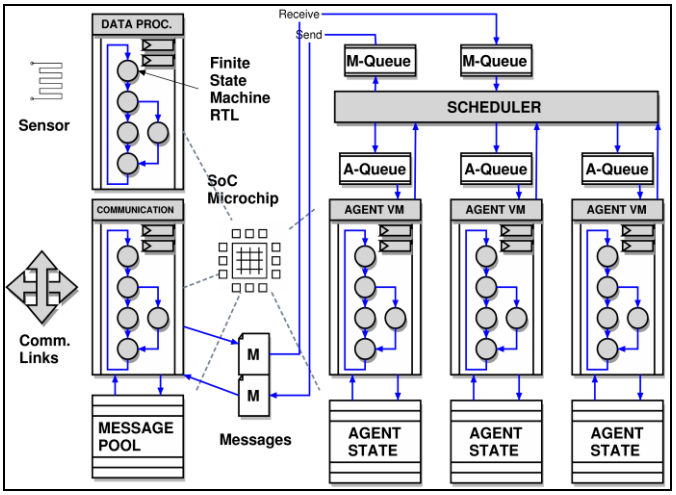

Figure 2. Sensor node building blocks providing mobility and processing for multi-agent systems: parallel agent virtual machines, agent-processing scheduler, communication, and data processing. All parts are mappable to digital logic on RTL and SoC system architecture.

The functional agent behaviour is implemented with a (non-mobile) finite state machine (virtual machine) built in the sensor node side, modelled with a high-level synthesis approach on an imperative multi-processing programming language level [3].

Inter-agent communication is provided by shared data structures, available on each sensor node. Each node is represented by a node agent, too, to ensure interaction and information exchange between mobile agents and the sensor node. All interacting agents must comply about the data structures and types, fixed at design time.

4. Approach II: multi-agent implementation using code morphing

Multi-agent systems providing migration mobility using code morphing can help to reduce the communication cost in a distributed system [4]. The second proposed hardware architecture and runtime environment is specifically designed towards the implementation of mobile agents by using dynamic code morphing under the constraints of low-power consumption and high component miniaturization. It uses a modified and extended version of FORTH as the programming language for agent programs. FORTH is a stack-based interpreted language whose source code is extremely compact. Furthermore, FORTH is extensible, that is new language constructs (called words, zero-operand functions) can be defined on the fly by its users. A FORTH program contains built-in core instructions directly executed by the FORTH processing unit and user-defined high-level word and object definitions that are added to and looked up from a dictionary data structure. This dictionary plays a central role in the implementation of distributed systems and mobile agents. Words can be added, updated, and removed (forgotten), controlled by the FORTH program itself. User-defined words are composed of a sequence of words. Again, the runtime environment is modelled on the behavioural level using the multi-process-oriented programming language and can be embedded in a single-SoC hardware design [4].

The principal system architecture of one **FORTH processing unit** (PU) part of the node runtime environment is shown in Fig. 3. A complete runtime unit consists of a communication system with a smart routing protocol stack, one or more FORTH processing units with a code morphing engine, resource management, code relocation and dictionary management, and a scheduler managing program execution and distribution, which are normally part of an operating system which does not exist here. A FORTH processing unit initially waits for a frame (a FORTH program) to be executed. During program execution, the FORTH processing unit interacts with the scheduler to perform program forking, frame propagation, program termination, object creation (allocation), and object modification.

The **scheduler** is the bridge between a set of locally parallel executing FORTH processing units, and the communication system, a remote procedure call (RPC) interface layered above SLIP, a fault-tolerant message-based communication system used to transfer messages (containing code) between nodes using smart delta-distance-vector routing [2].

The simple FORTH instruction format is an appropriate starting point for code morphing, i.e., the ability of a program to modify itself or make a modified copy, mostly as a result of a previously performed computation. Calculation results and a subset of the processing state can be stored directly in the program code which changes the program behaviour. The standard FORTH core instruction set was extended and adapted for the implementation of agent migration in mesh networks with two-dimensional grid topology. In our system, a FORTH program is contained in a contiguous memory fragment, called a frame. A frame can be transferred to and executed on remote nodes and processing units

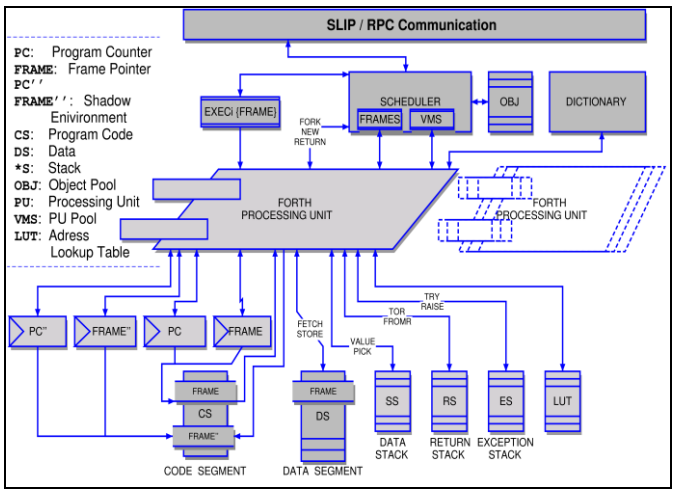

Figure 3. Mobile-agent runtime architecture providing code morphing, consisting of FORTH data processing units, shared memory and objects, dictionary, scheduler, and communication.

5. Comparison and Conclusions

In the following comparison, the first approach is abbreviated state-machine-based, the second code-based. Table 1

compares both runtime architectures and agent implementations. Both approaches allow the implementation of agent mobility and processing on hardware single-chip level. Flexibility and design time versus resource requirements is the main difference. The state-machine-based approach with fixed and hard implemented functional agent behaviour is well suited for a small set of different agents with simple algorithm complexity, whereas the code morphing approach is suited for a larger set of different agents with higher algorithm complexity.

A program controlled approach is less power efficient and requires more resources, but provides a higher lever of implementation and design freedom. The code morphing approach reduces communication complexity. One main issue addressed in the design of multi-agent systems is cooperation and communication of agents, and to ensure how can agents understand each other. Message based systems require some kind of communication language. Each node which processes agents must comply about well known data structures used for inter-agent communication, fixed at design time. There are only limited capabilities to handle data type inconsistency and the non-availability of expected data. In contrast, the code based approach uses named code and data words resolved by a dictionary, with a well known interface, and the capability to check and handle type inconsistency. The hardware implementation of the dictionary and the operational interface produces a fairly high overhead of the resources compared with the traditional shared data approach using memory references (as used in the state-machine-based approach I).

Future experimental investigations using real sensor networks with different classes of data processing algorithms should clarify the advantages and disadvantages of both approaches.

References

[1] Wooldridge, M., 2009, An Introduction to MultiAgent Systems, Wiley.

[2] Bosse, S., Lehmhus, D., 2010, Smart Communication in a Wired Sensor- and Actuator-Network of a Modular Robot Actuator System Using a Hop-Protocol with Delta-Routing, Proceedings of Smart Systems Integration Conference, Como, Italy, 23-24.3.2010.

[3] Bosse, S., 2011, Hardware-Software-Co-Design of Parallel and Distributed Systems Using a unique Behavioural Programming and Multi-Process Model with High-Level Synthesis, Proceedings of the SPIE Microtechnologies 2011 Conference, 18.4.-20.4., Prague, Session EMT 102 VLSI Circuits and Systems

[4] S. Bosse, S., Pantke, F., Kirchner, F., 2012, Distributed Computing in Sensor Networks Using Multi-Agent Systems and Code Morphing, ICAISC Conference, Prague.

[5] Kent, A., Williams (Eds.), J. G., 1998, Mobile Agents, Encyclopedia for Computer Science and Technology, New York: M.Dekker Inc..

[6] Pantke, F., Bosse, S., Lehmhus, D., Lawo, M., 2011, An Artificial Intelligence Approach Towards Sensorial Materials, Future Computing Conference.

[7] Peine, H ., Stolpmann, T., 1997, The Architecture of the Ara Platform for Mobile Agents, MA '97 Proceedings of the First International Workshop on Mobile Agents, Springer-Verlag London.

[8] Wang, A.I., Sørensen, C.F., Indal., E., 2003, A Mobile Agent Architecture for Heterogeneous Devices, Wireless and Optical Communications.

Table 1. Comparison of the two data processing approaches for mobile agents.

	I. State-Machine	**II. Code morphing**
Agent behaviour is ..	fixed, nodes must comply with previously defined common data types and structures as well as message formats	not fixed, nodes do not require knowledge of data structures and types in advance
Functional behaviour is implemented ..	statically in local data processing machine	dynamically in programming code
Implementation in ..	Hardware, single chip	Hardware, single chip
Agent state is kept in	data storage	code, stacks, and data storage
Message size depends on	full state size	full code size and partial state size
Hardware resources are ..	small (< 1M eq. logic gates including storage)	large (> 1M-3M eq. logic gates including storage)
Storage resources are ..	small (< 5000 register cells)	large (> 10000 register cells)
Speed is ..	high (1-2 clock cycles per statement)	medium (5-20 clock cycles per core word)
Power consumption is ..	low	medium