OPEN ACCESS

2nd International Electronic Conference on Sensors and Applications 15–30 November 2015

www.mdpi.com/journal/sensors

*Conference Proceedings Paper – Sensors and Applications*

# Agent-Based Solutions for Industrial Environments composed of Autonomous Mobile Agents, Modular Agent Platforms, and Tuple Spaces

**Stefan Bosse***

University of Bremen, Dept. of Mathematics & Computer Science, Robert Hooke Str. 5, 28359 Bremen, Germany

* Author to whom correspondence should be addressed; E-Mail: sbosse@uni-bremen.de

*Published: 5 November 2015*

---

**Abstract:** Design and Production processes become more and more complex. Today, industrial manufacturing environments consists of large-scale networks connecting smart sensors, embedded systems, server, desktop, and mobile computers. Mobile software Agents can be used in such strong heterogeneous environments with design, manufacturing, and logistics facilities. A unified agent model and a agent processing platform can overcome network and system barriers arising in such complex systems. The deployment of Industrial Agents can improve the scalability, productivity, and stability of modern adaptive and customized production processes, and aid the integration of sensor networks in cloud computing. This article shows the deployment and the relationship of industrial agents to industrial environments and common agent models, finally mapped on mobile program code executed by a low-resource and highly portable agent processing platform.

**Keywords:** Industrial Agents; Sensor Clouds; Internet-of-Things; Self-organization; Agent Platforms

---

## 1. Introduction

Future factory production and assembly environments require smart automation and are controlled by a massively increasing number of computers with sensorial feedback from machines, parts, products, and humans, consisting of a wide variety of different connected devices and software

programs, which can be considered as one big use case of pervasive and cloud computing with vanishing boundaries between the computing and the environment, and with a strong focus on decentralized distributed computing and information storage. These new complex information processing architectures are composed of hierarchical network graphs, and require some kind of self-organization and adaptability to overcome single-point of failure and robustness constraints. The data acquired from machines and sensitive products or parts are growing at a fast rate, leading to a large data volume that must be handled distributed with pre-processing, map- and reduce, and filtering algorithms.

Mobile Agents can be deployed in such large-scale and hierarchical network environments crossing barriers transparently, for example, industrial manufacturing and assembly environments, the Internet, Sensor Networks, and Cyber-Physical Systems (CPS). The networks can consist of high- and low-resource nodes ranging from generic computers to microchips, and the supported network classes range from body area networks to the Internet including any kind of sensor and ambient network. Mobile Agents can perform distributed computation in an autonomous manner, e.g., deployed in Structural Health Monitoring applications **[8]**.

In this work Agents are represented by mobile program code that can be modified at run-time by the Agents themselves. The presented approach enables the development of sensor clouds and smart systems 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, which embeds the control state and the private data of an agent, finally encapsulated in self-initializing and self-containing code frames.

This cross-platform interoperability is ensured by a modular and scalable Agent Processing Platform. The entire information exchange and co-ordination of Agents with other Agents and the environment is performed by using a Tuple Space database, unifying the platform and architecture specific data representation. The Tuple Space is used for any kind of information exchange including program code and directory and file system services mapped on the Tuple Space. Beside architecture specific hardware and software implementations of the agent processing platform, there is a *JavaScript* (JS) implementation layered on the top of a distributed co-ordination and management layer including distributed file and name services. The JS platform enables the integration of Multi-Agent Systems (MAS) in Internet server and application environments (e.g., WEB browsers). Agents can migrate transparently between different classes of computing devices and environments, ranging from hardware-level sensor networks (embedded in technical structures) to WEB browser applications or network servers without any required transformation.

## 2. The Agent Model of Computation and Industrial Agents

Traditional software is composed of objects and functions specified at design time. Though there are concepts of extending programs at run-time by dynamically loading libraries, these programs are mainly static. Usually, the execution of software is strongly coupled to a specific execution environments and operating systems. Interpreted script languages, e.g., *JavaScript*, are more loosely coupled and can be executed on a wider variety of host architectures, operating systems, and specific computers. Finally, a multi-program system requires communication regulated by protocols. Two programs participating in communication must have significant knowledge of the understanding,

meaning, and most of all the encoding of information encapsulated in communication protocols and messages.

Software agent technologies can overcome the limiting barriers in heterogeneous systems affected by technical unreliability. An agent is an operational unit, usually a software process, that satisfies the following properties:

*Autonomy*, *i.e.*, the execution of agents do not require continuously intervention of humans and machines, and agents have the control over their inner state and the actions they perform;

*Reactivity*, *i.e.*, agents respond to a an environmental perception, that can be the information of the state of a communication network and the connectivity, sensor input, or platform related properties (like available storage, architecture, other agents);

*Pro-activeness*, *i.e.*, the actions agents can perform do not depend only on the perception, they are planned based on goals an agent should reach, e.g., the delivery of sensor data from a source to a sink node in a network;

*Social ability*, *i.e.*, different agents can interact with each other to reach a group goal, based on the progress and actions of other agents.

A Multi-agent system (MAS) is a collection of individual agents capable to communicate and interact with each other, mainly by exchanging messages, and acting following a collaborative goal. Self-organizing is a common approach for a MAS to operate in unreliable environments with missing or incomplete world models.

*Agent Behaviour* models basically consisting of perception, reasoning, computation, and acting by performing discrete actions **[1]**, shown in Fig. **1**. Agent computation models are attractive in the context of sensorial and reactive systems found in manufacturing processes due to the inherent processing of sensing data that causes actions. Actions modify the environment, i.e., commonly data stored and computations performed outside of the agent (including data and activities of other agents), but they can affect the physical world, too, regarding CPS and machine interaction.

There are basically three major behaviour model classes suitable for industrial agents **[1]**:

***Reactive Agents.*** This agent model maps sensory input directly on a set of actions, i.e., the agents react immediately to perceptive data. This (sensor,action) mapping function can be ambiguous and conflicts and misbehavior can result. To overcome the incomplete sensor data reactivity and to provide run-time adaptability, *subsumption* architectures were proposed (Brooks, 1986), shown in in Fig. **1** on the upper right side. The agent behaviour is composed of different behaviours that can inhibit (block) lower prioritized behaviours, resulting in the selection of different (sensor,action) mapping functions. In this work, the agent behaviour can be transformed at run-time, which is a similar but more efficient and dynamic approach than inhibition used in the subsumption architecture.

***Deliberative Agents.*** Uses explicitly the more natural formulation of beliefs, desires, and intentions to select plans finally performing actions by a reasoner merging all these parts of a goal-orientated behaviour, shown in Fig. 1 on the lower left side. In the most common *Belief-Desire-Intention* (BDI) architecture [2] the beliefs relies on the sensory input accumulated over time, in contrast to pure immediate reactive behaviour. The desires represent the goals of the agent, and the intentions influence future plans and actions.

***Hybrid Agents.*** They combine methods and methodologies of reactive and deliberative agents with stacked layer architectures. For example, the *InteRRap* architecture (Mller, 1996) is composed of a world interface, behaviour, plan, and co-operation *layers* propagating sensory input upwards and actions downwards.

In this work, the agent behaviour is modelled using Activity-Transition Graphs (ATG) and agent interaction is performed by exchanging data tuples using a tuple space database. An activity represents a sub-behaviour that has a predecessor activity and a pre-condition basing on agent data as a result of previously sensory processing and computation, which is closely related to the reactive agent model, shown Figure 1 on the lower right side. An activity performs actions that can be classified in: (1) Modifying private agent data by computation (body variables of the agent); (2) Modifying global environmental data of host platforms and other agents by accessing a tuple space database discussed below; (3) Creating or destroying of agents; (4) Migrating of the agent to another host platform (a node in a computer or sensor network) taking a snapshot of the agent that continuos processing on the new platform. A behavioural transformation can be performed by adding or removing transitions between activities, and by removing or adding activities, enabling Dynamic ATG composition at run-time. The conditional transitions of the ATG and the ATG recomposition is comparable to the behaviour inhibition in the subsumption architecture (transitional reconfiguration). The inhibition can be considered as a behavioural transformation, too. In the BDI architecture, the state of an agent is given by the belief, desire, and intention databases. Regarding the BDI architecture, the ATG composition is related to the plans. The agent body variables mainly representing the beliefs, intentions, and desires. But the data of an agent is also contained in the tuple space, extending the state of an agent and couples the agent state with the states of other agents. The reasoner of the BDI architecture consists of different functions, the belief revision function, an option generation function, and an action selection function, which is mainly encapsulated in the activities and transitions of the ATG model.

Sensor input comes from and action output affects tuple-space database data, representing both the environmental world, the agent itself, and other agents state, or at least the public visible parts.

*Agent interaction* is usually performed by exchanging messages. A simple and robust communication model is the tuple space. A tuple space provides both synchronization and data exchange by a basic set of operations: {`in`, `rd`, `out`, `rm`, `mark`}. The `out` operation stores a data tuple consisting of one to *N* data values in the space (producer), the `in` and `rd` operations retrieve tuples and block the requesting agent if there is no matching tuple available (consumer), and the `in` operation destroys a tuple after reading (tuple atomicity), whereas the `rd` operation returns a copy only. The consumer operations base on patterns providing actual (which match a tuple) and formal parameters (which are replaced by values of a matching tuple). Tuple space communication is generative, i.e., the lifetime of a tuple can exceed the lifetime of the producing agent. To limit the lifetime of a tuple, the `mark` function can be used that assigns a time-out to the tuple that removes the tuple from the space after expiration.

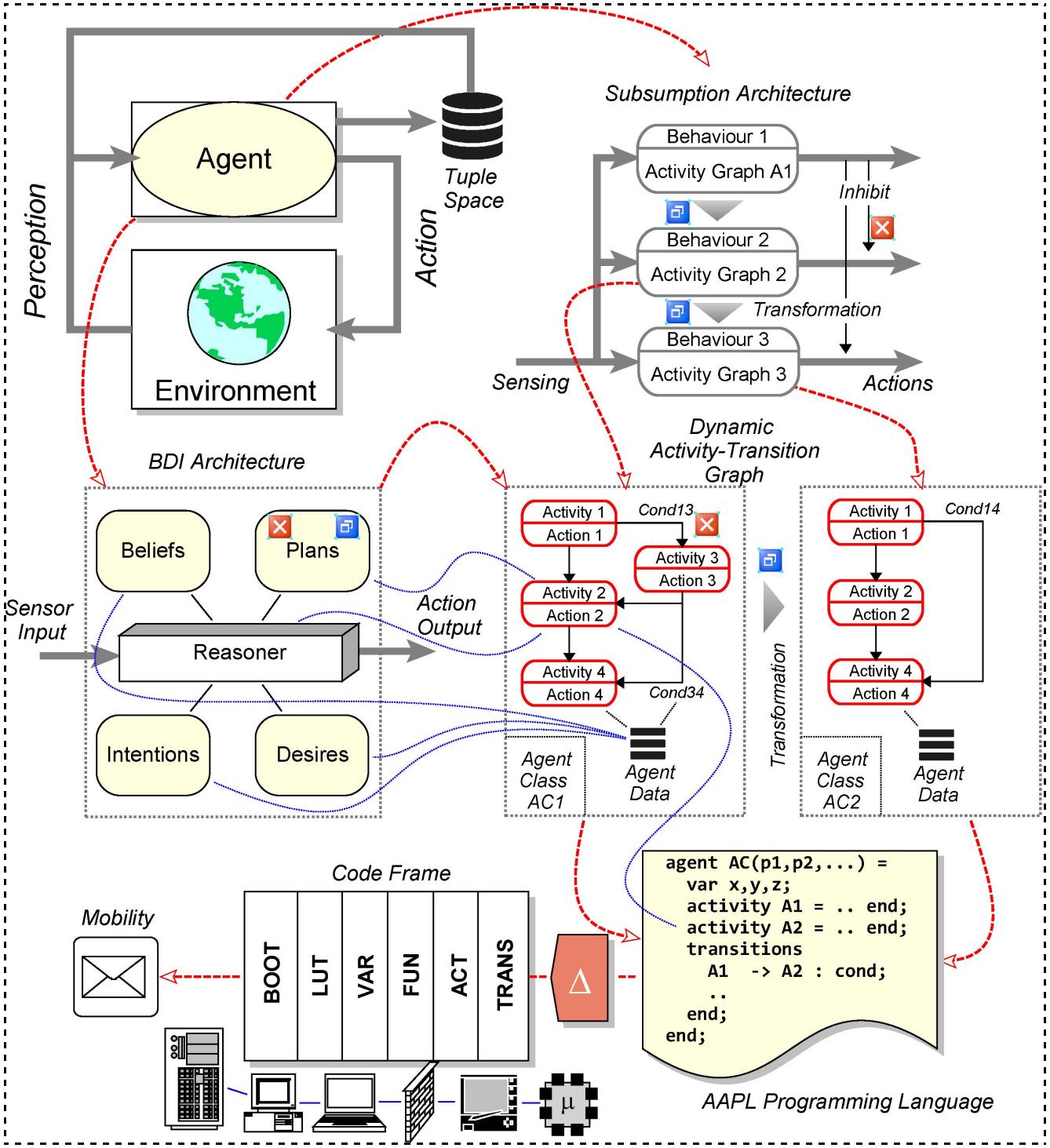


**Figure 1.** Agent Models: (Left, top) Basic Agent Model, (Right, top) Subsumption Architecture composed of different behaviour, (Right, middle) Reconfigurable (Dynamic) Activity-Transition Graphs (DATG) used in this work, (Left, middle) Relation to the Belief-Desire-Intention (BDI) Architecture, (Bottom) Programming model and code frame.

*Industrial agents* can cope with issues related to distribute processing, communication, customized and individual products, and planning tasks arising in modern manufacturing environments and a progressing globalization process [3]. The adaptability and learning features of agents can match the requirements in changing manufacturing and logistics environments. The tasks of industrial agents consists of sensor processing, planning, control, and management.

The sensing-aggregation-application layer model well covered by agents contains the tasks (1) Sensing; (2) Aggregation and Delivery; and (3) Application of sensory data and control of machines, offering perception of machines, products, and tools. This includes event-based sensor acquisition and sensor fusion (Sensing), data to information mapping (Aggregation), search of information sources and sinks, information delivery to databases (Application), delivery of sensing, design, and manufacturing information, propagation of new design data to and notification of manufacturing processes, notification of designers, end users, update of models and design parameters. Self-organizing aspects of agents can reduce the organization complexity significantly and can aid to improve overall system stability of the a for mentioned tasks [7]. Planning tasks of industrial agents, e.g., the co-ordination and scheduling of production resources and supply chains [6], is well matched by the BDI architecture and reflected by the Plans database.

## 3. The Agent Processing Platform, Agent Programming, and Mobile Code

Today, still many agent frameworks rely on the *Java* programming and execution platform, but the *Java* architectures do not match the constraints and requirements arising in multi-scale and multi-domain sensor networks, especially considering microchip scalability (e.g., **[9]**).

An agent consists of a behaviour and a state. The behaviour is given by the previously introduced ATG, and its state is given by the content of the body variables, the configuration, an identifier, and the control state (current activity). In this work the entire agent behaviour (ATG), the agent data, and the configuration and control state is encapsulated in program code organized in frames, see Fig. **1** (bottom, details in **[4]**). The code frames are self-initializing by a boot section.

A Virtual Machine Processor (Agent FORTH VM, *AFVM*, discussed in depth in **[5]**) is used to execute agent code on a traditional Multi-Stack *FORTH* processor architecture with an extended zero-operand word instruction set (α*FORTH*/*AFL*). This approach has the advantage of a large implementation variety ranging from microchips (System-on-Chip design) to WEB browser *JavaScript* implementations of the VM. This eases the deployment of agents in heterogeneous industrial environments and the Internet, i.e., Sensor and CPS clouds.

Most instructions operate directly on a data and a control return stack. A code segment stores the program code with embedded data. The program is mainly composed of words (stack functions). A word is executed by transferring the program control to the entry point in the code segment; arguments and computation results are passed only by the stack(s). The execution of agent code is token based, i.e., a unique token identifier is passed by queues to the VM processor, and passed back either to a VM processing or management queue of an agent manager, enabling fine-grained auto-scheduling and sharing of a VM by multiple agents.

The *AFVM* is a self-contained and stand-alone processing unit not depending on an operating system, *i.e.*, suitable for single microchip implementations embedded in sensor networks. To support the deployment of the *AFVM* platform in the Internet domain, the entire VM was implemented in *JavaScript* and a Distributed Co-ordination and Management Service with a Broker Server was added **[4]**. The Broker server enables the integration of pure client-side applications (like WEB browsers) in an Agent Processing Network (APN). All *AFVM* platform implementations are fully compatible on the operational level, i.e, code frames can be passed between different devices without transformation.

The core *FORTH* instruction set (containing basic computation and program flow control) was extended with agent specific operations: (1) Agent creation including forking and destruction; (2)

Tuple Space communication; (3) Mobility of agents; (4) Mapping of the ATG on program code by partitioning the code frame in activity word definitions and transitions tables containing code, too; (5) Behaviour modification based on ATG reconfiguration using code morphing. Though it is possible to write agent code directly in *AFL* that is compiled to a machine executable sub-set *AFM*, programming of agents require a more expressive programming language focusing on the ATG and agent interaction model and that is suitable for non-experts, which was finally provided by the *AAPL* programming language, discussed in **[5]**. The *AFL*/*AML FORTH* code can be easily compiled from *AAPL* sources using a simple compiler with a set of transformation and mapping rules only, as shown in Figure 1.

The program code frame of an agent encapsulates the entire state of an agent including data and configuration. Therefore, a snapshot of an agent can be migrated to another processing node by simply transferring the code frame to the new processing node in network messages using any communication protocol, which supports application-specific Sensor Networks like Sensorial Materials as well as Internet applications, finally boosting cloud-based agent frameworks for the industry.

## 4. Conclusions

A unified Agent Programming Model, Agents implementations using self-contained mobile program code suitable for low-resource platforms, and a modular and portable Agent Processing Platform can boost the deployment of industrial agents in strong heterogeneous design and manufacturing environments connected to the Internet significantly, finally enabling Cloud-based computing and storage. The information processing and distribution using industrial agents provides a higher Quality-of-Service than traditional static and stronger bound software approaches.

## Conflicts of Interest

The authors declare no conflict of interest.

## References

1. S. Bussmann, N, R. Jennings, M. Wooldridge, *Multiagent Systems for Manufacturing Control*, Springer, 2004
2. M. Wooldrige, *Intelligent Agents, in Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, G. Weiss (Ed), MIT Press, 1999
3. P. Leit £ o, S. Karnouskos, *Industrial Agents Emerging Applications of Software Agents in Industry*. Elsevier, 2015.
4. 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*, EUSPN 2015, Procedia Computer Science, 2015, DOI:10.1016/j.procs.2015.08.312
5. S. Bosse, *Design and Simulation of Material-Integrated Distributed Sensor Processing with a Code-Based Agent Platform and Mobile Multi-Agent Systems*, Sensors (MDPI), 15 (2), pp. 4513-4549, 2015, DOI:10.3390/s150204513.
6. F. Pantke, et al., *Symbolic discrete-time planning with continuous numeric action parameters for agent-controlled processes*, Mechatronics, Special Issue on System Integreated Intelligence, Elsevier (2015)
7. C. Sansores, J. Pav n, *An Adaptive Agent Model for Self-Organizing MAS*, in Proc. of 7th Int.

Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008), May, 12-16., 2008, Estoril, Portugal, 2008, pp. 1639 1642.

8. X. Zhao, S. Yuan, Z. Yu, W. Ye, J. Cao. (2008), *Designing strategy for multi-agent system based large structural health monitoring*, Expert Systems with Applications, 34(2), 1154 1168. doi:10.1016/j.eswa.2006.12.022
9. 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.