Smart dynamic software components enabling decision support in Machine-to-machine networks

Alexander Dannies[1]*, Javier Palafox-Albarrán[1], Walter Lang[1] and Reiner Jedermann[1]
[1] Institute for Microsensors, -actuators and -systems, University of Bremen
Bremen, Bremen, Germany

Abstract
The future Internet of Things will be extended by machine-to-machine communication technologies in order to include sensor information. The overwhelming amount of data will require autonomous decision making processes which are directly executed at the location where data is generated or measured. An intelligent sensor system needs to be able to adapt to new parameters in its surrounding unknown at the time of deployment. In our paper we show that Java enables software updates on mobile devices and also that it is possible to run algorithms required for decision making processes on wireless sensor platforms based on Java.

Keywords: *Machine-to-Machine communication, Internet of Things, autonomous logistics, Java, dynamic updates, OSGi.*

1. Introduction

Classical machine-to-machine (M2M) communication focuses on the supervision of large, expensive machinery, and on remote monitoring in a centralized point. But in the future, according to the vision of the internet of things (IOT), in which smaller physical objects will interact with each other through the use of the M2M concept, communication will become more and more ubiquitous.
The advancement of M2M communications from single machines to supervision of a network of objects will not be made by simply increasing the number of existing system and hardware solutions. After defining the requirements for integration of M2M into the IOT [1], adequate communication and software structures have to be found and programmed onto the hardware. In this paper we will discuss and demonstrate by our prototype implementation how ubiquitous M2M can be enabled by combining and reprogramming system components which are available in the market.

1.1 Combining cellular and infrastructureless networks

Nowadays, M2M communication is typically implemented by cellular radio networks (CRN) technologies, such as GSM and UMTS. The infrastructure of a commercial network operator consists of fixed base stations to cover large geographical areas. In order to make M2M technologies more ubiquitous, devices have to collect information from a high number of devices distributed in the environment. For such a detailed supervision CRN are rather disadvantageous for the following reasons:

- Communication costs have to be kept as low as possible.
- Network protocols have to be optimized for transmission of small packets of sensor data consuming as little energy as possible instead of enabling global communication.
- In many applications, such as the monitoring inside large buildings or rural regions, the supervised area will not be fully covered by the CRN of an external operator.

If wireless connectivity is required, local infrastructureless networks are the better solution for spatial monitoring of an area. Typical Ad-Hoc wireless sensor networks (WSN) using the Zigbee or the underlying 802.15.4 protocols, meet the requirements mentioned above. They provide coverage of even difficult areas by forwarding messages over multiple hops inside the network.

But on the other hand, pure WSN lack the ability to connect to global networks. Therefore, we suggest using a heterogeneous network combining infrastructure and infrastructureless technologies to enable future M2M networks which will not only supervise single machines, but be aware of their environment.

1.2 Local intelligence by Java-based dynamic software frameworks

The vast amount of data, provided by a distributed M2M network needs dedicated processing. According to the concept of cloud computing, the required computation resources can be provided as service by the network. The resources can be hosted by a stationary server farm as in [2] or [3], but in the case of M2M networks a more direct approach is to move the “cloud” into the network by processing collected data directly on the sensors. This approach entails advantages in regard to costs and robustness:

- The costs for the transmission of large amounts of unprocessed sensor data cannot be neglected, they have to be either paid directly to an external operator, or have to be calculated as service costs, for the current that the radio draws from the batteries. The communication volume decreases dramatically, if an intelligent processing directly on the sensor transmits only summaries, conclusions or warning messages about unexpected situations instead of the full raw data over the network.
- If the infrastructure or part of the WSN fails, processing can be continued by the remainder of the local network.

Immutable software, which is programmed in a static way and transferred to the sensor before distribution in the field, is unsuitable. The sensor software has to adapt to new situations, tasks and application fields, which were unknown at the time of deployment. This results in a permanent need for software updates. As a consequence the network nodes have to be equipped with an adequate operating system or software framework. In order to reduce the size of update files, the software should be structured in a modular way, whereby it is possible to update only single components of the software.

In [4], over-the-air and differential reprogramming in WSN is made; however, the applicability of the solution is limited to some hardware devices.

Java is the most common language that meets this requirement because dynamic class loading is one of its intrinsic features. It has penetrated more and more the realm of embedded systems. Optimized virtual machines have become available for several embedded controllers [5], [11].

1.3 Testing Java-enabled wireless sensor nodes and M2M platforms

Because of the enormous spreading and pervasiveness of Java, we focused on this language to implement an intelligent sensor node. Several Java-enabled wireless sensor nodes and CRN-enabled M2M devices were tested. Depending of the available resources of the device, two different software frameworks for handling of code updates were installed. The wireless devices were tested in regard to their ability to execute and update complex software algorithms within their computation and battery capability.

By measurement of required CPU time we could show that there are several Java-enabled wireless sensors platforms, which are capable of running complex algorithms as well as frameworks for automated software updates. Differences in the performance of the tested types of WSN hardware are evaluated by measurement of execution time for benchmark tests and example sensor data processing algorithms.

Our test bed shows how a combined network of infrastructure CRN and infrastructureless WSN can be installed. By local pre-processing on wireless nodes the network can provide a new quality of information to the end-user. World-wide access to the M2M system from the internet is provided by a web interface.

Furthermore, we could demonstrate by our prototype implementation the advantages of such an intelligent network for a logistic supervision and decision support tasks.

In section 2 we give an overview of the theory and background related to our paper. Subsequently, section 3 introduces the platforms for dynamic software updates and section 4 contains the performance measurements of the selected wireless sensor platforms. Section 5 describes the topic of software updates. Finally we summarize in section 6.

2. Background and vision of M2M and IOT

The initial idea behind the creation of the Internet of Things was to interconnect real-world objects globally. It emerged under a logistic point of view in which the items would be tracked over the existing internet. Its development of the communication technology has been built on top of it.

IOT means the connection of clearly recognizable physical objects (Things) with a virtual representation in an internet-like structure. Participants in the IOT are not only of human nature but also machines.
When the IOT concept was created, passive RFID (Radio Frequency Identification) and barcode were already mature technologies for item identification and tracking; the identification on the internet was made by manual inventory. RFID was used however because it does not require line of sight and requires less human intervention than barcode. Automated identification with the help of RFID is often considered as the foundation of the IOT. Its target is the minimization of the information gap between real world and virtual world.

Because some applications require communicating without human intervention, concepts such as
M2M communications came into mind as possible extensions of the IOT concept. M2M was an already existing technology which allowed automated exchange of

information among terminal devices like vending machines, vehicles or containers with a central point. M2M technology connects information and communication technology to build the Internet of Things (IOT).

2.1 Definition of M2M and available technical solutions

A M2M system consists of three basic components. A data end point (DEP) which can be a machine extended by a sender module with the main task of providing data. The second component is a communication network; this can be either wireless or wired. The data integration point (DIP) mainly plays the role of the gateway. It receives information from the DEPs and redirects it to a central point. Several commercial technical solutions for M2M can be found in the fields of industrial automation, transportation, smart energy and logistics.

M2M can be classified according to the physical transmission media it uses: The media can be either based on wired (Ethernet or optical), cellular (for example GSM, GPRS, UMTS, LTE-M and WiMAX) or "capillary" short-range technologies (for example Bluetooth, ZigBee, IEEE 802.15.4).

All of them offer advantages and disadvantages: wired communication offers the best reliability and highest data rates, but is expensive, complicated to install and not scalable. To bridge long distances the communication standards of 2G (GSM) or for higher data rates 3G (UMTS) can be used, but are expensive in maintenance and need a fixed infrastructure. Wireless sensor networks using protocols such as 802.15.4 at 2.4GHz are cheap, scalable, do not need infrastructure, but have drawbacks such as low coverage, security and data-rates and energy constraints.

2.2 The impact of M2M to logistic processes

The supervision of supply chains and logistic tasks is one of the main application fields of the IOT. The IOT differs from the idea of autonomous control in logistics [6]. A fully autonomous object requires basically only communication with its near-by neighbors and not necessarily internet-like networked structures.

Communication between RFID tags and a central point as in conventional M2M allowed complying with a ubiquitous necessity. Tracking the position of the identified items globally, seemed to solve it, however one consideration was missing: In the process of transportation damages such as spoilage or breakages may appear and it is required to know not only the position of the item but also whether its quality is acceptable.

That is the reason why the emergent technology of WSN, together with RFID, can be seen as the enabling concept of IOT. In WSN's the nodes have sensing, communication and processing capabilities and use M2M to communicate with each other and with the gateway.

Conventional M2M solutions require four basic steps: Data collection, transmission, assessment and response. But the gathering and transmission of all the available data alone can lead to a flood of information and asks too much of a human operator and is extremely costly. The entire decision making should be done autonomously, at best in the same location where the data is collected.

Our vision of an intelligent sensor network, from M2M to the IOT, proposes a change of paradigm in which the assessment (data processing) is performed locally in the wireless sensor nodes or on the gateway device. The concept of the intelligent container [7] includes the introduction of a decision support tool (DST) which can, as the name suggests, support humans in making decisions based on the sheer abundance of data occurring every second. The quality of perishable goods like fresh fruits or meat has to be monitored to ensure that the food reaches the end-consumer in the best state possible. On the other hand the economic aspect of the supplier benefits from the monitoring because losses due to reduced shelf-life caused by broken cool chains can be absorbed by intervening in logistics processes. Moreover, as mentioned in [1] the DST should provide device control, which includes activating, deactivating or updating the devices over the air.

Supervision of logistics processes is often limited to data-logging during the transportation and analyzing this data afterwards. Reactions to unexpected situations can only be triggered with long time delay or in the worst case an intervention is not possible anymore.

In order to be able to react to these events on time, it is necessary to monitor the cargo objects in a pervasive way, which means anywhere anytime. The "Internet of Things" can solve this problem. By creating a network of pervasive systems it becomes possible to collect real-time information with simultaneous consideration of the decision making process.

The objects or things use M2M communication to access the real-time data without human intervention. They can for example send an e-mail or SMS to a human with the condition information to be acted on at a reasonable price. Due to the increasing processing power (according to Moore's Law) on the one hand and the decreasing costs for hardware in general on the other hand, the feasibility for an implementation of omnipresent data processing by an advanced internet of things rises. The condition of the cargo, that may be for example signs of degradation, is

calculated by intelligent data processing algorithms in wireless sensor nodes.

As mentioned in [8], M2M enabled intelligent devices like the ones visualized in the concept of the IOT will impact the logistic process mainly in three ways: Self-aware products, delivery by product characteristics and proactive tendering.

In self-awareness, the cargo or thing is able to react to self-related problems as soon as they arise. The problems can be for example deterioration caused by fluctuations of environmental parameters such as temperature or humidity, leading to quality degradations such as decrease in aesthetic appeal. Algorithms to estimate the quality of the goods by biological models may be used. The Gompertz model will be introduced as example in section 4.2.2.

Quality of perishable goods such as fruit and vegetables are highly dependent of failures in the cold-chain. If the quality decreases, the delivery of the good has to be re-planned according to the actual product characteristics. New routes and alternate suppliers or buyers have to be found in accordance with the actual price of the cargo and with the aim of increasing the profitability.

In proactive tendering early information about the quality of the cargo will lead to take actions to reduce waste or to replan the supply orders. Prediction algorithms, such as Feedback-Hammerstein (section 4.2.1) can be applied to compute a model for temperature changes.

3. Platforms for dynamic software management of embedded devices

As mentioned before, our vision of an intelligent sensor network includes device control to react on dynamic changes in the environment. The sensor nodes must not only be able to update the software modules over-the-air, but also to do it dynamically during run-time. WSN is still an emergent technology; the research focus has been mainly on energy efficient algorithms. The question arises whether it is feasible to implement the mentioned solution in an energy-efficient way on resource-limited devices.

Typically, WSN nodes are programmed once, in native code such as nesC without taking into consideration neither modularity, over-the-air (OTA) programming or dynamic features.

In [9] three execution environments for software update management in sensor networks are compared: monolithic (TinyOS), modular and virtual machines (VM). VMs interpret symbolic or intermediate code instead of directly transferring and executing machine code. The size of a program in an intermediate code such as Java class files is between five and ten times smaller than the same program in machine code. Han [9] concludes that VM is the best one regarding the energy costs of network transmission. He also concludes that if VM is combined with a modular environment, the energy costs of updating a task are very low. The only disadvantage of using a VM environment is the cost of interpretation. As mentioned before, IOT should combine the best of capillary and cellular data transmission. Specialized VMs, which are written to run on sensor nodes, such as Maté [10], only cover the first one mentioned (short range) and are not suitable for cellular networks.

Beside these WSN-specificVMs, Java is a mature technology to run intermediate code on a VM. Java implements the concept of "write once, run anywhere". It is the most common language that meets the requirement of the ability to extend software with dynamic code segments through the use dynamic class loaders.

In this section we will discuss the advantages and hardware requirements of virtual machines and software frames for enabling dynamic updates.

3.1 Native Code versus virtual machine

On the selection of the software platform for dynamic updates, a series of figures of merit have to be taken into account. The software must support updates, be fast and able to run on diverse hardware platforms. Basically, there are two types of programming languages: the so called high-level and the interpreted ones, each one of them with their advantages and disadvantages.

When speaking about workstations, the high-level or native languages such as C or C++ have faster execution times and allow memory management but the code has to be compiled according to the hardware. On the other hand the interpreted languages such as Java or C# are platform-independent but the execution time is in general not optimal. This disadvantage can be compensated by Just-in-time compilers, which translate only those parts of the code to machine instructions that are most critical for the executions speed.

With the development of WSN as an emergent technology, it was clear that the solutions were not suitable to be used on the first sensor nodes available to the market because of their very constrained resources. Initially, native code such as nesC was used but they are not able to update software or run on different platforms.

Different VMs have been implemented for sensor nodes. One example is the above mentioned Maté [10]. It allows executing high level instructions by an interpreter. New application scripts can be sent over the air, requiring only

very small communication volume and user memory on the microcontroller.

In the recent years there have been lots of efforts to provide VMs for high level languages such as Java on sensor nodes [11]. Depending on the memory and CPU recourses on the sensor node, either the Java Micro or Standard Edition is supported.

3.2 The Java Micro Edition on sensor nodes

Because of the hardware-constrained nature of microcontrollers, there is not enough space on them to install a full operating system which can be used as a base for a virtual machine. In contrast microcontrollers are using a virtual machine which runs on bare metal. The kilo VM (kVM) requires only a few 100 kBytes of memory and runs on ARM processors. One example implementation is the Squawk VM used by Oracles SunSPOT [11]. It includes the functionally of the Java Micro Edition (JavaME) as part of the Connected Limited Device Configuration (CLDC). New software components can be uploaded in the form of software suites containing MIDlets (see section 3.4.3).

A further example for the implementation of a kVM is the Preon32 sensor node by Virtenio [12]. Its kVM does not exactly cover the whole CLDC standard but is close to it. As consequence it is not possible to install MIDlets on Preon32 nodes.

Floating point and double precision data types are not part of the original JavaME but were introduced in CLDC 1.1. Although the Java SE Math Library is not available by default, manufacturers, such as Oracle and Virtenio, have implemented their own library for mathematical functions.

3.3 The Java Standard Edition on sensor nodes

There is a variety of VMs supporting the Java Standard Edition (JavaSE) on the market, both open-source and commercial ones. We selected JamVM as a representative of the open-source type and JamaicaVM from AICAS as a representative of the commercial ones. Both of them are able to run on workstations or on embedded devices.

JamVM makes use of the GNU Classpath [5]. Their implementation is more suitable for sensor nodes, is extremely small but still able to support the full specification including, class-unloading and native support. It can be installed on several operating systems like Linux, Mac or Solaris as well as different hardware architectures like PowerPC, ARM or AMD64.

JamaicaVM of AICAS [13] provides Hard Realtime Execution, Realtime Garbage Collection, dynamic loading, multi-core support, and native support. It can be installed for diverse operating systems like Linux and Windows, and several architectures like x86 and ARM. Besides it offers the possibility of combining all files relevant for the application (a set of class files) and the Jamaica VM into a single executable file. The implementation offers a trade-off between run-time performance and code size.

JavaSE allows replacing the system class loader by user defined class loaders. This feature, which is not available in JavaME, is essential to control mutual access between different dynamic components (see section 3.4.1).

3.4 Java frameworks for dynamic code

Although it is possible to handle the dynamic loading of new software components by basic features of the Java VM, it is more efficient to use additional Java features or a software framework to handle updates:

- JavaME allows installing and executing new software components during runtime in the form of so called MIDlets [14]. It is commonly used for mobile devices such as cell phones, but also supported by up-to-date WSN hardware such as the SunSPOT sensor node [15].
- Agent platforms, such as MAPS, JADE [16] or Agilla [17], provide a framework which enables migration of software agents between different local platforms. The migration is in general based on an internet connection but can be adapted to the needs of WSN and CRN technologies.
- Whereas software agents have their focus on artificial intelligence and research, the Open Services Gateway initiative (OSGi) framework originates from industrial automation and building maintenance. New components can be installed as software bundles without the need to stop or restart the machine to perform a software update. Furthermore, OSGi provides methods to exchange information and services between different bundles.

Due to the dynamic features, efforts have been made to run OSGi on resource limited devices; OSGi has been tested in pervasive environments [18, 19 and 20] but not in sensor networks context, yet.

There is a major difference in the concepts behind agents and OSGi. OSGi components can be organized in a hierarchical structure. It is even possible to update components which are currently in use by another component. Software agents, on the other hand, are organized in a flat structure. They can exchange messages

for communication, but services are only provided by the framework, without any means to update or modify their implementation. Because we consider this feature to update components in a hierarchical structure as crucial for intelligent objects, we focus on OSGi as second example framework for dynamic code.

3.4.1 Inter component communication

The major task of a framework for handling dynamic software components is to install, upload and run different components in parallel and independently. But it is also necessary to provide some kind of communication between different components. The access to the code and memory of other components has to be restricted in order to avoid that one malfunctioning component can crash the whole system. The concepts range from full protection such as in Android, where apps can only access methods provided by the operating system, to controlled accesses by one of the following solutions:

- One solution to provide inter-component communication is the use of shared memory. As example we consider a system containing the three components “decision unit”, “sensor driver” and “radio driver”. The decision unit has to read sensor data and to communicate the result over the radio. Sensor and radio data are written to the memory by one component and polled by another component.
- A more efficient solution allows that certain parts of the code of one component can be invoked by another component. Only methods that are explicitly published as service can be accessed from the outside. If the component is updated, the framework has to redirect the service request to the new component. The decision unit calls services provided by the sensor and radio component.
- In a third solution, components can exchange messages or events by registering for a special service provided by the framework. For example, the sensor component informs the decision unit by an event if new measurement data is available. This third solution is the standard way of agent communication.

3.4.2 OSGi

OSGi was introduced in the 90’s to manage controller units for building maintenance remotely. The original idea was that a human operator can start, stop and update software components without being on site. But OSGi can also be used in a M2M way: a central unit or machine can control a remote unit by calling system functions provided by the framework. An agent-like migration is also possible: a component uploads its own code to another platform, starts it, and stops its own execution on the first platform.

OSGi provides two ways of inter-component communication. Components can call services published by other components. Or they can send an event to a blackboard. Other components can register as listeners for a certain type of event.

OSGi runs on top of a Java VM. The mutual access to code of different components by services is handled by user defined class loaders. Unfortunately, this feature is only available in the Java Standard Edition. As consequence, it is not possible to run OSGi on the SunSPOT platform. Furthermore, a typical OSGi framework needs at least 32 MByte of user memory, which is also not available on the SunSPOT.

There has been some effort to make OSGi services and dynamic uploads available for JavaME by a so called OSGiME framework [21]. This approach keeps the core features of OSGi like dynamic software updates but being compliant to Java ME CLDC means that user-defined class loaders are not available. To be able to fit on resource-constrained platforms the OSGi technology is simplified by removing unnecessary and semantically complex features like dynamic and optional imports.

A further minimal OSGi implementation by ProSyst, which requires only eight MByte of memory [23], does also not provide dynamic uploads.

3.4.3 JavaME and MIDlets

Recently JavaME has been integrated into WSN technology, for example the SunSPOT nodes use a proprietary Java VM, to offer a sensor node that can be programmed over-the-air, but only inside the sensor network. Updates over external global networks are not supported by the original software package.

shows the execution environment to enable MIDlets. On top of the host operating system sits the configuration which is extended by a profile and optional packages. As a configuration the Connected Limited Device Configuration (CLDC) is used, which contains a very small virtual machine (KVM). On top the Mobile Information Device Profile (MIDP) enables the execution of MIDlets. Writing MIDlets limits the programmer to the functions of JRE 1.3. After installation the MIDlet can be started, paused and destroyed by calling an interface function. The transition to the destroyed-state is irreversible.

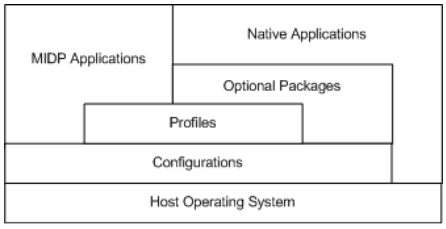

Figure 1: Structure of a MIDP environment

The SunSPOTs have modularity limitations inherent from any JavaME deployment because the MIDlets are unable to communicate among themselves. However, it is possible to overcome this disadvantage by using Record Management Storage (RMS) or programming them within the same suite. In theory Inter-MIDlet-Communication (IMC) is possible in MIDP 3.0, but currently a SDK is missing so that the current status is that MIDP2.0 is still used.

4. Java performance on WSN platforms

The interpretation of dynamic code by a VM requires some overhead, compared to precompiled "C" code. In order to decide whether this overhead hinders the application of Java VMs on WSN nodes we carried out performance measurements on different hardware platforms.

4.1 The hardware platforms

All hardware platforms we used are available off-the-shelf. They can be classified in two categories: telematics units and wireless sensor nodes. Table 2 shows selected characteristics for each hardware platform.

The telematics units are equipped with extended communications possibilities like 3rd generation mobile telecommunications (GSM / UMTS) and wireless LAN. Additionally, it is possible to get geodata via the global positioning system (GPS) and hard disk storage allows extensive data-logging.

The VTC 6200 from Nexcom is used as a reference platform. It is equipped with an Atom processor (1.6 GHz) and 2 GB of main memory.

The DuraNAV serves as an exemplary platform for lower power consumption. It utilizes an ARM architecture CPU (400 MHz) and 64 MB of RAM. Both can run different Java VMs (JamVM, Jamaica) and different OSGi implementations (Prosyst, Equinox).

Table 1: Telematics platforms

	DuraNAV	VTC6100
CPU (MHz)	PXA255 (400)	N270 (1600)
RAM	64 MB	1 GB
OS	Linux	Linux
Java Edition	SE	SE

In the category of wireless sensor nodes we chose three products, which are listed with its properties in Table 2.All these platforms enable the usage of the high-level programming language Java.

Table 2: 802.15.4 Wireless sensor platforms

	Imote2	Sun SPOT	Preon32
CPU (MHz)	PXA 271 (416)	SAM 9G20 (400)	Cortex-M3 (72)
RAM	32 MB	1 MB	64 kB
OS	Linux	None	None
JVM	any	Squawk	Custom
Java Edition	SE	ME CLDC 1.1	ME almost CLDC 1.1

4.2 Example algorithms

As example algorithms we utilized two synthetic standard benchmarks (Dhrystone and LINPACK), the inversion of a 20 by 20 matrix of double values as well as two real-world application algorithms.

Dhrystone is a synthetic benchmark using integer operations, whereas LINPACK uses floating point operations. The latter one calculates the average speed of floating point operations during solving a n by n system of linear equations Ax = b. In addition to the abstract LINPACK benchmark for matrix operations, we also tested the inversion of a 20 by 20 matrix in double precision as example for the computation needs of a more

complex sensor evaluation task. For the matrix inversion the functions of the JAMA library [24] were used.

4.2.1 Temperature prediction

The two real-world application algorithms can be applied in the field of logistics. The Feedback-Hammerstein-algorithm is used in the context of the transportation of bananas in a refrigeration container. Bananas are living organisms with a metabolism so they emit heat and gases such as CO_2 and C_2H_4 - a phytohormone which is responsible for the ripening process. The algorithm identifies the parameter for a model to predict the temperature curve during cooling. By taking the generated heat of the bananas into account the model accuracy is improved compared with a simple model that is based only on thermal time constants. The structure of the applied model is shown in Figure 2.

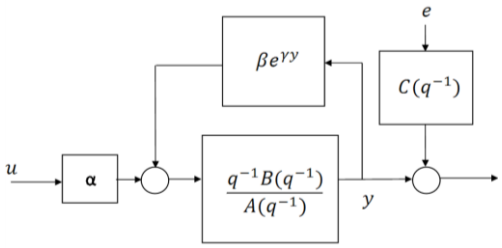

Figure 2: Feedback-Hammerstein Model

4.2.2 Gompertz-model

The Gompertz-model's application lies also in the carriage of fresh produce in the cold chain – transport of meat. A statement about the quality of meat can be done by using the bacteria count in a meat sample. The speed of bacteria growth is in general calculated as a function of temperature according to the law of Arrhenius for reaction kinetics. The bacteria count over time functions shows a lag phase during which the bacteria growth is delayed for a certain period of time. A combined model includes the Gompertz model to describe the lag phase and the Arrhenius model to describe to temperature dependency. By using several fixed values which are specific for a certain type of meat the number of bacteria can be estimated, which is correlated to the quality and the shelf-life [25]. The update of the model state for each measurement interval requires the calculation of three exponential and two logarithmic functions. If the temperature is the same as in the previous interval, the calculation of the logarithmic functions can be skipped.

4.3 Test results (execution speed and feasibility of algorithm)

The following Table 3 contains the results of the different benchmarks for the chosen reference platform (gateway device, Telematics unit VTC).

Table 3: Results of the chosen benchmarks on the reference platform (VTC)

Benchmark	Result
Dhrystone	523 ms
Linpack	45,778 Mflops/s
Feedback-Hammerstein	7 ms
Matrix inversion 20 by 20	1 ms
Gompertz model (single interval)	0,7 ms
Gompertz model (3450 intervals)	9.1 ms

In what follows all diagrams displaying the results of the measurements are relative values in comparison with the reference platform. Figure 3 shows the results of the Dhrystone-benchmark: A correlation between the processing power of the platforms and its available CPU and RAM is obvious. Even though Imote2 and SunSPOT have the same clock-rate of the CPU, the execution time seems to be linked to the available RAM of the systems. Consequently the order of the performances from slower to faster is the same as the amount of memory in ascending order.

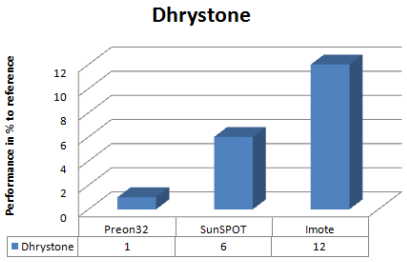

Figure 3: Results of the Dhrystone 2.1 benchmark

Unlike the first benchmark, Figure 4 shows a different correlation for floating point operations. In this case the influence of the amount of RAM at hand on the performance is less significant. The performance of the SunSPOT is better than that of the Imote2 for the LINPACK benchmark and for the matrix inversion. An explanation for this result could be the newer CPU architecture of SunSPOT, which seems to have improved floating-point processing power. The highest difference is observed for the inversion of a large 20 by 20 matrix. In this case the SunSPOT is eight times faster than the iMote at the same clock speed.

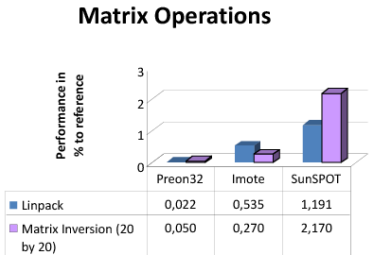

Figure 4: Benchmarks for LINPACK and matrix inversion

The results of the first tested exemplary real-world algorithm is shown in Figure 5. The Feedback-Hammerstein algorithm was executed with different orders. Similar to the previous benchmark the SunSPOT is three to four times faster than the Imote2 because of the newer CPU architecture.

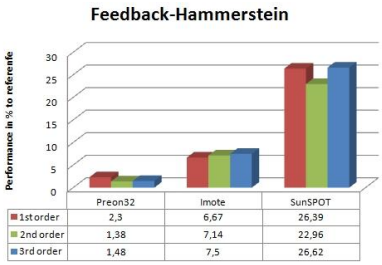

Figure 5: Results of the Feedback-Hammerstein algorithm

The results of the execution time of the second exemplary real-world-application algorithm are depicted in Figure 6. The Imote2 takes about five times more time for the execution than the SunSPOT.

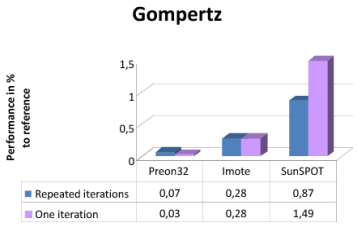

Figure 6: Results of the Gompertz-algorithm

In all five benchmarks the performance of the Preon32 was by far the lowest. But this result is no wonder because of the very resource-constrained platform. The CPU runs at only 72 MHz and only 64 kByte of RAM are available. In comparison to that the SunSPOT has a clock-rate of 400 MHz and 1 MB of RAM.

Even though the results in comparison with the other platforms at first appearance seem to be not that good, it is feasible to run these algorithms on the platforms: For example, the time required to execute 72 iterations of FH-algorithm required for three days of hourly samples is about two seconds. The Gompertz-algorithm takes about three to twelve seconds depending on the amount of temperature changes, which is also fast enough taking into account that a change in temperature is measured only once a minute.

4.4 Comparison of framework performance

In the previous section we have shown that all tested hardware platforms are capable of executing typical algorithms for sensor data evaluation. But it still has to be questioned, how much overhead is created by using a framework to manage software updates. To answer this question, the execution time for different algorithms was compared for a) direct execution as a Java class file and b) running them as software bundle inside the Equinox OSGi framework.

The only platforms able to perform these tests are the gateway devices and one of the sensor nodes – the iMote2. Because of the similar processing power we compared it with the DuraNAV system.
Figure 7 compares the execution time on Imote2 and DuraNAV of the benchmarks as a class-file or an OSGi-bundle.

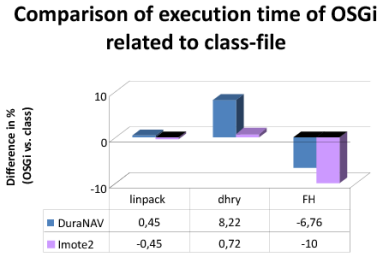

Figure 7: Difference in execution time in % between OSGi and class

The result of our test is that the execution time of the synthetic benchmarks was of the order of eight percent slower when the algorithm was run in the OSGi-environment instead of a direct execution of a class file. A completely different result is generated when comparing the real-world-example-algorithm of Feedback-Hammerstein. In this case the execution time of the OSGi-bundle was approximately seven to ten percent faster than running the algorithm directly from a class file.

From these results one can infer that the use of a framework can make sense. Not only the ability for dynamic software updates becomes possible but also the execution time is not increased in a way that the advantage is negated – even an improvement of execution speed is possible, depending on the type of algorithm.

Of course it remains to be seen if and how a framework can be introduced to resource-constrained wireless sensor nodes in the future and if the behaviour of this implementation will be similar.

5. Software updates over multi modal networks

Oracle offers for its sensor node SunSPOT a graphical user interface (Solarium) and also a command line tool based on ANT to deploy software connected to the computer via USB or over the air. The term over the air (OTA) programming means the distribution of new software updates to wireless devices without the need of a cable connection. By using one sensor as a base station it becomes possible to communicate with other sensors of the WSN in range of this platform.

The disadvantage of this approach is that in order to be able to use this software the user needs to install it on his machine. In contrast to that, our approach is of a web-based nature. As a result anybody can use it from a remote location via the internet.

5.1 Our test and demonstration system

The concept we used in our demonstration system is depicted in Figure 8.

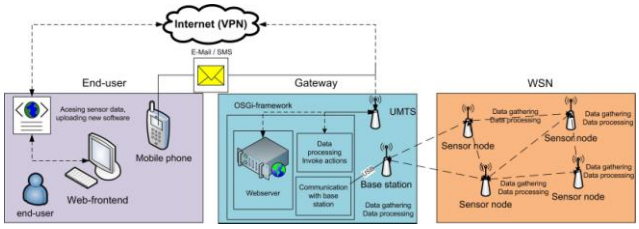

Figure 8: Concept of the demonstration system

From a remote internet connection the user opens a web browser and accesses a web interface, which is provided by the gateway device. This becomes possible by running an OSGi-Framework on the gateway device which hosts a bundle that publishes a web interface. The connection between the end-user and the gateway becomes possible due to the use of a virtual private network (VPN) connection which connects gateway device and end user in one space. The gateway device is connected to a sensor node (SunSPOT) that acts as a base station. In this way a connection through multimodal networks from the end-user (internet) to a sensor node (802.15.4 network) is possible.

The user can get information from any sensor in the network as well as update software on a specific sensor node remotely without being on site.

6. Summary

The Internet of Things is a concept in which objects use the infrastructure of the internet to communicate with each other in a global way. An essential part of the IOT concept is to enable objects to exchange data between each other autonomously, i.e. without human intervention. Autonomous communication between the objects requires sensing, evaluating and communicating. Environmental parameters are sensed, intelligent algorithms run on the sensor node using this acquired data and the result is transmitted wirelessly to a gateway. This leads to the ability to create autonomous decision making or supporting functionality to disburden human operators who otherwise would have to battle their way through a flood of raw data. The gateway which is the connection point to the outer world should have M2M communication capabilities with the sensors and with the internet infrastructure to allow pervasiveness. The impact of the use of all these technologies in the logistic process is mainly in three ways: Self-aware products, delivery by product characteristics and proactive tendering. However, the available technological solutions that make the IOT concept possible have their Achilles' heel in the sensor end-point. Over-the-air dynamic data programming is possible with off-the-shelf components. On the one hand the wireless sensor node SunSPOT from Oracle can be used as base station as well as part of the WSN. On the other hand commercial telemetric units such as DuraNAV and VTC, with Linux as operating system, can serve as a gateway device. In combination with OSGi as software framework a user can remotely update software in the WSN from any location using a web interface. Java on the sensor nodes is useful, because the communication volume for updating software bundles is lower than in the case of monolithic software. However, JavaME running on sensor nodes does not yet allow communication between MIDlets therefore the modularity is limited due to missing communication between different modules.

Acknowledgments

The research project “The Intelligent Container” is supported by the Federal Ministry of Education and Research, Germany, under reference number 01IA10001. The current study is also supported
by International Graduate School in Dynamics in logistics at Bremen University.

References

[1] S. Tompros(Ed.),Internet-of-Things Architecture IOT-A Project Deliverable D3.1 - Initial M2M API Analysis

[2] Integrating K. Ahmed, M. Gregory, Integrating Wireless Sensor Networks with Cloud Computing, Seventh International Conference on Mobile Ad-hoc and Sensor Networks,2011, pp. 364-366.

[3] W. Kurschl, W. Beer, Combining cloud computing and wireless sensor networks. In Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services (iiWAS '09). ACM, New York, NY, USA,2009, pp.512-518.

[4] N.F. Shafi, Efficient Over-the-air Remote Reprogramming of Wireless Sensor Networks,Master thesis,Queen's University Kingston, Ontario, Canada,2011.

[5] R. Lougher, 2010, JamVM [Online]. Available: http://jamvm.sourceforge.net/ [Accessed 20.07.2012].

[6] K. Windt, M. Hülsmann, Changing Paradigms in Logistics - Understanding the Shift from Conventional Control to Autonomous Cooperation and Control. In: Understanding

Autonomous Cooperation and Control - The Impact of Autonomy on Management, Information, Communication, and Material Flow, (M. Hülsmann, K. Windt, eds.) pp. 4-16, Springer, Berlin, 2007

[7] W. Lang, R. Jedermann, D. Mrugala, A. Jabbari, B. Krieg-Brückner, K. Schill, The Intelligent Container - A Cognitive Sensor Network for Transport Management. In: IEEE Sensors Journal Special Issue on Cognitive Sensor Networks, 11(2011)3, 688-698

[8] H.Sundmaeker, M. Würthele, S.Scholze, Challenges for Usage of Networked Devices Enabled Intelligence, Vision and Challenges for Realising the Internet of Things,CERP-IoT – Cluster of European Research Projects on the Internet of Things,2010, pp. 93-103. Available from http://docbox.etsi.org/tispan/open/IoT/CERP-IOT_Clusterbook_2009.pdf

[9] S. Han, R. Rengaswamy, R. S. Shea, M. B. Srivastava, Sensor Network Software Update Management: A Survey, Internation Journal of Network Management . 15 (2005) 283-294

[10] P. Levis, D. Culler, Maté: a tiny virtual machine for sensor networks. In Proceedings of the 10th international conference on Architectural support for programming languages and operating systems (ASPLOS-X). ACM, New York, NY, USA, 2002.

[11] D.Simon,, C. Cifuentes, D. Cleal, J.Daniels, D.White, Java(TM) on the bare metal of wireless sensor devices: the squawk Java virtual machine. In: Proceedings of the 2nd international conference on Virtual execution environments, Ottawa, Ontario, Canada, ACM, 2006. (doi: 10.1145/1134760.1134773)

[12] VIRTENIO. 2012. Available: http://www.virtenio.com/de/produkte/hardware/preon32.html [Accessed 20.07.2012].

[13] F.Siebert, Hard Realtime Garbage Collection. aicas GmbH, Karlsruhe, 2002.

[14] U. Breymann, M. Heiko, JAVAME Anwendungsentwicklung für Handys, PDA und Co. 2008

[15] Sun SPOT World. 2012 Available from http://www.sunspotworld.com/

[16] F. Bellifemine, G. Caire, A. Poggi, ,G. Rimassa, Jade - A White Paper. In: "EXP in search of innovation - Special Issue on JADE" TILAB Journal,3, (2003).

[17] F. Aiello, A. Carbone, G. Fortino, S. Galzarano, Java-based Mobile Agent Platforms for Wireless Sensor Networks, Proceedings of the 2010 International Multiconference on Computer Science and Information Technology (IMCSIT),2010 ,pp.165-172

[18] A.Ibrahim, L. Zhao, Supporting the OSGi Service Platform with Mobility and Service Distribution in Ubiquitous Home Environments. Comput. J. 52, 2 (2009) 210-239 DOI=10.1093/comjnl/bxn032 http://dx.doi.org/10.1093/comjnl/bxn032

[19] M. Desertot, S. Do, D. Donsez, M. Bui,Mobile Agents Platforms over OSGi,Proc. of 4th International Conference on Computer Sciences, Research Innovation and Vision for the Futur, RIVF'06, 2006.

[20] S. K. Lee, J. H. Lee, OSGi based service mobility management for pervasive computing environments. In Proceedings of the 24th IASTED international conference on Internet and multimedia systems and applications (IMSA'06), 2006, pp.159-164.

[21] A. Bottaro, F. Rivard, OSGi ME An OSGi Profile for Embedded Devices. OSGi Community Event 2010.

[22] PROSYST. 2010. The World's smallest OSGi Solution [Online]. Available:http://www.prosyst.com/index.php/de/html/news/details/18/smallest-OSGi/ [Accessed 20.07.2012].

[23] National Institute of Standards and Technology (NIST) 2005, JAMA : A Java Matrix Package. Availabe at http://math.nist.gov/javanumerics/jama/

[24] Determination of the shelf life of sliced cooked ham based on the growth of lactic acid bacteria in different steps of the chain J. Kreyenschmidt, A. Hubner, E. Beierle, L. Chonsch, A. Scherer and B.Petersen Faculty of Agriculture, Institute of Animal Science, University of Bonn, Bonn, Germany

Alexander Dannies received his Diploma in Electrical Engineering and Information Technology with specialisation in microelectronics / micro system technology from the University of Bremen in 2010. Since February 2011 he is a research associate of the Institute for Microsensors, -actors and –systems at the University of Bremen. There he is currently involved in the project "Intelligent container" and is researching the topic "Data interpretation in sensor networks".

Javier Palafox-Albarran has a Master of Science degree in information and automation engineering from the University of Bremen. Previously he has earned several years of industry experience working in industrial Automation. Currently he is pursuing a PhD in the Institute for Microsensors, -actuators and –systems (IMSAS). His research topic is on the analysis and prediction of sensor and quality data in food transportation supervision. He is also a member of the International Graduate School for Dynamics in Logistics.

Walter Lang studied physics at Munich University and received his Diploma in 1982 on Raman spectroscopy of crystals with low symmetry. His Ph.D. in engineering at Munich Technical University was on flame-induced vibrations. Science 2003 he is the head of the Institute for Microsensors, -actors and –systems at the University of Bremen. His research focus includes the manufacturing of miniaturized sensor components and the automated processing of sensor data.

Reiner Jedermann finished his Diploma in Electrical Engineering 1990 at the University of Bremen. After two employments on embedded processing of speech and audio signals, he became in 2004 a research associate in the Department of Electrical Engineering at the University of Bremen. He finished his Ph.D. thesis on automated systems for freight supervision end of 2009. His current research focus is the analyses of spatial temperature profiles and the implementation of automated decision tools for container supervision.