Reiner Jedermann and Walter Lang:

Mobile Java Code for Embedded Transport Monitoring Systems

In: Caspar Grote and Renate Ester (Edt.) Proceedings embedded world Conference 2006, February 14-16, Nürnberg, Germany, Franzis Verlag, Poing, Germany, Vol2. pp. 771- 777 (ISBN 3-7723-0143-6)

IMSAS Institute for Microsensors, -Actuators and -Systems, University of Bremen

Abstract: Mobile software agents are the method of choice if embedded system should dynamically adapt to variable tasks. The supervision of fruit transports is an outstanding example for such a system. An intelligent freight container has to adapt to the requirements of different transport goods. Which sensor range has to be supervised, how should deviations be assessed and at which warning levels should an appropriate reaction be triggered? These transport instruction travel together with the real object in form of a mobile freight agent along the supply chain from system to system. Modern runtime environments like Jamaica allow running JAVA on embedded systems without stalls by garbage collection. With some changes it is even feasible to run the Java Agent Development Framework JADE on ARM processors. Further technologies from the fields of RFIDs and wireless sensor networks are integrated into our autonomous monitoring system.

1 Autonomous transport monitoring systems

The increased processing power of embedded systems enables new solutions in supply chain management and logistics. Quality supervision along the supply chain can be fully automated by utilising a system that does much more than merely checking threshold values. By using the feature of JAVA to execute mobile code, these autonomous transport-monitoring systems can dynamically adapt to the special needs of each kind of good. As part of a collaborative research centre[1] on autonomous processes in logistics, we are developing such an intelligent sensor system that could be integrated into standard containers or reefers.

Goal of our system

The system improves the transport facilities for perishable goods. The owner writes a specific transport instruction into an electronic consignment note, which contains information about the impact of different environmental factors on the quality of the goods.

The dynamic quality model and the actual quality state are part of this electronic waybill. It can also contain a small program that is activated when a potential risk for the freight is detected. It might, for example, change the route planning or order a replacement delivery. For defining the transport instruction, the owner is supported by a framework API that defines a common base class for user-defined functionality. Additionally, the owner may change general or user-specific parameters.

Selecting a programming language

The electronic waybill accompanies the freight along the transport route. At each transhipment, the host system could change from a PC based server to an embedded system or vice versa. Platform independence is a major point in selecting the programming language. It cannot be assumed that the owner of the goods has special knowledge on embedded systems. It must be easy for him to edit and complete the waybill on a PC. Furthermore, the transport system will continuously be extended by new kind of goods which

[1] Autonomous Cooperating Logistic Processes – A Paradigm Shift and its Limitations, University of Bremen, see www.sfb637.uni-bremen.de

implies an equally dynamic framework. It must be possible to dynamically load and unload code at runtime.

JAVA meets all these requirements and has the additional benefits of being a very widespread and simple language. Nevertheless, it has rarely been used on embedded systems due to its high consumption of resources. But the increased power of up-to-date processors and special programming environments make it feasible to run complex JAVA code on embedded systems.

Fruit Logistics as example

Fruit logistics places high demands on the quality monitoring. Fruits and vegetables are even after harvest a living product, as part of the maturing process takes place during transport. The speed of these processes depends on various factors like temperature, humidity, and concentration of oxygen and carbon dioxide. Especially the gaseous ripening hormone ethylene has a considerable impact. Instead of recording the course of these environmental parameters in a data logger, the system informs the owner immediately of major changes. To know in advance about quality changes is crucial in fruit logistics so as to make corrections in the transport planning or warehouse keeping as early as possible.

Local data interpretation

The vast amount of sensor information cannot be handled by human resources alone. There is a high need for automated data interpretation. Communication systems for transport supervision are getting into the market[2] but they are still lacking intelligent solutions for data reduction.

Local data processing reduces the amount of data that has to be transferred over expensive mobile communication networks. The systems work autonomously, even if the external communication fails and guarantees a continuous supervision of the goods. In our work we present an adaptive interpretation system that can run on a standard PC as well as on a standard check-card-sized processor module.

2 Intelligent objects versus supporting environment

The idea of intelligent environments is now more than five years old. It was mainly applied in research on the adaptive house[3], where every light switch and radiator vale is equipped with microprocessor and communication. The inventory of the house automatically adapts to the needs of the inhabitants.

It seems to be natural to use a similar approach in logistics by making each object like pallets and boxes intelligent. But the hard cost pressure demands other solutions. On item level only standard RFID-Tags with a price of 10 or 20 cents are acceptable. With current technologies intelligent objects with processor, sensors, battery, and communication are far beyond this scope.

For these economic reasons it is more appropriate to integrate the infrastructure into the means of transport. They stay in the ownership of the transport company. The willingness to invest is much higher at this level. In our solution the means of transport provides a platform on which the objects can “run” their intelligence and monitor quality influence factors by extensive sensor equipment. Therefore, we would rather name it a “supporting environment”. The “intelligence” is shifted into a software representation of the object separated form the physical instance. The system concept contains three different layers

- the physical object, tagged with a standard RFID for identification,
- the representation of the object in form of a mobile agent or JAVA application and

[2] see Press note form IBM / Maersk in the RFID Journal (www.rfidjournal.com/articleprint/1884/-1/1)
[3] see Lesser 1998 for example

- the supporting environment, consisting of the processor and sensor platform integrated into the means of transport.

Other application fields

Instead of the described transport scenario, this solution could be applied to a shop floor for example. The work pieces are marked by RFID tags. Only the machines need to be equipped with a processor platform. The assembling instructions as representation of the objects are transferred to the machine that handles the object.

The need for mobile code

On their way through the transport or production chain, the physical objects are accompanied by their software representations. When the object is loaded to a new means of transport the representation has to be transferred as well. To be adaptive to multiple and even yet unknown kinds of good and materials, the platform has to be able to execute dynamic code. The representation has to support mobility for the transferring process.

3 JAVA as a “mobile” language

Execution of dynamic code is an essential feature of JAVA. Copying the intermediate class files to the destination system is the simplest way to achieve mobility. If the name of the transferred class is only known at runtime, the class can be invoked by the standard *java.lang.ClassLoader*. The *loadClass* method takes the class name as a parameter. By overwriting the *findClass* method, the user can implement his own mechanism to load classes that are transmitted in a special format, for example over a socket-stream. But JAVA offers more powerful tools to support mobility. The *java.net.URLClassLoader* searches classes and archives in a given path, which could contain local directories or remote ftp-servers. Each classLoader creates its own namespace. Using a separate classLoader for each transport instruction avoids conflicts through overlapping class names by different programmers.

To store the state of an object JAVA provides special features for serialization. The *ObjectOutputStream.writeObject* method writes recursively the current data of all member and parent objects to a stream. Because this feature was originally meant to locally store and reload objects only the names of the required classes are written along the object state. A hash code or fingerprint prevents that an object is reloaded with an incompatible version. In order to transfer a running program to another computer, it is necessary to add the class files to the serialized object. But there are some drawbacks. The serialized object can become very large because it contains the complete hierarchy of parent classes. Furthermore, the user has to make sure not to store data in the object that is only valid on the local system. To avoid redundancy it is better to only transfer the code and data of derived classes that are actually changed.

4 The agent framework

Agents as a software concept to run programs in a distributed network have been investigated for a couple of years. Agents are software units that perform autonomously a task on behalf of their owner. They communicate peer-to-peer to achieve their goals[4]. Some agents hop through the internet to search information on enabled remote platforms. In other applications they cooperate as virtual football players to win the Robocop. The ubiquitous computer viruses and worms might be regarded as software agents, too, although they don’t tend to clean up after their job is done.

In the context of this work, agents can be understood as extended concept of mobile software that uses a standardized communication language. The most common environment to test and implement agents is the JavaAgentDEvelopment[5] framework (JADE) that results

[4] A detailed discussion of software agents can be found in Bingus 2001 and Wooldridge 1995.
[5] http://jade.tilab.com/index.html

from a former EU research project[6]. We found that the functions of this framework are very useful to implement the autonomous transport monitoring system.

The architecture or structure of an agent is separated in different units or behaviours, for example to handle incoming messages, to request the sensor system and to calculate the quality modelling. The thread handling is provided by the framework. The agents as a whole are using pre-emptive threads. Inside the agent cooperative multitasking is used to run independent units. Therefore, the units do not need to synchronize to access the data fields of the agent. If a unit causes a deadlock only his agent rather than the entire system is blocked.

Agents communicate among themselves by asynchronous messages independently of their current location. Because raw data could be misinterpreted by external participants all messages are send in form of a subset of the standardized FIPA-ACL language[7]. After defining the vocabulary, objects are automatically translated by the framework into an XML-like string representation. The framework requires JAVA J2SE as an additional library.

JADE has a built-in mechanism for migration that seamlessly transfers a running agent as a serialized object to a remote server. Unfortunately, we could not use this mechanism directly in our logistic system. Because of the nested hierarchy of the agent base class the deserialization of the agent needs a huge amount of processor time. Furthermore, the migration is restricted to peripheral platforms that are permanently connected to their host. Therefore, the built-in migration cannot be used for autonomous units, which have to be fail-safe against communication disruptions.

5 Running JADE on embedded systems

In the beginning of the 90th JAVA was originally written to run on portable microcomputers to control household appliances[8]. Since then it has become one of the major languages to implement business logic on central servers. But new programming environments brought JAVA back to the embedded world. The unpredictable interruptions of the virtual machine by the garbage collector have been an obstacle to run JAVA on systems with real-time requirements. The Jamaica virtual machine[9] avoids stalls by using real-time garbage collection[10]. Pre-compiling the static part of the Java byte code to native machine code speeds up the execution of the program. Only dynamic parts of the code have to be run by the virtual machine.

A special version of JADE, the Light Extensible Agent Platform (LEAP), was tested on mobile phones and palms by Moreno[11] et. al. By means of a graphical interface, the user sends requests and monitors results that were retrieved by software agents. Although JADE-LEAP is well tested on PCs, an implementation for "industrial" embedded platforms has not been made available yet.

As an example for the future "intelligent" container we selected a Strong-ARM processor module[12] with 16 MByte Flash and 32 MByte SDRAM. The embedded Linux operating system requires less than the half of the memory resources. The processor operates at a clock rate of 200 MHz.

[6] In 2003 a board has been founded to promote the further evolution of Jade by TILAB and Motorola, as a follow-up of the European project IST-1999-10211.

[7] The Agent Communication Language (ACL) was defined be the Foundation for Intelligent Physical Agents (FIPA).
[8] The Green Project started 1990 at Sun Microsystems to develop the Star Seven microcomputer.
[9] The JamaicaVM is a product of aicas GmbH, Karlsruhe, Germany, http://www.aicas.com
[10] see Siebert 2002
[11] see Moreno 2003
[12] The DNP1110 from SSV EMBEDDED SYSTEMS, Hannover, Germay, http://www.dilnetpc.com/

The adjustment of JADE-LEAP for ARM processors turned out to be an extensive task. The source code is composed of more than 800 files. The embedded JAVA environment provides almost all functions of J2SE except for graphics. But some missing or problematic functions had to be detected and worked around in JADE.

Furthermore, the execution time of the framework was not satisfactory. Object oriented programming allows calling classes over an interface, which makes the program very flexible. The implementations of the interface can be exchanged and either statically or dynamically linked to the system. For example, a new UMTS communication class can replace a standard TCP-IP class without changing the method calls in the rest of the code. The JADE developers made extensive use of this approach. But the use of interfaces reduces the scope for the compiler to optimize the code. Beside the size of the code, this is the main reason why the execution of JADE is slow compared to straightforward-programmed applications.

The translation from JAVA objects like the working copy of the consignment note into a platform independent language was identified as the major bottleneck. The built-in mechanism of JADE adds member elements which are listed in a vocabulary or ontology definition file to the message text. To retrieve the elements the ontology is compared against a table of the objects getter methods. This table is generated by reflection, i.e. by a call to *java.lang.Class.getMethods()* that turned out to be very expensive. The performed optimization prevents redundant calls to *getMethods()* by caching the table after the first call. The speed of the translation process was thereby accelerated by a factor of three.

Furthermore the translation is done in two steps to keep the option to implement other languages than FIPA-ACL. The object is first transformed into a nested intermediate list. In the second step the list is either translated into a byte or string orientated language. By a direct translation and some other reductions of the framework's flexibility further optimization is possible.

6 Application in transport monitoring

To show how the described concept could be applied in transport logistics we installed the processor module into a model container. The module was equipped with the necessary peripherals for a transport monitoring system. Sensor nodes can be either connected directly by SPI[13] or wireless based on the new 802.15.4 low-power communication protocol. A unit for external communication that was constructed by a partner institute[14] automatically switches between different available mobile networks. The sensor concept was presented in detail on the Eurosensors 2005[15].

The detection of freight loading and unloading is done by an RFID-Reader. We had to dismiss our first idea to store the mobile agent on a tag attached to the freight item. The available memory of 100 Bytes[16] is far too small to hold the code of a typical monitoring agent with a code size of about 20 kBytes. Additionally, the time span to access the tag by a door reader is limited to few seconds given by the time a fork-lift needs to pass a gate. Therefore, only small data packages can be stored on the tag. In our current approach we store only an IP-address and the current quality state on the tag for quick access with a hand-held reader. From the IP-address the container knows where to request the mobile agent.

[13] A standard 4-wire serial bus (Serial Peripheral Interface)
[14] ComNets, University of Bremen
[15] See Jedermann, 2005
[16] Value for ISO 15693 tags with a transmission rate of about 1 kBit per second at a carrier frequency of 13.56 MHz. UHF technology will bring some improvements but no general change.

Transmission of mobile agents along the transport chain

The producer or manufacturer of the good defines how sensitive the monitoring agent reacts to changes in the monitored values. The electronic consignment note contains a JAVA archive with the derived classes for freight specific instructions. The agent accompanies the freight item along the transport chain. In the first step the agent is started on a server in the warehouse. He supervises the freight item as long as it is waiting for transport.

When the freight is loaded to a means of transport the following steps take place:

- The container permanently runs a management agent. He receives a notification from the RFID-reader that a new freight item has arrived and sends a *“doTransfer”* command to the IP that is stored in the tag.
- The instance of the monitoring agent in the warehouse receives this command. The agent stores its current state in the electronic consignment note and sends it along with its own class files as *“handOver”* message to the means of transport.
- The container’s management agent extracts the state and the class files from the consignment note and starts a new instance of the monitoring agent.
- Finally the instance in the warehouse terminates.

During the transport, the agent collects the sensor data and calculates the resulting stress for the good. If it detects a potential quality risk, it carries out the programmed action, for example sending a warning message to the owner or informing the driver to change the route. In our demonstrator, all warning messages are displayed by a graphical user interface.

On unloading, the current IP-address and quality state are written back to the tag. The agent waits for a request from the next means of transport.

7 Required resources and results

By running test cases for real-world scenarios we verified whether the resources of the typical embedded systems are sufficient to run the software for an autonomous container monitoring system. Our StrongARM[17] test system features 16 MByte flash and 32 MByte SDRAM at a clock rate of 200 MHz. The Linux operating system leaves 10 MByte flash and 15 MByte SDRAM to the user.

The JAVA environment links the virtual machine together with the native code containing the compiled classes into a single file for the target processor. For the application combined with the JADE framework the executable file has a size of 2.5 MBytes. The required minimum heap size of 12 MByte takes almost the whole free user memory.

The code size of the agents is less crucial. The class files of a complete monitoring agent comprise about 20 kByte. But only individual derived classes have to be mobile. A simple quality model based on a time-temperature-integration for example needs about 3 kByte.

Beside the dynamic classes, the electronic consignment note contains the freight state in form of a FIPA-ACL message with a size of about 1.2 kByte. A warning message that contains only state changes requires 600 Bytes. The transmission of the complete consignment note takes about 3 seconds. Warning messages need less than one second to be transmitted.

The total time to transmit a mobile agent encloses communication, loading dynamic classes, agent start-up, and restoring its state. With 15 seconds it falls short of our expectations. But in transport planning, time is not a critical parameter; therefore delays in the order of tens of seconds should be acceptable.

Compared to the migration time between two PCs of about 100 ms or 200 ms, the transmission time to the embedded system is slower than the ratio of clock rates indicates. The migration time will be reduced by switching to an XScale processor that offers double

[17] Intel's StrongARM SA-1110 is an ARM7 derivate.

the clock speed and an improved ARM9 architecture at the same power consumption of 1 Watt. Our aim is to reduce migration time to a few seconds by further optimization of the framework.

8 Summary

JAVA is a convenient language from a programmer's point of view, but it does not excel in modesty when it comes to hardware resources. It is simply a trade-off between increased hardware and reduced development costs. In our work we showed that it is feasible to use JAVA on embedded systems to run mobile code. We presented an implementation of the concept of "supporting environments". Freight items are accompanied by a software representation in form of a mobile agent. The means of transport provide a platform for the agents by an embedded processor module. The JADE framework that simplifies the programming of individual transport instructions into software agents was adapted to ARM processors.

Especially in transport logistics, this concept offers considerable advantages as compared to pure telemetric solutions. The system is less sensitive against communication failures. Local data processing guarantees permanent supervision of the freight. Furthermore, the costs of mobile communication are reduced.

9 References

Bigus,J.: Intelligente Agenten mit Java programmieren; Addison-Wesley; München; 2001

Jedermann,R.; Behrens,C.; Westphal,D.; Lang,W.: Applying autonomous sensor systems in logistics; Combining Sensor Networks, RFIDs and Software Agents; EUROSENSORS XIX; 11th-14th September 2005, Barcelona, Spain

Lesser,A. et. al.: A Multi-Agent System for Intelligent Environment Control; UMass Computer Science Technical Report 1998-40

Moreno, A.; Valls,A.; Viejo,A.: Using JADE-LEAP to implement agents in mobile devices; TILAB "EXP in search of innovation"; 2003; Italy; http://jade.tilab.com/papers-exp.htm

Siebert,F.: Hard Realtime Garbage Collection, aicas GmbH, Karlsruhe 2002

Wooldridge, M., and Jennings, N. R. (1995). Intelligent Agents: Theory and Practice. The Knowledge Engineering Review 10(2): 115-152.

Acknowledgements

This research was supported by the German Research Foundation (DFG) as part of the Collaborative Research Centre 637 "Autonomous Cooperating Logistic Processes".

Contact Address

Dipl.Ing. Reiner Jedermann;	rjedermann@imsas.uni-bremen.de;	Phone ++49/421/218-4908
Prof. Dr. Walter Lang;	wlang@imsas.uni-bremen.de;	Phone ++49/421/218-4701

University of Bremen, FB1, IMSAS (Institute for Microsensors, -Actuators and –Systems)
Otto-Hahn-Allee, Building NW1
D-28359 Bremen; GERMANY
Fax 0421/218-4774