Object Oriented Software Development in Embedded System Environment

József Tick

Budapest Tech, Bécsi út 96/B, H-1034 Budapest, Hungary, tick@bmf.hu

Abstract: This paper deals with software development possibilities in embedded system environment especially focusing on object oriented paradigms. In the beginning, this paper will give a classification of embedded systems based on their performance, then it will outline software development possibilities of the most commonly used mid-range embedded systems. The paper will detail the benefits of the application of object oriented paradigms, and some possible solutions. It will present the application possibilities of JAVA as the most suitable environment in the category of mid-range embedded systems.

Keywords: Object oriented software development, embedded systems, JAVA

1 Introduction

Embedded systems are small sized independent systems built on microcomputers, which control different equipments. These equipments are of a relatively wide range and play an important role in everyday life. We meet them in cellular phones, in microwave ovens, in cars, in video recorders, watches, toys and in several other equipments [1].

The development of embedded systems is unbroken and has overtaken the development of PCs recently. It is quite astonishing that approximately 100 million microprocessors have been sold and have been built into desktop computers, mainly in PCs lately. At the same time 3 billion microprocessors have been sold to be used in embedded systems.

This enormously large number already implies that developing software products to be used in embedded systems means a significant market. While, in case of software development of large systems, set methodologies, tools and software libraries are available for software developers, till then software developers for embedded systems meet again those problems that have been left behind in case of the evolution of large systems.

In case of small sized embedded systems the problems of restricted memory capacity, the relative slowness of machines, the restricted accessibility of

resources in all aspects, as well as the modest level of services in case of own-platformed developing systems show up again.

At the same time, the requirements from both the end-users and the application environment are very strict and high. These requirements can be noticed in three different fields:

1 In case of user interface the end-users have got used to the comfortable windows interface in case of large computers (PCs) and would like to use it in their small equipments. The windows based graphical handling is extremely resource demanding.

2 The processing speed is a serious criterion partly from the user side (s/he does not want to wait a lot), and partly from the side of the application environment (real-time applications).

3 The correct and exact handling of multitasking is an important requirement since most of these systems execute real-time controls. The realisation of such applications indicates a serious challenge in an environment with poor resources.

The performance of embedded systems rapidly grows. Their resources are gradually expanding, however, both the applications for large computers as the pulling force and the expectations from software developers are gradually growing thus setting high demands facing small-sized systems.

2 Classification of Embedded Systems

Embedded systems can be classified from different aspects. In our case the classification will be based on the resources and/or the performance of these systems. All over the world several types of systems are in use and myriad types of microcomputers are produced. It is almost impossible to consider each made, that is why only those types are taken into consideration which meet the requirements set in the development of embedded systems, are widely used and of which an adequate number of publications have been available so far. The examination cannot be expanded to those equipments that are produced for special purposes thus documentation is not or hardly accessible and only a few publications can be accessed.

Taking the above into consideration the following three categories can be set up:

2.1 Low Performance Embedded Systems

These systems are quite cheap, of low performance and of simple structure. Most of them contain a one-chip microcomputer. The processor itself is typically a 4,-8 bit one, its clock frequency varies from 0,1 MHz to 4-8 MHz. Its memory ranges from some hundred bytes to two Kbytes and the used I/O ports are generally of digital types in the number from four to eight. A typical example of this type is the MicroChip 16f84 which is widespread and most frequently used.

The software is developed at assembly level or with the help of a simple C compiler. The development at a foreign platform (generally PC) is typical due to comfort. The software can be loaded to the microcomputer via a simple serial interface or can be burnt in electronically or with a masking technology.

2.2 Mid-range Performance Embedded Systems

These systems offer a good price/performance ratio, their central element is a complex one-chip microcomputer.

The processor itself is a 16 bit processor with RISC architecture, its clock frequency ranged from 4-8MHz to 40 MHz. The RAM varies 2-8 Kbyte, the used I/O ports are analogue as well as digital ones, their number varies between eight and thirty two. They offer several other built-in services like multi-level interruption possibilities, different timing, encounters, rapid analogue-digital converters and several types of communication possibilities.

Considering that this category is preferred by the production side as well, four different types of processors are most commonly in use in the market. These ones are the older INTEL 8051 type, the ATMEL ATmega type, the Motorola 68 and the MicroChip 16f87x.

In general, the software used in these systems has been developed in C with the help of C compiler in most cases while in only a few cases it is developed at assembly level. The development at a foreign platform (generally PC) is typical due to comfort. The software can be loaded into the microcomputer via an intelligent interface or can be burnt in electronically or with a masking technology.

2.3 High Performance Embedded Systems

These systems are nearing to the bottom end of PC categories and their performance coincides with the performance of the bottom-end PCs. In this category no one-chip applications can be found, the goal is the possibility of expansion, the modular type structure and the compatibility with the PC world to a certain extent. The application field of these processors well differs from the ones

described above, and these ones are more frequently implemented in the industry or in semi industry complex controls.

PC104 can be mentioned as a typical application, which practically coincides with a lower performance and sized (90x96 mm) PC mother board with 104 expansion connection points (buses).

Examining this model from the software side, it is dominantly compatible with PC world, the reason of which is the demand for the use of software developed for PCs.

3 Conventional Software Development Possibilities in Mid-range Embedded Systems

In case of low performance systems, the supply of software is quite low, the application of object oriented paradigms cannot be expected due to lack of resources. In case of high performance systems the application of “pared down” solutions taken from PC environments are preferred thus it solves the problem. The real challenge appears in the case of mid-range performance systems that is why the software development possibilities of these systems will be examined hereafter.

The application of these systems mostly happens in real time environment [2], which requires the use of such an operation system, which enables multitask handling. These are called RTOS (Real Time Operating System). These small systems deal with the handling of hardware resources apart from task handling. Therefore, they are inserted between the physical and the application levels (see Figure 1).

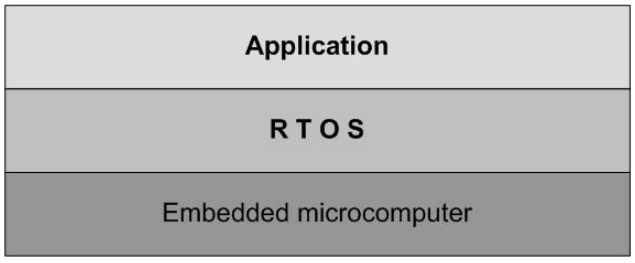

Figure 1
The application levels in case of compiled C, or assembly application

In conventional cases the application is typically a software product developed in C or at assembly level. In both cases the application is close to the microprocessor’s logical architecture, run time and memory size is minimised, the

expected functionality can be perfectly realised and time problems do not appear during operation. Development is executed during cross systems, which assures a comfortable environment for programme writing, editing, translating, simulation, debugging and tracing during simulation. The new system can be built up with the help of already developed and tested elements of existing programme libraries. The reuse of such software makes the development not only quicker but makes it a better quality and a more efficient development.

After development, the debugged programme can be downloaded into the microcomputer and can be thoroughly tested with the use of different tools in real environment. The modification in this case is also simple and quick.

This solution is widespread, the development happens in comfortable environment and the result is a good technical quality product.

The problem with this development methodology appears to be the high offset threshold, since a relatively large and thorough (machine level) knowledge is required in order to make applications. This knowledge depends on microcomputers, physical architectures as well as on languages. In case of a change in models, a large part of this knowledge cannot be used, a deep understanding and knowledge of the new model is required again.

The other problem with this development methodology seems to be that the systems are more and more complicated at micro levels, the controlling and processing tasks are becoming more complex and more difficult to manage. The object oriented paradigm has emerged due to these problems in large computer environments too.

4 Object Oriented Software Development Possibilities in Mid-range Embedded Systems

At the examination of object oriented software development the starting point should be that the solution applied in microcomputer environment, the language presentation and environment should be compatible with the environment of PCs, since development is carried out in that environment. The other essential aspect is that a well known, already introduced language, environment and philosophy can be much easily introduced and spread in the world of micros since the willingness of application is higher.

According to certain publications the number of object oriented languages now exceeds 150, however, only a few languages are really widespread and frequently used.

When selecting the language and application environment the following aspects have to be considered. The language and the environment should:

- Be well known, introduced and used widespread
- Have large desktop PC support
- Be a comfortable development environment
- Enable real-time taskhandling
- Enable the realisation of object oriented concepts to the largest potential possible
- Provide an adequately large development programme library
- Produce extremely concise code, since the size of memory is small in micro environment
- Not require large resource in running time, because it slows down the operation of application
- Be portable if possible, i.e. it should be linked to neither physical architecture nor operation system features

It can be declared that a language that meets all the requirements described above does not exist. JAVA is the language that fulfils the requirements to the largest extent.

4.1 The JAVA Standard Conception

JAVA is the success story of object oriented software development of the latest years [3].

The application of JAVA in mid-range embedded systems is supported by several facts: it is widely used, its realisation on several types of platforms, the wide range of programme library available, its portability and the well realised object oriented concepts.

From the aspect of our examination its originally imperative character, the intensive use of external class-library, the layer to assure platform independence, the demand for the use of the so called JAVA Virtual Machine (JVM) [6] are some drawbacks. These features make JAVA too large, memory demanding and too slow in mid ranged embedded systems. A typical JAVA application layer allocation would be as follows: (see Figure 2).

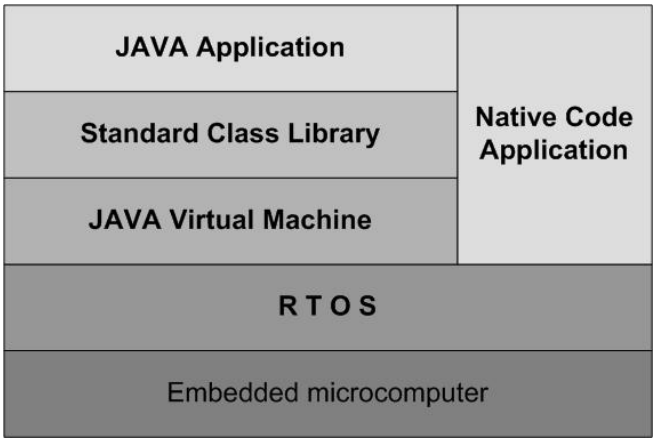

Figure 2
The application levels in case of standard JAVA application with pluged in native code application

As it can be seen in the table, in the case of a standard JAVA application there are too many layers, the process is two slow and quite a large RAM is required for the operation. In this embedded system category this way cannot be followed.

There is another possibility for JAVA application in order to replace the imperative feature, namely the use of Just-In-Time compiler, which translates from JAVA bytecode to immediately executable native code. The advantages of these compilers are, that they save the once translate codes due to reuse, thus similar code sets will not be translated again. As a result, speed will grow, although this requires larger memory size. It implies that this solution cannot be realised in restricted RAM environment.

4.2 A Possible JAVA Concept in Mid-range Embedded System

By giving up certain benefits a compromise can be reached, which enables the application of JAVA programmes in an environment with weak resources. As a first step the JVM concept must be given up, that is the imperative feature. The Just-In-Time compiler solution can neither be used. Instead, a cross developing system must be elaborated, which translates to a cross native code that is the executive code of the target environment, then it links those elements executing the necessary services from Realtime Service Library (RSL), which are needed at the running of the application (see Figure 3).

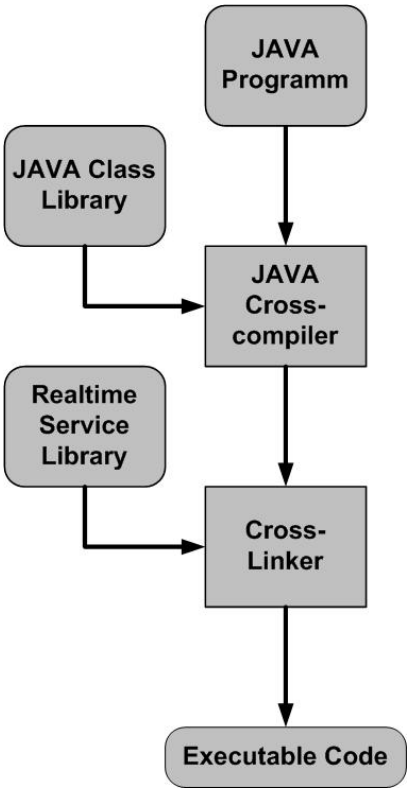

Figure 3
The development process

When creating RSL, the “kernel” part will be taken out from the conventional RTOS, which includes the main process altogether with the programme parts perforce needed for its operation. The parts covering the remaining services and the routines, which serve the required special services, make up the RSL. Accordingly, the application layers loaded into the microcomputers are as follows (see Figure 4).

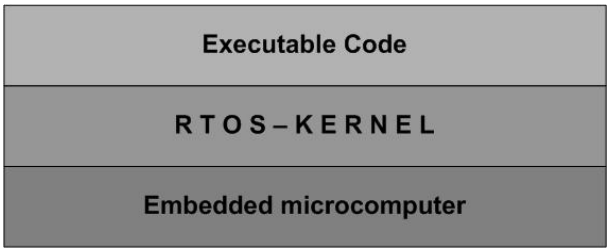

Figure 4
The application levels in case of JAVA application in midrange embedded systems

The solution made with compromise enables mid-range embedded systems to use the large number of existing JAVA programs and products. These programmes make our development work more efficient.

Conclusions

The outstandingly large number of mid-range embedded systems could be developed in assembly or in C so far. The solution outlined in this paper offers a possible alternative solution by separating the conventional RTOS into a Realtime Service Library and a RTOS kernel, which provides a minimal level of services. When splitting the RTOS, the process of assigning each unit to one of the parts and supplementing the RSL with such programme units that are necessary for efficient work, require thorough and in-depth planning. The implementation of this theoretical solution means the next step, which hopefully will prove the above presumption.

References

[1] Peter Marwedel: Embedded System Design, Kluwer Academic Publishers, 2003, ISBN-10 1-4020-7690-8

[2] Alberto Sangiovanni-Vincentelli, Grant Martin: Plattform-Based Design and Software Design Methodoogy for Embedded Systems, IEEE Design & Test of Computers, November-December 2001, pp. 23-33

[3] Robert Orfali, Dan Harkey: Client/Server Programming with JAVA and CORBA, John Wiley & Sons, Inc. New York, 1997, ISBN 0-471-16351-1

[4] Niklaus Wirth: Compiler Construction, Addison-Wesley, 1996, ISBN 0-201-40353-6

[5] Andrew W. Appel: Modern Compiler Implementation in JAVA, Cambridge University Press, New York, Cambridge, 1997, ISBN 0-521-58388-8

[6] Bill Venners: Inside the JAVA Virtual Machine, McGraw-Hill, 1999, ISBN 0-07-135093-4