Real-Time Concepts Implemented in Distributed System Programming Languages

Horia Ciocarlie

Computer and Software Engineering Department,
Faculty of Automation and Computers, “Politehnica” University of Timisoara,
Bd.. V. Parvan 2, 300223 Timisoara, Romania
Phone: +40-256-403285, E-mail: horia@cs.utt.ro

Abstract: The problem of time measuring and managing in distributed systems is important both as an independent problem and for reasons like: the maintenance of the consistency of distributed data, the elimination of duplicated data and the verification of the authenticity of a request.
After presenting the main problems that appear during distributed system design and programming, the paper continues with some issues concerning the domain of real-time programming. At the same time we emphasize the complex relations between real-time, concurrent and distributed programming. Then the requirements for the real-time programming are presented.
Although the paper's subject is based on the analysis of several specialized languages, the presented exemplifications are from Java programming language with RTSJ (Real-Time Specification for Java) extension.

Keywords: real-time programming, embedded systems, distributed systems, concurrent languages, Real-Time Specification for Java

1 Introduction

Most of the modern microprocessors are used in *embedded systems* that control various processes of the real world [1]. The field of applicability is continuously expanding and includes various domains of human activities from telecommunications to automation of fabrication processes.

The embedded systems are more and more connected by computer networks. Thus results the growth of the importance of distributed applications. It is expected that the number of these applications will grow exponentially.

Designing, programming and implementing distributed systems have risen from the beginning new problems [2]:

- the heterogeneity of the processors from the nodes of a network
- the various conventions of data representation
- the synchronization of the activities from the remote nodes
- avoiding the congestion of some communication paths or the network blocking
- maintaining the consistency of the data bases despite of transmission errors or the malfunctioning of processors or communication paths and devices
- the security of data, of transmission and of information exchange.

The distributed systems synchronization is based on a time ordering relation between events. Each computer has its own clock, but clocks can’t be perfectly synchronized. The absence of a global clock makes difficult the determination of the states of the programs that are executing.

Thus, the realization of the embedded real-time systems introduces more difficulties for designers.

2 The Relation between Real-time Programming and Concurrent Programming

Real-time programming includes programs with an execution time behavior, depending on the duration of the program’s component activities.

In common sequential programs [3] the time required by a certain activity has no influence on the program correctness. The execution time of the program is of interest only globally and first of all, from commercial point of view.

A real-time program is in the direct service of an application (or of a reduced number of related applications) from which it receives and/or to which it sends signals. The main feature of the program is its response time, expressing the promptitude in solving the services required by the application. There are **two typical situations** in real time programming:

1 Execution of the operations is required at some specified moments, or at certain time intervals (sampling periods, control intervals). The time intervals between the starting of two consecutive activities are known; so, at programming time, it is possible to verify and, if necessary, to adjust that the duration of an activity doesn’t exceed some allowed limit of time.

2 Execution of some operations is required at unforeseeable moments, depending on the arrival of external signals. To assure the correctness of the program behavior, one can act in one of the following two modes:

   - the delay (if possible) of a signal, until the end of the activity started by the precedent one;
   - the establishing of a minimal time interval between consecutive signals; all activities must fit in this interval.

The active components of a real time program are concurrent processes, which interact in the form of communication and synchronization; so we can say that *real-time programming implies concurrent programming.*

Thus, the real-time programming is the part of concurrent programming in which the triggering times and the execution times of program actions are significant and influent the synchronized activities of parallel processes and even the results correctness.

3 Requirements for Real-time Programming Languages

Like concurrent programs in general, real time programs were written, for long time, only in assembly languages. Three were the most important reasons for this:

1 The response time must be exactly determined; for a lot of applications it must be reduced below some limits, hardly or impossible to achieve in high level languages.

2 The programmer needs direct access to machine hardware (peripheral devices, real time clock, input/output ports, etc.); for a long time, high level languages provided no facilities in this direction.

3 The real time program is also a concurrent program, including its own policy for process management and synchronization. In this context, the supplementary implications of an operating system are disturbing.

Successive improvements in hardware and programming languages changed this situation. Today most opinions agree, that a real time programming language, must be first of all a performing and highly improved language, and this means implicitly, a high level language.

The major problem of most companies is that of developing new programs in a time that is as short as possible. Thus it is important for the production of embedded real-time systems to use the advantages of programming in a high level language, of using middleware or tools and methods that permit an increased productivity without affecting to much the quality of the service (QoS).

Many embedded real-time systems are still developed in C or C++. Writing programs in these languages is more productive then using assembler languages but is far from being optimal. Below are some of the causes:

- the lack of special facilities for distributed and real-time processing
- the mechanism of allocation and especially that of deallocation of the memory are a major source of errors
- C++ is a difficult language, it is hard to be learned and applied.

In conclusion, the embedded real-time software should be realized in high level languages that are dedicated for this domain.

Some basic criteria for the design of a real time language, can be formulated. The most important are [3]:

- To assure the necessary security and readability, real time programs must be first of all structured and modular. Consequently, the programming language must provide the corresponding structures, to describe both actions and data, including abstract data types.
- Real-time programs are often large and complex. So the programming language has to provide facilities for the so called "*extended programming*", this means first of all modularization.
- Real-time programs are concurrent; consequently the programming language must be proper for concurrent programming.
- Real-time systems use often nonstandard input/output devices. To program such devices, the language must provide facilities for direct access to the machine hardware.
- To ensure a maximal reliability of real time systems, the language must provide the handling of errors and exceptions, allowing so a normal functionality after incidents.
- The language must provide means for interrupt handling.
- For some functions, the program needs direct access to the real time. This can be solved by providing some facilities to determine the actual time, to measure a time interval, to delay a process for a certain time period, etc.
- For each implementation of a real time language, the user documentation must contain the execution time for each source statement. These times can be easily determined by the implementation, when designing the compiler (more exactly the object code generator).

As a conclusion, a real-time programming language must be a very performing concurrent language, with supplementary features to access and handle the real-time. It must provide, first of all, basic facilities for concurrent and distributed programming [4]:

- a notation for describing parallel activities (to specify concurrent processes);
- the possibility to create and eventually kill processes;
- efficient means for communication and process synchronization.

4 Concurrent and Distributed Java Programming

Although the language's syntax is based on the syntax of C++, Java is considerably simpler. There have been eliminated many key words and the programmer is aided by a very large developing tools library. Independent functions, global variables or the *goto* instruction don't exist anymore. The pointers arithmetic has also been removed. Thus many programming error sources have been eliminated. This leads to correct, robust and simple programs, an important requirement for distributed systems programming.

Java has network programming facilities as it was especially designed for the Internet [6]. At the same time it satisfies the requirements of distributed and concurrent programming formulated in §1 and §3 [7], as follows:

- the possibility of processes parallel execution (threads)
- higher security
- more ways of synchronization and communication: synchronization primitives, semaphores, mutual exclusion, etc.
- the implementation of *Client/Server* model and of distributed systems communication [8]: sockets, the transmission of messages, remote method invocation (RMI).

5 The Realization of Embedded Real-time Systems in Java

Because of the simplicity and safety [7], and especially because of the low maintenance cost, Java may be a solution for an efficient development of embedded real-time systems. Still, the traditional Java implementations have some important drawbacks [8]:

- the scheduling of Java execution threads is weakly specified
- the language includes the mechanism of garbage collector that is improper for real-time systems
- it allows the allocation of memory in heap, without controlling the type of memory in which the objects are allocated.

A part of these problems have received and satisfactory solving from the definition or Real-Time Specification for Java (RTSJ) [9]. The novelties brought by this system are:

- garbage collector can be replaced by other methods for memory management
- it permits the access to the physical memory
- allocation and the access to the memory are more adequate for real-time systems
- the semantics of execution threads is more rigorous.

In the present there are already RTSJ implementations that permit the utilization of Java in embedded real-time systems, like TimeSys [10] or jRate from Washington University [11].

There are also the first evaluations of these implementations, concerning the performances that can be achieved [12].

In principle, RTSJ extends the Java API and redefines the semantics of some constructors in order to support real-time applications. Here are some of the novelties that are being brought [9]:

- **Allocation of memory**: the classic dynamic allocation (from heap) is extended with *Immortal Memory* and *Scoped Memory*. The objects allocated in *Immortal Memory* have the same life time as the application. Each *Scoped Memory* area has a reference counter for the number of threads that are active in that area. The life time of the objects that are allocated in such an area depends on this counter, as they are deallocated globally when there are no more references to them.
- Extension of the **Java threading model** with two new types of threads, that are specific for real-time: *RealTimeThread* and *NoHeapRealTimeThread* correlated with memory allocation.
- **Schedulable objects**. RTSJ offers an execution scheduling of entities generally enough for implementing the most used scheduling algorithms [4]. Although, the best scheduling policy that matches with RTSJ is that based on execution priorities capable to distinguish between 28 different priorities. The executable entities are accessed by references.

- **Execution asynchronies**. RTSJ provides mechanisms of binding at execution when certain internal or external (asynchronous) events appear. There is also defined the concept of *Asynchronous Transfer of Control* that permits the asynchronous control transfer between executions.
- Access to **real time**. Embedded systems use timers for making some actions at a given moment of time or at periodic time intervals. RTSJ provides two timer types:
  - *One shot timer*: generates an event when the associate time interval expires
  - *Periodic timer*: generates events periodically.

Conclusions

Relying on his experience in implementing concurrent languages, the author discusses specific problems of real time programming, from the view of high level languages. The advent and development of concurrent and real time languages, produced a large diversification of real time programming methology and the application domains: process control, data acquisition, robotics, telecomunications, etc.

The main characteristic of a real time program is its promptitude in solving the desired services (its response time). It depends first of all on the nature of the application and is highly inflenced by the type of pheripherical devices, number and speed of processors, number and complexity of operations and not at least, on the manner in which real time mechanisms are implemented in the programming language.

All these requirements are satisfied by Java programming language with RTSJ extension.

References

[1] M. V. Micea, V. Cretu, L. M. Patcas: Program Modeling and Analysis of Real-Time and Embedded Application, Transactions on Automatic Control and Computer Science, Vol. 49(63), No. 3, Periodica Politehnica, Timisoara, 2004, pp. 207-212

[2] B. Ari: Principles of Concurrent and Distributed Systems, Addison Wesley, 1990

[3] P. Eles, H. Ciocarlie: Programarea concurenta in limbaje de nivel inalt, Editura Stiintifica, Bucuresti, 1991

[4] H.Ciocarlie: The Characteristic Features of a Concurrent Language Implementation in a Distributed Environment, International Conference on Computational Intelligence 2004, Istanbul, pp. 121-123

[5] I. Jurca: Programarea retelelor de calculatoare, Editura de Vest, Timisoara, 2000

[6] D. Petcu, V. Negru: Procesare distribuita, Editura Universitatii de Vest, 2002

[7] I. Athanasiu: Java ca limbaj pentru programarea distribuita, Editura Matrix Rom, Bucuresti, 2002

[8] H. Georgescu: Introducere in universul Java, Editura Tehnica, Bucuresti, 2002

[9] G. Bollella, J. Gosling, B. Brosgol, P. Dible, S. Furr, D. Hardin, M. Turnbull: The Real-Time Specification for Java, Addison-Wesley, 2000

[10] Timesys, Real-Time Specification for Java Reference Implementation, www.timesys.com/rtj, 2001

[11] R. Meersman, Z. Tari: The Design and Performance of the jRate Real-Time Java Implementation, Springer Verlag, 2002

[12] A. Corsaro, D. C. Schmidt: Evaluating Real Time Java Features and Performance for Real-Time Embedded Systems, Proc. Eighth IEEEE Real-Time Technology and Applications Symp., Sept. 2002