ISSN 0005-1179, Automation and Remote Control, 2011, Vol. 72, No. 5, pp. 1134–1139. © Pleiades Publishing, Ltd., 2011.
Original Russian Text © S.N. Zyl', 2008, published in Avtomatizatsiya v Promyshlennosti, 2008, No. 10, pp. 6–9.

# Real-time Java Has Become Already Real

**S. N. Zyl'**

*SWD Embedded Systems, St. Petersburg, Russia*

Received March 24, 2008

**Abstract**—Enhanced complexity of the designed real-time computer systems requires that the developers seek for efficient methods and tools to solve the posed problems. The most obvious and simple approach is to employ technologies that have demonstrated their applicability; such technologies should be adapted to the requirements of mission-critical systems. This paper is dedicated to specific features in real-time Java technology. They are considered simultaneously with characteristics of real-time computer systems.

**DOI:** 10.1134/S0005117911050237

Software developers for real-time systems face difficulties being natural in any field of software, notably, growing complexity of ready-made product functionality and decreasing the time required for ready-made product to enter the market. This is the case against the background of permanent or even reduced human resources and financial funds allocated for the system design. Way out of this situation is well-known, i.e., one should use modern methods and tools of design and development. However, real-time systems' developers can not afford the luxury of involving untested tools and techniques. Therefore, the beginning of certain technology usage in real-time systems design serves as a touchstone of its maturity. Java may be found among such real-time technologies.

Probably, some readers would be surprised to see the term "real-time" and Java mentioned together. Indeed, there is nothing strange in such state of the things; Java has proved its applicability and efficiency (as a platform, as a programming language, as a technology). It is independent of hardware platform or operating system. Moreover, efficient model of the components and model of distributed calculations have been created for Java, together with interacting technology for database management systems and embedding technology for devices with limited resources, as well as many other things. Needless to say about numerous ready-made freeware codes. Of course, it would be beneficial to utilize this "abundance" in real-time systems.

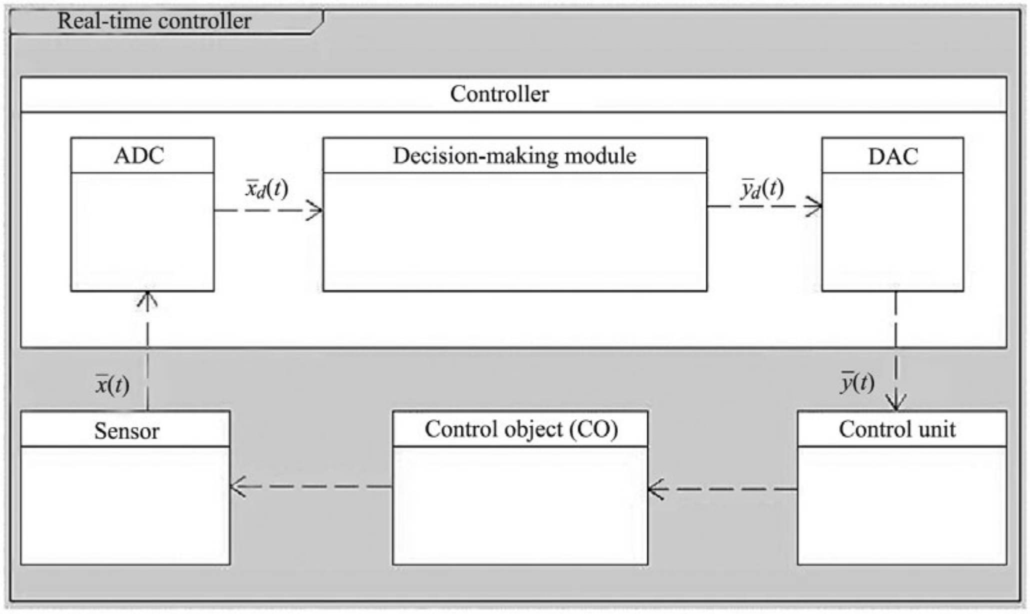
The beginning of real-time Java technology dates back to 1999 when National Institute of Standards and Technology (NIST) published the Requirements for Real-time Extensions for the Java Platform. Formed by interested enterprises and organizations, the working group issued Real-Time Specification for Java (RTSJ) standard. It was the base of real-time Java.

First, recall what are specific features of real-time computer systems and what were the obstacles to use Java platform in them. Let us address the circuit (see the figure below); it demonstrates classical real-time system, i.e., real-time controller.

The following elements of a certain real-time system are represented by the figure:

—control object (CO); actually, the customer is interested in control object. For instance, the latter may be antilock brake system (ABS) in automobile, gas turbine engine, assembly conveyor of the factory, etc.;

—controller; this component "knows" what CO should do for efficient execution of the corresponding functions (depending on certain factors). Computers often serve as decision-making



Real-time controller circuit.

modules in controllers to manage complicated CO. In this case, digital transformation of the signal coming from the sensor requires analog-to-digital converter (ADC); on the other hand, one needs digital-to-analog converter (DAC) to transform computational results to electric-type control signal;

—sensor; it is used by controller to "know" what are the current activities of CO, as well as what are the states of the factors affecting CO operation. As a rule, the sensor is placed under the action of the measured factor and transmits a certain electric signal to the controller (corresponding to the factor's value). Possible examples are pressure sensors, temperature sensors, smoke detectors, position sensors, etc.;

—control unit (CU); the latter is employed by the controller to operate CO. In other words, CU informs CO what actions (and when) should be performed, thus adjusting the behavior of CO. Common types of CU are represented by different drives, e.g., stepping motors, electromagnetic valves and so on.

Thus, what is the operating principle of real-time system having the circuit described by the figure above? The sensor reads parameters of CO and transfers the corresponding electric signal to the controller. Using ADC the signal is digitized and transmitted to PC via a certain input/output device (for instance, RS-232 serial port). Software receives the data and makes necessary computations based on it; the program may either involve additional information (data tables, operator commands, etc.) in computations. After that, input/output device passes the command to CU. In general case, digital command through DAC is transformed to a certain electric signal which activates CU. And finally, CU acts on CO.

In practice, CO can have either complex structure or complicated functions performed. In addition, numerous sensors may exist within the system; not all of them measure parameters of CO. In other words, the sensors can also measure parameters of the ambient environment (where CO operates), as well as parameters of the objects being affected by CO. For instance, controller of the ventilation and air-conditioning system should know the ambient temperature, while machine controller needs information on the completed workpiece.

No doubt, computer-based module of decision-making may be rather sophisticated. In particular, it can represent a computing cluster (instead of a single PC), include workstations of the operators, keep necessary parameters and initial data in database, as well as provide guaranteed level of reliability using "hot standby" computers, and so on. Among other things, it should be mentioned that real-time computer systems without human operator are referred to as automatic control systems, while those ones involving human operator are called by automated control systems. Key feature of any real-time system lies in that obtaining correct computing result seems of crucial importance in due time. Having been obtained late, correct result appears useless. An obvious example is ABS in automobile.

Now, having a clear view of real-time computer system, one may speak about mechanisms to-be-implemented in software, as well as about abilities provided by real-time Java technology to the program developers.

It is possible to read data from sensors following two methods, *viz*, periodic sampling and event processing. In the first case one needs timers used to activate sampling procedure for sensors (after specific time intervals). In the second situation data from the sensor enters input/output device and hardware interrupt is generated; it should be processed by software. Surely, the controller may simultaneously utilize both methods. Having been received from the sensors, the data should be processed according to the given algorithm. Note that processing time is limited; thus, the program or thread responsible for processing procedure must not be inferior to auxiliary processes or threads (an example of auxiliary problem consists in periodic data backup). Again, the commands may be produced either at certain periods or following specific event.

Summarizing the aforesaid, it could be underlined that an important software mechanism of real-time systems are timers and time service in general. In this sense, standard Java technology has two disadvantages, as follows:

—insufficient accuracy of time mechanism. Classical Java operates time with the accuracy up to 1 ms; in contrast, real-time software operates time intervals specified in micro- or even nanoseconds. Therefore, real-time Java technology is described by two fields, i.e., 64-bit field (used to store the number of seconds) and 32-bit field (used to store the number of nanoseconds in the current second);

—clock in standard Java does not ensure monotonic translatory evolution of time. RTSJ specification defines the clock with strictly monotonic time (it is called by "real-time clock").

As one could see, another important real-time mechanism is hardware interrupt processing. In real-time Java, this task is fulfilled by aperiodic events' handlers. The events in real-time Java may be associated with hardware interrupts, POSIX signals or timer operation. A single event may be processed by several handlers; similarly, a handler may be used to process many events.

Probably, the reader has focused the attention on the fact that timer operation represents a certain event, as well; it is processed by some handlers. In other words, reading data from sensors is anyway performed by event handlers (the difference is that these events may be generated by the timer after specific intervals or by hardware in aperiodic way). For the view of real-time Java, aperiodic events' handlers are particular case of dispatching objects. In addition to handlers, these objects include:

—thread (we mean common Java threads);

—real-time thread. Sometimes this type of the dispatching objects are called by soft real-time threads;

—no-heap real-time thread. It is also referred to as hard real-time systems.

We will discuss the differences between real-time threads and common ones in the sequel. For the time being, let us pay attention to the function of the threads in the considered real-time system; recall they serve to make necessary computations and (as the need arises) to transmit control command to CU. Two special features should be pointed out here. The first one consists in that the thread should be given the control so as it could start the computations (perhaps, it is also required to "take away" the control from less important thread). The second special feature is to eliminate possibility of "depriving" the processor from the thread engaged in valuable calculations in the favor of less important threads.

In standard Java, the problem of thread termination (transfer of control to another thread) is solved, to a certain degree, on the basis of the so-called "interruptable methods." This approach has been extended in real-time Java technology by suggesting the mechanism of Asynchronous Transfer of Control (ATC). Moreover, the mechanism of asynchronous abandoning of the threads has been added; it allows for application-safe termination of unnecessary thread.

In the view of competition for processor resources, advantages of the threads are given by priority-based dispatching. Of course, standard Java provides 10 priority levels for the threads. It appears insufficient for modern real-time software to use just 10 priorities; however, the major problem consists in a different thing. The matter is that classical Java has no guarantee that higher-priority thread would displace the one with lower priority. The reason is that (instead of regulating how Java priorities affect the parameters of "original" threads of operating system) classical Java leaves the final decision to the discretion of JVM developers. With the aim of eliminating these negative aspects, real-time Java includes at least 28 additional priority levels and a scheduler; the latter performs displacement-type dispatching based on fixed priorities. Opportunity to create schedulers involving alternative (non-prioritized) metrics to make dispatching decisions, as well as involving other dispatching algorithms (e.g., Earliest-Deadline-First) is provided *de bene esse*.

Concurrent threads (even those ones having different levels of priorities) may be used by the same data arrays (required for operation). A problem of synchronization arises when several threads have simultaneous access to any common resource. Within the framework of classical Java, this problem is solved through the so-called synchronized methods. Nevertheless, using the mechanisms of synchronization generates another problem, i.e., priority inversion phenomenon (being well-known among real-time software developers). To secure against priority inversion, scheduler in real-time Java is supplied with a certain parameter referred to as priority inversion secure policy. In particular, two types of the policy are supported:

—priority inheritance. In this case the thread which has captured the common resource is executed with priority being the highest among priorities of the threads *waiting* for release of the resource in question;

—priority ceiling protocol. Here the thread which has captured the common resource is executed with priority being the highest among priorities of the threads that *may* use the resource in question.

Now, let us understand why real-time Java includes several types of the threads, notably, common threads, real-time threads and real-time non-heap threads. In addition, we discuss their differences. Such "variety" of the threads has been developed to protect those threads engaged in urgent work from being eliminated by "garbage collector" (GC).

Generally speaking, garbage collection has always been posed by Java developers as a strong feature of the platform. This is true since Java enables solving the perennial problem of C++ (memory losses due to the fact that program developers forget to delete useless objects in the program). In Java, the memory required for the objects is automatically allocated (following their initiation) in the memory space controlled by Java Virtual Machine (JVM) and called by heap. In some handbooks on Java technology the authors directly use the term of "heap;" this is memory space used to initiate the objects. In certain intervals or as soon as the heap is filled up, the objects run the thread of "garbage collector;" it deletes the objects which have no pointers in the program.

A certain problem for real-time system consists in random character of garbage collection. In other words, the moment when "garbage collection" is required can not be assessed, while the number of objects without pointers (to-be-eliminated during every case of GC operation) could be hardly foreseen. Perhaps, the most upsetting thing is that the thread of "garbage collector" displaces the rest threads of classical Java. Such situation either has some grounds—the "heap" is overfilled and further operation of Java-program appears impossible without proper "garbage collection."

To fight against unpredictable garbage collection, real-time Java technology has been provided with two additional types of memory spaces (besides the "heap"):

(1) immortal memory, i.e., memory space outside the "heap;" having been initiated in it, the objects are eliminated only when the program is completed. Immortal memory is available for dispatching objects of all the types;
(2) scoped memory, i.e., memory spaces allocated outside the "heap" for those objects which have known lifetime. Each object in scoped memory possesses a certain counter of the pointers for this object within the program. When the counter's value is changed from 1 to 0, the object's destructor is activated simultaneously releasing the memory kept by the object under consideration.

No-heap real-time threads do not process objects from the "heap;" in other words, "garbage collector" would not be able to affect the thread. Contrariwise, soft real-time threads have weaker constraints. Sometimes, this leads to situations when the threads under consideration depend on GC (yet, to a smaller degree than common threads).

One should mention to the point that data exchange between real-time threads and common threads represents by no means an easy problem. Two special classes (for buffered data exchange) are provided in real-time Java to solve it. The first class is a certain buffer with synchronized write operation (while read operation is declared as non-synchronized). The second class has reversed situation with these operations. A small feature of the second class consists in that the reading thread may be informed about data availability through asynchronous event.

It seems important to focus the reader's attention on the fact that a certain approach called Real-Time Garbage Collector (RTGC) exists as a supplement to RTSJ specification. Different versions of real-time Java suggest different RTGC technologies; nevertheless, the common underlying idea of these approaches is to protect even standard threads against the "self-will" of GC (via alternative methods of garbage collection). This allows the program developer to avoid the usage of both real-time threads and mechanisms of scoped and immortal memory (note the latter appear rather nontrivial). Well-known approaches to RTGC are proposed within famous realizations of real-time Java, *viz*, JamaicaVM (by Aicas company), Java Real-Time System (Sun Corporation) and WebSphere Real Time (IBM Corporation).

And finally, the authors would like to mention access mechanism for physical memory as a crucial innovation provided by real-time Java. The necessity to operate physical memory directly from Java applications was observed long ago. In the first place, it was connected with shared memory (as one of the most popular mechanisms of interaction among the processes). In classical Java, to access the shared memory program developers can utilize Java Native Interface (JNI), i.e., technology of using API of the operating system directly from Java code. This makes programming complicated. Access mechanisms for physical memory enable seamless access to the shared memory directly from Java code. Moreover, they allow for involving DMA mode and device registers (the ones linked to the memory). Thus, from now on Java language may serve for creating the drivers for certain devices.

Today, real-time Java represents rather mature technology being applied in different computing systems with tough requirements to reliability and predictability of the software. Websites of the companies designing real-time Java software contain references to many application examples, notably, AN/FPS-85 aerospace radar system (the version that has been modified by ITT Industries Corporation), CSEM PocketDelta industrial miniature robots, multimedia systems for news and financial information used by PLC company (Reuters Group), as well as control systems for DDG 1000-class ships, etc. On the other hand, real-time Java technology keeps on moving; for instance, Distributed RTSJ specification has been developed. It allows for integrating real-time Java applications in software packages with network distribution. Another important direction consists in the design of optimization methods for resources used by multicore processing units in real-time Java.