Javolution - Solution for Real Time Embedded System

Ms. Rashmi P. Sonar
Department of Computer Science & Engineering
Prof Ram Meghe College of Engineering & Management, Badnera-Amravati

Ms. Rani S. Lande
Department of Computer Science & Engineering
Prof Ram Meghe College of Engineering & Management, Badnera-Amravati

***Abstract* - Embedded systems are highly used in every field. Embedded system often required to provide Real-Time response. As embedded systems become more and more complex, and the time to market becomes shorter; there is a need in the embedded systems community to find better programming languages that let the programmers develop correct code faster: The programming languages used today typically C and/or Assemblers-are just too error-prone. The Java technology has therefore gained a lot of interest from developers of embedded systems in the last few years [1].**

To use Java in Real-Time/ Safety-Critical systems calls for significantly green predictable code execution. Techniques to achieve this include Ahead-Of-Time compilation, incremental Garbage Collection and the use of RTSJ-compliant Virtual Machine. However those are not enough as time-predictability can without trouble be ruined through using the same antique library (lazy initialization, array resizing, and so forth.). To reap actual time predictability one should have a time deterministic library. This paper introduces such open deliver library known as Javolution suitable for embedded or server-aspect applications and used for safety important application global [2]. This paper also presents an overview of the most recent and important studies in the area of real-time Java computing. We will study and analysis of Java variations for Real-Time systems.

Index Terms - Embedded Systems, Real-Time Operating Systems, Real- Time Java, JIT Compiler, RTSJ.

I. INTRODUCTION

Dynamic adaptive conduct for flexibility and unexpected change within the runtime environment and Real-Time necessities for predictability and determinism of software's response times are common needs of these days software program.

Real-Time systems are software systems whose correctness relies upon on each logical and temporal element. The most important properties of real-time systems are their predictability and the determinism in their execution time. In the start devoted hardware was created for real-time systems; however with the advances in computer hardware, actual-time worries targeting the software layer. Real-time operating systems had been designed to offer deterministic hardware get admission to time; real-time scheduling algorithms were elaborated to avoid deadlines missal; and programming language have been designed or extended in order to permit programmers to develop real-time systems with high-degree abstractions.

A. Real-Time System

1) Definition and Concept: The Real-time system is one that must perform operations within rigid timing constraints i.e., processing must be completed within the defined constraints otherwise the system will fail. The correctness of real time system does not depend only on the logical correctness but also on the time it takes to produce the result [1].

Real-time systems have well defined, fixed time constraints. There are **two main types of real-time systems:** Hard Real-Time System (HRT), Firm or Soft Real-Time System (SRT).

In Hard Real-Time System requires that fixed deadlines must be met otherwise disastrous/ catastrophic situation may arise whereas in Soft Real-Time System, missing an occasional deadline is undesirable, but nevertheless tolerable. System in which performance is degraded but not destroyed by failure to meet response time constraints is called soft real time systems.

a) Soft Real Time system

- Deadline overruns are tolerable, but not desired.
- There are no catastrophic consequences of missing one or more deadlines.
- There is a cost associated to overrunning, but this cost may be abstract.
- Often connected to Quality-of-Service (QoS).

b) Hard Real Time system

- An overrun in response time leads to potential loss of life and/or big financial damage
- Many of these systems are considered to be safety critical.
- Sometimes they are "only" missioning critical, with the mission being very expensive.
- In general there is a cost function associated with the system.

c) Firm Real Time system

- The computation is obsolete if the job is not finished on time.
- Cost may be interpreted as loss of revenue.
- Typical examples are forecast systems.

2) Periodic Sporadic or Aperiodic: Depending on the tasks arrival pattern, timing constraints may be periodic, sporadic or aperiodic. Periodic tasks are invocated within regular time intervals, while arrival times in sporadic and periodic tasks are unknown; however, the time interval between two releases of an aperiodic task is only known to be

978-1-5386-2599-6/18/$31.00 ©2018 IEEE

greater or equal to zero, while sporadic tasks have a time interval between two releases always greater than or equal to constant.

B. Embedded System

An Embedded System: It is a combination of hardware and software to perform a specific task. An embedded computer is frequently a computer that is implemented for a particular purpose. In contrast, an average PC computer usually serves a number of purposes: checking email, surfing the internet, listening to music, word processing, etc... However, embedded systems usually only have a single task, or a very small number of related tasks that they are programmed to perform.

C. Operating System

An **Operating system** (**OS**) is basically an intermediary agent between the user and the computer hardware (for hardware functions such as input and output and memory allocation). It is a collection of software that manages computer hardware resources and provides common services for computer programs. The operating system is an essential component of the system software in a computer system. Operating systems can be found on almost any device that contains a computer such as cellular phones and video game consoles, supercomputers and web servers.

1) Purpose of an Operating System:

- It's a resource allocator i.e. manages the computer's resources (hardware, abstract resources, software).
- It is also used to control programs to prevent errors and improper computer use.
- It is interrupt driven.
- Users and Processes access the Computer's resources through the Operating System

2) OS Efficiency Functions:

a) Program Execution

An OS has to be able to load a program into memory and run that program. Also, this program needs to be able to end execution, whether normally or abnormally.

b) Resource Allocation

- Multiple users or jobs require that resources be allocated to each one.
- Some resources have a special allocation code, while others may have more general request and release code.

c) Input/Output (I/O) Operations

- A running program may require I/O in the form of a file or an output device.
- Due to a security or efficiency need, users often do not control I/O, so the OS must provide the means for I/O.

d) Protection and Security

- It should not be possible for one process to interfere with other processes or the OS itself. This means that all access to system resources is controlled.
  - Protecting the system from outside threats is done through authenticating users, and to defending external I/O devices.
  - Log files can help detect and track attempts of outside infiltration.

e) File- system Manipulation

- Programs need to able to read and write to files, also delete, rename, and remove them.
- It is also necessary that files have permission functionality to provide additional security.

f) Communication

- Often time's one process must communication with another process.
- Communication may be between the processes of a single computer, or computers connected via a network.
- Communication may be implemented via shared memory or through message passing.

g) Error Detection

- An OS should be constantly looking for errors, and know how to handle them.
- For each error, the OS should have an action to take to ensure correct and consistent computing.
- Debugging facilities enhance user and development usage of the system.

3) Types of Operating Systems:

a) Real-time: A Real-time operating system is a multitasking operating system that aims at executing real-time applications. Real-time operating systems often use specialized scheduling algorithms so that they can achieve a deterministic nature of behavior. The main objective of real-time operating systems is their quick and predictable response to events. They have an event-driven or time-sharing design and often aspects of both. An event-driven system switches between tasks based on their priorities or external events while time-sharing operating systems switch tasks based on clock interrupts.

b) Multi-user: A multi-user operating system allows multiple users to access a computer system at the same time. Time-sharing systems and Internet servers can be classified as multi-user systems as they enable multiple-user access to a computer through the sharing of time. Single-user operating systems have only one user but may allow multiple programs to run at the same time.

c) Multi-tasking vs. single-tasking: A multi-tasking operating system allows more than one program

to be running at the same time. A single-tasking system has only one running program. Multi-tasking can be of two types: *pre-emptive* and *co-operative*. In *pre-emptive* multitasking, the operating system slices the CPU time and dedicates one slot to each of the programs. *Cooperative* multitasking is achieved by relying on each process to give time to the other processes in a defined manner.

d) Distributed: A distributed operating system manages a group of independent computers and makes them appear to be a single computer. The development of networked computers that could be linked and communicate with each other gave rise to distributed computing. Distributed computations are carried out on more than one machine. When computers in a group work in cooperation, they make a distributed system.

e) Templated: In an O/S, distributed and cloud computing context, templating refers to creating a single virtual machine image as a guest operating system, then saving it as a tool for multiple running virtual machines. The technique is used both in virtualization and cloud computing management, and is common in large server warehouses.

f) Embedded: Embedded operating systems are designed to be used in embedded computer systems. They are designed to operate on small machines like PDAs with less autonomy. They are able to operate with a limited number of resources. They are very compact and extremely efficient by design.

B. Embedded Real – Time Operating System

An Embedded Real Time System possesses the characteristics of both an embedded system and a real-time system. The embedded systems are resource limited, the memory capacity and processing power in an embedded system is limited as compared to a desktop computer and response time is one of the most important requirements. Some embedded systems run a scaled down version of operating system called Real Time Operating System (RTOS).

A RTOS is a multitasking operating system designed to meet strict deadlines (Real-Time). Many embedded systems require software to respond to inputs and events within a defined short period. RTOS is designed to control an embedded system and deliver the real-time responsiveness and determinism required by the controlled device. Applications run under the control of the RTOS, which schedules allocated CPU time.

The choice of an operating system is important in designing a real time system. Designing a real time system involves choice for proper language, task portioning and merging and assigning priorities using a real time scheduler to manage response time. The depending upon scheduling objectives parallelism and communication may be balanced. The designer of scheduling policy must be determine critical tasks and assign them high priorities. However, care must be taken avoid starvation, which occurs when high priority tasks are always ready to run.

Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources. A scheduling algorithm is a set of rules that determines which task should be executed in any given instance. Due to the tasks criticality scheduling algorithms should be timely and predictable.

1) Basic Requirements of Scheduler in RTOS: The following are the basic requirements of Scheduler in RTOS.

- Multitasking and Pre-emptable.
- Dynamic Deadline Identification
- Predictable Synchronization
- Sufficient Priority levels
- Predefined latencies

a) Multitasking and Preemptable: To support multitasks in real time applications a RTOS should be multitasking and preemptable. The Scheduler should be able to preempt any task in the system and give resource to the task that needs it.

b) Dynamic Deadline Identification: In regulate to achieve preemption; an RTOS should be able to dynamically identify the task with the earliest deadline. To handle deadlines, deadline information may be converted to priority levels that are used for resource allocation. It is also employed for lack of a better solution and error less.

c) Predictable Synchronization: Multiple threads to communicate among themselves in a timely fashion, predictable inter-task communication and synchronization mechanisms are required. Predictable synchronization requires compromise. Ability to lock or unlock resources is the ways to achieve data integrity.

d) Sufficient Priority levels: When using prioritized task scheduling, the RTOS must have a sufficient number of priority levels, for effective implementation. Priority inversion occurs when a higher priority task must wait on a lower priority task to release a resource and turn the lower priority task is waiting upon a medium priority task. Two workarounds in dealing with priority inversion, namely priority inheritance and priority ceiling protocol, need sufficient priority levels.

e) Predefined latencies: The timing of system calls must be defined using the following specifications:

- Task switching latency or time to save the context of a currently executing task and switch to another.
- Interrupt latency or the time elapsed between the execution of the last instruction of the interrupted task and first instruction of the interrupt handler.
- Interrupt dispatch latency or the time to switch from the last instruction in the interrupt handler to the next task scheduled to run.

The objective of real-time task scheduler is to guarantee the deadline of tasks in the system as much as possible when we consider soft real time system. To achieve this goal, vast

researches on real-time task scheduling have been conducted. Mostly all the real time systems in existence use preemption and multitasking.

We provide an introduction to certain system concepts that carry a lot of significance in embedded real-time systems.

- *Periodic Tasks* - The period of a task is the rate with which a particular task becomes ready for execution. Periodic tasks become ready at regular and fixed intervals. Periodic tasks are commonly found in applications such as avionics and process control accurate control requires continual sampling and processing data.
- *Deadlines* - All real time tasks have deadline by which a particular job has to be finished. There are scheduling algorithms designed to allow maximum tasks to meet their deadline.
- *Laxity* - Laxity is defined as the maximum time a task can wait and still meet the deadline. It can also be used as a measure of scheduling necessity.
- *Jitter* - It is defined as the time between when a task became ready and when it actually got executed. For certain real time systems there is an additional constraint that all the tasks should have minimum jitter.
- *Schedulability* - A given set of tasks is considered to be schedulable if all the tasks can meet their deadline. In certain on-line scheduling algorithms a new task is subject to schedulability test, wherein it is verified that the new task is schedulable along with the already existing tasks. If the task is not schedulable the task is not permitted to enter the system.
- *Utilization* - It is the factor giving a notion of how much CPU is utilized by a given set of tasks.

Meeting deadlines, achieving high CPU utilization with minimum resource and time utilization are considered as the main goals of task scheduling.

C. Real Time Scheduling Paradigms

Real time scheduling techniques can be broadly divided into two categories: Static and Dynamic.

1) Static scheduling: Static algorithms assign all priorities at design time (before the tasks are entered into the system based on statically defined criterion like deadline, criticality, periodicity etc.). All assigned priorities remain constant for the lifetime of a task. The advantage of using static scheduling procedure is that it involves almost no overhead in deciding which task to schedule. Static scheduling of tasks in embedded real-time systems often implies a tedious iterative design process. The reason for this is the lack of flexibility and expressive power in existing scheduling framework, which makes it difficult to both model the system accurately and provide correct optimizations. This causes systems to be over constrained due to statically decided rules of procedures.

2) Round-robin method: The simplest of static scheduling procedures is round-robin method. The tasks are checked for readiness in a predetermined order with ready to execute task getting a CPU slice. Each task gets checked for schedulability once per cycle, with scheduling time bound by execution time of other tasks. Apart from simplicity this method has no advantages. The major disadvantage being that urgent tasks always have to wait for their turns, allowing non urgent tasks to execute before the urgent tasks. Also polling tasks for schedulability for readiness is not a good procedure at all. This type of scheduling works well in some simple embedded systems where software in the loop executes quickly and the loop can execute repeatedly at a very rapid rate.

3) Dynamic Scheduling: Dynamic algorithms assign priority at runtime, based on execution parameters of tasks. In a dynamic scheduling policy the tasks are dynamically chosen based on their priority dynamically, generally from ordered prioritized queue. The priorities can be assigned statically or dynamically based on different criterions like, deadline, criticality, periodicity etc. Dynamic scheduling can be *preemptive* or *non-preemptive*.

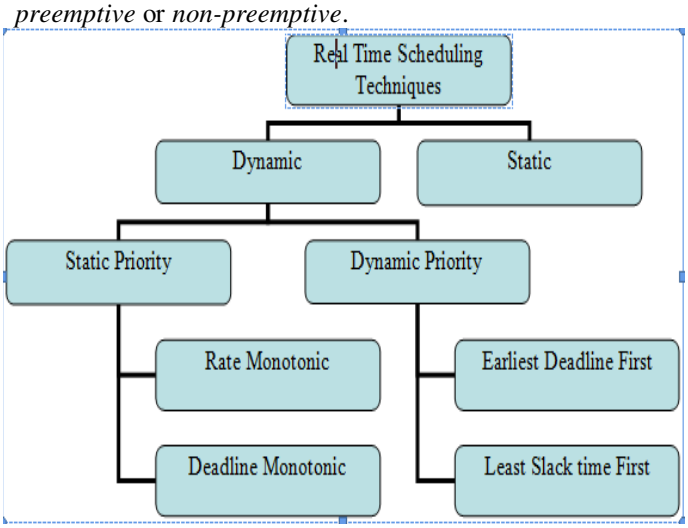

Fig 1: Types of Real Time Task Scheduling Techniques

RM (Rate Monotonic) and DM (Deadline Monotonic) are examples of dynamic scheduling with static priority EDF (Earliest Deadline First) and LST (Least Slack Time First) are examples of dynamic scheduling with dynamic priority. EDF and LST algorithms are optimal under the condition that the jobs are preemptive, there is only one processor and the processor is not overloaded. But the limitation of these algorithms is, their performance decreases exponentially if system becomes slightly overloaded [.

The scheduling is considered as *on-line*, if scheduler makes scheduling decision without knowledge about the task that will be released in the future. In the online case, the schedulability of the task set is analyzed at run time, whereas in the offline case, it is performed prior to run time.

In principle scheduling algorithms may be *preemptive* or *non preemptive*. In preemptive approaches a running task instance may be stopped (preempted) at any time and restarted (resumed) at any time later. Any preemption means some delay in executing the task instance, a delay which the RTOS has to take care of as it has to guarantee respecting the

deadline. In case of non-preemptive scheduling a task instance once started will execute undisturbed until it finishes or is blocked due to an attempt to access an unavailable exclusive resource. Non preemptive approaches result in less context switches (replacement of one task by another one, usually a very costly operation as many processor locations have to be saved and restored). This may lead to the conclusion that non preemptive approaches should be preferable in real-time scheduling. However, not allowing preemption imposes such hard restrictions on the scheduler's freedom that for most non-static cases predictable real-time scheduling solutions with an acceptable processor utilization rate are known only if preemption is allowed.

II. Real -Time Programming Languages

Using traditional technology and methodologies in real-time improvement is highly-priced and difficult. Thus, for the reason that early days of pc programming field, many programming languages had been used to broaden real-time applications. These languages assist the expression of timing constraints and deterministic behaviour in as a minimum certainly one of 3 unique ways:

- Eliminating constructs with indeterminate execution times,
- Extending existing languages, or
- Being constructed jointly with an operating system.

The most important requirement for real-time programming languages is the guarantee of predictable, reliable and well timed operation. For this cause, each software interest should be expressible within the language through time-bounded constructs; subsequently, its execution timing constraints can be analyzable. In addition, a real-time language should be reliable and robust and modularity. Process definition and synchronization, interfaces to get access hardware, interrupt handling mechanisms and error handling facility are also desirable features for real-time languages.

Assembly, procedural and object-oriented languages are most common general-purposed languages used for developing real-time systems. Despite of the lacking of most the high-level language features (such as portability, modularity and high-level abstractions), assembly languages provides direct access to hardware and an economic execution. Procedural languages, such as C and FORTRAN, BASIC, Ada and Modula extensions, offer desirable properties of real-time software, like versatile parameter passing mechanisms, dynamic memory allocation, strong typing, abstract data typing, exception handling and modularity. C++ and real-time extensions for Java are examples of object-oriented languages used in real-time development, which benefits from some procedural languages advantages and adds higher level programming abstractions. Even though these abstractions increase developer's efficiency and code reuse, mechanisms underlying them may introduce unpredictability and inefficiency into real-time systems.

A. Java versus Real-Time Systems

This section presents a brief overview of the Java Technology. In the first subsection we introduce Java and its main features. Then, in a second subsection, we show why it is standard form is not suitable for real-time applications.

1) An Overview of Java

Java technology was designed by Sun Microsystems in 1995 and consists of the Java language definition, a definition of the standard library and the definition of an intermediate instruction set, along with an accompanying execution environment. Originally, Java was created to facilitate the development of networked devices small embedded systems, but due to its portable and flexible capabilities, Sun released it to the general public for Internet and high –level interface applications development. Though the syntax of the Java's programming languages is based on C/C++, Java was designed to eliminate some error-prone features of these languages, such as pointer arithmetic, unions, goto statements and multiple inheritance, improving developer's productivity.

Java introduced a different execution model. First, Java programs are translated into a machine-independent byte-code representation. Then, this byte-code can run in any device which implements the Java Virtual Machine (JVM). JVM is a software system which understands and executes byte-code instructions. In the first implementations of the Java Virtual Machine, those instructions were interpreted, but for performance issues other translation techniques including ahead-of-time and just-in-time (JIT) compilation were included into later implementation.

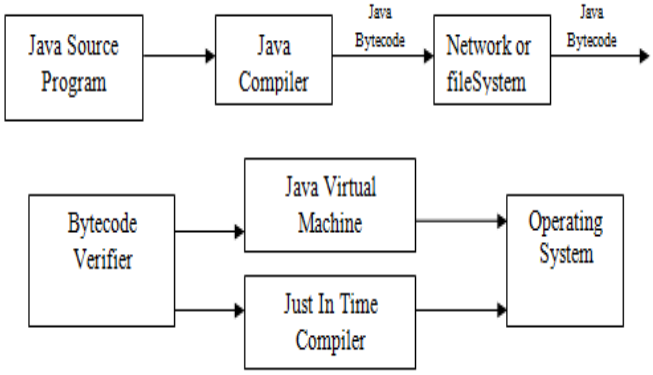

Fig 2. Simple Diagram of Java Program Execution and JVM

One may want to argue that the power of Java is its comprehensive standard library which includes components for networking, graphical user interfaces, XML processing, logging, database access and many other areas. These components are nicely examined, standardized (through the JCP) and available for any conforming implementation. The problem is that those additives had by no means been meant to be used safety critical structures. Even to this day, Sun Java license states explicitly that Java should not be used for the operation of "Nuclear facility".

Java has grown to be one of the maximum popular standard cause languages. This recognition is in element because of its portability, reusability, Security capabilities, ease of use, robustness, rich API set and automatic memory management. Java has many advantages over traditional languages for programming. However, the identical Garbage Collector that eases development is one of the predominant reasons why Java become not used to design critical, embedded and real-time application. Indeed, garbage collection introduces unpredictable execution times. As a, end result unique solutions were designed to improve the determinism of traditional java.

Java is a programming technology which changed into prolonged to be able to provide the timeliness required by way of real-time systems. In its standard form, Java provides numerous shortcomings which avoided its use in the format of real-time structures, along with garbage collection, dynamic lazy loading and unfastened scheduling mechanisms. The Real-Time Specification for Java is an extension to Java which adds real-time programming constructs and constraints to the Java environment.

2) Real-Time Java

Due to its interplay with the real-world, some real-time systems have to be dynamically adaptive, i.e. they need to be able to being modified and up to date at runtime. Software runtime versions may be parametric or compositional. Parametric means modification within the device variables; while compositional specify addition or elimination of device components. Many techniques were developed to guide software runtime version, collectively with aspect-oriented programming computational mirrored image. Service-Oriented Architectures (SOA) and Component –Based Software Engineering (CBSE) are a few other paradigms for the development of dynamic adaptive structures that are becoming very famous.

The need to separate SOA mechanism from company or functional code originated the idea of service-oriented component models, a programming version in which components are used to implement services. Some works were developed in the utility of dynamic adaptive systems strategies in real-time software program. Most of them are targeting Real-Time SOA or in Real-Time CBSE, however only some try to cope with service-orientated component models.

3) Java issues in RT Applications

Java has several features which might be ideal for developing real-time applications; but, in its standard form, Java is not well-suited for it. Some of the motives why Java is inadequate for the improvement of real-time software programs are:

a) Memory Footprint: Standard JVM's desires as a minimum ten of megabytes in memory, what is not good enough for embedded systems. Solutions addressing this problem are, as an instance, the Java 2 Micro Edition [J2ME], IVM and JVM hardware implementations. The formers are appreciably limited compared to the standard API, while the latter is a platform-specific solution.

b) Performance and Execution Model: Byte-code interpretation reduces the overall performance of Java applications. In order to remedy this problem, JIT compilers had been designed to compile Java byte-code into native code at run-time. However, running a compiler at run-time, except requiring a large amount of memory, raises scheduling issues, what implies in latency and lack of determinism.

c) Scheduling: Java defines a very loose behavior of thread and scheduling. Threads with higher priority are executed in preference to threads with lower priority. However, low priority threads can preempt high priority threads. Although this protects from starvation in general purpose applications, it violates the precedence property required for real-time applications and can introduce indeterminism in execution time. In addition, the wakeup of a single thread (through the notify) is not always precisely defined.

d) Synchronization: Synchronized code uses monitors to protect critical code sections from more than one simultaneous access. Even though Java implements mutual extension, it does not prevent unbounded priority inversions, an unacceptable condition for real-time systems.

e) Garbage Collection: Automatic memory management simplifies programming and avoids programming errors. At the same time, traditional garbage collection implies in pauses at indeterminate instances impose delays of unbounded duration.

f) Worst- Case Execution Time: Key concepts for object-oriented programming guide in Java are technique overriding and the use of interfaces for a couple of inheritances. However it usually requires a search on the class hierarchy or dynamic selection of functions at runtime, what complicates WCET analysis.

g) Dynamic Class Loading: In order to dynamically load classes, they should be resolved and verified. This is a complicated and memory-consuming task, which may additionally introduce an unforeseen delay in execution time depending on elements as the speed of the medium and the classes' size.

As we may also see, standard Java implementations do not longer provide mechanisms for the reliable and deterministic execution of real-time applications. However, most of these issues do not come from the language, however from the Java execution environment.

4) Real-Time Solutions for Java

The benefit of Java over languages traditionally used to design real-time systems led to several efforts in late 1990's in extending the language. Next section presents the main solutions developed to make Java more appropriate for real-time applications.

5) Early Work in Real-Time Java

Ander Nilsson and Torbjorn Ekman proposed an approach based on compiling Java into native machine code via **C** as an intermediate language. The **C** code generation process should also add close interaction with a fully pre-emptive incremental garbage collector and a small and efficient real-lime kernel. Tests performed on a small &bit microprocessor show that it is possible to use a modern object-oriented language with automatic memory management-such as Java-and yet generate fully predictable code that can be rim in very small devices with severe memory constraints [1].

JamaicaVM is a hard realtime Java VM with a fully pre-emptable, deterministic garbage collector. Depending on the hardware and operating system a submicrosecond jitter can be achieved. The JamaicaVM tool chain contains an application builder and profiler for optimizing applications. The J2SE language features are supported. It is optimized for intelligent systems and critical applications and widely used in the automotive, industrial automation, military, medical and financial application.

III. Real -Time Specification for Java

Java with its pragmatic method to object orientation and enhancements over C, were given very popular for desktop and server application development. The productivity increments of as much as 40% compared with C++ attracts also embedded systems programmers. However, standard Java is not practical on those usually small devices. The [4] paper presents the status of restricted Java environments for embedded and real-time systems. For missing definitions, additional profiles are proposed. Results of the implementation on a Java processor show that it is far not possible to develop applications in pure Java on resource constraint devices.

Java was first utilized in an embedded system. In the early '90s Java, which was originally known as Oak, was created as a programming tool for wireless PDA. The device (known as *7) was a small SPARC primarily based hardware tool with a tiny embedded OS. However, *7 was not issued as a product and JAVA was officially released in 1995 as a new language for internet (to be integrated into Netscape's browser). Over the years, Java era has become a programming tool for desktop applications and web services. With every new release, the library (defined as part of the language) continues to develop. Java for embedded systems was clearly out of focus for Sun. With the advent of mobile phones, Sun again become interested in this embedded market. Sun defined distinctive subsets of Java, which might be analyzed in this paper.

As the language become popular, with less difficult object oriented programming than C++ and threads defined as part of the language, its utilization in real-time systems were considered. Two competing groups began to define the way to convert Java to be used in those systems. Nilsen published the first paper on this subject in November 1995[5] and formed the Real- Time Working Group. The other group known as Real-Time Expert group, published the RTSJ (Real-Time Specification for Java) [6].

RTSJ was the first specification request under Sun's Java Community Process and gained much attention from educational and commercial research. This paper will give:

- An extended overview of actual specifications for Java for embedded and real-time systems.

The Real-Time Expert Group created the Real-Time Specification for Java (RTSJ) whose implementations provide extra mechanisms for building deterministic Java applications.

IV. Other Real -Time Solution for Java

Next to all software tactics, hardware implementation of JVM which is known as Java processor is also provided as a solution of real-time systems. Basically, a Java processor is a stack primarily based processor and executes JBC at once. Method cache and stack cache take the places of instruction cache and information cache, separately, internal of Java processors. It is possible that Java processors are designed to be deterministic in terms of execution time.

Komodo [7] is an early implementation of Java processors that offer the primary Java capabilities and support real-time tasks.

[8] continued working on Komodo processor with superior scheduling and event-handling algorithms.

SHAP [9] is another Java processor that is designed specifically for real-time systems. It implements rapid context switching and concurrent Garbage Collector (GC).

JOP [10, 11] is a well-developed Java processor which is WCET analyzable. Method cache in JOP simplifies the evaluation of WCET in control flow, due to the fact most effective thread switching can introduce cache misses. Tools for performing WCET analysis on JOP is provided in [12]. The excessive degree WCET analysis is primarily based on ILP and a low level timing model is provided by JOP properties.

Similar works have additionally been completed in [13, 14].

Besides, Harmon and Klefstad [15] adapted their work of WCET annotation to Java processors, and made it interactive to developers so that it will offer various feedbacks.

The research resources of real-time Java are quite constrained. Only a few real-time JVMs are completed and fewer of them are under an open-source license.

IBM [20] and Sun Microsystem (now Oracle) [21] are two important companies who provide well-developed commercial real-time Java products. Evaluation or academic version of their real-time JVM can be obtained from the Internet. However, the source code is unavailable.

OVM [22] is a good preference. It is open-sourced, supports most of RTSJ's features, and is still an active

research project [24, 16] with some documents. There are two most important issues with OVM: first, its JIT compiler is quite simple and incomplete (lack of dynamic class loading); second, OVM is obviously designed for a single processor. Extra work is needed if we want to do some research on multicore systems. An alternative may be jRate [17], whose source code is also available. jRate is an extension to the GNU GCJ compiler and a collection of runtime libraries. It implemented most features needed by RTSJ.

Another candidate is Jikes RVM [23]. Although Jikes RVM is not designed for real time purpose, it is the most completed open-source JVM. Actually the prototype of the Metronome garbage collector is implemented on Jikes RVM. So it is possible to develop real-time extension for Jikes RVM.

Aside from these software solutions, the JOP [19] Java processor is also open-sourced, in VHDL format. So it is possible to combine it with some simulators such as SimpleScalar or Trimaran so as to construct a multiprocessor system. There is some research work [18] specializing in multiprocessor, and this is a very promising research field.

V. The Javolution Library

A Java library named Javolution is an extended library on Sun Java implementing RTSJ. It is open-sourced. The Javolution library provides *time-deterministic* and *RTSJ-Safe* alternative implementations of the standard library interfaces. The Javolution collections as an example implement the standard collection interfaces and can be used as drop-in replacement. These collections have additional characteristics extremely valuable for real-time systems such as thread-safe without synchronization, support for custom key/value comparators, direct record iterations (no object creation), etc. Time-determinism behavior is achieved through incremental capacity increases instead of full resizing. In other words, resizing occurs more often but has less impact (on execution time or memory fragmentation). An important aspect of Javolution implementation is that all classes are RTSJ-Safe. If an object has to perform some lazy initialization or increase its capacity this is always done in the same memory area as the object itself. When high-level components are implemented using Javolution components, these high-level components inherit from the same real-time characteristics guaranteed by the Javolution components (time determinism and RTSJ Safety). On the other hand high level components based on standard components might suffer from the same kind of time unpredictability which plagues the standard library.

- Javolution classes are simple to use, even simpler than most JDK classes. You don't need to guess the capacity of a TextBuilder, FastTable or a FastMap, their size expand gently without ever incurring expensive resize/copy or rehash operations (unlike StringBuilder, ArrayList or HashMap).
- Developers may achieve true separation of concerns (e.g. logging, configuration) through Context Programming or by using classes such as Configurable.
- Javolution classes are fast, very fast (e.g. Text insertion/deletion in O[Log(n)] instead of O[n] for standard StringBuffer/StringBuilder).
- All Javolution classes are hard real-time compliant with documented real-time behavior.
- Javolution makes it easy for concurrent algorithms to take advantage of multi-processors systems.
- Javolution's real-time collection classes (map, table and set) can be used in place of most standard collection classes and provide numerous additional capabilities.
- Any Java class can be serialized/deserialized in XML format in any form you may want, also no need to implement Serializable or for the platform to support serialization
- Javolution provides Struct and Union classes for direct interoperability with C/C++ applications.
- Javolution is fully integrated with OSGi but still can be used as a standard Java library.
- Javolution can be either a Pure Java Solution or a Pure Native Solution (C++ mirror), small (less than 400 KBytes jar) and fully produced through maven (Java and C++).

A. Features

Javolution real-time goals are simple: To make your application faster and more time predictable!

- **High-Performance** - Hardware accelerated computing (GPUs) with ComputeContext.
- **Minimalistic** - Collection classes, supporting custom views, closure-based iterations, map-reduce paradigm, parallel computations, etc.
- **Optimized** - To reduce the worst case execution time documented through annotations.
- **Innovative** - Fractal-based structures to maintain high-performance regardless of the size of the data.
- **Multi-Cores Ready** - Most parallelizable classes (including collections) are either mutex-free (atomic) or using extremely short locking time (shared).
- **OSGi Compliant** - Run as a bundle or as a standard library. OSGi contexts allow cross cutting concerns (concurrency, logging, security,...) to be addressed at run-time through OSGi published services without polluting the application code (Separation of Concerns).
- **Interoperable** - Struct and Union base classes for direct interfacing with native applications. Development of the Javolution C++ library to mirror its Java counterpart and makes it easy to port any Java application to C++ for

native compilation (maven based) or to write Java-Like code directly in C++ (more at Javolution C++ Overview)).

- **Simple** - You don't need to know the hundreds of new Java 8 util.* classes, most can be built from scratch by chaining Javolution collections or maps. No need to worry about configuration, immutability or code bloating!
- **Free** - Permission to use, copy, modify, and distribute this software is freely granted, provided that copyright notices are preserved (BSD License).

VI. Conclusion

Real-time embedded systems have increasingly become integral to our society. Real-time applications range from safety-critical systems such as aircraft and nuclear power plant controllers, to entertainment software such as video games and graphics animation. Recently, there had been growing interests in the usage of Java for a wide variety of both soft- and hard-real-time systems, primarily due to Java's inherent features such as platform independence, scalability and safety. However, to permit real-time Java computing, the computation time of Java applications must be predictable, which is particularly crucial for hard real-time and safety-critical systems. This paper surveys this relatively new research area, which is expected to help researchers understand the state-of-the-art and to advance the real-time Java computing. In this work, we have reviewed the RTSJ and the WCET analysis of Java applications at both the byte code level and the architectural level. Since garbage collection can disrupt the time predictability, we have surveyed the state-of-the-art solutions of real-time Java GC for both uniprocessors and more than one processors.

Due to the importance of JIT compilation on the overall performance of Java applications, we have also discussed the compiler issues for real-time Java applications. In addition to the software based solutions to achieve time-predictable Java computing, we have also briefly explained the current work in designing real-time Java processors. To help new researchers in deciding appropriate real-time Java experimental framework, this paper also listed a number of current open-source and private real-time JVMs, libraries and soft Java processors. As can be seen in this overview, real-time Java computing is a lively and promising research field. There are many research challenges and opportunities as well. Based on this survey, further investigation remains needed in the following directions:

- Time-predictable dynamic compilation for real-time Java applications
- Real-time Java on multiprocessor (multiple uniprocessor OR uniprocessor +Java processor)
- Low level Java WCET analysis with architectural timing information (cache, branch prediction, etc.)
- Multiprocessor/multicore real-time GC algorithms

Ensuring bounded reaction time is of interest to any interactive utility even non real-time. But for protection crucial packages it is vital. As we have visible in this paper using a RTSJ VM isn't sufficient. One might also wish that increasingly more consideration may be given to time-determinism whilst implementing Java specifications. The Javolution project has confirmed to be quite popular and is presently being leveraged by way of builders from many professional agencies (Raytheon, Sun, IBM and so on.) Finally, it has to be noted that real-time is not incompatible with excessive overall performance. In many times Javolution classes are faster than their general opposite numbers, proving that you can be both real-time and real-speedy.

References

[1] A. Nilsson, "Deterministic Java in Tiny Embedded Systems", IEEE Conference on Object Oriented Real Time Distributed Computing, ISORC - 2001.

[2] Jean-Marie Dautelle, "Fully Time Deterministic Java™", American Institute of Aeronautics and Astronautics SPACE Conference and exposition, AIAA SPACE Forum 18-20 September 2007.

[3] "The Javolution Library" [Online] Available: http://javolution.org.

[4] M. Schoeberl, "Restricitons of Java for Embedded Real-Time Systems", in Proceedings of the 7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, ISORC 2004, Austria, Vienna, May 2004.

[5] K. Nilsen. Issues in the Design and Implementation of Real-Time Java. Java Developer's Journal, July 19, 1996.

[6] G. Bollella, B. Brogsol, P. Dibble, S. Furr, J. Gosling, D. Hardin, M. Turnbull, R. Belliardi, D. Locke, S. Robbins, P. Solanki, and D. de Niz., "Real Time Specification for Java". Addison-Wesley, November 2001. [Online] Available: http://www.rtsj.org/docs/rtsj_1.0.2spec.pdf.

[7] U. Brinkschulte, C. Krakowski, J. Kreuzinger, and T. Ungerer, "A multithreaded Java microcontroller for threadoriented real- time event handling," in Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, Newport Beach, CA, 1999, pp. 34-39.

[8] J. Kreuzinger, U. Brinkschulte, M. Pfeffer, S. Uhrig, and T. Ungerer, "Real-time event-handling and scheduling on a multithreaded Java microcontroller," Microprocessors and Microsystems, 2003, vol. 27, no. 1, pp. 19-31.

[9] M. Zabel, T. B. Preuber, P. Reichel, and R. G. Spallek, "Secure, real-time and multi-threaded general-purpose embedded Java microarchitecture," in Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, Lubeck, Germany, 2007, pp. 59-62.

[10] M. Schoeberl, "JOP: a Java optimized processor for embedded real-time systems," Ph.D. dissertation, University of Technology, Vienna, Austria, 2005.

[11] M. Schoeberl, "A Java processor architecture for embedded real-time systems," Journal of Systems Architecture, 2008, vol. 54, no. 1-2, pp. 265-286.

[12] M. Schoeberl and R. Pedersen, "WCET analysis for a Java processor," in Proceedings of the 4th International Workshop on Java Technologies for Real-Time and Embedded Systems, Paris, France, 2006, pp. 202-211.

[13] C. Z. Lei, T. Z. Qiang, W. L. Ming, and T. S. Liang, "An effective instruction optimization method for embedded real-time Java processor," in Proceedings of the International Conference on Parallel Processing Workshops, Oslo, Norway, 2005, pp. 225-231.

[14] Z. Chai, W. Zhao, and W. Xu, "Real-time Java processor optimized for RTSJ," in Proceedings of the ACM Symposium on Applied Computing, Seoul, Korea, 2007, pp. 1540- 1544.

[15] T. Harmon and R. Klefstad, "Interactive back-annotation of worst-case execution time analysis for Java microprocessors," in Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Daegu, Korea, 2007, pp. 209-216.

[16] A. Armbruster, J. Baker, A. Cunei, C. Flack, D. Holmes, F. Pizlo, E. Pla, M. Prochazka, and J. Vitek, "A real-time Java virtual machine with applications in avionics," ACM Transactions on Embedded Computing Systems, vol. 7, no. 1, article no. 5, 2007.

[17] jRate, http://jrate.sourceforge.net/.

[18] C. Pitter and M. Schoeberl, "Towards a Java multiprocessor," in Proceedings of the 5th International Workshop on Java Technologies for Real-Time and Embedded Systems, Vienna, Austria, 2007, pp. 144-151.

[19] M. Schoeberl, "JOP: a Java optimized processor for embedded real-time systems," Ph.D. dissertation, University of Technology, Vienna, Austria, 2005.

[20] IBM WebSphere Virtual Machine, http://www-306.ibm.com/software/webservers/realtime/.

[21] Sun Microsystem Real-Time Java System, http://Java.sun.com/Javase/technologies/realtime/.

[22] Open Virtual Machine, Purdue University, http://www.cs. purdue.edu/homes/jv/soft/ovm/.

[23] Jikes RVM, http://www.jikesrvm.org/.

[24] J. Baker, A. Cunei, C. Flack, F. Pizlo, M. Prochazka, J. Vitek, A. Armbruster, E. Pla, and D. Holmes, "A real-time Java virtual machine for avionics: an experience report," in Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium, San Jose, CA,2006, pp. 384-396.

[25] Ms. R. S. Lande and Dr. M. S. Ali, "Synchronization in Embedded Real-Time Operating Systems", in International Journal of Advance Engineering and Research Development, June 2014, Vol 1, Issue 6.