Java for Embedded, Real-Time Systems

Muhammad Azeem Khan  mkn05006@student.mdh.se
Gurjodh Singh Dhillon  gdn05001@student.mdh.se

4 june 2007
Abstract

Java paradigm to object oriented programming, memory management and secure management of complex system design make it popular for desktop applications. It is true for embedded system, today’s embedded system extend the complexity of system design with timing and deployment constraints. J2ME is considerably used in small embedded device, its Connected Limited Device Configuration is used for devices with limited memory while Connected Device Configuration profile is used for shared network devices. Further deployment of embedded system with real time system imposed more focus on predictability and timing issues. The Real-Time Specification for Java addresses many features specifically for real-time domain while maintaining the application optimization with limited resources and without affecting the timing predictability. The report describes the challenges faced by the Java for its implementation in real-time systems, then it describes the detail RTSJ specification and its overcoming. Finally for archiving overcoming of RTSJ and additional functionality extra profiles are described.
Contents

1 Introduction 4
  1.1 Overview of Java ........................................... 4
    1.1.1 Characteristics of Java ............................. 4
  1.2 Object Oriented Development .............................. 6
  1.3 Embedded Systems ........................................ 7
    1.3.1 Embedded Software Architecture .................... 7
  1.4 Real-Time Systems ......................................... 8

2 Java 2 Micro Edition (J2ME) for Embedded Systems 9
  2.1 J2ME Architecture .......................................... 11
  2.2 Performance Tuning for the J2ME Applications .......... 11

3 Challenges to Java for Real-time Systems 13
  3.1 Thread Model .............................................. 13
  3.2 Memory Management ........................................ 14
  3.3 Asynchrony ................................................ 14
  3.4 Dynamic Semantics ........................................ 15

4 Real-time Specification for Java (RTSJ) 15
  4.1 Memory Management: ....................................... 16
  4.2 Time Values and Clocks: ................................... 18
  4.3 Scheduling: ............................................... 18
    4.3.1 Scheduler: ............................................ 19
    4.3.2 Scheduling Parameters: .............................. 19
  4.4 Real-time Threads: ........................................ 20
  4.5 Asynchronous Events and Event Handling: .............. 21
  4.6 Asynchronous Transfer of Control: ...................... 22
  4.7 Synchronization and Resource Sharing: ................. 22
  4.8 Physical Memory Access: .................................. 23

5 Real-time High Integrity Profile 24
  5.1 Ravenscar-Java Profile: .................................... 24
    5.1.1 Predictability of memory utilization: ............. 25
    5.1.2 Predictability of timing: ............................ 25
    5.1.3 Predictability of control and data flow: ........... 26

6 Conclusion: 26
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>J2ME Architecture Layers</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>Classes Supporting Memory Management</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>Memory References</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>Time Classes</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>Scheduling Classes</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>Real-time Thread and its Parameter Classes</td>
<td>21</td>
</tr>
<tr>
<td>7</td>
<td>Two execution phases</td>
<td>25</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Overview of Java

Java technology was developed to provide standard programming technique for developing the platform, machine and vendor independent server side applications [1]. Development of Java language started with Oak Project by Sun Microsystems in the year 1991. Earlier it was intended to be used as a programming language for embedded devices [2]. In 1995 the Oak was renamed as Java and launched as Java Development Kit (JDK) which can be downloaded from Internet making Java an easy choice. Further with the introduction of Java Servlets proved to be a replacement for CGI (Common Gateway Interface) which initiates a revolution in server side programming making Java a popular server side technology [1]. In late 1998 with the release of Java 2 Platform, new versions had multiple configurations built for different platforms [3]. For example, Java 2 Platform Enterprise Edition (J2EE) offers a standard platform for the development of enterprise applications; Java 2 Platform Micro Edition (J2ME) is for developing mobile applications; Java 2 Standard Edition (J2SE) is used for developing and deploying applications on desktops and servers [4]. Java Runtime Environment (JRE) is a software bundle consisting of Java Virtual Machine and Programming interface to run a Java application and it is just a subset of the SDK. The primary distinction between Java Standard Development Kit and JRE is that in JRE the compiler is not present [4]. According to Sun Microsystems, Java is a “simple, object-oriented, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, and dynamic language.”

1.1.1 Characteristics of Java

Some of the important visible characteristics of Java language that make Java an ultimate choice for a programmer are described below:

1. Object-Oriented:

Every Java program needs to have at least one class [6], thus class is used as a building block in Java which makes it pure Object-Oriented Language. There are no stand-alone constants, variables, or functions in Java and everything is accessed through classes and objects [7]. It is possible in Java to reuse previously written classes and create new classes with more functionality which can cut the amount of code and time needed to develop an application. Thus Java provides well structured and object-oriented programming environment which facilitates
easy system development [5].

2. Platform Independent:

Java is platform independent, so the applications developed using Java are portable across multiple platforms. The code written in Java can be compiled into bytecodes for Java Virtual Machine (JVM) that interprets and executes generic Java bytecodes [3] [5]. These Java bytecodes can run on different platforms having JVM. In beginning, Java used an interpreted virtual machine to achieve portability which is slower and hinder performance of the language. Using techniques such as just-in-time compilation and dynamic recompilation, latest Java Virtual Machine implementations produce programs that run much faster improving the performance of Java language [3].

3. Automatic Memory Management:

Automatic Memory Management or Automatic Garbage Collection is significant development in Java platform which manages the objects lifecycle sparing the programmers from the burden of performing manual memory management. In Java whenever there are no references to an object, Java garbage collector deletes the unreachable objects, freeing memory and preventing memory leak [3]. It can also affect programming strategy since programmer can construct objects freely if allocating and recollection of memory is cheap [3].

4. Network Support:

Java supports multilevel network communication with its rich library for network programming. The Java API is implemented through java.io package and it is implemented in the form of several streams dedicated for filtering and pre processing input and output data streams [5]. The Application programming interface (API) developed on this basic networking streams implement complex high level networking capabilities such as distributed objects, remote connections to database servers, directory services etc [5].

5. Robustness:

Robustness is one of the critical features of Java language that makes it the first choice for a programmer. In Java, strict error checks are
performed both at execution-time and compile-time. Also early detection of errors in code can be possible by type checking, and exception handling in Java is strong [9]. Java's pointer model makes it a more robust language by removing the problem of memory overwriting and data corruption.

6. Security:

Security is an important issue in Java since this language is intended to be used across networks. Security in Java is implemented in following ways [8]:

- Source code for Java compiler and interpreter are accessible for inspection.
- It does not use pointer arithmetic thus restricting programmers from forging a pointer to memory and restricting users to access private information.
- Java has strict language definition, e.g. two java compilers always give same result for a same program.
- Automatic Garbage Collection feature of Java also enhance security.
- Strict compile-time checking is performed by Java compiler to check errors and violation of security barriers.

1.2 Object Oriented Development

Since Java is an Object-Oriented Language, for developing any application in Java one has to undergo through the three phases of Object-Oriented development. These three phases are Object-Oriented Analysis, Object-Oriented Design, and Object-Oriented Programming. Following is the brief description about these phases:

1. Object-Oriented Analysis:

Object-Oriented Analysis (OOA) is done to analyze the problem domain and model it. Problem domain can be divided into sub-domains each constituting particular area of the problem depending upon the magnitude of problem. Requirement documentation, use cases, and UML diagrams are used to illustrate the problem domain. OOA describes the problem using a conceptual model which constitutes of concepts and relationships between the concepts.
2. Object-Oriented Design:

Next phase in Object-Oriented development is Object-Oriented Design (OOD), in which logical solutions to the problem domain are set out using objects. In OOD, implementation constraints are added to the conceptual model, and concepts in the conceptual model are mapped to concrete classes, to abstract interfaces in APIs and to roles that the objects take in various situations [10]. This phase describes in detail how to build the system using objects.

3. Object-Oriented Programming:

The last phase of Object-Oriented development is Object-Oriented Programming (OOP) or coding, in this objects are used to design applications. In OOP, each object can receiving messages, process data, and send messages to other objects, thus each object can be viewed as an independent little machine with a distinct role or responsibility [11]. It is much easier to code in an object-oriented language, also it is easier to understand and handle complex problems.

1.3 Embedded Systems

An embedded system is a special-purpose computer system designed to perform a dedicated function [12]. Embedded systems are usually small in size and have limited resources like limited memory and low processing power. Software used in embedded systems is known as Firmware and is stored in the flash memory or ROM chip. Embedded system can be a part of a large system in few cases where they provide standard computing services [12]. Typical example of an embedded system is a router which has integrated flash memory, microprocessor, and RAM. Other common examples are mobile phones, automatic washing machines, PDA, MP3 players, cruise missiles, pacemakers, GPS receivers etc.

1.3.1 Embedded Software Architecture

It is same as the simple control loop scheme, but in this architecture the loop is hidden in an application programming interface [13]. Advantage of non-preemptive multitasking is that new software can be added easily just by creating a new task. There are various software architectures available for use in embedded systems, following are few general architectures common in embedded systems:
1. Simple Control Loop:

In this architecture there is a loop in the software, and loop calls subroutines which handles a part of software or hardware [12]. The advantage of this design is its simplicity and its fastness. This design is mostly used on small devices having specific task to perform with stand-alone microcontroller [13]. Some of the disadvantages of this are that its difficult to maintain and its timing is not predictable.

2. Preemptive Multitasking:

In this design, tasks can be switched based on the timer. Since one task can overwrire other tasks data they are separated from each other, and access to the shared data is controlled by using synchronization strategy [12]. In these circumstances there is a need for an operating system to handle multiple tasks.

3. Non-Preemptive/Cooperative Multitasking:

It is same as the simple control loop scheme, but in this architecture the loop is hidden in an application programming interface [13]. Advantage of non-preemptive multitasking is that new software can be added easily just by creating a new task.

4. Interrupt Controlled System:

This design is event triggered and tasks are executed when triggered by other events e.g. interrupt generated by a timer in a predefined frequency [12]. It is useful in the situations where event handlers are short and need low latency. Also the interrupt latency should be kept minimum by keeping tasks short in the interrupt handlers [12].

1.4 Real-Time Systems

Real-time system is a system with real-time constraint and its tasks should be performed within the time restraints. More detailed definition of real-time systems is given by Donald Gilies:

A real-time system is one in which the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time in which the result is produced. If the timing constraints are not met, system failure is said to have occurred.
There are two general key terms in the context of real-time systems, first is Predictable which means every task should have to meet the deadline and second is Deterministic which means time behavior should be within a certain range and it can be predetermined [14]. Real-time systems can be classified into hard real-time systems and soft real-time systems. Hard real-time systems have hard deadlines to meet and failure can result in a catastrophic effect. Some examples of hard real-time systems are heart pacemakers, missile control systems, elevators etc. On the other hand, soft real-time systems have soft or less strict deadlines which mean that they don't result in catastrophic failure even if they are not able to meet the deadlines. Some examples of soft real-time systems are telecommunication systems, live audio-video systems etc.

2 Java 2 Micro Edition (J2ME) for Embedded Systems

Java 2 Platform, Micro Edition (J2ME) a reduced edition of Java Development Kit (JDK) with low profile library and JVM proved to be a second revolution in the Java industry. Developed to cater the multimedia domain of handheld devices with limited resources, Java 2 Platform, Micro Edition is a platform to develop applications for embedded devices like mobile phones, GPS systems and PDAs. J2ME provides the methodology that can make functioning of small devices very similar to PC (Personal Computer). There are three major elements of J2ME, namely profiles, configurations and optional APIs. Configuration of a particular device is designed according to its specifications like memory, and processing power. Configuration also includes the JVM (Java Virtual Machine) which can be ported to any particular device. Profiles define the additional APIs for developing the GUI (Graphical User Interface), persistent storage and all other functionalities which are essential for the application. Profile is totally based on the configuration. Optional APIs are included to embed some extra functionalities which are not essential for the application but certainly can improve the application [15].

There are two types of configurations in J2ME, first is the Connected Limited Device Configuration (CLDC) and second is Connected Device Configuration (CDC). They are described as follows:

1. Connected Limited Device Configuration (CLDC):

   CLDC is used for the devices like cell phones and PDAs which have lim-
CLDC devices usually use 16 bit architecture. CLDC devices cannot use the JVM so these devices use the low profile version of the JVM known as KJava Virtual Machine (KVM). KVM has no support for floating point, Java Native Interface, asynchronous exceptions, thread groups, reflection, finalization, weak references, and user-defined class loaders [20]. CLDC HotSpot Implementation(tm) promotes faster application execution and effective resource management, which result in better performance than KJava Virtual Machine (KVM) [16].

CLDC only has only one profile i.e. MIDP (Mobile Information Device Profile) which has java APIs that are responsible for providing the user network connections and user interface for a media player, LC displays, and a game API [15] [20]. When CLDC is coupled with Mobile Information Device Profile (MIDP), it provides a solid Java runtime environment for developing applications for embedded mobile information devices, such as mobile phones, PDAs etc. [18]. Higher versions of MIDP have new features like improved user interface, multimedia and game functionality, greater connectivity, over-the-air (OTA) provisioning, and end-to-end security [18].

2. Connected Device Configuration (CDC):

Connected Device Configuration (CDC) is a Java framework used to build and deliver applications that can be shared across a range of network connected consumer and embedded devices [17]. CDC devices use 32 bit architectural design and the minimum memory required by the CDC devices is 2MB. CDC devices can use JVM so they incorporate the full version of the Java Virtual Machine (JVM) [15]. CDC HotSpot Implementation(tm) is most suitable for embedded systems and other resource constrained devices as CDC HotSpot Implementation(tm) provides portable interfaces, dynamic compiler, and solid reliability for multi-threaded and low-memory conditions [17] CDC has three profiles as listed below [20]:

- Foundation Profile- This profile consists of Java APIs that support embedded devices without standards-based GUI system. It supports basic class libraries of standard Java.

- Personal Basis Profile- This profile consists of Java APIs that support embedded devices with standards-based GUI framework based on light weight components. It has partial support of Abstract Window Toolkit (AWT).
- Personal Profile- Personal profile has support for the applet interface and it has complete AWT libraries.

From above discussion it is clear that J2ME is quite appropriate for embedded systems but it is not suitable for embedded real-time systems because of its low profile library and JVM which do not correspond to Real-Time Specifications for Java (RTSJ).

2.1 J2ME Architecture

J2ME (Java 2 Micro Edition) architecture is not an operating system in itself but it is a combination of layers above the native operating system that can provide run-time environment for portable or embedded computing devices [15]. Thus J2ME platform has the capability to offer benefits and power of java technology to any embedded device independent of the operating system installed. It is because of the modular design of J2ME architecture that it is possible to build an application that can be scaled depending upon the constraints of a device [15]. According to Sun Microsystems:

> J2ME architecture defines configurations, profiles and optional packages as elements for building complete Java runtime environments that meet the requirements for a broad range of devices and target markets.

J2ME architecture defines configurations, profiles and optional packages as elements for building complete Java runtime environments that meet the requirements for a broad range of devices and target markets.

There are 3 software layers above the operating system that comprises J2ME architecture as shown in figure 1. The first or the lower most layer is called configuration layer, in this layer JVM (Java Virtual Machine) directly interacts with the operating system. This layer also controls interactions between JVM and profile layer. The second layer is known as profile layer, it comprises of least number of APIs (Application Programming Interface) which can be utilized in embedded devices. The top most or third layer is called MIDP (Mobile Information Device Profile) [15].

2.2 Performance Tuning for the J2ME Applications

Performance has always remained the major issue in the applications for embedded devices. Embedded devices usually do not have large resources as compared to computers like the processing power of these devices is lower than general computers by many folds and same is the case with memory. So
programmer has to be extremely cautious while developing any application for compact devices. Some of the tips for increasing the performance of J2ME applications are given below [19]:

1. Minimizing Memory Use:

As discussed earlier in this report that memory remains the core issue in compact devices so the programmer needs to focus on this point specifically. Java environment provides you the feature of garbage collector. The main responsibility of the Java garbage collector is to remove the unreferenced objects from the memory. This mechanism is automatic and programmers do not have to worry about the garbage collector activation. But the process of garbage collection can still make your application slower. Following are some techniques to minimize the usage of memory [19]:

- Creating a new object always requires memory, so one needs to be selective while creating new objects. Especially in the loops, a programmer should minimize the creation of new objects. This phenomenon can be analyzed from the sample code below:

- Strings and string buffers both point to the character array in the memory and when you modify any of them then the compiler creates a new character array, and copy the old character array to the new one. So the best practice is to use character array for minimizing the memory usage. Modification of strings gets more severe in the loop.

```java
Object[] myobj = new Object[100];
Int i = myobj.length();
```
for(int i=0; i<size; i++)
{
    Object obj = new Object(test no + i);
    Myobj[i] = obj;
}

Bad Programming Technique:

Object[] myobj = new Object[100];
for(int i=0; i<myobj.length(); i++)
{
    Myobj[i] = new Object(test no + i);
}

Good Programming Technique:

2. Remain Careful While Coding:

The best practice that the programmer should practice is to remain watchful on the constraints of memory and he should try to release the resources as soon as he is finished with them, so that garbage collector can remove the unreferenced memory objects as soon as possible.

3. Prepare for Failure:

As the compact device always has some memory constraint so the application can face failure while requesting for more memory. Application should be able to deal with such situations and one way of dealing with this is to detect java.lang.OutOfMemoryError exception while creating the new objects [19].

3 Challenges to Java for Real-time Systems

The problem for Java as real-time programming languages is focused on the following areas.

3.1 Thread Model

Concurrent programming in know evolved quite a lot and Java is considered as new developments in the concurrent object-oriented programming model. Java concurrency model is based on active objects in the object-oriented framework. Two methods run() and starts() are defined in the Thread class.
A thread is created when its relative object is created. `starts()` begin the new thread execution by calling `run()` method. But Java Language Specification says [21]:

*threads with higher priority are generally executed in preference to threads with lower priority. Such preference is not, however, a guarantee that the highest priority thread will always be running, and thread priorities cannot be used to reliably implement mutual exclusion.*

In Java it is impractical to ensure that deadlines of the thread will meet due its flexibility. In various situations the making thread in ready states when lock is released is not based on thread priority and no priority inheritance or priority inversion protocol is implemented to prevent the priority inversion problem. Java have only 1 to 10 priority ranges which are insufficient and unrealistic. Even the `sleep()` [22] method implementation is ambiguous and not adequate.

In certain situation the preference to monitor access is not given to the thread those waiting for them instead access is transferred to the thread accessing the monitor first time. Nested monitor calls and thread safe objects are difficult to identify because methods can still hold synchronized statements instead they are not labeled as synchronized. Java also lack for support for condition variables.

### 3.2 Memory Management

Memory management is one of Java strong feature, unlike other object-oriented languages Java provide automatic memory reclamation known as Garbage Collection [22]. Garbage collector works as part of JVM and handle all memory allocation and releases [23]. It provides flexibility and easiness to the programmer but impose extra overheads and predictability issues. In real-time application predictability has core importance and can not be negotiable. For that reason conventional garbage collector can not work with real-time and whole new architecture is required.

### 3.3 Asynchrony

Real-Time application has to communicate with the asynchronous event either software, hardware or events real world, and thus asynchronous transfer of control asynchronous thread termination is also required. Java asynchronous event handler required dedicated separate thread and its `stop()` and
destroy() [22] methods are deprecated and can cause problem in various situations. In asynchrony and asynchronous transfer of control (ATC) Java is not well deterministic. Its interrupt() method is no sufficient and required polling as well.

3.4 Dynamic Semantics

Run-time flexibility of Java is one of its major advantages at other object-oriented programming languages. Its optimized code generation during the run-time execution is due to its advanced compiler and its run-time dynamic class loading [23]. All these features are not appropriate for real-time system and by using Java with static semantic decreases its flexibility and performance.

Java also lack in features like unsigned integers, strongly typed scalars and enumeration types, which can put extra help in real-time applications. Operator overloading [22] and generic templates are also omitted in Java. The Java strongly integration with object-oriented paradigm also introduce some extra problem is real-time aspects like dynamic binding of instance methods and garbage collection. The Java API has no support for low level system hardware access which are essential for real-time systems. JVM also introduce extra overheads for the system performance and its predictability. Java has lack of support for identifying timeout occurrence and delaying for absolute time.

4 Real-time Specification for Java (RTSJ)

Java is mostly distinguished as high integrated programming language for most developers. Its rich integrity with object oriented paradigm makes it suitable for large and complex application on web and distributed environments. In Real Time environment Java fall behind due to its memory management especially garbage collector, dynamic class loading and no support for Real-Time threads.

To address these issues SUN release new specification of Java for real time systems in 1999. RTSJ particularly enhance the java in the following areas[27].

- Memory Management
- Time Values and Clocks
4.1 Memory Management:

In non Real-Time applications systems are provided with substantially large amount of memory and developer are quite independent of their usages and get relief with low-level resource allocation issues [28]. Operation like variable access, register access and I/O access are handled by the language but Real-Time application are built on embedded system and consist of limited memory. The mechanism is needed to accurately manage and control the memory usage. Effective memory management increases the application performance and predictability.

The RTSJ allow effective memory management without or limiting the affects of garbage collector [27]. It introduces memory areas some of them reside outside standard [27] java memory model. The garbage collector can be pre-empted by the Real-time Thread. The RTSJ implement abstract Memory Class from which these memory areas are derived[27] [29]:

1. HeapMemory:

   In heap memory objects are declared like standard java heap.

2. ImmortalMemory:

   Objects created in immortal memory never get delay from garbage collector, threads and schedulable objects can use this memory in the application. These objects release when the program terminate its execution.

3. ScopedMemory:

   Objects are declared with well defined life time spam and created upon application need. A reference count is used to keep track the number
of real-time entities are been used. It has two types, in *VTMemory* (Variable Time Memory) objects allocation can take variable time and *LTMemory* (Linear Time Memory) where object allocation is directly proportional to the object size. *SizeEstimator* class is used to estimate the size needed by the object in scope memory.

Collection mechanism of memory area is different for each one, the RTSJ implies assignment restrictions on each memory area. Otherwise dangling reference [27](a dangling reference is a object that is collected when scope memory is reclaimed) can occurs.
4.2 Time Values and Clocks:

In Real-Time system various system and user takes perform action upon specified interval of time and such event are schedule by timers[28]. Standard Java uses the wall clock and it imitates the UTC coordinate while RTSJ encapsulate the clock with 64 bit millisecond and 32 bit nanosecond accuracy[29]. All time classes in RTSJ are derived from the Time abstract Clock class. The time relation to given epoch is express as Absolute time and time interval measure by some clock is relative time[27]. A rational time is another abstract time and it represent the rate is which certain event occurs.

4.3 Scheduling:

Scheduling is fundamental to any real-time system [28]. In Standard Java priorities are assigned to the threads but it does not guarantee that scheduling is always carried out according to threads priorities because JVM rely of its host operation system to implement its threads which may have no support for pre-emptive priority based scheduling and only 10 threads priority level are defined for threads. In real-time application such restriction imposes unbearable consequences and needed to me redefined. The RTSJ introduce new architecture to implement scheduling. First Schedulable Objects are been identified away from traditional threads and each schedulable object is specified with additional details which are required by scheduler[27]. The RTSJ has two schedulable objects Real-time Threads and asynchronous event handlers. Further RTSJ categorized into two divisions Scheduler and Scheduling Parameters.
4.3.1 Scheduler:

A scheduler schedules the schedulable objects. A RTSJ support the priority scheduling through the PriorityScheduler class which is derived from Scheduler abstract class is derive, yet other user defined scheduler can be used like EDF unless compromising the security RealtimeSecurity class. A fixed priority scheduling (FPS) is used in the PriorityScheduler which required that:

Execution of the schedulable objects is based on the active priority of the schedulable object. Priorities are assigned at the time of object creation while Priority Inheritance protocol is used for resource sharing [27].

4.3.2 Scheduling Parameters:

1. Releases requirements:

Releases requirements to the schedulable object is specified by the RealeaseParameters class and further with the PeriodicParameters, AperiodicParameters and SporadicParameters classes respectively for periodic aperiodic and sporadic objects (events). All parameters classes include the cost and deadline value which represent the maximum time needed for each instance of the object and the time in which object should finish its execution and start value for periodic objects and inter-arrival value for sporadic objects. The cost and deadline values can be changed by the setIfFeasible method if the new values are still schedulable by the scheduler but always return null for aperiodic schedulable object. Null missHandler or overrunHandler indicates no action should be taken when object miss its cost or deadline.

The RealtimeThread class will only call its method waitFornextPeriod if RealeaseParameters are associated with the object. No monitor is implemented in RTSJ to monitor the processing time consumed by the schedulable objects because real-time JVM depend on host operating system and currently no OS support this feature.

2. Memory Parameters:

The maximum amount of memory used by the object in its default memory area, immortal area and in heap memory. All parameters are given in the application while real-time JVM monitor the memory usage.

3. Scheduling Parameters:
Figure 5: Scheduling Classes [27]

The SchedulingParameters class is abstract class with nothing in it but its underlying PriorityParameters class provides 28 priority levels and ImportanceParameters class provide additional information to some scheduler in some overload conditions amongst in same priority objects.

4.4 Real-time Threads:

Real-time threads are one type of schedulable objects in RTSJ. RealtimeThread and NoHeapRealtimeThread are two different real-time threads implemented in the RTSJ and have their associated releases, schedule, memory and processing group parameters.

In standard Java a thread has to wait for subjectively undetermined amount of time due to the garbage collector, to overcome this issue RTSJ introduce NoHeapRealtimeThread but no-heap real-time thread priority should be higher than other heap based real-time threads [29]. A start method keep track that no heap memory allocated to the thread but it will increase the system overheads. In real-time system activities are based on periodic, aperiodic and sporadic events. The RTSJ distinguish them by the threads parameters of RealeaseParameters class and further with the PeriodicParameters AperiodicParameters and SporadicParameters classes respectively.
for periodic aperiodic and sporadic objects (events) [27]. For aperiodic and sporadic events RTSJ provide no solution for cost and deadline miss other than their first release.

4.5 Asynchronous Events and Event Handling:

Application entities representing real world event or internal events (through call fire()) are called Asynchronous Events [29] [28]. The application has to define some mechanism to interact with these events. Event handlers are coupled with these events and bound them with real-time threads. Relationship amongst the events and event handler can be many to many relationship. The RTSJ implements events and their event handlers with the AsyncEvent and AsyncEventHandler classes. Asynchronous events are schedulable objects in RTSJ and hence comes with their release, memory and scheduling parameters same as RTSJ real-time threads. AsyncEventHandler class comes with various constructors for providing different parameters and consequently heap memory access can be prevented by using the appropriate class constructor [27]. The RTSJ implement the cost budget and if cost is overturn the event handler is descheduled.
4.6 Asynchronous Transfer of Control:

Asynchronous Transfer of Control (ATC) happen when executing schedu-

able object sis changed by the other schedulable object or by the JVM [27].

The RTSJ asynchronous transfer of control comes with the ideas of excep-
tional handling and thread interruption. In a case when schedulable object
is interrupted the asynchronous exception AsynchronouslyInterruptedException (AIE) is thrown to the thread in thread poll for interruption[22]. RTSJ

provide the following guide line for ATC:

- In synchronized code the ATC must be deferred.
- Arbitrary actions can be occurs if ATC does not return from where it
initiated.
- In exceptional handled based ATC, exception should not catch by the
unintended handler.

An instance of AIE is thrown by triggering thread when ATC is initiated
in the target thread and only occur when statement code explicitly permit
it. The ATC constructor take Interruptible as parameter, while Interruptible
define the method run() and interruptAction(). The target thread constructor
passes to triggering thread and invoked doInterruptible on Interruptible
schedulable object while Time class implements the timeout for Interruptible
[22].

4.7 Synchronization and Resource Sharing:

Resource sharing is fundamental issue in multithreaded concurrent systems
and act important rule of timing predictability of the application [28]. Stan-
dard Java communication and synchronization is based on the mutually ac-

cess to shared data through monitor locks but it also introduce the inversion
problem.

The RTSJ address priority inversion problem by two means Priority Inversion
Algorithms and non-blocking communication.

1. Priority Inversion:

RTSJ support priority inversion problem by Priority Inheritance and
Priority Ceiling Emulation. The PriorityInheritance class and Priority-
CeilingEmulation classes inherit from the abstract Monitor class. Syn-
chronized method should kept short and loops should not be included.
Schedulable objects consist of no- heap and heap based threads and
some time they have to share data, at such situation garbage collector indirectly block the thread for unpredictable time. RTSJ also put following guidelines to prevent occurring this problem.

- No-heap schedulable objects must have higher priority than heap schedulable objects.
- Share synchronized objects should use Priority Ceiling Emulation.
- Memory requirements of objects should be preloaded.
- Objects should use primitive data types.

2. Non-blocking Communication:

The RTSJ use non-blocking (wait-free) communication to prevent priority inversion especially for the garbage collector to communication without any delay with non real-time threads and schedulable objects. RTSJ provide three classes \textit{WaitFreeWriteQueue}, \textit{WaitFreeReadQueue} and \textit{WaitFreeDequeue} while maintain the following conditions.

- Wait queue should be in immortal memory.
- Passes objects also must be in immortal memory

4.8 Physical Memory Access:

All Real-Time System can consist of different memory types, it can be static or dynamic random access memory (DRAM, SRAM), read only memory (ROM, EPROM) and hybrid memory (EEPROM) \cite{28}. Physical Memory availability in such system is quite limited due to performance, coast and predictability issues and great care is needed to sufficiently handle it. Moreover a system has to map different register of its input/output devices to memory location and it can be more complicated if DMA is involved with it. Standard Java handles all these activates itself and provide no mechanism to access individual memory locations.

The RTSJ provide direct architecture to access the physical memory and I/O. RTSJ access the physical memory by memory manager and memory filter. Memory manager provide direct access to the programmer to interact particular characteristics of physical memory while memory filter control the access to that particular physical memory location. Physical address of each memory type is hidden from memory manager and only viewable by the memory filter. Memory filter only contain the access type and mechanism of each memory type, and only one memory filter is used for each memory.
Before accessing physical memory location, memory area is created which define the object scope and considered as non-heap memory with individual memory characteristic [27]. Raw memory is accessed when external process outside java write on it and when it is used by memory mapped I/O devices. RTSJ interact with raw memory by outside the object model but within the memory manager through primitive data types while maintaining the java integrity. Byte ordering of the hardware is determined by the \textit{RealtimeSystems} class. A chunk of raw memory or particular raw memory location is accessed by the \textit{RawMemoryAccess} class and floating raw memory location is accessed by the \textit{RawMemory-FloatAccess} class. The JVM and external process must share the same floating-point format.

5 Real-time High Integrity Profile

Now a days use of high integrity softwares are increasing rapidly where failure can cause substantial lose or harm. High integrated systems specifically emphasis on reliability, robustness, traceability and maintainability[24]. Java features which make it popular for general application development make is inappropriate for high integrated systems. Its object-oriented support, inadequate support for real-time threads and garbage collection make it infeasible for real-time high integrated system. RTSJ improve its support for real-time application and overcomes various ambiguities. But on other hands RTSJ architecture becomes too complex for such system and imposes reliability constraints.

No language fully can be used for all domains. Ada is most popular language for high integrated systems but it introduces many radical changes before its adoption for real-time high integrated system. Ravenscar profile is subset of real-time use of ada and becomes industry most reliable specification for real-time and high integrated systems. Ravenscar profile defines a conceptual model for application to analyze timing property on single processor through response time analysis (RTS).

5.1 Ravenscar-Java Profile:

The Ravenscar-Java Profile [25] build upon the idea of developing concurrent java programming model also capable for developing high integrity applications. To achieve reliability some language features are removed and JVM is also been restrict for predictability. Off-line scheduling is used for schedulability analysis with fixed priority scheduling. Object scheduling is based on
pre-emptive priority based scheduling and such objects periodic or sporadic. To prevent priority inversion, priority ceiling protocol is used. Execution is carried out in two phases initialization and mission phase shown in figure 7 [25]. In initialization all non-time-critical activities like real-time threads, memory objects, event handlers, scheduling parameters are carried out while in mission phase application is executed and multithreading is carried out.

The Ravenscar-Java Profile provide helps in following domains:

- Predictability of memory utilization
- Predictability of timing
- Predictability of control and data flow

Although restriction should be consider imposed on Java and RTSJ.

5.1.1 Predictability of memory utilization:

Long term required objects are created by the initialization phase in the immortal memory area [26] and created in the constructor of the class. Objects needed for each activity are created in scope memory and all scope memory should be created by the initialization phase. Its also possible to disallow nested scoped memory areas for ease of memory management.

5.1.2 Predictability of timing:

All concurrent entities are considered periodic or sporadic in the Ravenscar-Java while in RTSJ has only real-time thread and asynchronous event han-
dbers are only concurrent entities. In Ravenscar-Java profile scheduling analysis is performed of-line with 28 unique priority levels but additional 10 standard priority for regular threads are not available, the profile does not support regular threads because uses of heap memory. The RealtimeThread and AsyncEventHandler classes are not directly accessible also due to heap memory and dynamic binding.
Thread invoke waitForNextPeriod method of the RealTimeThread class and end of there main loop body. BoundAsyncEventHandler class manage the event triggered activities, both event and event handler must remain unchanged after being setup for predictability reasons. The profile uses the priority ceiling emulation for archiving synchronization.

5.1.3 Predictability of control and data flow:
For static analysis techniques predictability of control and data flow in requires. A worst-case execution time analysis is based on these static analyses. Ravenscar-Java Profile use no garbage collector so heap memory is treated same as immortal memory.

6 Conclusion:
Java provides a better development environment to cover the vast development of complex system applications for different platforms. Since embedded systems use variety of different platforms, which makes Java an ideal choice for them yet Java being an architectural-neutral, is also object-oriented, secure, robust and portable. Java 2 platform new multiple configurations suit- ing benefits the the needs of various sectors such as J2SE, J2EE is for enterprise applications, and J2ME is for mobile and embedded applications and RTSJ for real-time systems.
Java 2 Micro Edition (J2ME) comes with two types of configurations, Connected Limited Device Configuration (CLDC) is for devices like cell phones and PDAs which have limited memory, slow network connectivity, and run by batteries and Connected Device Configuration (CDC) is for building and delivering applications shared across a range of network embedded devices. Modular design of J2ME architecture makes it possible for an application to be built according to the constraints of a particular embedded device.
RTSJ is considerably a well structures specification for real-time systems but there a lot of issues have to be resolve. Memory management in RTSJ is totally changed with memory areas and Asynchronous transfer of control is another remarkable implementation. Although RTSJ provide increased func-
tionality indeed there are still room to improve in the following areas. RTSJ implement only fixed priority based scheduling, there is need for other scheduling techniques like EDF. Importance Parameters are provided to scheduling parameters and but still there is need for differentiating the safety-critical and non safety-critical systems. Real-time clock is good addition to RTSJ but still a clock is need to measure the consumed execution time of schedulable objects. The interrupt handling mechanism is too complex and increase system overheads and should needed to be modified.
References


28


[29] Martin Schoebel, Restrictions of Java for Embedded Real-time Systems