

Simplifying the dualized threading model of RTSJ

李昌术

lichsh06@sei.pku.edu.cn

2009-10-09

About The Paper

- Title
 - Simplifying the dualized threading model of RTSJ
- Appeared on
 - 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC)
- Subject
 - Real-time Threading Model

Contents

- Background
 - Current Threading Model in RTSJ v1.1
- Motivation
 - Drawbacks and Limitations in Current Threading Model
- A Proposal by The Authors
 - RealtimeThread++ extension
- Conclusion

Background

- Popularity of java in software industry
 - Portability, Programmer Productivity
 - The initiative of RTSJ(Realtime Specification for Java)
- Threading Model in RTSJ v1.1
 - Dualized threading model
 - Lacks flexibility and the programmer has to deal with complex mechanisms to avoid priority inversion

Threading Model in RTSJ

- Threading
 - Thread, RealtimeThread, NoHeapRealtimeThread,
 - AsyncEventHandler, BoundAsyncEventHandler,
 - DistributedRealtimeThread,
DistributedNoHeapRealtimeThread,
DistributedEventHandler
 - Expedited Real-time Thread (XRTH) by IBM researchers

Threading Model in RTSJ

- Data Sharing & Communication
 - Synchronized
 - Reduce and minimize the priority inversion by using PIP (Priority Inheritance Protocol) and PCP (Priority Ceiling Protocol)
 - RealtimeQueues
 - WaitFreeReadQueue & WaitFreeWriteQueue
 - Signals
 - Limitation: Programmer can not accompany data with signals

The RealtimeThread++ extension

- Observation
 - The Nature of RealtimeThread and NoHeadRealtimeThread are almost the same except for their relationships to the GC
 - The relationship maintained with the GC is decided when the thread is created
- The Idea
 - The relationship between GC and RealtimeThread can be dynamically changed

RealtimeThread++: Programmer's Point of View (1/3)

```
package javax.realtime;
public class realtimeThread extends Thread{
    public static void enterHeap(Runnable r){. . .}
    public static void enterNoHeap(Runnable r){. . .}
    public static boolean isRunningInHeap() {. . .}
        . . .
}
```

Figure 1: New methods introduced by RealtimeThread++

RealtimeThread++: Programmer's Point of View (2/3)

```
01: Runnable r1 = new Runnable(){
02:     public void run(){
03:         System.out.println("Running in heap");
04:     }
05: };
06:
07: Runnable r2 = new Runnable(){
08:     public void run(){
09:         System.out.println("Running in Noheap");
10:         RealtimeThread.enterHeap(r1)
11:     }
12: };
13: RealtimeThread.enterNoHeap(r2);
```

Figure 2: A simple example on the use of two nested `enterHeap` and `enterNoHeap` methods

RealtimeThread++: Programmer's Point of View (3/3)

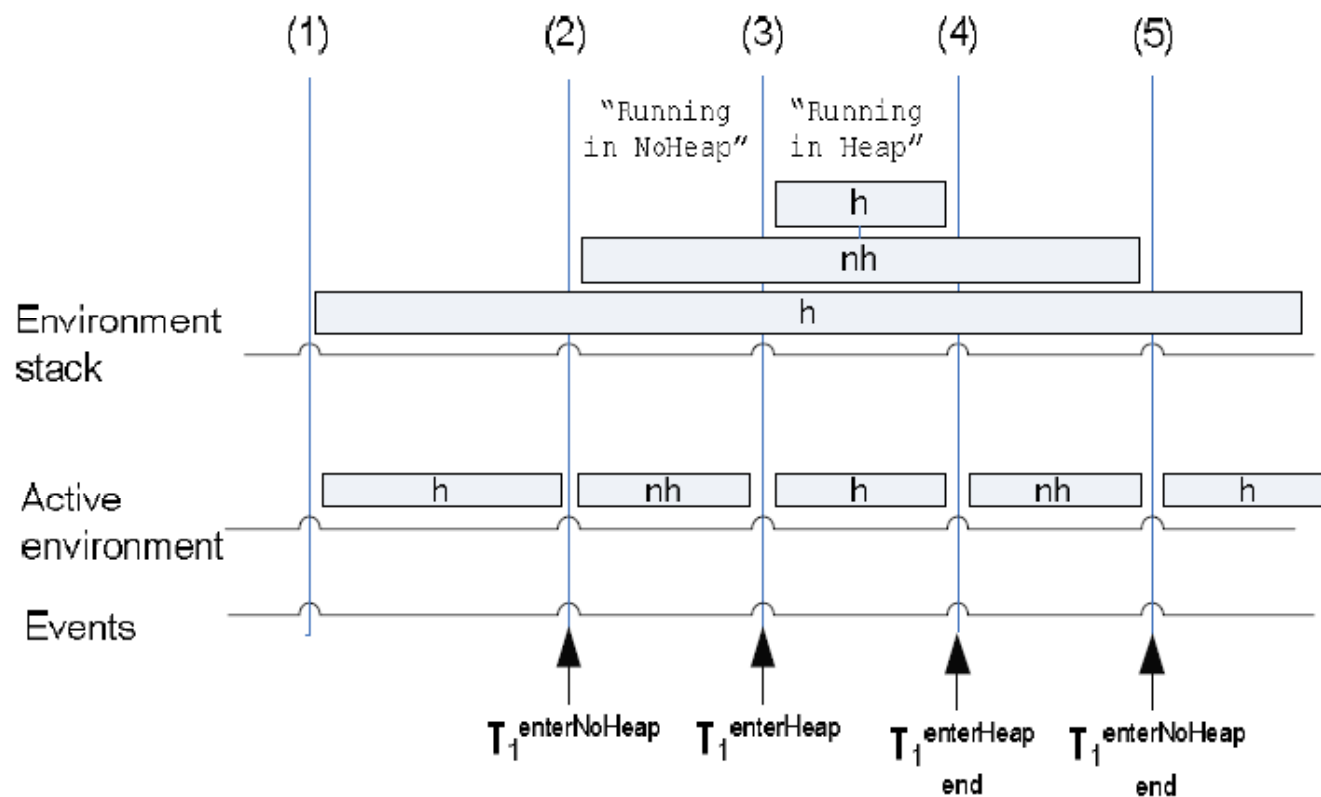


Figure 3: Details on the internal execution of Figure 2 example

RealtimeThread++: Implementor's Point of View

- Guarantee at least two things:
 1. The first is that any thread running under the no-heap constraints never holds a reference to a heap-allocated object.
 2. The second is that the garbage collector does not affect or does not cause interferences on threads running under no-heap constraints.
- The Implementation should:
 1. Manipulating the runtime barriers related to the region model
 2. Activating and deactivating runtime checks
 3. Manipulating the relationship maintained with the garbage collector

Key Advantages of RealTimeThreads++

1. Sharing data between heap and no-heap entities using `enterNoHeap`
2. Improving current RTSJ
 1. Event Model
3. Improving on-going distributed real-time Javaspesification (DRTSJ)

Data Shareing between Heap and No-heap entities 1/3

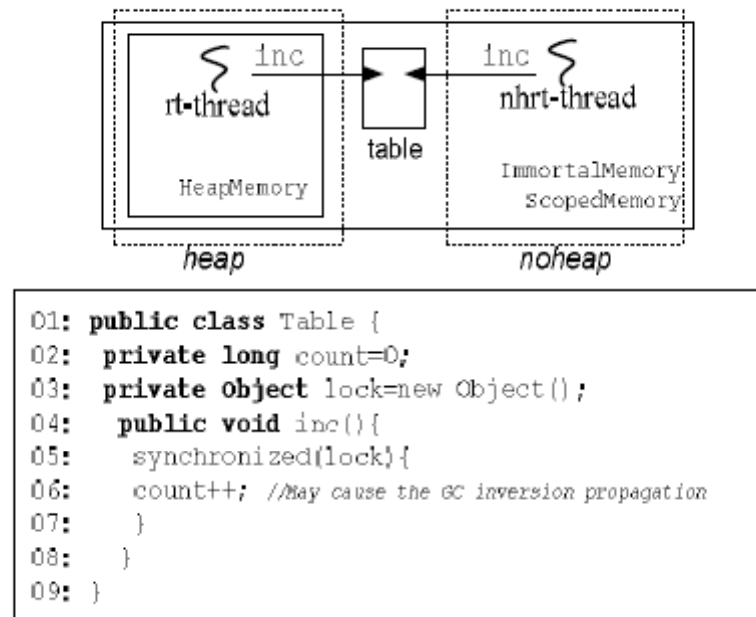
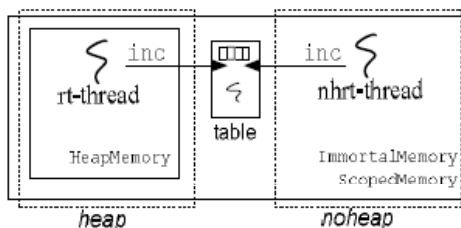


Figure 7: Priority inversion propagation of the garbage collector

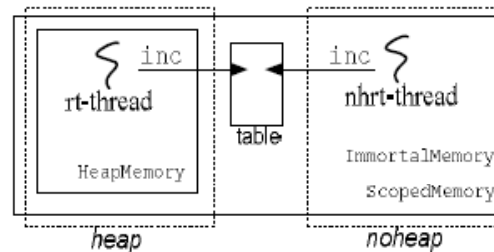
Data Shareing between Heap and No-heap entities 2/3



```
00: public class Table{
01:     private long count=0;
02:     private Object lock=new Object();
03:     private WaitFreeReadQueue wfq=
04:         new WaitFreeReadQueue(5, false);
05:     public Counter(){
06:         th.start();
07:     }
08:     private NoHeapRealtimeThread th=new
09:         NoHeapRealtimeThread(){
10:     public void run(){
11:         do{
12:             wfq.waitForData();
13:             wfq.read();
14:             count++;
15:         }while(true);
16:     };
17:     public void inc(){
18:         wfq.write(lock); //OK
19:     }
20: }
21: }
```

Figure 8. Using queues to leave out the collector

Data Shareing between Heap and No-heap entities 3/3



```
01: public class Table{
02:     private long count=0;
03:     private Object lock=new Object();
04:     private Runnable r=new Runnable(){
05:     public void run(){
06:         synchronized(lock){ //GC OK
07:             count++;
08:         }
09:     };
10:     public void inc(){
11:         RealtimeThread.enterNoheap(r); //Ok
12:     }
13: }
```

Figure 9: Using the RealtimeThread++ extension to leave out the collector

Improving current RTSJ: Event Model

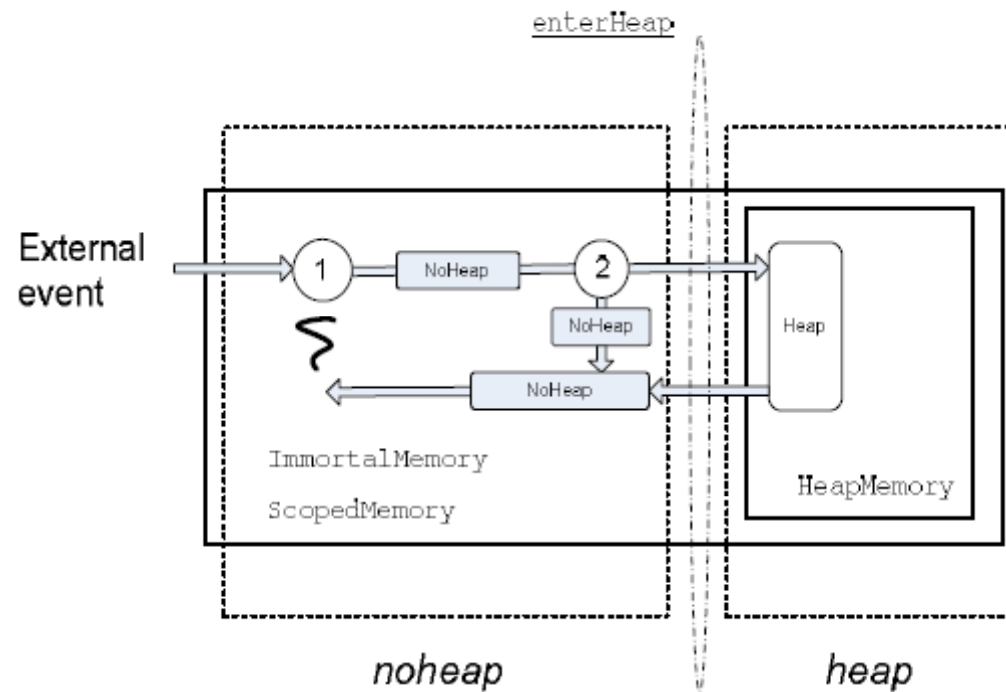


Figure 10: Advanced handling model

Improving on-going (DRTSJ) 2/2

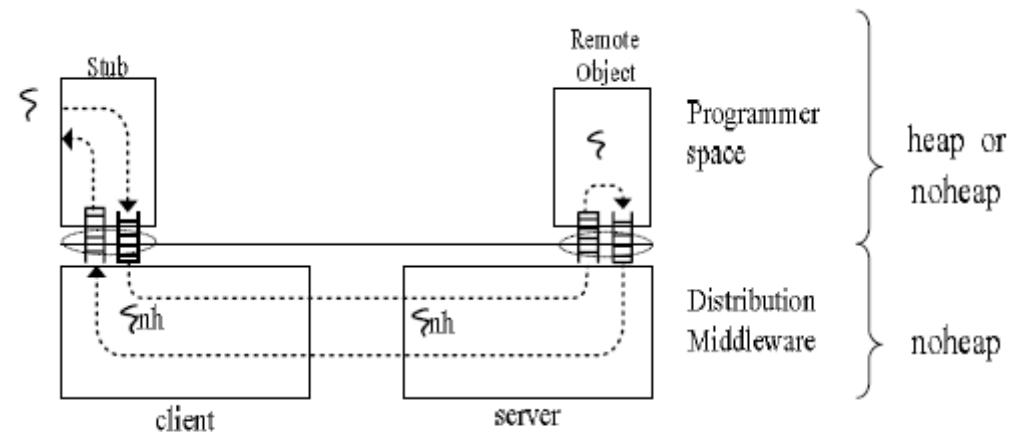


Figure 12: Achieving with the current RTSJ isolation in a communication middleware

Conclusion

- The Current Threading Model in RTSJ is Complicated and lacks Flexibility
- RealtimeThread++ extension
 - Enable real-time thread to dynamically choose the relationship maintained with the GC
- Advantages:
 1. Simplifying data sharing between heap and no-heap entities
 2. offering novel architectures for the on-going real-time Java