

# Gestione della Memoria e Garbage Collection in Real-Time Java

Roberto Aloi  
<roberto.aloi@email.it>

1 giugno 2006

Ai miei genitori  
per non avermi tagliato i viveri

# Indice

|   |             |
|---|-------------|
| <b>Indice</b>   | <b>ii</b>   |
| <b>Prefazione</b>                                       | <b>v</b>    |
| <b>Nota dell'autore</b>                                 | <b>viii</b> |
| <b>1 Java e i sistemi real-time</b>                     | <b>1</b>    |
| 1.1 I sistemi real-time                                 | 1           |
| 1.1.1 Caratteristiche dei sistemi real-time             | 2           |
| 1.1.2 Componenti di un sistema real-time                | 3           |
| 1.1.3 Programmazione real-time                          | 4           |
| 1.2 Il linguaggio Java                                  | 6           |
| 1.2.1 Caratteristiche di Java                           | 6           |
| 1.2.2 Filosofia di Java                                 | 7           |
| 1.2.3 Java e la gestione della memoria                  | 9           |
| 1.3 Java verso i sistemi real-time                      | 9           |
| 1.4 Java VS i sistemi real-time                         | 11          |
| 1.5 I sistemi real-time verso Java                      | 13          |
| 1.5.1 Le <i>Java Native Interface (JNI)</i>             | 15          |
| 1.5.2 Le <i>Real-Time Specification for Java (RTSJ)</i> | 15          |
| 1.5.3 Real-Time Core Extensions (Core)                  | 18          |
| 1.5.4 I Real-Time Garbage Collectors (RTGC)             | 18          |
| 1.5.5 La <i>via dura</i> del PERC                       | 19          |
| 1.5.6 Vie alternative                                   | 20          |
| <b>2 Gestione della Memoria</b>                         | <b>22</b>   |
| 2.1 Definizioni e Concetti di Base                      | 22          |
| 2.2 Allocazione dinamica e memoria heap                 | 22          |
| 2.3 Il problema della frammentazione                    | 23          |
| 2.3.1 Politiche di piazzamento                          | 24          |
| 2.3.2 Liste Separate                                    | 28          |

---

|          |  |           |
|----------|--|-----------|
| 2.3.3    | Sistemi Buddy                                      | 28        |
| 2.3.4    | Liste Concatenate con Indirizione                  | 30        |
| 2.4      | Conclusioni  | 33        |
| <b>3</b> | <b>Garbage Collection</b>                          | <b>34</b> |
| 3.1      | I Fondamenti della GC                              | 34        |
| 3.2      | I compiti del Garbage Collector                    | 35        |
| 3.3      | Definizioni e Concetti di Base                     | 37        |
| 3.4      | Garbage Detection                                  | 37        |
| 3.4.1    | Reference Counting                                 | 38        |
| 3.4.2    | Il Tracing   | 41        |
| 3.4.3    | I Garbage Collector Incrementali                   | 44        |
| 3.5      | Barriere   | 49        |
| 3.6      | Garbage Collector Generazionali                    | 50        |
| 3.7      | GC Hardware-Assisted                               | 54        |
| 3.8      | Conclusioni  | 54        |
| <b>4</b> | <b>La JVM Juice</b>                                | <b>56</b> |
| 4.1      | Perché Juice?                                      | 56        |
| 4.2      | Breve introduzione a Juice                         | 58        |
| <b>5</b> | <b>Gestione della memoria in Juice</b>             | <b>59</b> |
| 5.1      | La memoria heap                                    | 60        |
| 5.2      | La politica di allocazione in Juice                | 61        |
| 5.3      | Accesso ai dati                                    | 68        |
| 5.4      | Calcolo del WCET                                   | 68        |
| 5.4.1    | Allocazione di un oggetto                          | 68        |
| 5.4.2    | Accesso ai dati                                    | 70        |
| 5.5      | I problemi di Juice                                | 71        |
| 5.5.1    | Il numero limitato di fields/elementi degli arrays | 71        |
| 5.5.2    | La frammentazione                                  | 74        |
| <b>6</b> | <b>Garbage Collection in Juice</b>                 | <b>75</b> |
| 6.1      | L'idea di base                                     | 75        |
| 6.2      | Il <i>Pay-Per-Use</i> garbage collector            | 76        |
| 6.2.1    | Write Barrier Marking                              | 77        |
| 6.2.2    | On-new Collection                                  | 78        |
| 6.2.3    | After Collection Marking                           | 79        |
| 6.3      | Predicibilità e calcolo del WCET                   | 80        |
| 6.4      | Ottimizzazioni                                     | 80        |
| 6.5      | Problemi di concorrenza                            | 82        |

---

|          |  |            |
|----------|--|------------|
| <b>7</b> | <b>Analisi Prestazionale</b>                           | <b>86</b>  |
| 7.1      | Tempi di allocazione . . . . .                         | 86         |
| 7.2      | Tempi di accesso ai dati . . . . .                     | 89         |
| 7.2.1    | Tempo di accesso agli elementi di un vettore . . . . . | 89         |
| 7.2.2    | Tempo di accesso ai fields di un oggetto . . . . .     | 93         |
| 7.3      | Frammentazione . . . . .                               | 93         |
| 7.3.1    | Wasted Space . . . . .                                 | 95         |
| <b>8</b> | <b>Conclusioni</b>                                     | <b>99</b>  |
| <b>A</b> | <b>Juice e NUXI</b>                                    | <b>101</b> |
| A.1      | Breve introduzione a NUXI . . . . .                    | 101        |
| A.2      | Un succo di frutta per una noce . . . . .              | 103        |
|          | <b>Ringraziamenti</b>                                  | <b>105</b> |
|          | <b>Elenco delle figure</b>                             | <b>106</b> |
|          | <b>Elenco delle tabelle</b>                            | <b>108</b> |
|          | <b>Bibliografia</b>                                    | <b>110</b> |
|          | <b>Indice analitico</b>                                | <b>112</b> |

# Prefazione

“Over 90 percent of all microprocessors are now used for real-time and embedded applications.”

Tali parole non costituiscono semplicemente la frase di apertura del JTRES 2003[?], il workshop sulle “Java Technologies for Real-Time and Embedded Systems” tenutosi a Catania nel novembre di quell’anno.

Esse rappresentano una grande verità, una realtà che non può venire in alcun modo ignorata o, ancor peggio, dimenticata e che ha assunto un’importanza, se possibile, ancor più grande nel corso di questi ultimi due anni.

Con queste parole siamo in grado di spiegare le attenzioni di un numero sempre crescente di ricercatori, di comunità ed industrie informatiche per l’argomento, di motivare il successo dello stesso JTRES, di comprendere le ragioni di una scelta: quella di realizzare questa tesi.

Queste stesse parole hanno la capacità ed il merito di far riflettere il lettore che, osservando il titolo della tesi, ne è rimasto turbato.

## *Real-Time Java*

Non è un ossimoro. Non un atto di superbia o un’irraggiungibile utopia. E’ una possibilità, che sta divenendo, di giorno in giorno, sempre più concreta.

Rappresenta la necessità di un mondo informatico che, stretto nella morsa del *time to market* e dei costi di sviluppo, deve ricorrere a delle soluzioni alternative, rendendo possibile ciò che sino a poco tempo fa sembrava un’inafferrabile chimera: l’adozione di un linguaggio come Java nei sistemi embedded con caratteristiche di tipo real-time.

La tesi si occupa proprio di spiegare come sia possibile realizzare questo tanto bramato connubio tra Java ed i sistemi in tempo reale.

Dopo un’ampia introduzione al problema, nel quale verranno introdotti proprio i sistemi real-time, il linguaggio Java e le principali problematiche derivanti dalla loro unione, l’attenzione verrà focalizzata su una ben determinata implementazione di Java Virtual Machine, *Juice* (JVM destinata proprio

a sistemi embedded con caratteristiche di tipo real-time, alla realizzazione della quale mi sono dedicato nel corso del mio tirocinio), che verrà analizzata in dettaglio.

Verrà presentata, dunque, la tecnica adottata in *Juice* al fine di garantire predicibilità temporale e possibilità di determinazione del *WCET* (*Worst Case Execution Time*) per le operazioni relative alla gestione degli oggetti (i.e. allocazione, accesso ai dati, etc.).

Tale tecnica, basata sulla suddivisione della memoria *heap* in chunks di dimensione prefissata e sull'introduzione di una struttura gerarchica mutuata dal mondo del file system di Unix per la gestione degli oggetti, consentirà un'esecuzione "controllata" dei bytecodes Java relativi alle operazioni su oggetti ed array, che avverrà in tempi costanti o, comunque, lineari, consentendo, così, un'agevole determinazione di un *upper bound* temporale.

Alcuni risultati notevoli, infine, ottenuti da una serie di test effettuati sulla stessa JVM *Juice*, verranno riportati alla fine della tesi, a suggellare le teorie proposte.

Ma vediamo meglio la struttura della tesi.

Il **Capitolo 1** introduce i sistemi real-time, evidenziandone caratteristiche e peculiarità. In esso spiegheremo come possa rivelarsi interessante l'adozione di un linguaggio come Java, cui è stata riservata una brevissima introduzione, in relazione a degli ambienti in tempo reale. Verranno illustrati gli inevitabili (ed apparentemente insormontabili) ostacoli che una simile scelta presenta. Alcune possibili soluzioni al problema saranno analizzate.

Il **Capitolo 2** affronta il problema della gestione della memoria in maniera generica per una Java Virtual Machine. Saranno analizzate le più diffuse tecniche di gestione della memoria. Verrà introdotto il concetto di *memoria heap*. Particolare attenzione sarà riservata alla struttura degli oggetti Java ed al problema della frammentazione. Cercheremo di evidenziare la strada da seguire nel processo di design e di sviluppo di una JVM "destinata" ai sistemi di tipo real-time.

Il **Capitolo 3** tratta la cosiddetta "Garbage Collection". Dopo una introduzione ai concetti base della Garbage Collection, verranno presi in considerazione i principali tipi di Garbage Collector, di cui saranno analizzati pregi e difetti, anche in relazione ad una loro eventuale adozione in un sistema con caratteristiche di tipo real-time.

Il **Capitolo 4** ci consente di passare da un ambito più generico ad uno decisamente più specifico. Verrà, infatti, presa in considerazione una

ben determinata JVM, *Juice*[?], sviluppata “just for fun” secondo le specifiche di *Sun Microsystem*[?] in linguaggio C e pensata ad hoc per dei sistemi embedded e con caratteristiche di tipo real-time.

Il **Capitolo 5** descrive in modo dettagliato e, speriamo, esauriente le tecniche di gestione della memoria utilizzate nella Java Virtual Machine *Juice*. Particolare enfasi verrà attribuita alla caratteristica struttura di memorizzazione degli oggetti ideata “ad hoc” per *Juice* ed alla politica di allocazione per gli *Java Objects*.

Il **Capitolo 6** è dedicato al garbage collector di *Juice*, battezzato dai suoi autori *Pay-Per-Use Garbage Collector*. Di esso verranno analizzati caratteristiche ed algoritmi di funzionamento e saranno studiati pro e contro. Alcune possibili ottimizzazioni per il miglioramento delle caratteristiche di predicibilità in relazione al suo utilizzo nell’ambito dei sistemi real-time verranno presentate.

Il **Capitolo 7** è fondamentalmente costituito da alcuni grafici, accompagnati da un ricco commento, che rappresentano il risultato di analisi e statistiche relative a *Juice*. Particolare attenzione verrà riservata ai dati relativi al grado di frammentazione interna del sistema, ai tempi di allocazione/collezione degli oggetti ed ai loro rispettivi limiti superiori, al tempo di accesso ai dati, etc..

Il **Capitolo 8** contiene un piccolo riassunto delle principali *conclusioni* cui siamo giunti nel corso dell’intera tesi. In esso verranno riportati i principali pro e contro dell’approccio adottato, già analizzati nel corso dei vari capitoli e qui riuniti per fornire al lettore una più corretta visione d’insieme.

L’ **Appendice A** consiste in un piccolo approfondimento sul processo di integrazione tra *Juice* e NUXI<sup>1</sup>[?], “An operating system in a nutshell”, sistema basato su micro-kernel, progettato per piattaforme Intel-based x86 e destinato a sistemi embedded e con caratteristiche di tipo real-time.

*Roberto Aloï*

---

<sup>1</sup>Liberamente scaricabile all’indirizzo: <http://nuxi.iit.unict.it>