La specifica Java Real Time

Il crescente sviluppo di applicazioni in tempo reale ha richiesto l’introduzione di linguaggi di programmazione di più alto livello anche per questi sistemi, con il fine di poter gestire meglio la loro crescente complessità.

Java, considerando anche la sua larga diffusione nei sistemi embedded, è uno dei linguaggi più appetibili per la nuova generazione di applicazioni in tempo reale. Non essendo nato per questi peculiari scenari applicativi, tuttavia, occorre apportare pesanti modifiche alla sua macchina virtuale affinchè sia efficacemente utilizzabile in questi sistemi.

La specifica RTSJ (Real Time Specification for Java), successivamente implementata da vari vendors, propone estensioni e modifiche alla Java Virtual Machine standard per renderla adatta ad sistemi in tempo reale.

Nella sezione storia si mostrano le tappe che hanno portato dalla prima stesura della specifica, fino all’ implementazione da parte di Sun utilizzata in questa tesi; sucesivamente si mostreranno i principi che stanno alla base della realizzazione della specifica. Dopodichè si procederà ad illustrare come sono stati risolti i problemi che rendevano java inutilizzabile per sistemi in tempo reale, con particolare riferimento all’inizializzazione ed alla compilazione delle classi, alla gestione della memoria ed all’attività di garbage collection, alle politiche di scheduling per concludere col le tematiche relative alla sincronizzazione fra thread. Infine si mostrerà il contenuto del package javax.realitme, il quale contiene classi ed interfacce necessarie per realizzare un’applicazione real – time in Java.

# Storia

Alla fine degli anni novanta il NIST (National Institute for Standards and Tecnologies), un'agenzia del governo degli Stati Uniti d'America che si occupa della gestione delle tecnologie attraverso il lavoro con l'industria per sviluppare standard, tecnologie e metodologie che favoriscano la produzione e il commercio ha coordinato un gruppo di esperti con lo scopo di raccogliere le linee guida ed un primo insieme di requisiti per le estensioni real-time di Java.

Successivamente la prima release della specifica RTSJ (Real Time Specification for Java) venne pubblicata nel 2002.

La prima versione di Sun venne lanciata nel 2005. La versione attuale, la 2.2, risale al 2009.

# Principi guida

La specifica RTJS si ispira ai seguenti principi:

**Applicabilità a particolari ambienti:** La specifica non include caratteristiche che ne limitino l’utilizzo ad ambienti particolari come Java Micro Edition o particolari versioni del JDK, al contrario lo scopo degli implementa tori deve essere quello di allargare il più possibile l’applicabilità dei loro prodotti

**Compatibilità**  La spcifica non deve impedire l’esecuzione di programmi Java non real time sulla nuova virtual machine: su di essa devono poter girare contemporaneamente sia applicazioni in tempo reale sia applicazioni java standard

**Write once, run everywere**  pur riconoscendo l’importanza dell’approccio write once, run everywere, data la difficoltà di mantenerlo su sistemi real time, si è deciso di non sacrificare la predicibilità dell’esecuzione in favore della portabilità

**Estendibilità** La specifica deve permettere l’utilizzo delle tecniche più conosciute per la gestione dei sistemi real time. Al contempo, deve permettere la futura implementazione di nuove tecniche e nuovi algoritmi

**Nessuna estensione sintattica** Allo scopo di semplificare il lavoro degli sviluppatori non sono stati introdotte nuove keyword e non è stata fatta alcuna estensione sintattica al linguaggio Java.

**Libertà di implementazione** Con lo scopo di permettere alle aziende implumentatrici di venire incontro alle specifiche esigenze dei propri clienti,la specifica riconosce che le varie implementazioni possono divergere in un gran numero di decisioni, quali, ad esempio, l’uso di diversi algoritmi per eseguire lo stesso scopo o trade – off tra l’occupazione di memoria e velocità di esecuzione. In ogni caso la specifica si propone di non vincolare in nessun caso all’uso di uno specifico algoritmo, ma si limita a descrivere i requisiti semantici che le operazioni devono rispettare.

# Gestione della memoria e garbage collection

Molti sistemi in tempo reale hanno a disposizione solo una quantità limitata di memoria, ciò è dovuto a considerazioni relative a problematiche di costo o a vincoli di natura fisica (dimensioni, potenza, peso…). Inoltre possono essere disponibili tipi diversi di memoria (con differenti caratteristiche d'accesso) può essere necessario far sì che certi tipi di ritardo risiedano esclusivamente in una certa zona di memoria.In certi scenari è quindi necessario controllare come la memoria viene allocata in modo da poterla usare efficientemente.

Java real - time permette di definire delle aree di memoria nelle quali allocare oggetti al di fuori dell'heap tradizionale. Quando si entra in una aree di memoria tutti gli oggetti vengono allocati all'interno di questa area. Queste aree di memoria sono la ScopedMemory e l’ImmortalMemory.

La zona di memoria ScopedMemory è stata pensata per avere un tempo di vita ben specificato: un reference counter è associato ad ogni scopedMemory e tiene traccia di quante entità real - time stanno attualmente utilizzando la zona di memoria; quando il suo valore passa da uno a zero tutti gli oggetti residenti nella scopedMemory vengono distrutti e la memoria viene liberata. Java real - time prevede due tipi di ScopedMemory: VtMemory, dove le allocazioni richiedono un tempo variabile e LTMemory, dove le allocazioni di memoria richiedono un tempo proporzionale alla dimensione dell’oggetto da allocare.

L’ immortal memory è una zona di memoria che non è mai coinvolta nelle operazioni di garbage collecting. Questa zona di memoria deve il suo nome al fatto che questa zona viene liberata solamente quando termina l’applicazione. Esiste una sola zona di immortal memory che, quindi, è condivisa da tutti i thread dell’applicazione. La gestione di questa zona di memoria è affidata all’utente, che deve provvedere a finalizzare gli oggetti non più utilizzati in modo da liberare la memoria da essi occupati.

Java real time mette snche a disposizione l’astrazione di memoria fisica, consentendo si mappare i dati in zone con indirizzo di memoria specificato. Unito all’utilizzo di clasi che facilitano la scrittura e la lettura di dati tipati, ciò serve qualora si voglia controllare dove i dati vengono memorizzati, cosa utile nel caso si voglia ottimizzare un sistema utilizzando memorie con caratteristiche differenti le une dalle altre.

Per quanto riguarda il garbage collector la specifica non impone nessuna strategia o nessun algoritmo in particolare. Si limita a prevedere che il garbage collector del sistema real - time, qualunque esso sia, abbia una funzionalità tale da non compromettere la predicibilità dell'esecuzione dei processi real - time.

# Tempo ed orologi

in un sistema real - time, misurare con precisione, efficacia ed efficienza il tempo è fondamentale. La specifica Java real - time viene incontro allo sviluppatore fornendo una serie di classi di supporto per esprimere misure temporali. In figura vengono mostrate le principali classi del package javax.realtime coinvolte nel misurare il trascorrere del tempo.

La classe Clock è la classe base per tutti gli orologi presenti nel sistema: in ogni sistema, infatti, è presente almeno un orologio: l'orologio di sistema con precisione al nanosecondo. Tuttavia, per ragioni di efficienza, la specifica prevede che possono esistere altri orologi con precisioni differenti. Ogni misura di tempo, sia relativa che assoluta, è associata ad un orologio. Se non specificato diversamente l'orologio associato è quello di sistema.

La classe HighResolutionTime è la classe madre di tutte le misure di tempo. Questa classe, in sostanza, fornisce le funzionalità per comparare misure di tempo e per estrarre da esse i valori di millisecondi e nanosecondi.

La classe AbsoluteTime è deputata a rappresentare uno specifico istante nell'asse temporale ossia un riferimento temporale assoluto. Oltre le funzionalità ereditate dalla classe madre, questa classe espone una serie di metodi che permettono di sommare o di sottrarre ad un AbsoluteTime sia un RelativeTime che un altro AbsoluteTime, ottenendo come risultato un RelativeTime o un AbsoluteTime.

La classe RelativeTime serve per esprimere un tempo relativo. Di conseguenza viene usata per esprimere periodi o differenze tra istanti di tempo differenti. Anche la classe RelativeTime, oltre ai metodi ereditati dalla classe madre, espone una serie di metodi che consentono di sommare o di sottrarre altre misure temporali.

# Scheduling

L’attività di schedulingdi java Real time si basa sull’idea di associare ad ogni oggetto schedulabile una serie di parametri in grado di caratterizzarne l’esecuzione.

# Sincronizzazione

L’implementazione di Sun

# Inizializzazione e compilazione

Come spiegato nel capitolo “perché Java non è una piattaforma valida per i sistemi real time” i meccanismi di lazy initialization e di just in time compilation previsti per la java virtual machine standard sono inadatti per i sistemi in tempo reale in quanto possono introdurre dilatazioni dei tempi di esecuzione di un’applicazione in momenti impredicibili.

La soluzione ideata per il sistema Java real – time consiste nel fornire alla virtual machine una lista degli elementi da inizializzare e da compilare prima di far partire un’applicazione. In questo modo non sarà più ritardata da eventuali fasi di compilazione o di inizializzazione durante la sua esecuzione.

Per quanto riguarda la compilazione è possibile indicare un file che contiene la lista delle classiche devono essere inizializzate prima che inizi l’inizializzazione tramite l’opzione -XX:PreInitList=<preinit-file-name> . E’ anche possibile far creare il file automaticamente dal sistema grazie all’opzione -XX:+RTSJBuildClassInitializationList. Questa opzione fa si che la Virtual Machine generi un file che contiene la lista delle classi delle classi referenziate durante l’esecuzione. Il file viene generato in maniera incrementale, ciò significa che , se in una successiva esecuzione dell’applicazione viene referenziata una classe non utilizzata in precedenza, questa viene inserita in fondo al file già formato. A titolo di esempio, si mostra uno stralcio del file in questione

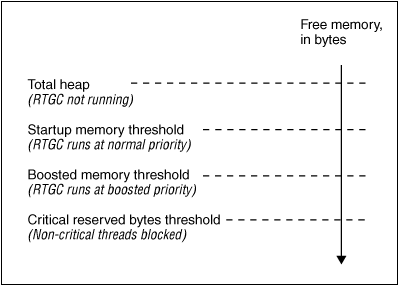
# Garbage Collector

Occorre ricordare come l’attivita di Grabage Collection sia fondamentale per preservare l a funzionalità di un sistema Java: permette di riciclare la memoria altrimenti occupata da oggetti non più utilizzati e di metterla a disposizione del sistema per l'allocazione di nuovi oggetti.

Per quanto importante, in un sistema real - time occorre che questa attività non vada a modificare sensibilmente i tempi di esecuzione delle applicazioni in corso con il rischio che queste sfornino i vincoli temporali a loro assegnati.

Per questo motivo Java real - time prevede l'introduzione di un nuovo garbage collector in grado di coesistere con le applicazioni real - time: può essere infatti impostato affinché non eserciti preemption su i processi real - time e, in caso di sistemi multiprocessore, affinché lavori su una sola cpu senza bloccare l'esecuzione dei processi di Java sugli altri processori.

Entrando nello specifico, come mostrato nella figura a fianco, il garbage collector ha tre modalità di funzionamento, basate sulla quantità di memoria libera residua:

* Finchè la quantità di memoria libera nell'heap resta sopra la soglia di StartupMemory non viene eseguita nessuna azione di garbage collecting.
* Quando la memoria libera scende sotto la soglia di startup mermory il garbage collector alla priorità normale, quando l'opera a questa priorità e il garbage collector blocca solamente i thread non real - time.
* Se la memoria libera scende oltre la soglia di BoostedMemory, il garbage collector esegue ad una priorità maggiore rispetto a quella normale detta BoostedPrioriy. Con il garbage collector a questa priorità vengono bloccati anche alcuni il thread real – time (quelli con priorità inferiore alla boosted priority) che il sistema di conseguenza considera non critici.
* Se la memoria libera scende ulteriormente fino ad occupare anche la quantità riservata ai processi critici il garbage collector entra nella modalità deterministica. In questa modalità il garbage collector lavora sempre con priorità boosted, tuttavia le richieste di memoria di thread non critici (con priorità inferiore a quella critica) vengono bloccate finché la quota di memoria libera non torna sopra alla soglia. In questo modo si cerca di garantire la funzionalità dei processi più importanti per il sistema, in quanto solo i processi che eseguono a livello critico sono in grado di allocare memoria proveniente dalla quota riservata appositamente per questi processi.

Tutti i parametri descritti finora (le priorità e le soglie di memoria) vengono calcolate automaticamente dalla virtual machine; in alternativa si possono indicare manualmente tramite opzioni.

La figura sottostante mostra un esempio di funzionamento del garbage collector, in particolare si evidenzia come quando la memoria libera scende sotto la soglia di boosted memory threshold, il garbage collector esegue a priorità maggiore esercitando preemption anche su thread real - time, ma non critici per il sistema.

