Introduzione a 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.

In questa sezione STORIA, 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.

# 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

# 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.

# Scheduling

# Sincronizzazione

# Il package javax.realtime