HarmonyArchitectureItalian

Architettura proposta di Harmony

Basato su "*Harmony: Il progetto J2SDK di Apache*" (http://www.jguru.se/jguru/control/Developers/Harmony) Con alcune modifiche prese dalla *Harmony mailing list* (http://mail-archives.apache.org/mod_mbox/incubator-harmony-dev). Siete pregati di inviare ogni commento e discussione là, e di includere le modifiche quando si raggiunge il consenso.

Harmony è un progetto della Apache Fundation mirato a creare una implementazione stabile ed opensource del Java 2 SDK. Non è un application server Java 2 Enterprise Edition (J2EE), nè gestisce aplicazioni Java 2 Micro Edition (J2ME).

Requisiti Iniziali

I principali requisiti del Progetto Harmony sono evidenti e sono dati dal JDK Technology Compatibility Kit (TCK). Vi sono pero diversi altri requisiti:

1. Semplicità di utilizzo e porting dei servizi nativi dell'OS. Questo requisito può essere gestito introducendo un OS Abstraction Layer (OAL) che è l'unico punto di contatto tra la JVM ed l'OS nativo, o le tool library e l'OS nativo rispettivamente. E' vitale che l'OAL sia snello e che i suoi confini possano essere spostati all'interno della JVM per portare performance ottimali sul maggior numero possibile di OS. 2. Ridurre la confusione nelle librerie richieste e promuovere un'aumento dei tool del JDK. Questo requisito può essere soddisfatto usando una libreria comune per tutti i tools. Riducendo il numero delle dipendenze top-layer e mantenendo un'API pulita, possiamo aumentare il troughput degli sviluppatori nel progetto puntando a fornire, diciamo, versioni GUI dei tools standard del JDK.

Siete pregati di fornire ulteriori suggerimenti per i requisiti - e consentiteci di mantenerli misurabili da ora in poi. 🙂

Architettura Iniziali e Sottoprogetti

Propongo un'architettura iniziale ed alcuni sottoprogetti per Harmony delineato di seguito. Ogniuno dei (sotto)progetti dovrebbe aderire al modello di progetto standard di Apache, a meno che una particolare ottima ragione non ci suggerisca di fare altirmenti. Prendendo spunto dalle discussioni iniziali sulla mailing list di Harmony, ho creato alcune bozze della struttura del progetto.

http://www.jguru.se/jguru/Channel_Html/generic/images/developers/harmony/overall.jpg

La principale struttura interna del progetto della JVM e della class library è illustrato di seguito. Secondo le discussioni sulla lista di sviluppo:

- **Il progetto classpath** mira a creare un'implementazione delle API standard di Java che sia opensource e cleanroom. GNU Classpath (http://www.gnu.org/software/classpath) potrebbe venire utilizzato per colmare le richieste del progetto classpath, ammesso che le licenze possano venire rispettate da entrambe le parti, e che vengano dati gli appropriati riconoscimenti.
- **Il progetto JVM** mira a creare un'implementazione cleanroom ed opensource della Java Virtual Machine. Ci sono state discussioni per implementarla in Java, C/C++, o se usare un framework nel quale ogni componente possa venire scritto in diversi linguaggi. Attualmente ci manca una buona scelta per il codice di base della JVM. Potenzialmente dovremo reimplementarla da zero.
- **Il progetto OS Abstraction Layer (OAL)** ha lo scopo di definire ed implementare un sottile strato wrapper che connette la JVM ed il classpath ai servizi del sistema operativo nativo. Questo strato è l'unica parte ad interagire direttamente con l'OS per le versioni non di sviluppo della JVM. Attualmente l'Apache Portable Runtime (APR) (http://apr.apache.org) sembra la scelta migliore.
- The development facilities project aims at creating APIs and tools to facilitate debugging and development-time profiling in order to track state within the JVM. Development facilities should be stripped from the build for non-development releases. Development facilities must be adapted to the JVM, and will therefore be defined pending the choice of JVM.

JVM subprojects

The main Harmony project in terms of volume is likely the JVM project as illustrated in the overall project structure above. We should therefore split the development effort of the JVM project into a few subprojects. as illustrated below. http://www.jguru.se/jguru/Channel_Html/generic/images/developers/harmony/jvm_projects.jpg

It seems reasonable that we can identify (at least) three subprojects with a number of development goals inside the JVM. These are

- ClassLoading. This project provides facilities for loading Java bytecode streams, verifying integrity and optionally performing pre-runtime optimizations. (I.e. all activities before any bytecode is actually executed).
    - Loader. Loads bytecode streams as requested by a java.lang.ClassLoader
    - Verifier. Verifies the integrity of the loaded bytecode stream.
    - Optimizer. Performs simple optimizations on loaded bytecode in order to reduce its size or rearrange operations as required by the native processor or OS.
- Runtime. This project provides facilities for executing Java code already verified by the ClassLoading facilities. This includes performing runtime optimizations if applicable, and interfacing with requested native OS services as requested by the Java class.
    - MemoryManager. Garbage Collection, Memory allocation and automatic Object storage management [i.e. full heap management].
    - ThreadManager. Manages memory and services on a per-thread basis.
    - JIT. Standard Just-In-Time compiler converting pre-optimized bytecode to executable machine code for the platform in question.
    - Native Interface. The Java Native Interface connection to the services of the native OS.
- Statistics. No JVM is complete without facilities to measure and tune its performance in runtime. This is the task of the statistics project.
    - Profiling. Standard profiling interface which provides information about resource allocation and usage for the running JVM. I recommend that we provide facilities to modify runtime parameters on the fly through this interface.
    - Crash Talkback. It is important to receive crash dump information from JVMs to provide statistical information about runtime bugs, faulty optimizations or other such data. The talkback services sends this information over the internet to our statistics collection server.

As usual, these may not be the optimal or final project or development streams. Your comments are welcome.

Development Facilities Subprojects

Development facilities exist to enhance the overall development process, and should be stripped from the build in production releases (or, at least, non-development and -debug releases). Development facilities may explore state or data flow in any layer of the JVM, class library or OAL.

http://www.jguru.se/jguru/Channel_Html/generic/images/developers/harmony/devfacilities_projects.jpg

I recommend (at least) the following projects under the Development Facilities umbrella.

- Internal Tools. This project provides development facilities checking state and delivering development state profiling information in the running JVM.
  - Trace. Investigates state in all parts of the JVM; a "debugger-on-steroids" interface.
  - Thread Profiler. Collects statistics information from all running threads in the JVM and provides mechanisms to alter Thread runtime parameters and/or priorities.
- Analyzer. Creates simple-to-understand reports from reports retrieved from the net or local JVM.
  - Talkback Analyzer. Reads a set of talkback reports and collects relevant statistics from them. This developer stream should perhaps not be located in the Development Facilities project, but has been placed here for now and for lack of a better place.
  - Introspection GUI. The biggest problem with obtaining profiling information from several current JVMs is the complexity of the information interface. I propose that we create a slim and simple GUI for the Harmony development facilities where such information may be visualized in a simple manner.

Again, feedback is welcome - but use the Harmony mailing list for the feedback.

J2SDK Toolkit Architecture

In addition to the JVM and classpath, the standard JDK tools must be implemented for Harmony. These tools includes:

- An appletviewer
- A Browser Java plug-in
- A Java compiler (javac). Possible candidates is Eclipse and Jikes.
- A jar utility & signer
- A javadoc utility
- more?

These utilities can either be written in Java using the classpath project, or written in C/C++. In the later case we should, to promote reuse and facilitate deployment, strive for a single distribution library (called Common Tools Library or CTL in the image below) on top of the OAL from the JVM project.

http://www.jguru.se/jguru/Channel_Html/generic/images/developers/harmony/tools.jpg

The overall tool goal is to permit or promote augmented clients in addition to the standard-named ones. I believe most of us would like to see an augmented keytool that could - say - provide native OpenSSL CA management directly to the default keystore. Given a decent Swing-based GUI for the job, key management annoyances would likely be greatly reduced compared to today's text-based equivalent. Alas, we would strive for the situation illustrated below:

http://www.jguru.se/jguru/Channel_Html/generic/images/developers/harmony/tools_example.jpg