CoroutineJSR

Section 1. Identification

Submitting Member:
Name of Contact Person:
E-Mail Address:
Telephone Number:
Fax Number:

Specification Lead:
E-Mail Address:
Telephone Number:
Fax Number:

Initial Expert Group Membership:

Supporting this JSR:

Section 2: Request

2.1 Please describe the proposed Specification:

The performance of many applications depends heavily on the efficiency of context switch operation of their underlying threading model. Java applications almost exclusively use java.lang.Thread (or some abstracted form thereof) as threading model, which is a heavyweight, operating system level facility.

Fast, user-level context switching in the form of coroutines would be beneficial for any application that has to deal with concurrent sessions or interactions or other types of program units that need to be interleaved.

Many dynamic languages that have JVM-based runtime environments already support coroutine-like features. Thus, coroutine support would also help these runtime environments provide an efficient implementation of these features.

2.2 What is the target Java platform? (i.e., desktop, server, personal, embedded, card, etc.)

Coroutines can be implemented very efficiently. They can be applied to all platforms that at the moment support threading, and could be used as a lightweight alternative to threading on platforms that don't.

2.3 The Executive Committees would like to ensure JSR submitters think about how their proposed technology relates to all of the Java platform editions. Please provide details here for which platform editions are being targeted by this JSR, and how this JSR has considered the relationship with the other platform editions.

2.4 Should this JSR be voted on by both Executive Committees?

2.5 What need of the Java community will be addressed by the proposed specification?

- JVM-based language implementation need coroutines to efficiently implement coroutine-like language features.
- Server applications can use coroutines to keep more sessions alive, which leads to less latency and better resource utilization.
- Coroutines provide an easy way to inverse algorithms, e.g. turn a callback-based algorithm into one that can be polled for the next result.
- Coroutines are a natural control abstraction for numerous problems like scanner/parser combinations.
- It is often convenient to write algorithms in multiple units that perform synchronous communication.
- These non-parallel algorithms can avoid exposing parallelism by using coroutines instead of threads.
- Coroutines can be used to hide the inherent complexity of asynchronous I/O and to avoid having to split the program into small pieces.

2.6 Why isn't this need met by existing specifications?

The current java.lang.Thread system is designed for preemptive, operating-system level context switching. This inherently is orders of magnitude slower that user-level context switching.

2.7 Please give a short description of the underlying technology or technologies:

Coroutines as a programming concept have been around since the early days of computer science, the term "coroutine" itself was coined in 1963 by Melvin Conway (http://dx.doi.org/10.1145%2F366663.366704). They are available in numerous programming languages and runtime environments.

From a language design perspective, coroutines are a generalization of subroutines. When they are invoked, they start execution at their first statement and they can subsequently transfer control to some other coroutine (via an operation often called \texttt{yield}).

The state of the local variables and the expression stack at the transfer point is preserved.

When control is transferred back, the coroutine resumes the preserved state and continues to run from the point where it relinquished control.

Coroutines can also be seen as a limited form of continuations, called \textit{one-shot continuations}. 

From a VM perspective, coroutines can be seen as a limited version of threads that happen to transfer control only at some user-specified points in the program. For the sake of this document, this is the most useful definition, because it can be used to infer sensible memory model and runtime semantics for coroutines.

Coroutines have been seen as an inferior alternative to full-blown threads, because of the manual context switching. However, there are situations in which manual context switching makes sense or is even desired.

2.8 Is there a proposed package name for the API Specification? (i.e., javapi.something, org.something, etc.)
2.9 Does the proposed specification have any dependencies on specific operating systems, CPUs, or I/O devices that you know of?
2.10 Are there any security issues that cannot be addressed by the current security model?
2.11 Are there any internationalization or localization issues?
2.12 Are there any existing specifications that might be rendered obsolete, deprecated, or in need of revision as a result of this work?
2.13 Please describe the anticipated schedule for the development of this specification.

We hope to deliver the final specification, reference implementation, and TCK in XXX. A rough estimate of the schedule would be

XXX Expert group formed
XXX Expert draft
XXX Early Draft review
XXX Public Review
XXX Proposed Final Draft
XXX Final release (aligned with Java EE 6 dates)

2.14 Please describe the anticipated working model for the Expert Group working on developing this specification.

The primary means of communication will be email and conference calls. Face-to-face meetings will be scheduled if needed. We will solicit feedback from the community. See section 2.15 for the transparency measures we will use.

2.15 It is important to the success of the community and each JSR that the work of the Expert Group be handled in a manner which provides the community and the public with insight into the work the Expert Group is doing, and the decisions that the Expert Group has made. The Executive Committees would like to ensure Spec Leads understand the value of this transparency and ask that each JSR have an operating plan in place for how their JSR will address the involvement of the community and the public. Please provide your plan here, and refer to the Spec Lead Guide for a more detailed description and a set of example questions you may wish to answer in your plan.

The Expert Group will publish early drafts for review by the community at convenient points during the development of the specification. The Expert Group will solicit feedback from the community via public fora (e.g., public presentations, blogs, and other online postings).

2.16 Please describe how the RI and TCK will be delivered, i.e. as part of a profile or platform edition, or stand-alone, or both. Include version information for the profile or platform in your answer.

2.17 Please state the rationale if previous versions are available stand-alone and you are now proposing in 2.13 to only deliver RI and TCK as part of a profile or platform edition (See sections 1.1.5 and 1.1.6 of the JCP 2 document).

2.18 Please provide a description of the business terms for the Specification, RI and TCK that will apply when this JSR is final.

Section 3: Contributions

3.1 Please list any existing documents, specifications, or implementations that describe the technology. Please include links to the documents if they are publicly available.

- Prototype implementation: part of the Da Vinci Machine Project (MLVM) [http://wikis.oracle.com/display/mlvm/Home](http://wikis.oracle.com/display/mlvm/Home)

3.2 Explanation of how these items might be used as a starting point for the work.

The thesis should provide a starting point for discussions about the features that a coroutine implementation should provide, and the prototype implementation can be seen as a proof-of-concept that can be used to explore the API design and performance characteristics.

Section 4: Additional Information (Optional)

4.1 This section contains any additional information that the submitting Member wishes to include in the JSR.