**Special Issue on Java Technologies for Real-Time and Embedded Systems JTRES2013**

*Geoffrey Fox, School of Informatics and Computing, Indiana University, Bloomington, IN, 47401*

This editorial describes a special issue of papers from the 2013 workshop on Java Technologies for Real-Time and Embedded Systems [1]. There are 2 papers in this special issue.

The first paper [2] discusses software locking mechanisms that commonly protect shared resources for multi-threaded applications. This mechanism can, especially in chip-multiprocessor systems, result in a large synchronization overhead. For real-time systems in particular, this overhead increases the worst-case execution time and may void a task set's schedulability. This paper presents two hardware locking mechanisms to reduce the worst-case time required to acquire and release synchronization locks. These solutions are implemented for the chip-multiprocessor version of the Java Optimized Processor. The two hardware locking mechanisms are compared with a software locking solution as well as the original locking system of the processor. The hardware cost and performance are evaluated for all presented locking mechanisms. The performance of the better performing hardware locks is comparable to the original single global lock when contending for the same lock. When several non-contending locks are used, the hardware locks enable true concurrency for critical sections. Benchmarks show that using the hardware locks yields performance ranging from no worse than the original locks to more than twice their best performance. This improvement can allow a larger number of real-time tasks to be reliably scheduled on a multiprocessor real-time platform.

Safety Critical Java (SCJ) is a profile of the Real-Time Specification for Java that brings to the safety-critical industry the possibility of using Java. SCJ defines three compliance levels: Level 0, Level 1 and Level 2. The SCJ specification is clear on what constitutes a Level 2 application in terms of its use of the defined API, but not the occasions on which it should be used. The second paper [3] broadly classifies the features that are only available at Level 2 into three groups: nested mission sequencers, managed threads, and global scheduling across multiple processors. It then explores the first two groups to elicit programming requirements that they support. The paper identifies several areas where the SCJ specification needs modifications to support these requirements fully; these include: support for terminating managed threads, the ability to set a deadline on the transition between missions, and augmentation of the mission sequencer concept to support composability of timing constraints. We also propose simplifications to the termination protocol of missions and their mission sequencers. To illustrate the benefit of our changes, we present excerpts from a formal model of SCJ Level 2 written in Circus, a state-rich process algebra for refinement.

We thank Kelvin Nilsen and Fridtjof Siebert for their work on this special issue.

**References**

1. The 11th International Workshop on Java Technologies for Real-Time and Embedded Systems JTRES 2013, October 9-10, Karlsruhe, Germany
2. Torur Biskopstø Strøm, Wolfgang Puffitsch, and Martin Schoeberl, “Hardware Locks for a Real-Time Java Chip-Multiprocessor”, Concurrency and Computation: Practice and Experience [this issue].

3. Matt Luckcuck, Andy Wellings and Ana Cavalcanti, “Safety-Critical Java: Level 2 in Practice”, Concurrency and Computation: Practice and Experience [this issue].