Scoped Types and Aspects for Real-Time Java

Chris Andreae[3], Yvonne Coady[1], Celina Gibbs[1],
James Noble[3], Jan Vitek[4], and Tian Zhao[2]

[1] University of Victoria, CA
[2] University of Wisconsin–Milwaukee, USA
[3] Victoria University of Wellington, NZ
[4] Purdue University, USA

Abstract. Real-time systems are notoriously difficult to design and implement, and, as many real-time problems are safety-critical, their solutions must be reliable as well as efficient and correct. While higher-level programming models (such as the Real-Time Specification for Java) permit real-time programmers to use language features that most programmers take for granted (objects, type checking, dynamic dispatch, and memory safety) the compromises required for real-time execution, especially concerning memory allocation, can create as many problems as they solve. This paper presents Scoped Types and Aspects for Real-Time Systems (STARS) a novel programming model for real-time systems. Scoped Types give programmers a clear model of their programs' memory use, and, being statically checkable, prevent the run-time memory errors that bedevil models such as RTSJ. Our Aspects build on Scoped Types guarantees so that Real-Time concerns can be completely separated from applications' base code. Adopting the integrated Scoped Types and Aspects approach can significantly improve both the quality and performance of a real-time Java systems, resulting in simpler systems that are reliable, efficient, and correct.

1 Introduction

The Real-Time Specification for Java (RTSJ) introduces abstractions for managing resources, such as non-garbage collected regions of memory [4]. For instance, in the RTSJ, a series of *scoped memory* classes let programmers manage memory explicitly: creating nested memory regions, allocating objects into those regions, and destroying regions when they are no longer needed. In a hard real-time system, programmers must use these classes, so that their programs can bypass Java's garbage collector and its associated predictability and performance penalties. But these abstractions are far from abstract. The RTSJ forces programmers to face more low-level details about the behaviour of their system than ever before — such as how scoped memory objects correspond to allocated regions, which objects are allocated in those regions, how those the regions are ordered — and then rewards any mistakes by throwing dynamic errors at runtime. The difficulty of managing the inherent complexity associated with real-time concerns ultimately compromises the development, maintenance and evolution of safety critical code bases and increases the likelihood of fatal errors at runtime.

This paper introduces Scoped Types and Aspects for Real-Time Systems (STARS), a novel approach for programming real-time systems that shields developers from many

D. Thomas (Ed.): ECOOP 2006, LNCS 4067, pp. 124–147, 2006.
© Springer-Verlag Berlin Heidelberg 2006

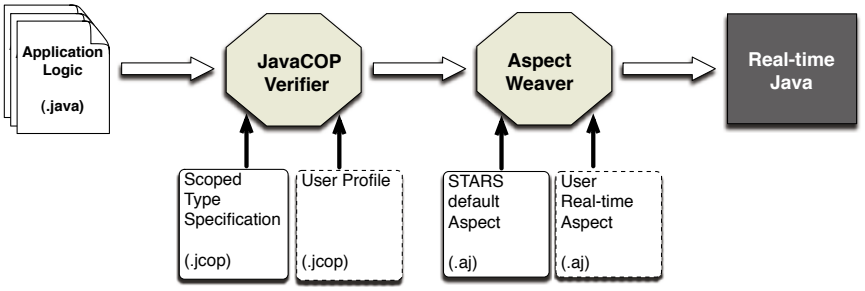

Fig. 1. Overview of STARS. Application logic is written according to the Scoped Types discipline. The JAVACOP verifier uses scoped types rules (and possibly some user-defined application-specific constraints) to validate the program. Then, an aspect weaver combines the application logic with the real-time behaviour. The result is a real-time Java program that can be executed on any STARS-compliant virtual machine.

accidental complexities that have proven to be problematic in practice. Scoped Types use a program's package hierarchy to represent the structure of its memory use, making clear where objects are allocated and thus where they are accessible. Real-Time Aspects then weave in allocation policies and implementation-dependent code — separating real-time concerns further from the base program. Finally, Scoped Types' correctness guarantees, combined with the Aspect-oriented implementation, removes the need for memory checks or garbage collection at runtime, increasing the resulting system's performace and reliability. Overall, STARS is a methodology that guides real-time development and provides much needed tool support for the verification and the modularization of real-time programs.

Fig. 1 illustrates the STARS methodology. Programmers start by writing application logic in Java with no calls to the RTSJ APIs. The code is then verified against a set of consistency rules — STARS provides a set of rules dealing with memory management; users may extend these rules with application-specific restrictions. If the program type checks, the aspects implementing the intended real-time semantics of the program can be woven into the code. The end result is a Real-time Java program which can be run in any real-time JVM which supports the STARS API.

The paper thus makes the following contributions:

1. **Scoped Types.** We use a lightweight pluggable type system to model hierarchical memory regions. Scoped Types is based on familiar Java concepts like packages, classes, and objects, can be explained with a few informal rules, and requires no changes to Java syntax.
2. **Static Verification** via the JAVACOP pluggable types checker [1]. We have encoded Scoped Types into a set of JAVACOP rules used to validate source code. We also show how to extend the built-in rules with application-specific constraints.
3. **Aspect-based real-time development.** We show how an aspect-oriented approach can decouple real-time concerns from the main application logic.
4. **Implementation in a real-time JVM.** We demonstrate viability of STARS with an implementation in the Ovm framework [2]. Only minor changes (18 lines of code in all) were needed to support STARS.