

ABSTRACT

We present the rationale, design, and implementation of Relaxed MultiJava (RMJ), a backward-compatible extension of Java that allows programmers to add new methods to existing classes and to write multimethods. Previous languages supporting these forms of extensibility either restrict their usage to a limited set of programming idioms that can be modularly typechecked (and modularly compiled) or simply forego modular typechecking altogether. In contrast, RMJ supports the new language features in a virtually unrestricted form while still providing mostly-modular static typechecking and fully-modular compilation. In some cases, the RMJ compiler will warn that the potential for a type error exists, but it will still complete compilation. In that case, a custom class loader transparently performs load-time checking to verify that the potential error is never realized. RMJ's compiler and custom loader

As a consequence of MultiJava's insistence on fully modular typechecking, there are several useful forms of extensibility that are simply disallowed. For example, MultiJava does not allow an external method to be

a concrete subclass of `Shape` is loaded that does not override the abstract `area` method declaration, then a load-time verification error will be reported. RMJ's combination of compile-time and load-time checking is sufficient to ensure that all operations are properly implemented. Therefore, a program that passes RMJ's compile-time and load-time

While RMJ supports glue methods belonging to external operations like `area`, it currently does not support

opposite perspecti

The preloader starts by registering all the glue listed in the `rmj.glue` property, just as `RMJClassLoader` does. The preloader then exhaustively explores all classes transitively referenced from the `<Main>` class, ignoring the application's actual flow of control. The preloader performs all the RMJ load-time checks as it visits each class. Even though classes may be visited by the preloader in a different order than in a real execution, and more classes may be visited by the preloader than in a real execution, the preloader is guaranteed to discover all potential load-time hazards of a real execution, with one caveat described below. If the preloader reports a hazard, then the program is not safe to execute.

The visitor design pattern [Gamma et al. 95] is a programming idiom that allows new operations to be added to existing classes without modifying existing code. However, the visitor pattern has several drawbacks that are not shared by open classes in RMJ, as discussed in Section 4.1. Most importantly

- [Cartwright & Fagan 91] Robert Cartwright and Mike Fagan. Soft Typing. *SIGPLAN Notices*, 26(6):278D292, June 1991. Conference on Programming Language Design and Implementation.
- [Castagna 95] Giuseppe Castagna. Covariance and Contravariance: Conßict without a Cause. *ACM Transactions on Programming Languages and Systems*, 17(3):431D447, May 1995.

[Steele Jr. 90] Guy L.