particular, special Spaces are used to allocate JIT-generated code, the Jikes RVM Bootimage, and low-level VM implementation objects such as TIBs. There are 15 Plans (i.e., 15 different GC algorithms) predefined and distributed in MMTk version 3.0. Furthermore, a number of other Plans have been developed and described in the academic literature by the MMTk research community.

MMTk and its hosting runtime system interact through two narrowly defined interfaces that specify what API MMTk exposes to the host virtual machine and what virtual machine services MMTk expects the host to provide. To maintain MMTk's portability to multiple host virtual machines, the build process strictly enforces these interfaces by separately compiling MMTk against a stub implementation of the VM interface. Perhaps the most complex bit of these interfaces is the portion that is used to allow MMTk and Jikes RVM to collaboratively identify the *roots* to be used for a garbage collection. Most of MMTk's Plans represent tracing collectors. In a tracing collector, the garbage collector starts from a set of root objects (typically the program's global variables and references on thread stack frames) and does a transitive closure of their reference fields to find all reachable objects. Reachable objects are considered live and as such are not collected. Any objects that are not reached in the transitive closure are dead; these may be safely reclaimed by the collector and used to satisfy future memory allocation requests. In Jikes RVM, the roots for garbage collection come from registers, the stack, the JTOC, and the boot image. The references in the boot image are determined from the root map, and the root map compresses all of the offsets in the boot image that contain references. To determine the references on the stack and in registers, the method and the location within it are determined in the same manner as is used for the exception model (see the earlier section "Exception Model"). From the method the compiler is determined, and it can give an iterator to MMTk that processes the registers and stack returning the references.

Jikes RVM integration

Jikes RVM integrates with MMTk during the initial creation of object representations, providing iterators to process references, object allocation, and barrier implementations. Object allocation and barriers can influence performance; as MMTk is written in Java, the associated code can be directly linked into the code being compiled for efficiency. Barriers are necessary in garbage collection schemes for a variety of reasons. For example, read barriers are necessary to catch the use of objects that are possibly being copied by a concurrent garbage collector, and write barriers are used in generational collectors where a write to an old object means the old object needs to be considered as a root for collection of the young generation.

Summary

MMTk provides a powerful and popular set of precise garbage collectors. The ease of being able to link together different modules of Jikes RVM and MMTk eases development and reduces overhead, with Jikes RVM inlining parts of MMTk for performance reasons. Writing garbage-collection algorithms in Java allows the garbage collector implementor to ignore what occurs