Dynamic Class Loading Inhibits Performance

Many modern runtime environments, including Java, have the ability to be extended dynamically. This can be useful in an environment where the user wants to plug together different parts of a system; in the case of Java, the components may be downloaded from the Internet. Although useful, this technique can mean the compiler cannot make certain assumptions that it otherwise could if all information about an application were available to it. For example, if a method of an object were being called and there was only one possible class for that object, it would be preferable to avoid doing dynamic method dispatch and directly call the method. There are specific optimizations to deal with this situation in optimizing runtime environments, and we describe them later in "On-stack replacement."

Garbage Collection Is Slower Than Explicit Memory Management

Automatic garbage collection is an advanced area of computer science research. Memory is requested and reclaimed when no longer needed. Explicit memory management requests and then reclaims memory using explicit commands to the runtime environment. Although explicit memory management is error-prone, it is often argued that it is needed for performance. However, this is overlooking the many complications caused by explicit memory management. With explicit memory management, it is necessary to track what blocks of memory are in use and maintain lists for those that are not. When this is combined with many threads concurrently requesting memory, the problem of memory fragmentation, and the fact that merging smaller regions can yield larger regions of memory to be allocated, the job of an explicit memory manager can become complex. The explicit memory manager also cannot move things around in memory—for example, to reduce fragmentation.

The requirements of a memory manager are application-specific, and in the context of a metacircular runtime, a simple JIT compiler doesn't need to perform much memory allocation, so either explicit or automatic garbage collection schemes would work well. For a more sophisticated optimizing compiler, the picture isn't as clear, other than the fact that garbage collection reduces the potential for bugs. For other parts of the runtime system there are further complications, which we describe later in "Magic, Annotations, and Making Things Go Smoothly." Although we haven't been able to refute the claim that garbage collection may be slower than explicit memory management, one thing it definitely improves is the "hack-ability" of the system.

Summary

As development tools are themselves applications, our original question of how best to develop an application becomes self-referential. Managed languages design out faults and improve the productivity of the developer. Simplifying the development model, exposing more opportunities to optimize the application and the runtime, and doing this in a metacircular way allow the developer to gain from the features they introduce without encountering