By Kim Clohessy,
Vice President,
Object Technology, Inc., a wholly owned subsidiary of IBM.

Development Tools for Embedding Java

Computing hardware, network infrastructure, and market expectations have aligned to propel pervasive, embedded computing to the status of "next big thing." The field is poised for explosive growth. Yet significant challenges and obstacles face the developer of embedded systems and threaten to stunt this market. A highly scalable, production environment for developing embedded systems is necessary for the embedded Internet to emerge and for embedded computing to scale to its potential. One way of understanding the process for developing embedded systems is to view embedded systems as comprising two sides: the development tools and the runtime components. This presentation focuses on the development tools, while giving a contextual overview of the runtime components. Following is a discussion of a development environment for creating Java™ based embedded systems. This product includes the following features, which solve challenges inherent in creating systems for embedded Internet devices:

1. ***Cross-platform debugging.***
2. ***Profiling performance and runtime issues on the small embedded device, to allow application optimization.***
3. ***Managing team-development issues (i.e., configuration control, version management).***

Re-using code, thus simplifying the complicated world of embedded systems, where you may have many different client devices that are based on variety of microprocessors which can all have many different microprocessor/RTOS configurations.

INTRODUCTION

The Java™ programming language and platform is rapidly moving squarely into the mainstream of embedded programming, where Web years and a networking focus require reliable and high-performance systems delivered in far shorter time frames than in the past. How did such a new and radically different programming language so quickly have such an impact in embedded development? The answer is simple - embedded programmers need an object-oriented language without the size and complexity of C++, but with ready-made networking and user interfaces. The use of objects enable developers to achieve data abstraction, design better systems, and achieve a higher level of cooperation and communication between applications and systems.

And behind all of the programming advantages is the potential for achieving the silver bullet - reusable code. Thanks to an execution model that enables Java bytecode to run on a Java virtual machine, or VM, Java programs can be run on multiple platforms and operating systems. The key to code reusability for embedded systems is a VM that provides full Java compatibility and optimized class libraries that provide the ability to run well on small and resource-limited embedded microprocessors.

But Java development tools have not kept pace with the needs of embedded system developers. Existing Java development tools are satisfactory for learning how to program with and use the Java language, and are fine for developing and deploying on powerful desktop computers and servers. But embedded systems have unique development, debugging, and performance requirements that are simply not met by most existing Java development tools.

Embedded developers require similar tools to those already in use for embedded systems programming, such as cross-platform debuggers, performance analysis and profiling features, and a host-target development model. But thanks to the growing complexity of embedded systems, they also need tools borrowed from enterprise development, such as team-based source code control.

CROSS-PLATFORM DEBUGGING

Tools for debugging Java applications are primitive today compared to those available for C/C++ programming. Part of the reason is that the VM hides

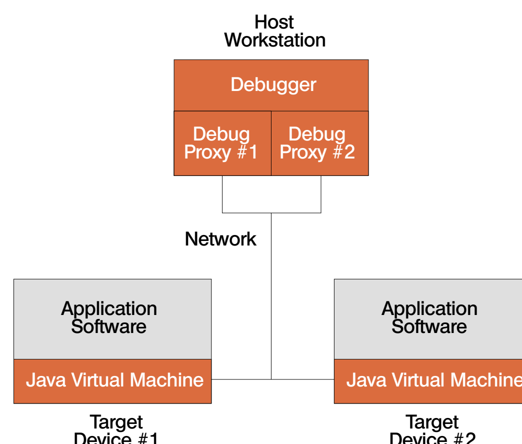

Figure 1.

Copyright 2001 by Dedicated Systems Magazine - 2001 Q1 (http://www.dedicated-systems.com)

much of the behavior of the Java programs from the underlying system. And because Java is a platform-independent language, many programmers believe that if a program runs correctly on the workstation IDE, it will run correctly on the target.

But this doesn't mean that Java programs don't have to be debugged on the target. Memory and processor limitations on the target can result in different program behaviors, and performance must be monitored to ensure that system requirements are met.

Embedded Java debugging tools have to be specifically designed to identify and distinguish between Java program and VM behavior. IBM's VisualAge Micro Edition Remote Debugger provides the programmers with the ability to debug embedded programs on the VM, either directly on the embedded target or simulated in the VisualAge Windows runtime environment.

The VisualAge Micro Edition J9 VM runs on the embedded target. When debugging Java code designed to run on the target device, it waits for a connection from the Remote Debugger on a specified port. After a connection is made, the VM runs the Java program according to the instructions from the Remote Debugger. The programmer can step through the program, set breakpoints, or simply run to completion. The target VM may be running on a remote embedded device, a remote machine running the Windows virtual machine, or on the same machine as the host system.

The debug proxy portion of the Remote Debugger runs on the development workstation. The debug proxy assists the VM by translating specific VM commands into Java Debug Interface (JDI) understood by the Remote Debugger. The target debug VM support can be kept very small by offloading debug processing to a proxy located on the development workstation.

The Remote Debugger runs on the host development workstation. The debugger tool supports all the standard debugging capabilities such as single-step, back-up, inspect, modify, and watched variables, and it complies with Java Debug Interface 1.2 level support. The Remote Debugger can connect to multiple embedded target devices from a single development workstation if desired.

Java programs can be packaged as class files, JAR files, or a binary image file on the embedded target, depending on the memory and performance requirements of the device. All of these file formata can be debugged using the Remote Debugger tool from the development workstation.

PERFORMANCE PROFILING

A critical part of embedded debugging and testing is performance profiling. This is particularly important for Java programs, which execute by interpreting Java code on a VM software layer. Unoptimized but adequate code isn't an option on a resource-constrained embedded system with real time response requirements.

The IBM Visual Age Micro Edition Analyzer tool provides visibility into the embedded target, showing information about how Java applications and the VM are running, including thread state as execution occurs, and timing about the application and VM running on the target.

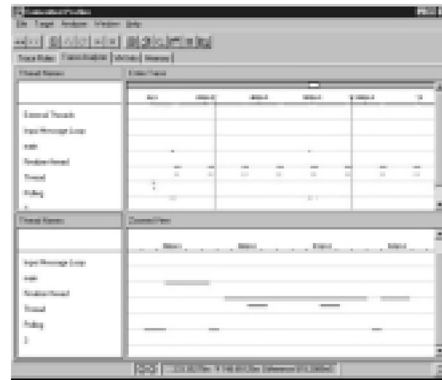

Figure 2.

The Analyzer shows the actual thread state as execution occurs. The developer can see how the real-time operating system and garbage collection functions interact, when threads are waiting on monitors, and when threads are running and performing their expected tasks. The Analyzer shows data on different types of events, such as JNI calls, thread start and end events, garbage collect events. It also lets the programmer specify which portions of the code to show up in a program trace.

The Analyzer also provides important information about the duration of events on the target, without the overhead typical of non-realtime profiling tools. This information is important because the performance and manner of execution on the target can differ greatly from that on a development workstation. For example, a block of code that uses floating-point arithmetic might perform fast on a development workstation. Moving that same code to an embedded platform with only emulated floating-point arithmetic, however, would result in substantially slower performance.

The Analyzer includes information about memory usage - which threads are properly utilizing the memory allocated to them, and which may be using their memory allocation inappropriately. It provides the programmer with the data needed to optimize memory allocation, an especially important consideration on memory-constrained embedded devices.

TEAM DEVELOPMENT

In the past, most embedded systems were developed by individuals or small teams. The relative simplicity of devices meant that there was no need for large teams, individual specialization, complex development plans, and sophisticated source code control.

That picture has changed with the new generation of modern, 32-bit, multi-function embedded systems. Source code can easily run into the tens or even hundreds of thousands of lines, and the long service life of embedded systems means that the code base may

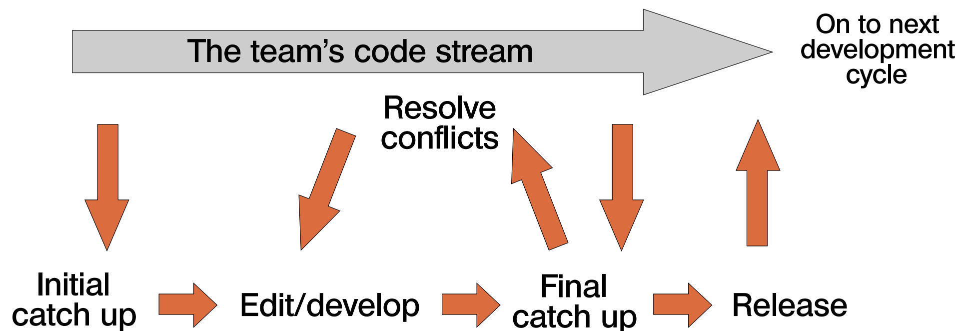

Figure 3.

have to be maintained for a decade or more. Embedded system development tools typically don't scale well to this level of complexity.

Today, an embedded products company will not offer just a single product or application. Instead, it will likely have a number of different products, or a family of products. Many of these products or applications will probably have a similar focus, but be targeted at different users or price points. The company will have multiple teams working concurrently on different members in the family. Traditional configuration management used in enterprise development products can be cumbersome, rather than efficient. The solution is a new team environment that allows concurrent development across the range of products, or different versions of a specific product.

VisualAge Micro Edition provides the full range of team development capabilities found on its enterprise development counterparts. It features a repository-based development environment with source and version control that enables development teems to keep track of all source code changes made over the life of the project. The VisualAge Micro Edition source code repository is updated automatically each time a change is made to the source code. A history of all changes made to the Java application is kept within the repository, enabling any programmer in the development team to back out any unwanted changes.

As embedded communications products begin to be driven by Web years, it becomes more and more important to achieve productivity in a team environment. Java helps here by code reuse, decreased complexity, and overall productivity. This is doubly important if the customer has a product line that mutates rapidly with new derivative products, which may be delivered in a matter of weeks apart from each other.

VisualAge Micro Edition leverages the advantages of rapid application development using Java with a flexible code repository for managing code reuse situations like rapid derivative product situations. This means that when the team finishes the primary applications, it also has a significant head start on new applications that build on the original code base.

A ROBUST VM MEANS CODE REUSE

The VM, the platform on which a Java program executes, models a computer whose instruction set is Java bytecode. This model has many implications, both positive and negative. On the positive side, it makes Java programs highly portable between a variety of embedded devices, and makes it possible to use the same code base for many derivative applications. But some perceive that this execution model increases the footprint of the program, requiring more expensive hardware for implementation.

Some analysis shows that this reasoning is faulty. Instead, think of the VM not as a part of individual Java programs, but as a static overhead for all programs written in Java. If the VM, its supporting native code components, and the class libraries are considered Java "overhead," then the application code would be considered "footprint." When considered in this manner, Java makes a small footprint go a long way.

The size of supporting components varies, but for the purposes of this example, assume that the graphics component requires on the order of a megabyte of code. Other components, which are highly dependent on the application, might total another megabyte. Once the overhead code is included, individual Java programs are tiny in comparison. Even moderately useful programs will be only a few KB in size. Even highly complex programs would be unlikely to compile into over 100 KB of bytecode.

For an embedded device that runs several separate programs, the overhead averages out among these programs. This means that an embedded Java developer needs two megabytes to execute any program at all, but very little more in order to execute many programs, even of moderate complexity. First, the JAR (Java Archive) file containing the classes can be compressed to conserve space, or left uncompressed to improve performance. That makes a 30- to 50-percent difference. Second, Java assumes that it is operating with a file system. It allocates a buffer to hold the class, reads it into the buffer, and digests the buffer into its internal representation of a class. If the device doesn't have a file system, the class file or JAR is kept in ROM. Most of the code can run directly from ROM, and doesn't have to be relocated to RAM before running.

Copyright 2001 by Dedicated Systems Magazine - 2001 Q1 (http://www.dedicated-systems.com)

This assumes that the VM and embedded development environment is flexible enough to assemble just the VM components required for the applications to be run by the embedded device. IBM's VisualAge Micro Edition VM provides a standards-based platform specifically designed for the unique requirements of embedded systems.

An important feature of an embedded VM is the ability to access a range of peripherals. The Java Native Interface (JNI) gives software developers a way to directly manipulate things like customer device drivers or RTOS functions. The VisualAge VM native methods support JNI, so programmers will always be able to devise access mechanisms for all the unique hardware peripherals and connections a device may have.

The VisualAge Micro Edition VM has carefully reduced memory requirements in the VM design. And programmers can optionally configure the VM to further pare out unutilized features and libraries, achieving a bare minimum footprint for an embedded platform.

Dreaded memory leaks are avoided in the VisualAge Micro Edition virtual machines, as memory allocation and de-allocation is automatically managed. The VisualAge Micro Edition J9 virtual machine includes algorithms to ensure full memory compaction and no memory leaks, while providing user control for unique realtime situations that may be encountered by the developer.

Another key issue for running Java on embedded systems is real time response. By fully utilizing a robust thread capability provided by the underlying real-time OS, the J9 VM can make use of a large range of thread priorities and a highly reliable scheduler. This provides the programmer with a wide latitude in scheduling and executing code to meet real time requirements. Currently, that RTOS is QNX/Neutrino, which is available on Intel, PowerPC, MIPS, and ARM processor families.

A COMPLETE PACKAGE SPEEDS EMBEDDED JAVA DEVELOPMENT

A complete development system for embedded Java, therefore, has to include a full range of tools, include compiler, debugger, and code profiler, all integrated into an IDE, with a reliable and flexible VM. The development environment must also take into account an order of magnitude increase in complexity in modern embedded systems, along with the ability to maintain and enhance that system code over the life of the project.

IBM's VisualAge Micro Edition is one such Java application development environment. For programmers seeking ways to leverage the networking, portability, and productivity advantages of Java on embedded systems, VisualAge Micro Edition provides a unique combination of tools and features. With Java development for embedded systems just starting to take off, such an embedded development environment will improve code quality while speeding time to market ■

Kim Clohessy is Vice President of OTI, a wholly owned subsidiary of IBM, managing the Embedded Systems Group. OTI has provided Smalltalk-based tools to a wide range of embedded projects for over 11 years. OTI is now building upon expertise in tools and virtual machine technologies to bring integrated Java development to embedded systems.

Kim has over 20 years in the embedded systems industry. Prior to OTI, Kim was VP Technology at DY-4 Systems which builds VMEBus products for industrial and military applications.

Kim was the editor of the ANSI/VSO VME64 Specification, and has chaired several other IEEE standards committees related to embedded systems. Kim has a B. Engineering degree from the University of Western Australia.

Dedicated Systems Experts can assist you with:

- Audits
- Courses
- Consultancy
- Support

Contact Nico Van Wijmeersch at +32-2-520.55.77 or find more information on our Web-site at http://www.dedicated-systems.com