# Design of a Real-Time Virtual Machine (RTVM)

Steven Cavanagh and Yingxu Wang

Theoretical and Empirical Software Engineering Research Center
Department of Electrical and Computer Engineering
University of Calgary
2500 University Dr. NW, Calgary, AB, T2N 1N4
Email: *steve@jpe.ca, yingxu@ucalgary.ca*

**ABSTRACT**

The commercial software industry has a number of different operating system vendors offering various features. Real-time operating systems (RTOS's) are primarily proprietary, closed source, expensive, and uniquely designed to specific applications. Software designers must examine a given real-time operating system to ensure an RTOS will meet the real-time requirements for an intended platform. An RTOS does often not meet the original real-time constraints, when product evolves to new hardware platforms. This makes the evolution of real-time systems a cumbersome and difficult problem.

In this paper we will describe a new approach to real-time system development. Instead of reinventing software for a real-time system, we provide a generic and platform independent virtual machine. We will discuss the design, architecture, and implementation of a Real-Time Virtual Machine (RTVM). The RTVM provides a generic real-time platform, where real-time applications using the Java paradigm can operate on various platforms without re-writing the code.

**Keywords:** *Real-time systems, operating systems, virtual machine, portability, configurability, Java*

## 1. INTRODUCTION

Typically, a commercial real-time operating system provides specific features suitable to a single application. It would be obsolete when the platform or target machine is changing. Therefore, existing software must be "ported" to another real-time operating system to meet new requirements, and add new features. To solve these common development issues, a real-time virtual machine (RTVM) is proposed. RTVM provides an adaptable platform with customisable real-time characteristics and predictable behaviors. The generic real-time application platform allows simplified software and product line evolution within an organization.

This paper discusses specific components of the RTVM architecture and its real-time characteristics, independent of the underlying hardware platform, while providing consistent application performance. By providing a generic real-time platform, the evolution process of real-time applications and products is simplified. Because the RTVM is designed as open source, it can also be easily customized to meet specific performance requirements, while the legacy application, core services, and hardware interfaces remain constant.

## 2. THE ARCHITECTURE OF RTVM

A virtual machine provides a operating system abstraction, which realizes common core services. Core services such as memory management, scheduling, and code interpretation are defined within a common virtual machine pattern [1, 6]. A number of issues remain in the design of an RTVM. If an application is developed for a virtual machine, is it possible to provide consistent timing results, on different hardware platform? Regardless of the application structure to interact with the virtual machine services, can the CPU and peripheral features be configured to perform consistently? The following sections discuss techniques for solving the above problems.

### 2.1 RTVM Hardware Interface

The architecture of the RTVM is shown in Fig. 1.

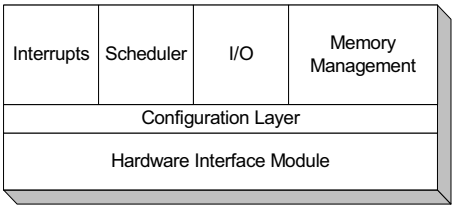


Fig. 1 The RTVM architecture

The fundamental component of the RTVM architecture is the hardware interface module (HIM). HIM is the lowest layer interface between the core RTVM services and the underlying hardware platform. HIM allows RTVM to abstract hardware idiosyncrasies from the RTVM core services. Because the HIM is independent of the core, a HIM module can be easily "plugged in" to the RTVM

architecture. A consistent set of primitive interfaces are designed to interact with the core. By maintaining these interfaces between the RTVM core and HIM, a HIM module can be easily replaced to support another platform. This architecture also provides a consistent, well understood behaviour to adapt a virtual machine to other platforms. HIM defines a well known set of procedures to configure and initialise hardware consistently to interact with the core services. HIM formalizes a platform pattern for the RTVM. For example, the memory management policy must be efficiently executed by the underlying hardware memory management mechanisms. HIM defines a consistent pattern to initialise the hardware in support of memory management and other core services.

### 2.2 RTVM Core Services

The RTVM core services consist of the following:

- Interrupt handling
- Event handling
- Memory management
- The scheduler

Interrupt and exception are handled through the common configuration of the interrupt vector table within HIM. RTVM configures a minimal set of interrupts and exceptions to support application system calls, task scheduling, and memory management. Other common devices such as mouse and keyboard are not configured, to minimize interaction with the core, intended for mobile real-time embedded devices.

Event handling within RTVM is based on the Linux signal handlers. Processes can interact with RTVM using POSIX defined signal handling.

The RTVM memory management is designed similarly to the Linux paging system. Memory management does not use segmentation, and applies common 4kB pages, which provide efficient disk access and page swapping. The RTVM allocates memory for a process via the "fork" system call as per the configuration module, such as stack size. Further research is ongoing to measure performance of applications using this memory management policy. To achieve greater performance improvement, a configuration item may be added to allow RTVM to support a flat memory model without paging.

The scheduler is based on the Linux operating system, which provides an overall fairness algorithm. Each task in the execution context is given a time slice to execute. This type of scheduling provides a "soft" real-time response for an application. Applications, which require "hard" real-time response may configure the RTVM to modify the scheduling policy and priorities, based on Linux real-time processes.

RTVM defines a set of scheduler interfaces to change the scheduling characteristics of a process. These interfaces allow a developer to improve the "real-time" nature of a process as required.

The core services provided by the RTVM, interrupt, event handling, memory management, and scheduling do not change when porting the RTVM. This set of core services provides a consistent platform for real-time applications. Other virtual machine architectures "plug-in" a real-time operating system to provide core services. This is a key aspect of the RTVM, to provide a consistent, well-understood platform. Applying a third party solution re-introduces the complete learning cycle to understand the development environment and characteristics of the RTOS.

### 2.3 The Configuration of RTVM

As seen in Figure 1, RTVM uses a configuration module to define its behaviours. A resource file pattern [1] is used to allow developers to configure aspects of the RTVM, such as task threshold, process memory allocation, and scheduler policy, to name just a few. The task threshold defines a high water mark for the maximum number of tasks that the RTVM may run at a single instant. Heap memory allocation defines a high water mark for heap memory allocation. Process memory allocation defines the code, data, stack, and heap size used when a process is created. There are many configurable items, which can be applied to the RTVM configuration module.

## 3. THE PLATFORM OF RTVM

### 3.1 Application Portability

As mentioned in previous sections the HIM abstracts the RTVM hardware idiosyncrasies from the core services. This provides a consistent and well-understood set of components to execute applications.

The RTVM also provides a well-defined set of interfaces to the core services. This design is analogous to the Linux system call interfaces. A well-defined set of interfaces are defined by the RTVM to access core services, such as tasks, signal handling, and memory allocation. These interfaces follow the same POSIX standard interfaces as defined in Linux. POSIX is the "Portable Operating System" interfaces standard [3].

Applications developed for the RTVM use well-defined interfaces and shared libraries for the C and C++ programming languages. The GNU compiler is used to compile applications linked against C and C++ shared libraries. These libraries are built-in to the RTVM architecture. The RTVM uses a simple RAM disk to allow applications to dynamically link against native libraries. Because the development and RTVM libraries are identical, applications can be easily ported. The libraries can be optimised to reduce the footprint required, more suitable for highly embedded devices.

This allows applications compiled on any native platform to run on the RTVM platform. As long as the shared libraries are compatible, applications are completely portable. Further research is ongoing to provide a portable object format using shared libraries across Intel and PPC platforms.

### 3.2 Platform Portability

RTVM provides a well-defined hardware module to provide a consistent approach to configure hardware. Another feature of the RTVM is loadable modules. In order to support different hardware platforms, RTVM allows a developer to configure modules to support different hardware peripherals. Because RTVM is based on Linux version 2.4 and 2.6, modules can be easily loaded using the same Linux technique. Modules are loaded via the RTVM configuration module. This feature provides great flexibility to support various hardware platforms. It is essential to note, that a module can adversely affect the real-time performance of the RTVM. It is recommended that modules are matured and thoroughly reviewed before configuration into the RTVM.

## 4. THE JAVA VIRTUAL MACHINE

The Java virtual machine is designed for enabling an application to "write once, run anywhere" on various hardware platforms. This virtual machine was not designed to address real-time issues, although originally intended for embedded devices. Often, virtual machine design sacrifices application performance for portability. The Java Real-Time Specification (JRTS) has been developed to addresses real-time response and predictability issues. Various virtual machines have been developed, which address specific footprint requirements for embedded systems and custom implementations in an attempt to provide real-time characteristics. Most variants of virtual machine provide an adapter interface to a commercial real-time operating system to create real-time predictable behaviour. Custom virtual machine implementations provide specific components, such as assembly code and native compilations to improve the real-time characteristics of the virtual machine. Real-time capable virtual machine designs are highly inconsistent. Various techniques, such as Dynamic Adaptive Compilation (DAC), Ahead of Time compile (AOT), Java Native Interfaces (JNI), and Just-in-Time (JIT) compiles are used to improve real-time characteristics.

This section describes specific components of virtual machine architecture, which will form the basis of a real-time capable and predictable system. We will describe a re-usable virtual machine architecture, independent of a specific real-time operating system. This approach will allow researchers and developers to create a real-time virtual machine using a consistent pattern.

To consider virtual machine predictability, a pattern is necessary to create consistent well defined components. By creating a virtual machine "predictability" pattern, core components are defined, which will provide consistent behaviour. We expand on this approach by presenting the Microkernel [5] pattern to define core virtual machine components.

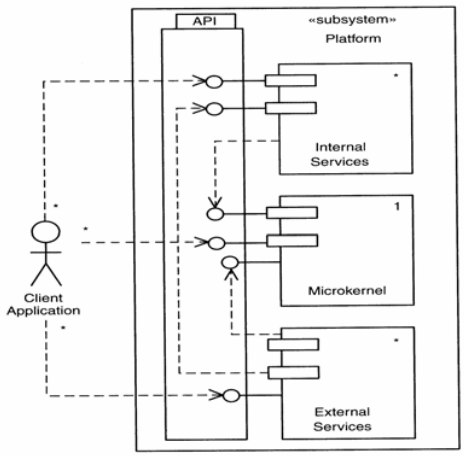


Fig. 2. The MicroKernel pattern

### 4.1 Memory Management

There are various resource management schemes that have been developed for virtual machines. Some varieties of garbage collection (GC) are:

- Precise GC
- Batch GC
- Concurrent GC
- Conservation GC
- Incremental GC

The most fundamental aspect of a virtual machine is the resource management algorithm. Resource management must provide predictable behaviour as being concurrent, and either precise, conservative, or incremental. Research based on work completed for incremental GC [2] has shown promising results for its efficiency. Resource management, which is performed concurrently using a pre-emptible platform, will produce consistent, predictable results. A balance must be reached using further research to ensure that an incremental GC can produce predictable results, without resource starvation. An incremental algorithm should be scheduled with an efficient period to reduce the probability of starvation incidents.

### 4.2 The Scheduler

Research has shown that the rate monotonic scheduling algorithm provides consist and predictable scheduling. Tasks whose ratio of execution time to scheduling period is guaranteed to run to completion based on the rate monotonic theorem:

$$\frac{C_1}{T_1} + \cdots + \frac{C_n}{T_n} \leq n(2^{1/n} - 1) = U(n) \qquad (1)$$

To provide predictable behaviour with a run-to-completion model, a scheduler must run in a fully pre-emptible environment. The underlying operating system

must provide a pre-emptible mechanism, which is highly independent of resource usage. Typical Linux operating systems, responsiveness drops in proportion to the resource usage, such as running tasks and RAM usage. This problem requires the adaptation to provide consistent and predictable behaviour. Linux 2.6 for example has created a pre-emptible kernel [5], which yields positive results to be applied in a predictable virtual machine pattern.

### 4.3 Hardware Interface

To provide predictable behaviour at the hardware layer, hardware must be consistently configured to support the virtual machine core services. The development of RTVM provides a hardware interface module (HIM), which creates a hardware configuration pattern to support the core services. This provides a consistent configuration of the underlying platform mechanisms to support the virtual machine core services.

### 4.4 Application Interaction

Defining well structured interfaces is a core feature of predictable VM behaviour. As defined in the Java programming language, Java 2 specification provides a clearly defined set of classes to interact with the Java virtual machine.

Analogous to this approach is the design of the RTVM, which defines a set of POSIX compliant interfaces to the RTVM core services [5]. A well defined API provides adaptable and consistent support for applications.

### 4.5 Compilation

To achieve predictable behaviour, virtual machines need to employ consistent approach to executing applications. The Java VM, can interpret compiled bytes codes to execute an application, but also provides interfaces to compile code natively via the Java Native Interface (JNI). Other virtual machine implementations, such as Jeode [4], have developed Dynamic Adaptive Compilation (DAC). This technique compiles often executed code to native format to improve VM performance.

Another technique presented is to employ a consistent compiler to provide native code. Native code will consistently execute faster than adaptive and dynamic compilation techniques. The RTVM design sacrifices application footprint for responsiveness by using the GNU C/C++ compiler to natively compile applications. This approach has two advantages:

- A consistent compiler
- Native code execution

Because the GNU compiler is the only compiler available to build Linux applications, compiler output is consistent for the RTVM platform. Native code execution and consistent compile are critical aspects to consistent, predictable VM behaviors. Further research is ongoing within the RTVM project to provide a portable object format executable across Intel and Motorola architectures.

## 5. CONCLUSIONS

Real-time applications are becoming more widely spread with personal digital assistants or PDAs, cellular phones, and other mobile handheld devices. Although, these applications are not considered hard real-time devices, such applications require the adaptability of a real-time virtual machine (RTVM). The RTVM, will be useful for a dynamic marketplace with the use of various mobile devices. These devices require soft real-time responsiveness and adaptability.

This paper has described the development of the RTVM to provide soft real-time and adaptable embedded systems. The RTVM will help embedded device manufacturers and software developers to maximize re-use of applications, which behave consistently and predictably. By employing configurable, open source hardware and software interfaces the RTVM core services remain consistent and highly adaptable for application development and product evolution.

## ACKNOWLEDGEMENTS

The authors would like to acknowledge the Natural Science and Engineering Council of Canada (NSERC) for its support to this work. We would like to thank the anonymous referees for their comments.

## REFERENCES

[1] Noble, J., Weir, C., Small Memory Software: Patterns for Systems with Limited Memory, Software Patterns Series, Addison Wesley, 2001.

[2] Blunden, B., Memory Management Algorithms and Implementation in C/C++, Wordware Publishing Inc., 2003.

[3] Bovet, D., Cesati, M., Understanding the Linux Kernel, O'Reilly Publishing, 2003.

[4] INSIGNIA Solutions, Jeode Platform White Paper, April 2001.

[5] White, B., Linux 2.6: A Breakthrough for Embedded Systems White Paper, Sept. 2003.

[6] Wang, Y. (2002), "A New Approach to Real-Time System Specification", *Proceeding of the 2002 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE'02),* Winnipeg, MB, Canada, May, pp.663-668.