Real-Time Java: Does it belong in the IoT?

Randy Rorden, Software Development Director, PTC Inc.

Embedded Systems History

When you talk about the "things" of the Internet of Things, you are talking about embedded systems. Application-specific digital processing has been "embedded" in things since the early days of computing. The first SCADA (Supervisory Control and Data Acquisition) systems were based on mainframes and minicomputers in the 1950s and ’60s. They automated factories, water treatment plants, oil and gas production, electricity distribution, and other complex, labor-intensive processes. When microprocessors came along, they were put to use controlling stuff [Fig. 1]. As a hardware engineer in the ’70s and ’80s, I designed and built computers, workstations, and embedded systems using 16 and 32-bit microprocessors. I wrote firmware, device drivers, and graphics libraries for those machines. Early on we had to design a custom board with a processor, memory, and peripheral controllers. Manufacturing a new board was expensive and time consuming. Software consisted of "Roll Your Own" (RYO) kernels and executives that ran on bare metal, much of it written in assembly. As the IBM PC became the standard for office computing in the ’80s, semiconductor manufacturers began creating chipsets and reference designs that made their
way into embedded systems. Second-tier vendors built "merchant computer boards" out of those chipsets and collaborated to define form factor and bus standards so they could connect and talk to one another. Then developers could buy embedded hardware off the shelf and adapt it for their needs. For software, they turned to commercial Real-Time Operating System (RTOS) vendors so they wouldn’t have to roll their own. RTOS’s with names like VRTX, VxWorks, pSOS, QNX, LynxOS, INTEGRITY, and others proliferated. There are embedded systems that have been running this software quietly in the background for many decades.

More recently, commoditization of hardware in Smartphones has created faster, cheaper processors running at lower power with more memory and microelectromechanical systems (MEMS) sensors. Multicore System-on-Chips reduce parts counts and open source board designs such as Arduino, BeagleBoard, and Galileo yield fully functional computers the size of a credit card for less than $50. Embedded Linux and high level languages are common, leaving the RTOS’s to run only the hardest of real-time and safety critical applications.

Figure 1

Technology Convergence to the IoT

The Internet of Things, like most innovations, is the result of the convergence of many unrelated technologies that came together at the right time to create something new and larger than the sum of its parts [Fig 2].

Figure 2

First, Embedded Systems technologies have been commoditized to the point where computing and storage can be sprinkled around the world like magic dust. Cheap, low-power sensors can run for years on a coin cell battery. Intelligence is the new ingredient going into everything we use: appliances, cars, public infrastructure, as well as creating new applications such as fitness trackers and smart doorbell cameras.

Second, Cloud Computing and the technologies that make it possible: virtualization, blade servers, etc., deliver computing and storage resources on demand with no overhead to the user. Infrastructure as a Service, Platform as a Service, and Software as a Service models are available from competing cloud vendors that will host your code in their data centers at reasonable prices. With commoditized IT infrastructure, companies are free to spend more of their time and money on the value created by their software and less on the copper and silicon required to run it.

Third, Big Data and Deep Learning make it possible for companies to gain insights and understanding from massive amounts of unstructured information. Advances in artificial intelligence and neural networks yield self-driving cars, speech-recognizing digital assistants, and computer vision systems. These have been aided by advanced Graphical Processing Units (GPUs) to do the number crunching tasks of machine learning.

And fourth, ubiquitous connectivity is reaching more homes, schools, offices, factories, and public spaces than ever before. Companies like SpaceX, Google, and Facebook are working to bring the Internet to poor and rural regions from the upper atmosphere or low
earth orbit. Wireless standards such as Bluetooth LE, ZigBee, 6LoWPAN, and LoRa connect the “Things” of the world to the each other and to the IoT Cloud.

Each of these technologies on their own are making people and organizations become incrementally more productive, but together they enable new kinds of smart, connected products and services that would not be possible without convergence. In some cases, business models are getting turned upside down, with companies deploying smart, connected products to the field and charging the end user only for their actual use: the so-called “Product as a Service” model.

Intelligence at the Edge

A simplistic view of the Internet of Things consists of a centralized IoT “Cloud” surrounded by end-devices in the form of sensors, controllers, and human-machine interfaces at the “Edge,” each feeding data into the Cloud and acting on commands coming from the Cloud. This model quickly breaks down as the number of devices scales up. There isn’t enough bandwidth in all the networks and not enough storage in all the data centers of the world to handle the 50 billion connected devices predicted by the year 2020. The solution is intelligence and storage at the edge of the network, where data can be aggregated, filtered, analyzed and acted on as necessary, forwarding only relevant information to the Cloud. Cisco coined the term “Fog Computing” for these intelligent edge devices. Some “Fog Nodes” may be custom-built with sensors and actuators closely coupled to the software running in the device. Others may be built with standard hardware, interfaces, and pluggable components, such as cellular modems, serving as gateways for a network of lower-level sensors and actuators in a home, on a factory floor, or along a roadway. Intelligent gateways are also important to securely connecting legacy end-devices to the IoT.

Adaptability: The Key to Survival

Along with intelligence at the edge, an important characteristic of the IoT is the ability to adapt and improve products in situ and on the fly through “Over-The-Air” (OTA) software updates. This is another concept borrowed from the smartphone market. But we aren’t just talking about bug fixes. Entirely new features and capabilities can be added to a product that the original developers may not have anticipated fully. A good example is Tesla’s software updates to their Model S all-electric vehicle adding autonomous driving features.

The Value of Java

To satisfy the requirements of sophisticated intelligence, faster time-to-market, and over-the-air updates in IoT Edge devices, developers are turning more to the Java language, runtime, and ecosystem. A survey of 800 engineers by VDC Research (sponsored by Oracle) shows an increase in the use of Java in their current project from 12% in 2008 to 27.4% in 2015. Java improves upon C or C++ as a high-level object-oriented language with automatic garbage collection, robust exception handling, built-in threading model, and extensive libraries for doing all the things an IoT Edge device requires. Programming in Java is safer and more reliable by avoiding common coding errors, such as dereferencing stale pointers and trampling memory. The VDC report shows that for a typical ARM-based project shipping 1M units, using Java would save 40% in software development costs compared to C. Standard frameworks such as the Open Systems Gateway Initiative (OSGi) add capabilities to Java that further enhance its usefulness in the Internet of Things. OSGi bundles are modular components that can be downloaded, installed, started, stopped, and uninstalled without having to restart the Java Virtual Machine. This can be a huge benefit for systems that must operate 24/7.

Traditional Java and Real Time

Something traditional Java cannot handle well is real-time. Real-time systems have to guarantee that they will respond to an event or input within a deadline. Most Java programmers have experienced the unexpected delays that can occur when a “stop-the-world” garbage collection cycle runs. Depending on the size of heap memory and the fragmentation of live objects, GC pauses can last...


hundreds of milliseconds up to several seconds\(^1\). But GC pauses are just one of many of traditional Java’s real-time shortcomings. For example, the Just-in-Time (JIT) compiler may delay execution of a critical piece of code. Java thread priority is treated as little more than a hint to the operating system scheduler, which may unpredictably favor low-priority threads over high-priority threads. Worse yet, a priority inversion can occur, blocking a high priority thread from running indefinitely. Add to this the effects of page faults, random preemption by other processes, and the lack of precise timing APIs and you might be convinced that Java could never guarantee a program will ever finish, let alone meet a deadline.

But you’d be wrong. It is possible to build a Java Virtual Machine that supports real-time response – it just doesn’t come from a traditional Java SE vendor. The PTC Perc product has been deployed in real-world, real-time systems since 1998. It is used in telecommunications, industrial control, aerospace, automotive, and defense applications. And now it is taking on the Internet of Things as a key technology for edge devices that need to react within milliseconds to critical events.

**When Milliseconds Matter**

Think about an advanced driver assistance system that uses a camera and steering wheel micro-movement detection to diagnose driver fatigue. It needs to react quickly to signs of drowsiness with escalating seat and wheel vibrations, audible warnings, and if necessary, to take control of the steering and brakes to correct for lane excursions or prevent a collision. This is a scenario in which a sophisticated algorithm must make a decision within milliseconds. It doesn’t have time to communicate with a Cloud server or wait for a Java VM garbage collector to finish.

---

**Real-Time Features**

Table 1 shows the features of traditional Java Standard Edition vs. PTC Perc.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Java Standard Edition</th>
<th>PTC Perc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Just-in-Time (JIT) Compiler</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Ahead-of-Time (AOT) Compiler</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Real-Time Garbage Collector (GC)</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Priority Inheritance Protocol</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Precise Timing APIs</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>GC Monitor and Control APIs</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Real-Time Scheduling</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Page Locking</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

PTC Perc supports Just-in-Time (JIT) compilation of Java byte codes to native code but adds an Ahead-of-Time (AOT) compiler that allows developers to pre-compile their applications and Java libraries to native code and link them into the Perc Virtual Machine just like a C/C++ toolchain. This avoids unpredictable JIT delays.

Perc uses a patented real-time garbage collection algorithm that allows high priority threads to interrupt GC activity at any time to perform time-critical tasks. When the task is finished, the GC continues running. There are no "stop the world" collection cycles.

Perc implements the Priority Inheritance Protocol when threads contend for shared locks, thereby preventing the priority inversions that can occur in traditional Java SE.

Additional Timing APIs in Perc permit developers to schedule jitter-free periodic tasks, sleeps, or waits. The new APIs use an absolute uptime-based scheduling mechanism so it is not subject to disruptions caused by a change to the system clock.

---

\(^1\)https://plumbr.eu/blog/garbage-collection/revealing-the-length-of-garbage-collection pauses
The Perc VM lets the user set the priority and timing parameters of the garbage collector to control when it runs and how much CPU time it is allowed to consume in relation to Java threads. On SMP processors, multiple GC worker threads run concurrently with Java threads and the user can assign which cores are to be used for GC, thus reserving cores only for Java.

Perc allows the user to set real-time scheduling policies in Linux to avoid preemption by other processes. The real-time priorities supersede all normal threads.

And the Perc VM can be configured to lock all memory pages into physical RAM rather than wait for page faults to occur.

The answer to the question posed by this whitepaper is a resounding “yes!” Real-time Java does belong in the Internet of Things. PTC Perc is a robust, real-time Java solution for intelligent IoT Edge devices that have millisecond response time requirements. Learn more about PTC Perc at [www.ptc.com/developer-tools/perc](http://www.ptc.com/developer-tools/perc).

© 2016, PTC Inc. All rights reserved. Information described herein is furnished for informational use only, is subject to change without notice, and should not be taken as a guarantee, commitment, condition or offer by PTC. PTC, the PTC logo, Product & Service Advantage, Creo, Elements/Direct, Windchill, Mathcad, Arbortext, PTC Integrity, Servigistics, ThingWorx, ProductCloud and all other PTC product names and logos are trademarks or registered trademarks of PTC and/or its subsidiaries in the United States and other countries. All other product or company names are property of their respective owners.

J6905–PercRealTimeJava–EN–0416