JCP EC Java ME Working Group
Eclipse
Fujitsu
Gemalto
London Java Community
MicroDoc
Oracle
V2COM
Werner Keil

Java on Embedded

Advancing Java on embedded devices

OVERVIEW

The JCP EC established the Java ME Working Group to address the concern of EC Members about the current state of Java ME, and this document is its deliverable: a set of use cases and requirements for Java on embedded devices.

In the January 2017 JCP EC Face to Face Meeting, V2COM presented their current view of the Java ME platform. When Java ME 8 was launched in 2015, Oracle stated[1] that its goal was to keep Java ME and SE releases synchronized and keep bringing the two versions closer together, regarding functionality and language features. But time has passed and no activity could be seen from Oracle regarding Java ME: content for JavaOne 2016 was high on Java SE 9, Java EE 8 but no word on the next iteration of Java ME. Some minor releases (8.1, 8.2 and 8.3) were made, but as Oracle started filing JSRs and announce Java SE 9, there was no public Java ME activity.

V2COM started raising concerns about this and requested an update from Oracle, much like what the JCP EC members requested about Java EE 8. While Java EE 8 concern was addressed in EC meetings and JavaOne 2016, no update was give about Java ME for months, until the JCP Chair announced [in December 2016?] that Oracle had changed management on Java ME to Georges Saab and we could come with a guiding document on our set of requirements for Java on embedded devices and feature phones.

GOALS

1. Describe technical and non-technical requirements for Java to be relevant and useful in embedded devices and IoT.
2. Provide grounds for discussion with Java leadership within Oracle on how to improve Java ME, or the next version of Java for Micro devices.

1

USE CASES

There are many different IoT use cases, with completely different set of technical and business specifications. We're focusing on two different device classes that group some of the desired characteristics, but by no means this is an exhaustive list and a Java embedded platform should be thought as ranging from low power[capability?] and/or low memory platforms such as ARM Cortex M4 with 1MB RAM and 1MB Flash to high power such as Intel Joule SoC[2] with 3 GB RAM and beyond

Specialized Device

A specialized device is targeted to a specific function, and it runs on specialized hardware. This means that its overall capabilities won't change over its lifetime, but there still is a need for small updates on business logic and bug fixing.

These devices are also constrained regarding cost: manufacturers will try to squeeze its costs to as low as possible, for these devices are for mass production and every penny matters.

We're considering that these devices don't offer a standalone OS: its firmware is normally a merge of OS and business functionality, and updates are done changing all code at once.

They might have external storage memory, but that's not always the case. Regardless, all memory (volatile and storage) is premium and usually there's "just enough" memory for it to work.

Gateway device

A gateway device has a broader scope of function, when compared to a specialized device. It also has a more generic hardware, with enough I/O and storage to allow it to handle different functions over its lifetime.

As its name implies, gateway devices also can serve as a bridge or connector between two device networks, usually between a closed/proprietary/local network and the Internet. Besides bridging the networks, it can run tasks on behalf of these devices, such as report status, and aggregate data for local analysis.

TECHNICAL SPECIFICATIONS

Performance

Specialized devices are associated with tasks that require fast startup time - such as car infotainment systems. Hence, it's very important that the VM can be started with the least amount of time possible.

Specialized devices are also sized to have just enough processing power, to save on costs. This brings the overall capabilities down, and has an impact on start up time as well as other parts of the VM such as

[2] Intel Joule home page

garbage collector pauses and multithreading. Special consideration has to be taken on VM level to address this low power devices.

Gateway devices normally don't suffer performance penalties, as they normally have hundreds MHz processors and one or more cores.

Size

There are two size constraints on specialized devices: RAM and ROM/Storage.

RAM consumption on specialized devices is critical and when working on such code, one has to take extra care on object creation and heap size. And in order to make most memory available to the application, the VM should take as few RAM as possible, with lazy initialization of mostly everything. (It does come to a trade-off between size and performance...). Java SE 8 Compact Profile 1 has a footprint of many MB of used RAM just to print "Hello World", which is too much for specialized devices.

ROM (Flash or Disk storage) is also a critical constraint: storage memory on these devices is costly and in many processors, it's built in into the processor. So there is not much space to "waste" on unused features of the VM. Case in point, the Java ME 8.2 release[3] for FDRM-K64F[4] board uses all the 1 MB available in the processor, and requires the user code to be loaded from an external memory. That leads to added BOM cost and hardware development time.

For gateway devices, this is not always an issue, but when one considers Over-The-Air Updates of the VM, a 100MB download to update the whole VM can be expensive and even prohibitive.

With Java 9 and modularity, we may be able to have a smaller base VM that can try and fit in specialized devices. We're still missing information on how Java SE Embedded 9 package definition differs from current JDK 9[5], but this might be an area for concern.

Something that might be better for specialized devices would be to merge both binaries - VM and User Code, "gutting" the VM and taking out unused classes. This might make the VM as a non-Java compatible VM, but that's not important for some use cases: the developer is not trying to sell a VM, but instead want a single binary that's able to fit in a small processor flash memory and run its business code.

Java ME/SE language level compatibility

Java ME and Java SE, historically, were too different APIs. But, with the release of Java SE and ME 8, they are converging. Java ME 8/MEEP was a huge step to be more SE like and the SE profiles made SE more like ME when comparing sizes. But what gets out of this is too static. For example, one can either use a very small runtime without JNI or a bigger runtime without some of the ME frameworks. Although

[3] Java ME 8.2 Download page
[4] NXP's FRDM-K64F
[5] JavaDoc for Java 9 java.base package/module

ME now supports most of the SE 8 language features, some are still missing. The situation with class libraries is similar.

Comparing Java ME to Java SE, these are the following missing pieces that are considered relevant in embedded design. Maybe these can start as separate, optional JSRs for the Java ME platform.

- Collection streams
- Reflection
- Runtime Annotations
- Concurrency utilities
- Collections and Math APIs

Comparing Java SE to ME (MEEP), basically everything that makes MEEP a relevant framework for IoT:

- Software provisioning
- Software management
- Application concurrency (MVM)
- Inter-application communication (IMC)
- Events
- Service Provider/Consumer pattern

Native Code support

One of the more concrete things missing for embedded development is better operating system/C/C++ integration. Libraries like JNA[6] and Device I/O[7] is a step in the right direction, but these aren't part of the Spec or the default platform.

IoT specific or new APIs/JSRs

There are emerging standards for IoT communication that today are supported in Java via JSRs (like JSON) or not (like MQTT and COAP). Java Standardization of such standards can be done to guarantee that the platform support is available for them with the IP protection granted by the JSR process.

Such standards are, not exhaustively:

- REST client
- Communication protocols such as MQTT[8] and/or COAP[9]
- Device I/O[10] (it does exist today, but it's not a JSR)
- LWM2M[11]

[6] JNA's home page
[7] Device I/O example in Java ME 8.3 documentation
[8] MQTT is a standard already
[9] COAP is also a standard
[10] Device I/O's home page
[11] LWM2M implementation as an Eclipse project

These efforts shall not be done exclusively by Oracle, but other JCP members like V2COM can step up and work to get these as separate JSRs, that can be eventually merged into a future Java Embedded release.

As part of the new APIs and JSRs, there is also need to update old JSRs.

Updating old JSRs

Some old JSRs need either an update or rework to address evolving technology.

JSR 177

Security is a critical aspect of IoT, and JSR 177[12] provides Java ME applications with APIs for security and trust services through the integration of a Security Element. Since its latest maintenance release in 2007, new technologies and needs have emerged, access to TEE (Trusted Execution Environment), SE (Secure Element), services like secure storage and handling of credentials for TLS.

JSR 361

JSR 361[13] updates IMP(-NG) to align with state-of-the-art features and current embedded device market requirements. This brought many needed updated, but some use cases are still hard to be address, specifically when considering multi-MIDlet environments.

BUSINESS SPECIFICATIONS

Licensing

Current business model for Java ME does not scale. It still uses the same licensing model, when Java SE moved on to a more open/public implementation that is a frictionless technology onramp where technologies can be implemented, tested and standardized.

Development Model

Similar development concept for Java ME as for Java SE (→OpenJDK + JSRs). Which implies that everybody can get the source and port it to a target system.

For business reasons, key technologies still need to be defined as JSRs – standards are important to ensure safety and compatibility for the whole community, as development as a JSR protects implementers from litigation.

Board support packages

[12] JSR 177's home page
[13] JSR 361's home page

Current access to the technology blocks adoption, as there's very few downloads available in board support package. Access will increase demand and adoption.

Because embedded development is not a box package, there's the need for easier download availability from hardware manufacturers. Hence, there is a need for a binary to download with porting compatibility so that hardware vendors can easily port the standard implementation to their hardware/boards

OTHER CONSIDERATIONS

Different set of requirements

There are different set of requirements for every vertical and even use case in IoT. As such, the current way of sharing a single platform/binary that contains the whole JSR specification may be outdated. There is a structured way to get a personalized VM for the current Oracle Java Embedded, where you get the JDK with functionalities that you need. But that is still too coarse. A better way would be a legal and safe method to pick-and-choose the functionalities/classes that we need for our target environment.

Low Java ME 8 adoption

Business related aspects contributed to low Java ME adoption immediately following the release. The technology met needs, but the costs to migrate were too high. In the meantime, most of these issues have been resolved. So a next version, with updated technology and licensing terms might see better adoption.

JSR Leadership

How does Oracle want others to contribute? Does it want to lead or allow others to lead and collaborate in this space? It is possible for others to lead, and others have led JSRs in the Java ME space in the past. Most members on this Working Group would be willing to participate in JSR activity, but not necessarily lead. Eclipse is interested as a potential source for communing members interested in leading JSRs and V2COM and Gemalto are interested in leading some of the JSR efforts in Java ME and Embedded domain.

But some JSRs are either touching Oracle IP (such as Device I/O) or are already lead by Oracle (such as JSR 177), so we need a clear position from Oracle on how to resolve such issues.

Market demand

Java is the only standardized platform which scales well. Many people are not aware of this. Java as an IoT platform also offers the developer appeal of an end of end technology platform. Not only program servers and backend applications, but also small devices. This is something that no other platform can do. These aspects together can create a business demand for Java as IoT platform.

Also, there might still be a case for Java ME on feature phones for the developing work. Many of these populations are too poor to afford smartphones but can afford inexpensive feature phones. Without

JavaME mobile apps have to be written in handset specific C/C++ which makes supporting multiple feature phones very challenging. Also, for humanitarian and charitable work apps that help improve individual users lives or protect them from danger or provide access to education and information can be implemented in such platform and distributed in such markets.

Competing technologies

There are plenty of other technologies that developers can choose from to develop their embedded devices. We want to learn from them and adapt Java to continue its relevance in this market!

Some of these technologies are perceived as either less expensive, hip, mainstream and/or accessible.

Node.JS (and embedded JavaScript)

Gateways with Node.JS and similar technologies are being used to make it easy to server-side developers develop embedded applications and be "end-to-end" in a single language. But this in fact is Java's value proposition. As such, with no clear Java-specific message to developers, Java is losing ground in this space.

Android/iOS

There are some de facto developer and runtime environments for smartphones with Android and iOS, so there's no perceived need Java ME in this area.

Android Things[14]

Android Things aims to take the role of Java ME on embedded development: taking care of most embedded development tools and needs and make it available for mobile Android programmers. The same value proposition of Java ME: use the server side language and tools to develop for embedded. From their website: "Android Things makes developing connected embedded devices easy by providing the same Android development tools, best-in-class Android framework, and Google APIs that make developers successful on mobile."

Windows IoT[15]

Windows 10 IoT Core is a version of Windows 10 that is optimized for smaller devices with or without a display, and that runs on both ARM and x86/x64 devices. Windows 10 IoT Core utilizes the Universal Windows Platform (UWP) API for building solutions. This is, in some ways, very analogous to the Android Things example above.

Ubuntu Core[16]

While not directly a competitor, it does make using competing technologies easier.

[14] Android Things website
[15] Windows IoT website
[16] Ubuntu Core website

Eclipse Edje[17] and “small Java VMs”

Eclipse Edje defines a standard and portable high-level Java API called Hardware Abstraction Layer (HAL) for accessing hardware features delivered by microcontrollers. This is then implemented by a small Java ME targeting small processors (such as ARM Cortex M0-M4) that implements only a subset of the Java API.

mbedOS[18]

mbedOS from ARM has the same goals as Java ME for ARM processors: have basic IoT APIs and device management capabilities implemented in a stack that makes it easier to create new embedded devices. Its reach is broader than Java (as it defines a server-side component as well), but its role is directly related to what Java ME is on embedded devices. Many related standards are also implemented, such as the Thread stack, in a manner similar to how Java ME enables standards to be easily accessed by applications.

OSGi[19]

OSGi can be seen as a competing framework to Java ME, as it can be used to supplement Java SE with Java ME-like services (provisioning, configuration etc).

[17] Eclipse Edje page
[18] mbedOS website
[19] OSGi Alliance website