# RFP 189—Resource Enforcement

Draft

11 Pages

## Abstract

The purpose of this RFP is to enable the enforcement of resource restriction on bundles to ensure the robustness of the framework. The resources of a bundle may include CPU use, memory consumption, number of threads, network connections, and other limited resources. These are not just limited to the runtime environment (Java VM, OSGi Framework, etc), but can also include any resource required by a service.

JSR 282 and the Java security manager already provide mechanisms for such resource enforcement at a granular level, e.g., threads. Industrial solutions also make them available at the bundle level. This RFP advocates standardizing the set of features and their API in OSGi specifications.

This API should complement the existing resource monitoring API.

# 0 Document Information

## Table of Contents

## Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in .

`Source code is shown in this typeface.`

## Revision History

The last named individual in this history is currently responsible for this document.

| Revision | Date | Comments |
|---|---|---|
| Initial | April-05-2018 | Initial versions from aicas GmbH. |
| 0.1 | April-13-2018 | Update from internal review |
| 0.2 | April-17-2018 | Update from OSGi Meeting |
| | | |
| | | |

# 1 Introduction

Applications, executed on an OSGi platform, need hardware resources (CPU, memory, disk storage space) and software resources (sockets, threads). As these resources are limited,; applications have to share them in order to preserve system quality of service. This is particularly true of IoT systems.

Providing well defined resource management features is crucial for OSGi to succeed in the IoT market, as IoT requires high robustness even for systems open to third party applications. In this perspective, the framework administrator has to offer well defined guarantees to every actor sharing the platform and the system must be robust against faulty bundles. These guarantees must consider both the resources available and the importance of each bundle.

Resource enforcement is also vital to Cloud Computing scenarios where a management agent needs to ensure that SLAs agreed around the cloud offering are met. When a cloud node becomes overloaded or fails this can affect the agreed SLA and action needs to be taken. In a Cloud Computing scenario this may imply starting additional nodes, adjusting the provisioning state of the system by moving or adding deployments or indeed shutting down some nodes if the system has become quiet. To be able to handle such scenarios the management agent will need to have visibility of the resource utilization of the cloud system as a whole, which encompasses a multiple of nodes and runtimes.

For the moment, existing OSGi specifications do not provide resource enforcement mechanism ensuring a well defined resource sharing between bundles and applications. The underlying JVM provides only some standard mechanisms at a level that is too fine-grained, e.g., classes, objects, methods. Since the bundle is the smallest deployment unit of interest for platform administrators and application providers, this RFP describes needs and requirements of such features at the OSGi bundle level.

This specification is not only of interest for the realm of embedded devices but might be useful to other domains, such as Enterprise, where robustness is also important.

# 2 Application Domain

Resources of environments are always limited and entities that share such environments should be aware of that. This is not different for OSGi environments. Each bundle consumes resources of varying types. Some of them are required for the very basic operation, some others are optional, but all of them can run out and lead to situations where the bundle, a set of bundles that form an application, or even the framework as a whole is no longer operational.

Problematic situations arise when a software unit binds important resources but does not release them after normal operation. This can be caused by faulty implementations, incorrect error handling, or by intention in case of malware. Especially in environments with very limited resources or with a huge number of bundles from varying vendors, it is crucial to enforce limits on the resource use of each bundle.

## 2.1 What are resources?

There are some obvious, basic resources like CPU, memory, disk-space, communication bandwidth. But new applications might introduce the need for new types of resources that are required for their normal operation, e.g., the presence of external services and devices. In order to support this flexibility, it is impossible to provide a complete list of potential resources here, rather a general mechanism for resource enforcement must be provided.

## 2.2 Most common and crucial resources

Applications uses hardware and operating system resources. Some core resources are

- CPU
- Memory
- Disk storage space
- Bandwidth on connected networks

JVMs allocate these resources when applications call Java standard APIs. They may provide resource monitoring mechanisms such as

- Java Management Extension (JMX), now provided by all J2SE-v5-compliant JVM,
- JVM Tool Interface Interface(JVMTI) and JVM Profiler Interface (JVMPI),
- Proprietary resource management API, e.g., IBM J9, Oracle Java Embedded Client, /K/ Embedded Mika Max, Myriad Jbed, and
- The Reatlime Specification for Java (RTSJ)[10]., e.g., JamaicaVM.

The latter provide strict algorithms that charge bundles with consumed resources. There are two known algorithms [3].:

- direct accounting—the resources consumed during bundle interaction are accounted to the code provider, i.e., the CPU used by a code that belongs to bundle A will be accounted to A, even if it is the bundle B that called this code through a public interface;
- indirect accounting—all the resources consumed by the threads belonging to a bundle are accounted to this same bundle, thus, in service interaction there is no resource consumption accounted to service providers.

Java and OSGi enables CPU monitoring per bundle on any VM (without any VM customization) [3].[4]., but not CPU use enforcement; however, the RTSJ 2.0 provides ProcessingGroups, which can provide such enforcement.

Though, memory enforcement can be supported on standard VMs, they would need significant additional memory for tracking live objects. The overhead is on the order of O(n) in the number of live objects. Again, RTSJ 2.0 provides MemoryGroups, which could improve enforcement efficiency.

## 2.3 What is Robustness?

Robustness of an entity (service, bundle, set of bundles, or the whole framework) is meant as the state where the entity is operational as it was specified and cannot be negatively impacted by other parts of the system. The correct operation of such an entity is often strongly related to the availability and to the state of the resources the entity needs to do its work. That means an entity that does not have or can obtain the required resources is not robust. There might also be intermediate states where mandatory resources are there, but some optional ones are not available.

Other reasons for the lack of robustness are potential failure situations either inside the entity itself or in their environment. Sometimes such conditions cause shortage of other resources, which at the end affects other entities as well. Being able to kill rogue applications is essential to maintaining the robustness of the rest of the system.

So, in order to ensure the robustness of entities, the first step is to ensure that no entity can use more resources than it is allotted. This could be via throttling or by denial. Then there must be an instance that ensures that attempts to overrun resource allotments are handled to ensure system overall robustness.

## 2.4 Terminology and abbreviations

**Application**
A set of bundles needed to render a full application to the user.

**Resource Context**
An entity that is subject of resource monitoring. In the scope of this document this can be a framework, a bundle or a set of bundles.

**Health**
The state of an observable that describes its ability to work as specified.

**Resource**
A limited source or supply of physical or virtual goods that are used by bundles in order to provide their service(s).

**Fault**
The term fault is a defect at the lowest level of abstraction, e.g., a memory cell that always returns the value 0.

**Error**
A fault may cause an error, which is a system state.

**Failure**
An error, in effect, may lead to a failure, meaning that the system deviates from its correctness specification or an error from which an application cannot recover.

# 3 Problem Description

OSGi platforms host several applications which are executed concurrently. These applications have to share limited resources between them. The current framework relies on cooperative execution of bundles. The means that If any of the bundles hoards resource by demanding too much of them, other bundles will fail to run properly. The result is that a rogue or misconfigured bundle can take down the entire system. For a system to be robust, this must be prevented. A solution must provide for both bundles that are aware of the solution and those that are not.

## 3.1 Base Mechanism

The base mechanisms should prevent a bundle from using more resources than it is allowed to use. Most resource overruns are indicated with an error, such as OutOfMemoryError. However, CPU use (and possibly other rate based resources, such as bandwidth) is a special cased. Here RTSJ priorities can be used to delay execution to prevent overuse.

Core virtual machine resources, such as CPU and memory use, need to supported by the underlying Java virtual machine. Most other resources can be enforced with a modified security manager. The only other exception may be rate-based limits such as bandwidth use, which may need more than just an enhanced security manager.

In any case, a standard means of extending a security manager for nonstandard resource enforcement should be provided. However, the limits for any given bundle should be set at bundle start, so that a reasonable admission policy can be provide for deciding how many bundle may be running at a given time.

In extreme cases, the framework should be able to terminate a bundle while releasing any bound resources.

## 3.2 Enforcement-Aware Applications

A bundle should be able to find out what resource limits are being imposed on it. This would enable a bundle to use the resource monitoring API to provide warnings before the limits are reached. An enforcement-aware application could then modify its behavior before it receives an enforcement error. Of course, an aware application could also handle enforcement errors directly.

# 4 Use Cases

When an OSGi framework is used to control a device, it should not be possible for a rogue bundle to freeze the framework by hoarding any resource. The resources available to each bundle must be declared before the bundle starts and if the bundle tries to exceed any of those limits, it should be prevented from doing so, either passively via rate limiting or actively via a resource limit error. Any bundle that becomes nonresponsive should be killed.

## 4.1 Passive Example

A bundle tries to use more CPU than it is allowed to use. Here CPU use is determined via some share of the compute cycles within a fixed, usually short time period. In this case, the priority of all threads in the bundle are dropped to a priority under the idle thread until the period end. The priority is then restored to the original priority.

## 4.2 Active Example

A bundle tries to use more memory than it is allowed to use. The bundle receives an OutOfMemoryError on that allocation. It is up to the application to respond to the error properly.

## 4.3 Unresponsive Bundle Example

A bundle becomes unresponsive. This could be due to not making progress on some activity, failing to respond properly to a resource limit error, or just some other failure. The system should be able to detect this and terminate the bundle safely.

# 5 Requirements

R1: The solution MUST provide at least one accounting algorithm for each resource type, using the the existing monitor specification where appropriate.

R2: The solution MUST monitor resources per bundle or group of bundles and take action to prevent resource limit violations.

R3: The solution MUST provide a mechanism to enforce limits on the following resources, where relevant on the underlying (hardware and software) platform:

- CPU
- Memory
- Disk storage space
- Bandwidth on any network connection.

R4: The solution MUST provide a mechanism to obtain a list of resource types that are limited and what those limits are.

R5: The solution MUST enforce CPU limits over a defined period.

R6: The solution MUST define thread thresholds as a number of threads.

R7: The solution MUST define socket thresholds as a number of opened sockets.

R8: The solution SHOULD be able to defined policies for handling resource violations.

R9: The solution MUST define means for specifying a bundle’s resource limits.

R10: The framework MUST be able to terminate a bundle without loosing resources.

R11: As many existing bundles as feasible that do not require resource management SHOULD run unmodified.

R12: Predeployment analysis tools MUST be provided to help ensure system robustness.

R13: The system SHOULD be able to enforce that only vettered code may runnable on it.

# 6 Document Support

## References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

[3]. T. Miettinen, D. Pakkala, and M. Hongisto. A method for the resource monitoring of osgi-based software components. In Software Engineering and Advanced Applications, 2008. SEAA '08. 34th Euromicro Conference, pages 100 {107, 3-5 2008.

[4]. Y. Maurel, A. Bottaro, R. Kopetz, K. Attouchi. Adaptive Monitoring of End-user OSGi-based Home Boxes. In Component Base Software Engineering, 15th ACM SigSoft International Symposium on Component-Based Software Engineering (CBSE 2012), Bertinoro, Italy, June 2012

[5]. C. Larsson and C. Gray. Challenges of resource management in an OSGi environment. In OSGi Community Event 2011, Darmstadt, Germany, September 2011.

[6]. N. Geofray, G. Thomas, G. Muller, P. Parrend, S. Frénot, and B. Folliot. I-JVM: a java virtual machine for component isolation in osgi. In Proceedings of the 2009 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, pages 544{553. IEEE, 2009.

[7]. Java Community Process, Java Specification Request 163, Java Platform Profiling Architecture, final release, September 2004.

[8]. *The Java Virtual Machine Specification*, Second Edition by Tim Lindholm and Frank Yellin. Addison-Wesley, 1999, ISBN 0-201-43294-3.

[9]. *The Java Language Specification, Third Edition, May 2005, ISBN 0-321-24678-*

[10]. *The Realtime Specification for Java 2.0., JSR 282, http://www.aicas.com/cms/rtsj*

## Author’s Address

| | |
|---|---|
| Name | James J. Hunt |
| Company | aicas GmbH |
| Address | Emmy-Noethe-Straße 9, 76131 Karlsruhe, Germany |
| Voice | +49 721 663 968-22 |
| E-mail | jjh@aicas.com |

## End of Document