Abstract
The purpose of this RFP is to enable the monitoring of the health of bundles and the framework and provides means to recover from irregular states. The state of health of a bundle can include memory consumptions, number of threads, network connections, and other limited resources. These are, however, not limited to the runtime environment (Java VM, OSGi Framework, etc), but can also include any kind of resources that are important to the services itself.

Java APIs already enables the monitoring of such resources at a granular level, e.g., threads. Several industrial solutions also make it available at the bundle level. This RFP advocates the need to standardize the set of features and their API in OSGi specifications.
0 Document Information

Table of Contents

0 Document Information..................................................................................................................2
  0.1 Table of Contents....................................................................................................................2
  0.2 Terminology and Document Conventions...............................................................................2
  0.3 Revision History.....................................................................................................................2

1 Introduction..................................................................................................................................3

2 Application Domain....................................................................................................................3

3 Problem Description...................................................................................................................4

4 Use Cases...................................................................................................................................4
  4.1 Resource Monitoring................................................................................................................4
  4.2 Eventing..................................................................................................................................4
  4.3 Error......................................................................................................................................5

5 Requirements.............................................................................................................................5

6 Document Support......................................................................................................................6
  6.1 References...............................................................................................................................6
  6.2 Author’s Address......................................................................................................................6
  6.3 End of Document......................................................................................................................7

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.
### 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 a general fact in the Enterprise and Residential markets.

Providing fair resource management features is crucial for the Smart Home to emerge as
Residential players are opening their gateway (or box) execution environment to third party applications. In this perspective, the framework administrator has to fairly offer the same guarantees to every actor sharing the platform.

Resource Monitoring 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 gets overloaded or fails this can affect the pre-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 multiplicity of nodes and runtimes.

For the moment, existing OSGi specifications do not provide monitoring resource mechanism ensuring a fair 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. The bundle being the smallest deployment unit of interest for platform administrator 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 provide service to other domains like Enterprise as well.

## 2 Application Domain

Resources of environments are always limited and entities that share such environments should be aware of that. This is not different in OSGi environments. Each bundle consumes resources of different types. Some of them are required for the very basic operation, some others are nice to have, 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 not operational anymore.

Problematic situations arise when a software unit binds a lot of resources but does not release them after normal operation. This can be caused by wrong implementations, wrong error handling or by intention in case of malware. Especially in environments with very limited resources and/or with a huge number of bundles/vendors it is crucial to monitor the state of bundles and their resource consumption and also to provide mechanisms to react on detected failures.
2.1 What are resources?
There are some obvious, basic resources like CPU, memory, disk-space, bandwidth. But new applications might introduce the need for new, different types of resources that are required for their normal operation (e.g. the presence of certain external services and devices, room temperature etc.). Because of that it is impossible to provide a complete list of potential resources here. The following figure tries to illustrate that:

![Figure 1: Origins of resources](image)

Every circle in this picture stands for a certain resource. As illustrated these resources can come from:
- the same OSGi Framework (e.g. service instances, exported packages ...),
- the same Java VM (e.g. threads, memory ...),
- the device (e.g. USB-Ports, network interfaces/ports ...),
- the local environment (e.g. room temperature, power consumption of the device, geo-location ...),
- or from completely external locations (e.g. special external services like maps, dictionary ...)

2.2 Most common and crucial resources
Applications uses hardware and operating system resources. Targeted 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)

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. In other words, 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. Therefore in service interaction there is no resource consumption accounted to service providers.

Java and OSGi enables CPU Management per bundle on any VM (without any VM customization) [3],[4]. Memory management per bundle requires the customization of VMs [5]. [6].

2.3 What is Healthiness?

Healthiness 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 will be for the foreseeable future. The correct operation of such an entity is often strongly related to the availability, and perhaps a certain quality, of resources that the entity needs to work. That means an entity that doesn’t have or get the required resources is not healthy. There might also be intermediate states where mandatory resources are there, but some optional ones are not available.

Other reasons for non-healthy entities 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.

So, in order to ensure the healthiness of entities the first step that should be done is to ask themselves, “how do you feel?” As a second step it is important to know for entities their resource requirements and to monitor their availability.

2.4 Terminology and abbreviations

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

Observable
An entity that is subject of Health 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 usually used to name 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 category of the system state.

Failure
An error, in effect, may lead to a failure, meaning that the system deviates from its correctness specification.

3 Problem Description

OSGi platforms host several applications which are executed concurrently. These applications have to share limited resources between them.
3.1 Cooperative applications

These mechanisms should also allow to estimate the severity of the situation and to decide for required actions to recover the intended state. Ideally, this should be done in cooperation with the bundle that causes the failure. If a failure situation is detected and can be assigned to a certain bundle, then first this bundle should have the chance to take actions to come back to a healthy state. If this is not successful, then appropriate actions must be taken by another entity.

Due to the wide range of potential failures and the definition of resources as very generic and application specific, this can not be achieved by a fixed and inflexible mechanism that handles a fixed set of predefined problems.

Needed is a flexible framework that allows dynamic provisioning of modules to:

- collect information about resource requirements, and further, the normal, intended states of the monitored entities,
- monitor those resources (as defined above) and ask services for their health status,
- warn interested and legitimate applications when monitored consumptions are above thresholds.
- evaluate the severity of deviations of the currently monitored state from the intended state,
- take decisions and perform actions to recover the intended state,
- control/monitor the success of the actions taken.

For the monitoring part the MonitorAdmin service looks like a logical candidate, perhaps extended with new features that might be necessary.

3.2 Less cooperative or legacy applications

In case of an application consumes too much resources, it may affect the quality of service of the other applications installed on the platform. Those situations have to be prevented by OSGi platforms.

As described in the previous chapter, JVMs may provide resource management mechanisms. However, all these solutions are designed to monitor low granularity elements: e.g., threads, classes, objects or methods.

As such, these data are of limited interest and there is a need to raise the abstraction to the
primitive deployment unit in OSGi, bundles and applications (or sets of bundles). This encourages the specification of a standard unified OSGi-level API managing resources of bundles and sets of bundles installed on the platform.

4 Use Cases

Resource Monitoring
Applications, hosted on the OSGi platform, use implicit standard Java APIs for requesting resources such as CPU, memory, disk storage space, threads and sockets. The OSGi platform administrator monitors resource allocation for every bundle and application considered as a set of bundles.

Resource Monitoring activation and deactivation on a bundle basis
Monitoring mechanisms are inactive most of the time and the resource manager triggers the more accurate monitoring mechanisms to monitor the suspected part of the system – a bundle or a set of bundles – only when a problem is detected.

Eventing
The OSGi platform raises events when the resource consumption of a bundle has reached a threshold or a resource is going to become unavailable.

These events could be used by administrator applications to execute appropriate actions, e.g., stop non critical bundles that consumes too many resources, in order to preserve application quality of service.

Errors and resource quotas
The OSGi platform assigns resource quotas to bundles or to a set of bundles according to platform resource constraints and according to requirements declared in bundle metadata. If the execution of one of the bundles or of the set of bundles exceeds a quota, the OSGi platform throws an exception or an error during the execution of the relevant method and makes the resource no more available for the calling bundle. The targeted application may take any appropriate action to reduce resource usage.
Resource requirements declaration

A bundle defines its needs in terms of a special resource (e.g. availability of certain TCP/IP ports) and wants to be notified, as soon as those resources become available.

Taking actions when problems are detected

![High-Level flowchart of monitoring and resolving cycles](image)

The following figure shows an exemplary high-level sequence of operations that such a framework could have.

**Figure 2 High-Level flowchart of monitoring and resolving cycles**

This framework may include the bundle itself into the resolving process. Bundles may have a chance to solve their problems before some stronger actions might be taken by a privileged entity, e.g. restart/stopping of that certain bundle.

The example above just describes a very basic policy. Of course the mechanism should have advanced algorithms and policies that might take more of the history into account. Also for some well-known and always interesting types of resources there might be standard entities that work on behalf of other bundles.
5 Requirements

R1: The solution MUST provide resource monitoring OSGi mechanisms with direct accounting algorithm or indirect accounting algorithm or both algorithms.

R2: The solution MUST monitor resources per bundle or per bundle set.

R3: The resource monitoring solution MUST be configurable, enabled and disabled at runtime per bundle or per bundle set.

R3: The solution MUST monitor the following resources, if relevant on the underlying (hardware and software) platform:

- CPU
- Memory
- Disk storage space
- Bandwidth on any connected network

R4: The solution MUST provide a mechanism to list the resource types that can be monitored on the underlying (hardware and software) platform.

R5: The solution MUST allow the setting of a warning threshold and an error threshold per bundle or set of bundles.

R6: The solution MUST send events while a bundle or a bundle set is exceeding one of the two thresholds defined by R4.

R7: The solution MUST define CPU thresholds as a percentage of use over a configurable period.

R8: The solution MUST define memory thresholds as bytes.

R9: The solution MUST define disk storage space thresholds as bytes.

R10: The solution MUST define thread thresholds as a number of threads.

R11: The solution MUST define socket thresholds as a number of opened sockets.

R12: The solution MUST be able to lower bundle thread priorities while CPU error threshold is
R13: The solution MUST raise an error (e.g., OutOfMemoryError) and MUST prevent further memory allocation while memory error threshold is reached.

R14: The solution MUST raise an error (e.g., IOException) and MUST prevent further disk storage space allocation while disk storage space error threshold is reached.

R15: The solution MUST raise an error (e.g., InternalError) and MUST prevent further thread activation while thread error threshold is reached.

R16: The solution MUST raise an error (e.g., IOException) and MUST prevent further connected-state socket while socket error threshold is reached.

R17: The solution MUST define means for bundles to define their intended resource usage.

R18: The solution MUST allow OSGi applications to monitor bundles, evaluate their states and take decisions to react gracefully.

R19: The solution MAY define optional means for a bundle to resolve its own conflicts based on the decisions of the entity introduced in R17 For the OSGi framework such a mechanism is mandatory.

R20: Thanks to notification from R5, an application able to monitor the success of R18 MAY take actions, if the conflicts are not resolved after a period of time. Default action MAY be that the framework mechanism (R18) MAY resolve this conflict.

R21: The solution MUST provide a mechanism that allows to plug application specific components to evaluate application specific resources.

6 Document Support

References


Applications, 2008. SEAA '08. 34th Euromicro Conference, pages 100 {107, 3-5 2008.


Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Steffen Drüsedow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Deutsche Telekom AG</td>
</tr>
<tr>
<td>Address</td>
<td>Telekom Innovation Laboratories, Winterfeldtstrasse 21, 10781 Berlin, Germany</td>
</tr>
<tr>
<td>Voice</td>
<td>+49 30 835358-139</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:Steffen.druesedow@telekom.de">Steffen.druesedow@telekom.de</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Andreas Kraft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Deutsche Telekom AG</td>
</tr>
<tr>
<td>Address</td>
<td>Telekom Innovation Laboratories, Winterfeldtstrasse 21, 10781 Berlin, Germany</td>
</tr>
<tr>
<td>Voice</td>
<td>+49 30 835358-185</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:Andreas.kraft@telekom.de">Andreas.kraft@telekom.de</a></td>
</tr>
<tr>
<td>Name</td>
<td>Alexander Manecke</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>Company</td>
<td>Deutsche Telekom AG</td>
</tr>
<tr>
<td>Address</td>
<td>Telekom Innovation Laboratories, Winterfeldtstrasse 21, 10781 Berlin, German</td>
</tr>
<tr>
<td>Voice</td>
<td>+49 30 835358-356</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:alexander.manecke@telekom.de">alexander.manecke@telekom.de</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Andre Bottaro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>France Telecom Orange</td>
</tr>
<tr>
<td>Address</td>
<td>28 Chemin du Vieux Chêne, 38240 Meylan, France</td>
</tr>
<tr>
<td>Voice</td>
<td>+33 4 76 76 41 03</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:andre.bottaro@orange.com">andre.bottaro@orange.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Arnaud Rinquin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>France Telecom Orange</td>
</tr>
<tr>
<td>Address</td>
<td>28 Chemin du Vieux Chêne, 38240 Meylan, France</td>
</tr>
<tr>
<td>Voice</td>
<td>+33 4 76 76 45 59</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:arnaud.rinquin@orange.com">arnaud.rinquin@orange.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Gregory Bonnardel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>France Telecom Orange</td>
</tr>
<tr>
<td>Address</td>
<td>28 Chemin du Vieux Chêne, 38240 Meylan, France</td>
</tr>
<tr>
<td>Voice</td>
<td>+33 4 76 76 45 59</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:gbonnardel.ext@orange.com">gbonnardel.ext@orange.com</a></td>
</tr>
</tbody>
</table>

End of Document