Monitoring the Distributedness of Java Applications

Grzegorz Wianecki, Mateusz Wójcik, Włodzimierz Funika, Marcin Smętek

[1]Institute of Computer Science AGH, Mickiewicza 30, 30-059 Kraków, Poland

[2]Academic Computer Centre CYFRONET, Nawojki 11, 30-950 Kraków, Poland

email1@cyf-kr.edu.pl, email2@cyf-kr.edu.pl

1 .Introduction

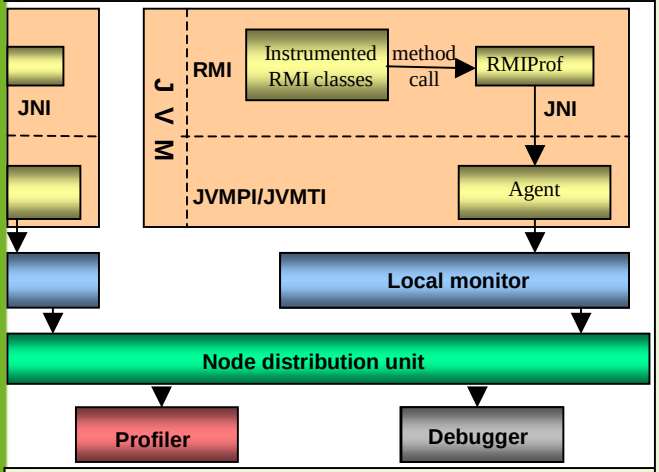
When designing distributed systems there are additional elements that need to be taken into consideration compared to sequential systems:

- communication
- synchronization
- load-balancing

2. Goals

The monitoring of distributed systems needs to focus on the mentioned aspects and allow to correlate data and events to form a complete picture of the application. The RMI extension to J-OCM was designed to monitor both RMI-based application and the internal RMI mechanisms with emphasis on Activation.

3. RMI Extension to J-OCM architecture



4. Implementation concept

The approach used in the RMI extension to J-OCM relies on Java classes instrumentation. Several RMI and Activation classes were modified to notify the RMIProf class, which in turn passes the events via JNI to the JOCM agent running within the Virtual Machine.

The following RMI classes were modified:

- `sun.rmi.registry.RegistryImpl`
- `sun.rmi.server.ActivatableRef`
- `sun.rmi.server.Activation`
- `sun.rmi.server.LoaderHandler`
- `sun.rmi.server.DGCImpl`

5. Entities identification

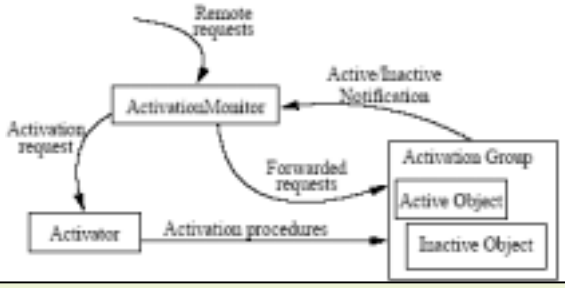
Both RMI application and RMI system itself run on several Virtual Machines. The RMI and Activation entities are accessed from different Virtual Machines and need to be consistently identified in the system.

The identification mechanism used in the extension is based on *hashcode* field of each Java object. RMI specification states that remote references to RMI and Activation objects that point to the same object have the same *hashcode* number. This allows the entities to be identified from different Virtual Machines.

6. Activation mechanism

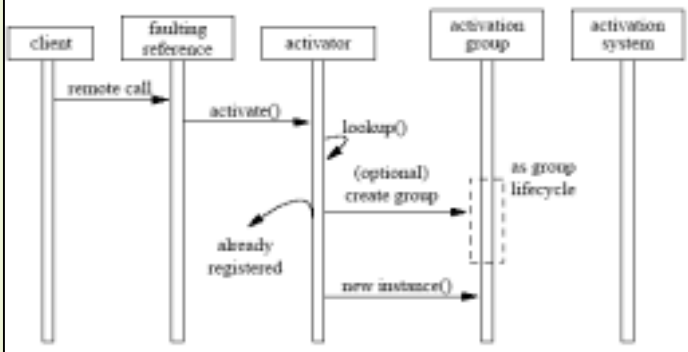
Activation is the process of delaying object initialization until it is used for the first time. It also allows to free the object's resources when it is not used while storing its persistent state.

The following entities are involved in Activation operations:



7. Activation flow

The following diagram presents the activation execution flow:



8. Execution procedure

Java Virtual Machine needs to be instructed to load the modified classes and the J-OCM agent to work with the monitoring system. As the application and RMI systems (*rmiregistry* and *rmid* – RMI activation daemon) run in different Virtual Machines need to have J-OCM support enabled. In addition as *rmid* spawns Virtual Machine for each ActivationGroup created, J-OCM needs to be enabled also for these ones.

```
rmiregistry –J-Xdebug –J-XrunAgent –J-Xbootclasspath/p:$INSTRUMENTED_CLASS_DIR

rmid –J-Xdebug –J-XrunAgent –J-Xbootclasspath/p:$INSTRUMENTED_CLASS_DIR
     –C-Xdebug –C-XrunAgent –C-Xbootclasspath/p:$INSTRUMENTED_CLASS_DIR

java –Xdebug -XrunAgent –Xbootclasspath/p:$INSTRUMENTED_CLASS_DIR MainClass
```

9. References

[1] Marcin Smetek, "OMIS-based Monitoring System for Distributed Java Applications", M.Sc. thesis, AGH, Krakow, 2003

[2] Grzegorz Duda, "Monitoring RMI calls in Java applications", M. Sc. thesis, AGH, Krakow, 2003

[3] The Java Virtual Machine Specification, Sun Microsystems.

[4] http://java.sun.com/docs/books/vmspec/