Dynamic Instrumentation of Distributed Java Applications

Włodzimierz Funika, Paweł Świerszcz
Institute of Computer Science AGH, Mickiewicza 30, 30-059 Kraków, Poland

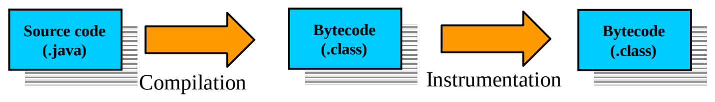
Dynamic Instrumentation

Instrumentation is a process in which existing program's code is enriched with additional functionality by injecting instructions implementing some new features, mainly intended for monitoring.

Original application's behaviour is not altered. Dynamic instrumentation means that running systems are be enhanced on-the-fly without stopping and restarting.

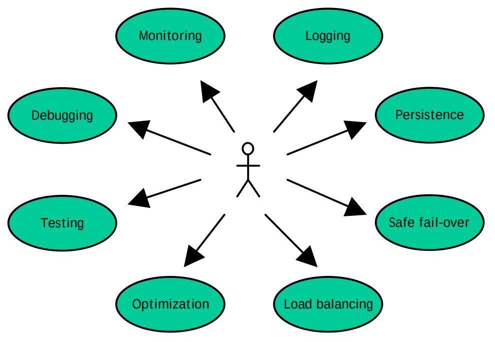
Goals of project

The main goal of this project is to present a proposal of a system that will support the developer in instrumenting Java applications. This support consists of automating some of the tasks that are required steps of instrumentation process. In this way the developer can concentrate on a higher abstraction level – designing instrumentation elements, choosing instrumentation spots, combining the created additional functionality with existing application classes in order to create a functional execution unit.



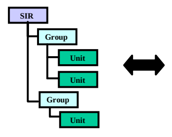
After being instrumented classes are indistinguishable from those not instrumented.

Use cases

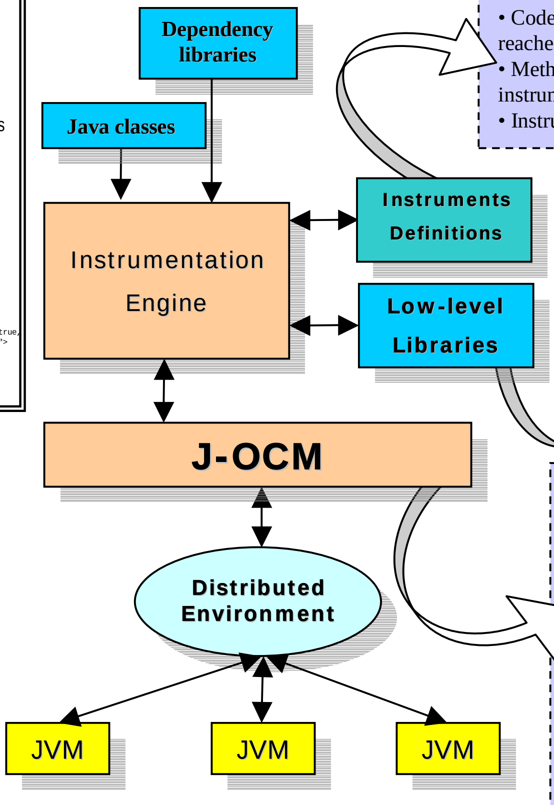


SIR

SIR (Standard Intermediate Representation) specifies an uniform way of representing an application's code structure. It consists of tree hierarchy with a **SIR** root element for whole application and it's nested subelements describing code details (classes, methods, fields, etc. In case of Java). It is generic and allows for representing applications in many languages, in an extendable way and can easily be transformed into XML.



```
<sir>
  <language>java</language>
  <group name="tests" type="package">
  <group name="tests/TestApp" type="class">
    <variable name="staticString" isField="true,
type="java/lang/String;" parameterIndex="-1">
    <unit name="logStack" arguments=""/>
  </group>
</sir>
```



- Define hook points in the application where an instrument can be applied
- Code to be executed when execution flow reaches hook point
- Methods and fields required by the instrument to work properly
- Instrument parameters

- Used to perform low-level operations like XML processing (**Castor**) or bytecode introspection and modification (**Java Object Instrumentation Engine**, **Bytecode Engineering Library**)

J-OCM is a system for monitoring distributed Java applications conforming to **OMIS** specification with **J-OMIS** extensions.

It is used to:

- gather information on distributed application
- transfer instrumented classes to their target locations
- redefine classes in remote **JVM**s
- return results from instruments

Usage scenario

- The instrumentation environment is prepared (classes to be instrumented, instruments' definitions)
- Instrumented classes are loaded and examined, a SIR tree is created for instrumented application
- Instrumentation design is prepared (instruments connected to classes)
- Classes are modified, instruments injected into methods
- J-OCM system initialized, application tokens obtained
- Classes redefined on remote nodes with J-OCM
- Instruments now active and working transparently, possibly returning results via J-OCM event services

Bytecode modifications

The nstrumentation system works entirely on binary classes. This means that for each instrument injection point, bytecode instructions specific for that hook have to be located. Then a special bytecode section is prepared taking into consideration a code location, hook point context and instrument parameters. This section is then inserted into methods code so it passes execution flow to instrument along with all data needed (like the current method's name or variable value, depending on the particular instrument). This has to be performed very carefully not to violate class correctness or application stability.

CYFRONET
kbn STATE COMMITTEE FOR SCIENTIFIC RESEARCH