DO178C Virtualization Software
Java 32bit Processor Use Case

IS2T / Capgemini / DGA
JERTIF

CTIC 2013, Toulouse, FRANCE

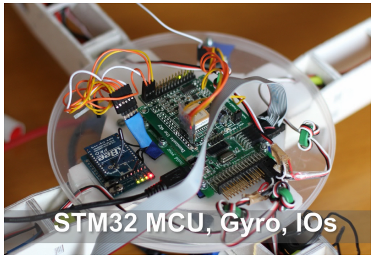
May 2013

Fred RIVARD (fred.rivard@is2t.com)
Frédéric RIVIERE (friviere@is2t.com)
Hugues Bonnin (hugues.bonnin@capgemini.com)

v1.0/D

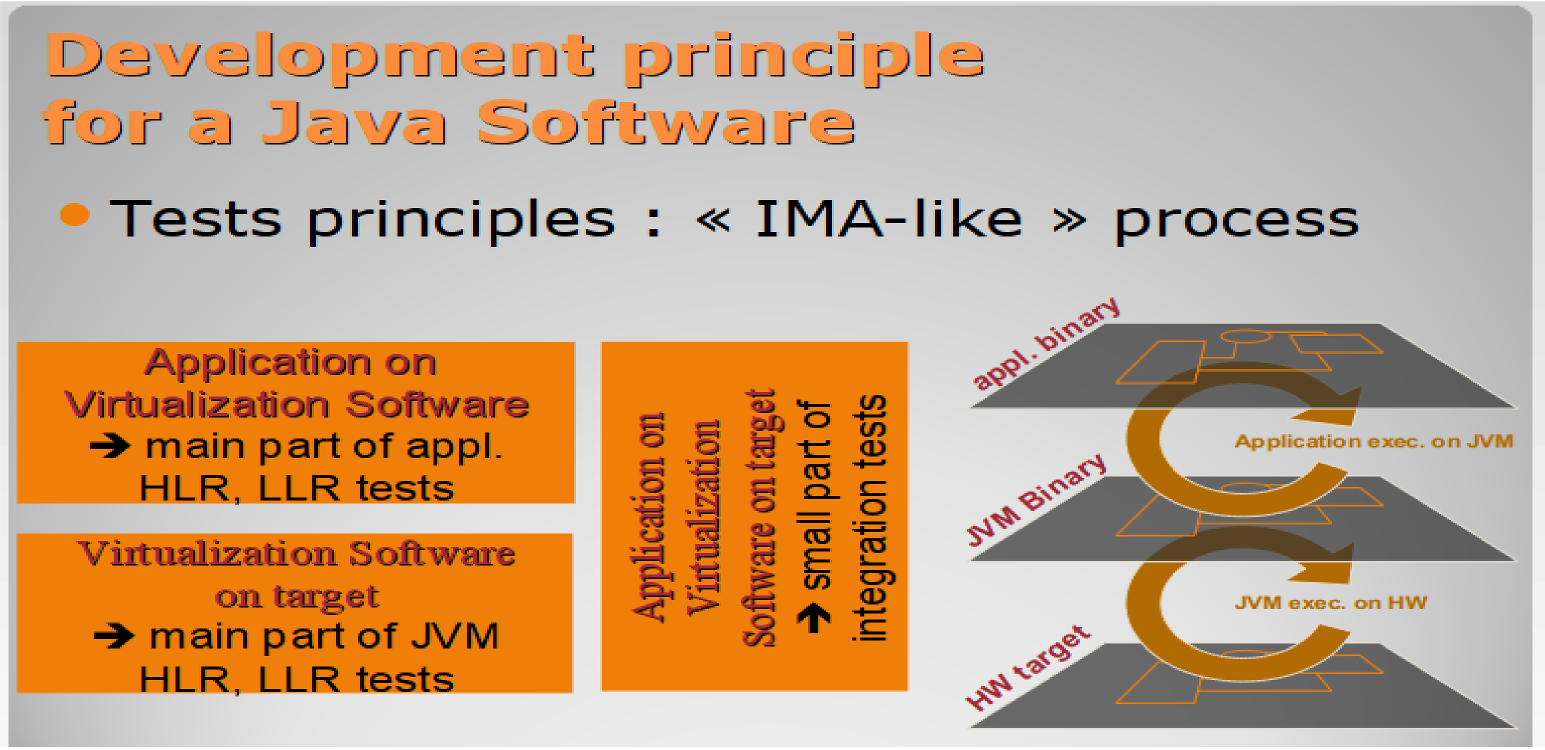
Agenda

- **DO178C/OOT supplement**
  - Virtualisation Software
  - Memory Management Infrastructure
  - Hard Realtime Java
- **JERTIF Project**
  - Activities
  - Main aspects used in of the standards ED12-C/DO178-C, ED217/DO332
- **Particular studied points of interests**
  - Specification, Verification & Validation of a Java Virtualization Software
- **Full HRTJ Quadrone device realization**
  - Hardware, Java VM, Java application
  - Video
- **Conclusion & Future**



Virtualization Software

- **DO178C & DO332 (OOT Supplement)**
  - OO.4 “The target environment is either a target computer or **a combination of virtualization software and a target compute**r. Virtualization software also needs to comply with DO-178C/ED-12C and applicable supplements”



Development principle for a Java Software

- Tests principles : « IMA-like » process

Application on Virtualization Software ➔ main part of appl. HLR, LLR tests

Virtualization Software on target ➔ main part of JVM HLR, LLR tests

Application on Virtualization Software on target ➔ small part of integration tests

Process Activities
Planning, Configuration, SDS

- **DO178C & DO332**
  - OO.4.2 m. : « Describe any planned use of virtualization » and « This data [byte code] should be treated as executable code »
  - OO.D.1.7.1 : main vulnerability is « incorrectly categorizing programming instructions as data. Consequently, tracing may be neglected, requirements may be inadequate or missing, and verification may be insufficient. »
  - OO.11.7 g., OO.11.8 f. : standards (design and code) must include constraints on usage of virtualization

- **Memory Management Infra.**
  - (a) Ambiguous References
  - (b) Fragmentation Starvation
  - (c) Deallocation Starvation
  - (d) Heap Memory Exhaustion
  - (e) Premature Deallocation
  - (f) Lost Update and Stale Reference
  - (g) Time bound Allocation or Deallocation

Java proc.

Technique	Sub-objectives (OO.6.8.2)						
	a	b	c	d	e	f	g
Object pooling	AC	AC	AC	AC	AC	N/A	MMI
Stack allocation	AC	MMI	MMI	AC	AC	N/A	MMI
Scope allocation	MMI	MMI	MMI	AC	AC	MMI	MMI
Manual heap allocation	AC	AC*	AC	AC	AC	N/A	MMI
Automatic heap allocation	MMI	MMI	MMI	AC	MMI	MMI	MMI

MMI : Memory Management Infrastructure **AC : Application**

Why using virtualization ?

- **Software productivity**
  - Virtualization has multiple known interests for productivity and industrialisation
  - Software / Hardware loosely coupled
  - Simulation made easier
  - Portability improved
- **Safety reasons**
  - Breakdown of increasing software complexity
    - « divide and conquer »
    - Each layer only manipulates entities that makes sense to it
  - In case of Java : the virtualization software is ultra stable (+10 years) with formal proof of the binary-Java-code verifier semantic (bytecode verifier)
- **Reduce integration costs & ease the use of "agile process"**
  - Reduce cycle-time, reduce batch-size, manage complexity "step by step", perform activities as early and often as possible, provide feedbacks

HRTJ principles

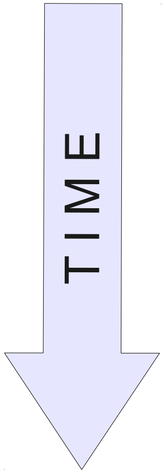
- **Purely cyclic tasks based system**
  - 2 phases: 1. initialization (mono-task); 2. mission (multi-task, cyclic)
  - Scheduling method: priority ceiling protocol
  - HRT task:
    - Period and priority are compile-time constants (no change at runtime)
    - The run()method is executed without interruption, except when the task is preempted by a higher-priority task
    - No blocking method (halt, sleep, wait, etc.)
- **+ one more cyclic task for the MMI activity**
  - MMI task: a HRT task with the lowest priority
    - => MMI activity is preemptible
    - => MMI activity is executed when all other tasks are done
- **Bounded live memory**
  - The maximum size of the live memory is known
  - For formula, see Baker (1992), Schoeberl (2006)

$$\boxed{T_{GC} \leq \frac{H_{CC} - 2\sum_{i=1}^{n} a_i l_i - 2\sum_{i=1}^{n} a_i}{2\sum_{i=1}^{n} \frac{a_i}{Ti}}}$$

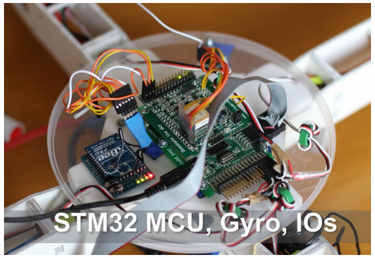
JERTIF Goal & Time line

- **Use DO178 to design a HRTJ Virtualization Software**
  - (1) IS2T provided its Java technology and processes
  - (2) Capgemini was responsible for the software application design and implementation of the small quadrotor UAV
  - (3) ACG Solution and DGA provided their strong expertise

- **18 months project: ended in Dec. 2012**



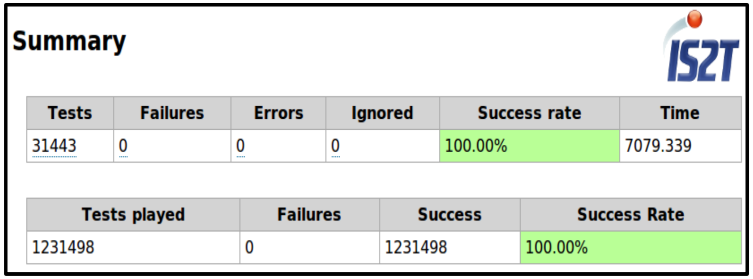
  - (1) Audit of IS2T current process
  - (2) Analyze of the gap & Actions plan
  - (3) PSAC for a Java Virtualization Software DO178-C Level A
  - (4) Design of a Java Virtualization Software for a quadrotor and its application
    - Only the MMI design using the DO178C-Level A process
  - (5) Tests DO178-C level A for the MMI
  - (6) Final dry-run audit



STM32 MCU, Gyro, IOs

Existing process

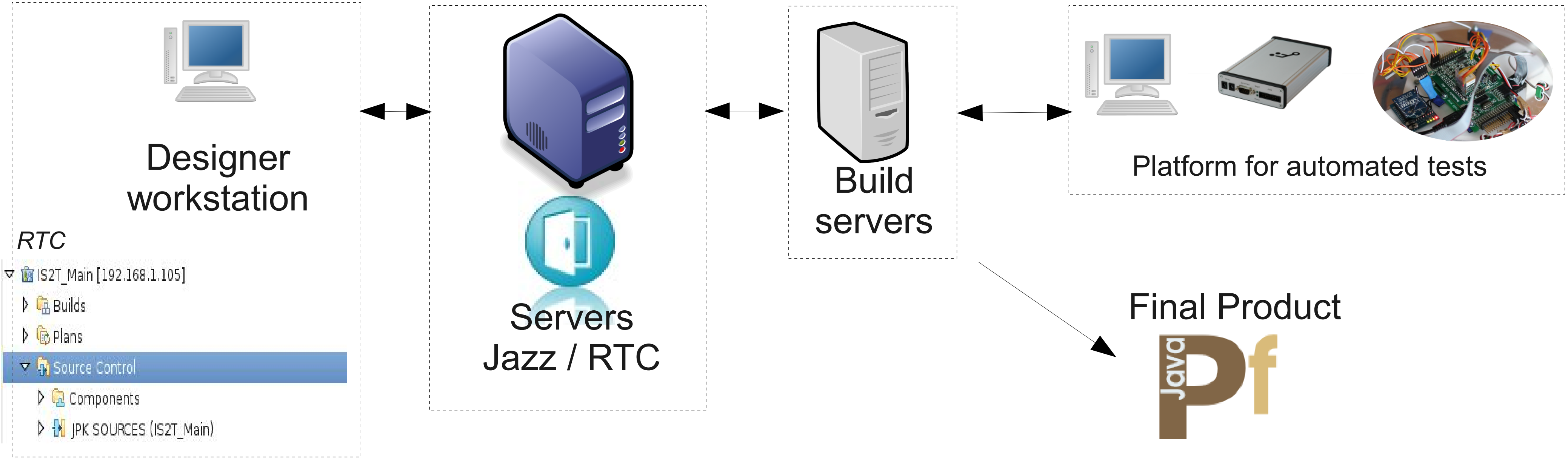
- **Automatic build addiction & automated tests suite on targets**
  - Source Control, Work Items & Requirements, Plannings, Builds
  - Automated tests
    - Typical numbers : more than 30.000 tests, run more than 1.000.000 times for one Java Virtualization Software
    - After each incremental V sprint, replay all the tests
  - Binary Level code coverage
  - Inline with DO178 spirit & “way of doing”



Summary

Tests	Failures	Errors	Ignored	Success rate	Time
31443	0	0	0	100.00%	7079.339

Tests played	Failures	Success	Success Rate
1231498	0	1231498	100.00%



Designer workstation

RTC

- IS2T_Main [192.168.1.105]
  - Builds
  - Plans
  - Source Control
    - Components
    - JPK SOURCES (IS2T_Main)

Servers Jazz / RTC

Build servers

Platform for automated tests

Final Product

Requirements for the Java Virtualization Software

- **Purpose of the Java Virtualization software (== Java processor)**
  - To run the Java binary instructions of Java application on the MCU
  - To give access to hardware capabilities through libraries (UART, SPI, ...)

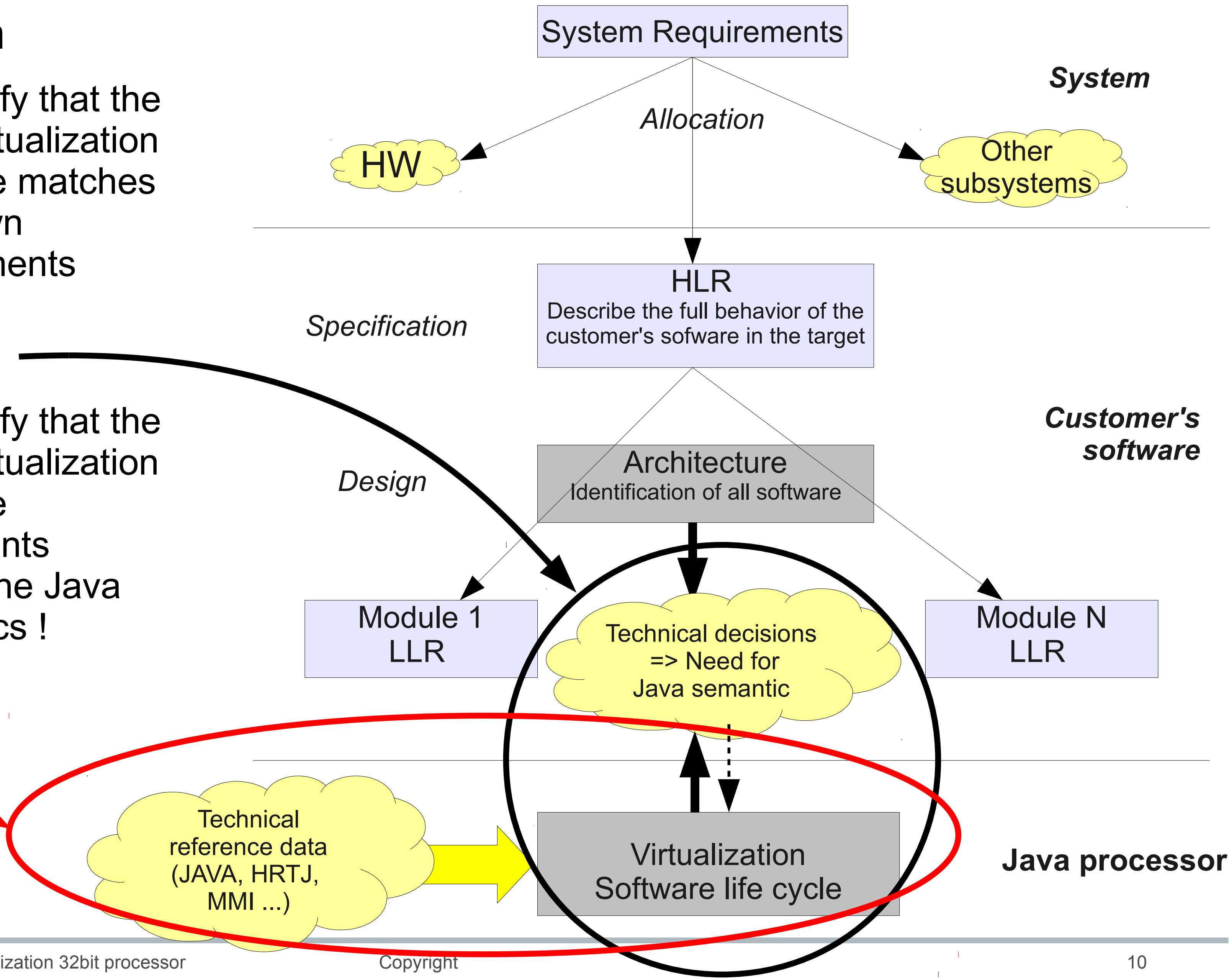
- **No system requirements**

- **No application requirements**

- **Semantic requirements from the "J2VM blue book"**
  - The "abstract" Java 32bit processor
  - Real implementation requirements from available technical literature for Java virtualization software components & from IS2T know-how
  - Low complexity of such requirements
    - Only one level of requirements
    - No derived requirements

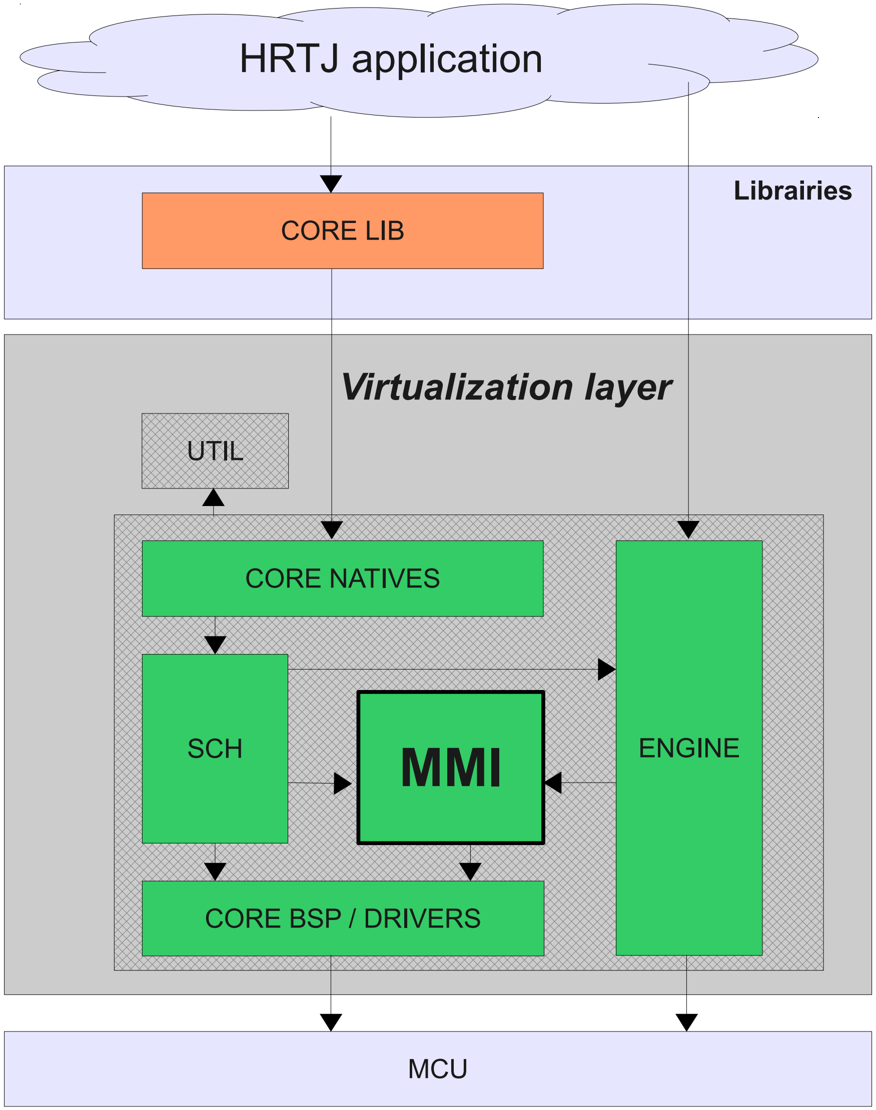
Verification & Validation

- **Verification**
  - To verify that the Java Virtualization Software matches all its own requirements
- **Validation**
  - To verify that the Java Virtualization Software implements indeed the Java semantics !

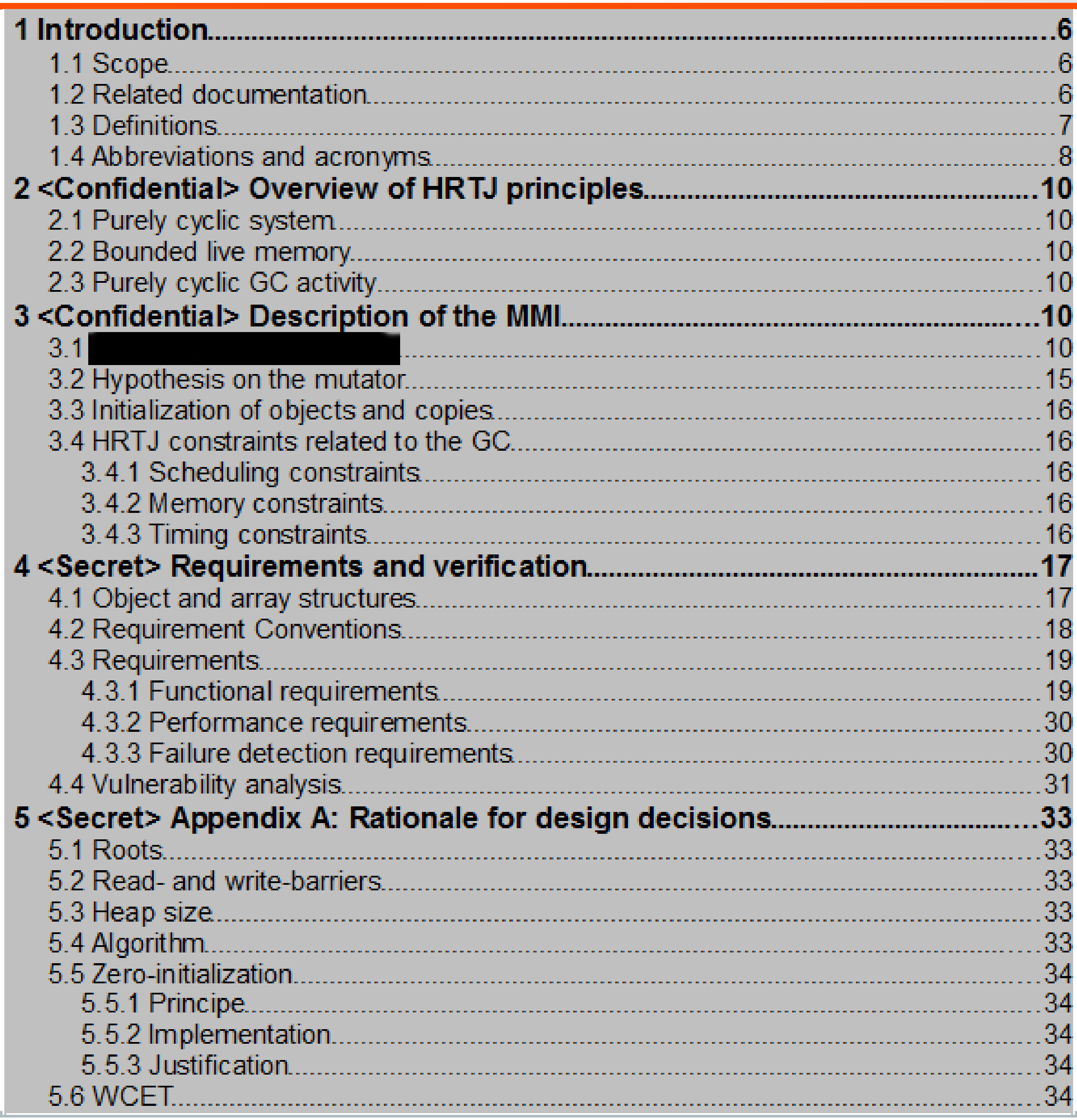

System Requirements
System
Allocation
HW
Other subsystems
HLR
Describe the full behavior of the customer's sofware in the target
Specification
Customer's software
Architecture
Identification of all software
Design
Module 1 LLR
Technical decisions => Need for Java semantic
Module N LLR
Technical reference data (JAVA, HRTJ, MMI ...)
Virtualization Software life cycle
Java processor

Architecture Overview

- **Libraries (Java classes):**
  - Core: methods to interact with the Virtualization Software
  - Platform: access to HW functionalities (e.g. I/O library)
- **Virtualization Software**
  - Utilities: Asm/C classes grouping basic functionalities
  - Natives: wrapper libraries (translation of Java interfaces into C interfaces)
  - Engine: runtime system for execution of the Java binary code
  - MMI (Memory Management Infrastructure): allocation and release of objects, defragmentation
  - Scheduler: synchronization of tasks
- **Board Support Package / Drivers**
  - Monitoring of input/output peripherals of the MCU



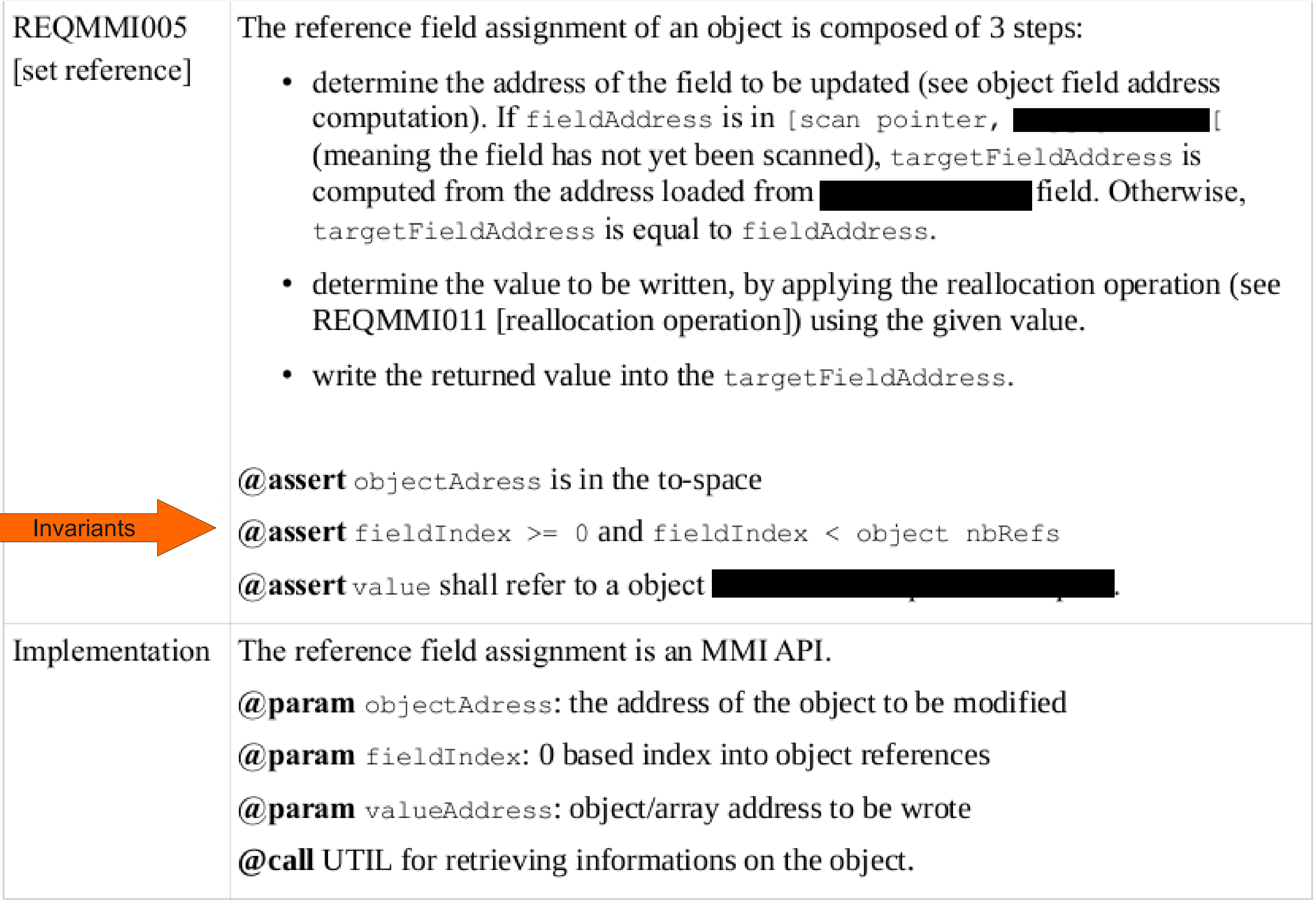
MMI Specification: CER-0074



A MMI Requirement sample

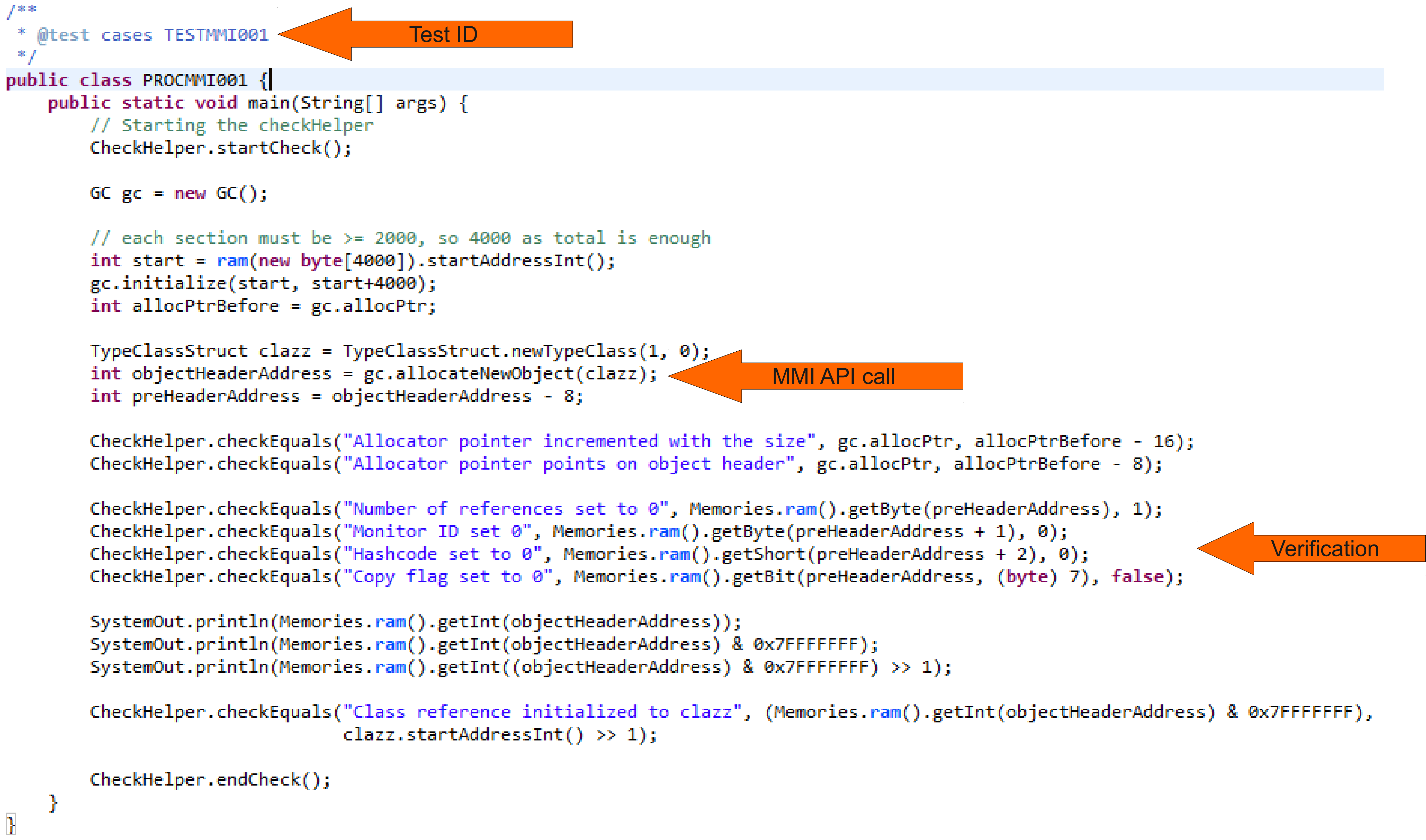
ID

Invariants



REQMMI005 [set reference]	The reference field assignment of an object is composed of 3 steps: • determine the address of the field to be updated (see object field address computation). If `fieldAddress` is in `[scan pointer,` [redacted]`[` (meaning the field has not yet been scanned), `targetFieldAddress` is computed from the address loaded from [redacted] field. Otherwise, `targetFieldAddress` is equal to `fieldAddress`. • determine the value to be written, by applying the reallocation operation (see REQMMI011 [reallocation operation]) using the given value. • write the returned value into the `targetFieldAddress`. **@assert** `objectAdress` is in the to-space **@assert** `fieldIndex >= 0` and `fieldIndex < object nbRefs` **@assert** `value` shall refer to a object [redacted].
Implementation	The reference field assignment is an MMI API. **@param** `objectAdress`: the address of the object to be modified **@param** `fieldIndex`: 0 based index into object references **@param** `valueAddress`: object/array address to be wrote **@call** UTIL for retrieving informations on the object.

A MMI Test sample



```
/**
 * @test cases TESTMMI001
 */
public class PROCMMI001 {
    public static void main(String[] args) {
        // Starting the checkHelper
        CheckHelper.startCheck();

        GC gc = new GC();

        // each section must be >= 2000, so 4000 as total is enough
        int start = ram(new byte[4000]).startAddressInt();
        gc.initialize(start, start+4000);
        int allocPtrBefore = gc.allocPtr;

        TypeClassStruct clazz = TypeClassStruct.newTypeClass(1, 0);
        int objectHeaderAddress = gc.allocateNewObject(clazz);
        int preHeaderAddress = objectHeaderAddress - 8;

        CheckHelper.checkEquals("Allocator pointer incremented with the size", gc.allocPtr, allocPtrBefore - 16);
        CheckHelper.checkEquals("Allocator pointer points on object header", gc.allocPtr, allocPtrBefore - 8);

        CheckHelper.checkEquals("Number of references set to 0", Memories.ram().getByte(preHeaderAddress), 1);
        CheckHelper.checkEquals("Monitor ID set 0", Memories.ram().getByte(preHeaderAddress + 1), 0);
        CheckHelper.checkEquals("Hashcode set to 0", Memories.ram().getShort(preHeaderAddress + 2), 0);
        CheckHelper.checkEquals("Copy flag set to 0", Memories.ram().getBit(preHeaderAddress, (byte) 7), false);

        SystemOut.println(Memories.ram().getInt(objectHeaderAddress));
        SystemOut.println(Memories.ram().getInt(objectHeaderAddress) & 0x7FFFFFFF);
        SystemOut.println(Memories.ram().getInt((objectHeaderAddress) & 0x7FFFFFFF) >> 1);

        CheckHelper.checkEquals("Class reference initialized to clazz", (Memories.ram().getInt(objectHeaderAddress) & 0x7FFFFFFF),
                                clazz.startAddressInt() >> 1);

        CheckHelper.endCheck();
    }
}
```

Application: Quadrotor & Video

Software characteristics

- Full HRTJ application
  - Initialization phase, then mission phase
- 5 tasks (all tasks allocate a few objects)
  - (1)Estimation : roll + pitch+ yaw + altitude
  - (2)Regulation : motors regulation (high-level)
  - (3)Motor : motors regulation (low-level)
  - Communication : (4) MAVLink com. : 50 Hz and (5) Earth log : 10 Hz

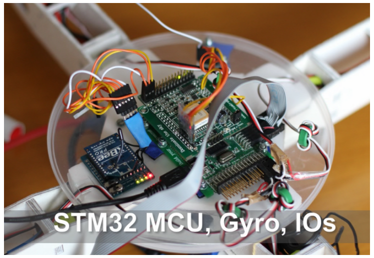
Cyclic HRTJ task		T(ms)	F(Hz)	WCET(ms)
(1)	Estimation	3,00	333	1,15
(2)	Regulation	5,00	200	0,71
(3)	Motor	5,00	200	0,24
	MMI	100,00	10	1,67
Overal CPU load				58,97%

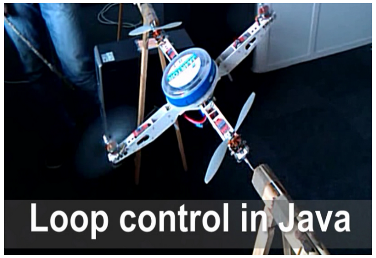
Software footprint

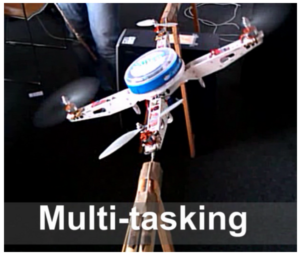
	KB
Java Application	73
Java Strings	23
Java Libraries (Java+natives)	34
HRTJ VM baremetal	**42**
BSP (drivers+libFloat)	48
Total	220

Hardware characteristics

- STM32F407, 168 Mhz
- Cortex-M4 (32bit FPU)
- FLASH=1024K / RAM=196 K
- SPI, UART, PWM, Tri-Axis gyro, Tri-Axis accelero, LEDs, Baro, GPS, 4 motors





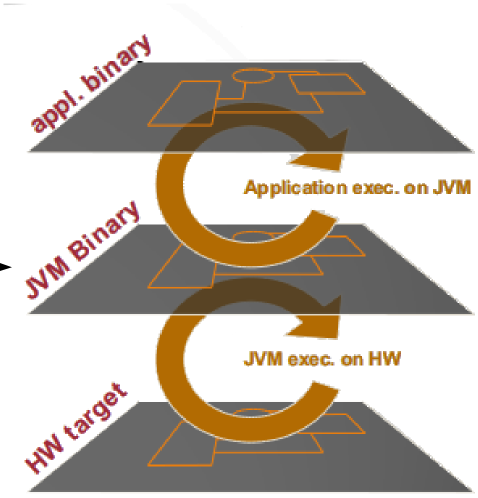


Video

Outcome & Conclusion

- **Existing process for Java Virtualization Software fit well the DO178 Core**
  - Requirements = API for library specification definition
  - Increase the traceability between tests and requirements

- **Clear classification between semantic layers**
  - Two different "qualification kits"
  - DO178C & DO332 : for the Java libraries
  - DO178C Core : for the Java Virtualization Software



- **DO178C & DO332 ready for industrialization**
  - No blocking points for Virtualization Software usage
  - Reduce complexity of a software system
  - Less costly safety-related activities

- **Applicable from "small" to "large" systems**

Thank You

Q & A ?

STM32 MCU, Gyro, IOs

Fred RIVARD (fred.rivard@is2t.com) / Frédéric RIVIERE (friviere@is2t.com)