



Real-Time Data Access

Wolfgang Hartmann, Siemens A&D ATS1

Copyright (c) 2001 J Consortium[®], Inc. All Rights Reserved

J Consortium and the J Consortium logo are trademarks or service marks, or registered trademarks and service marks, of J Consortium, Inc. in the U.S. and a trademark in other countries.

Java[™] is a trademark of Sun Microsystems in the United States and other countries.



Overview

- **J Consortium**
- **Motivations for a uniform data access**
- **RT Extensions**
- **I/O Data Access**
- **Configuration**
- **Interrupt Handling**
- **Interrupt Response Time Evaluation
(prototypical implementation)**



J Consortium: The History

- **Jun 1998 – Jan 1999 NIST-Workshops (National Institute for Standards and Technology)**

37 companies (SUN, IBM, NewMonics, MITRE, HP, Siemens A&D,...) defined the “Requirements for Real-time Extensions for the Java Platform”.

- **Nov 1998 Sun establishes the JCP (Java Community Process)**

The SUN licensees joined the Real-Time for Java Expert Group (JSR 00001).

- **Jan 1999 NCITS (National Committee for Information Technology and Standards)**

The non-SUN licensees joined the Real-Time Java Working Group (RTJWG) in Nov 1998. The NCITS members rejected the proposal of the RTJWG to define a standard for real time extensions to Java.

- **1999 May J Consortium foundation**



J Consortium: The Working Groups (1)

- **RTJWG: Real-Time Java Working Group**

Technical committee focusing on Real-Time extensions for the Java platform submitting the “Real-Time Core Extension” specification.

- **RTAWG: Real-Time Data Access Working Group**

Data access API to be employed by RT and nonRT applications in the area of industrial automation. This committee submits the “Real-Time Data Access” specification.

- **HIP: High Integrity Profile Task Group**

Subset of the “Real-Time Core Extension” specification that is suitable for use in high integrity and safety critical applications.



J Consortium: The Working Groups (2)

- **Automotive Task Group**

The proposed specification will describe a suitable profile for the use of Java for automotive control systems. It will be defined as an addendum to the HIP specification.

- **RTDM-WG: Real-Time and Embedded Middleware**

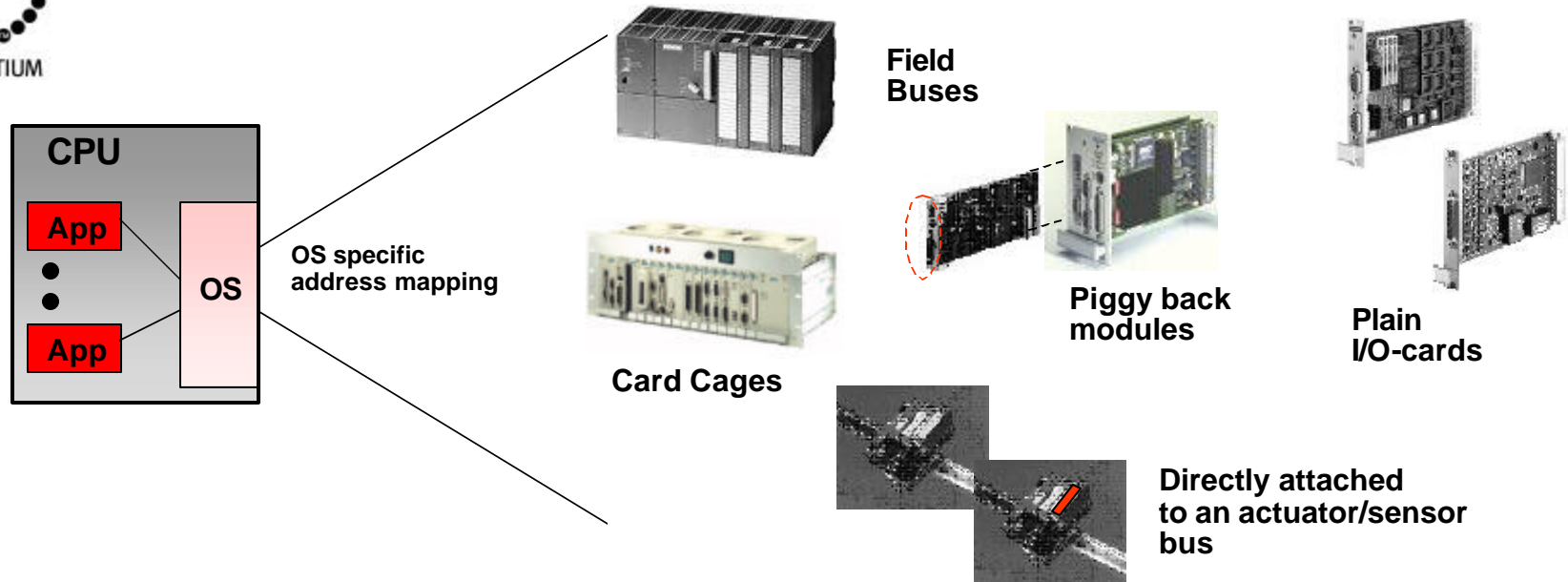
Technical committee focusing on internet appliance systems using Java.

- **JEFFWG: Java Executable File Format Working Group**

Technical committee focusing on a portable binary representation for efficient dynamic class loading. This working group has already completed Version 1.0 of the JEFF specification.



Motivations for a uniform data access: The starting point



To have an API for data access and event notification that

provides for reuse by

- an arbitrarily abstractive data access (endian approach, access variations, derived/composed data, location transparency, etc.)
- separating configuration and application

scales well in both directions

- minimal footprint for small systems
- efficiency for systems with number of IO-variables > 10000

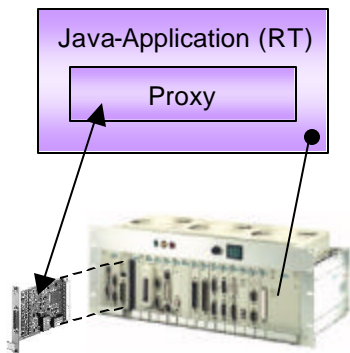
can be employed for RT and for non-RT applications as well



Motivations for a uniform data access: The overall scope

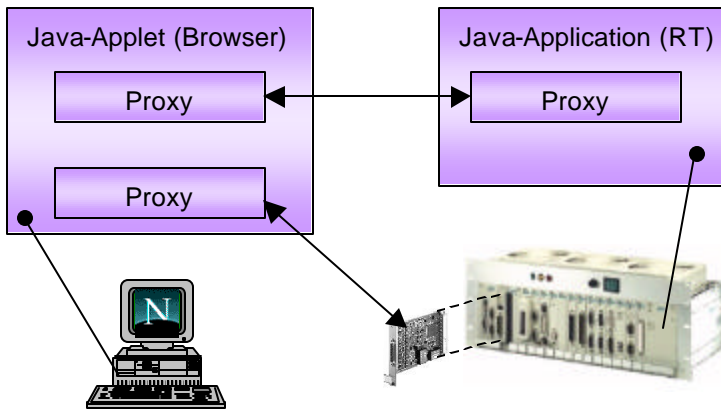
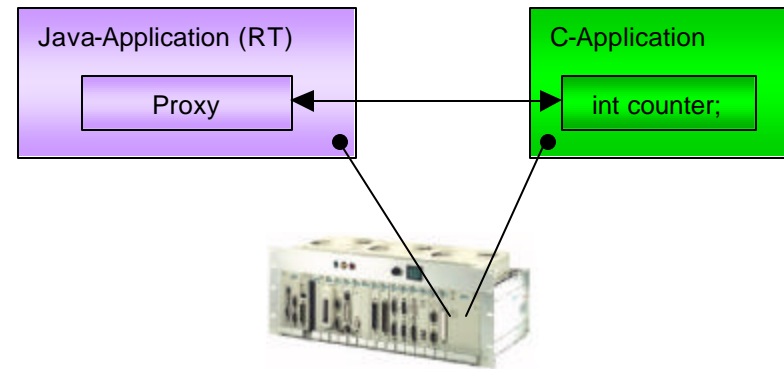
Accessing I/O-Boards

①



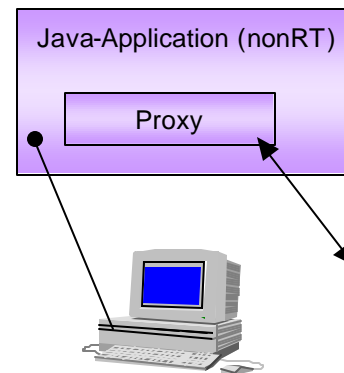
Accessing legacy systems

②



Monitoring and data display

③

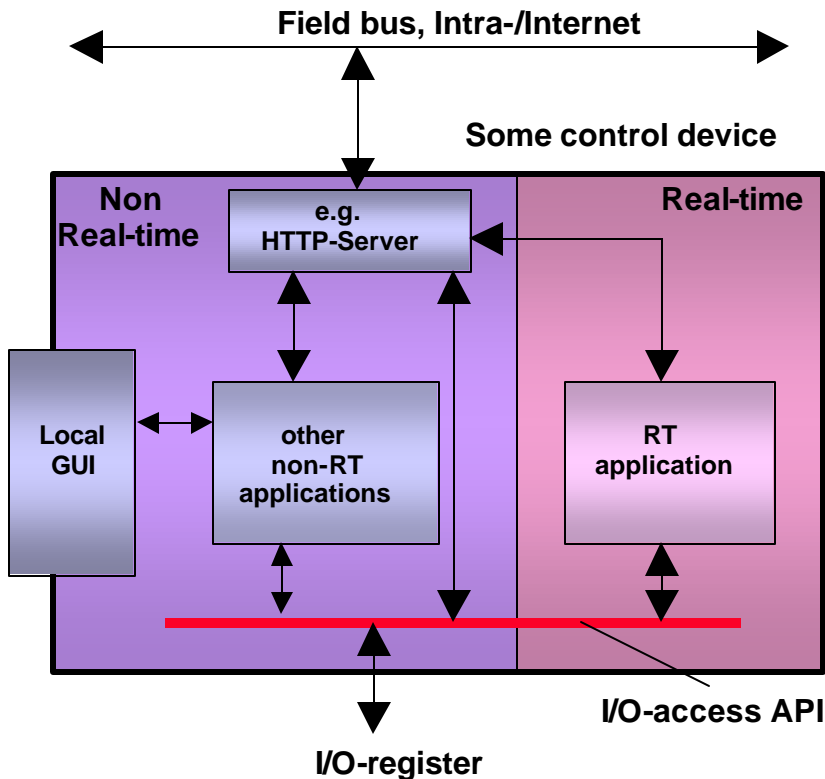


Data simulation

④

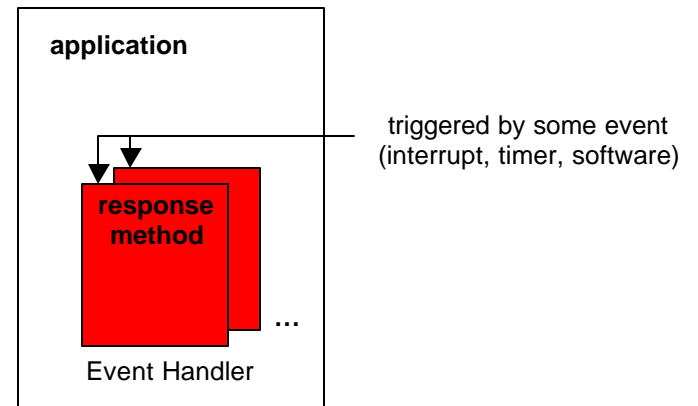


Motivations for a uniform data access : Typical RT/nonRT scenario



Industrial Automation:

- All applications need IO-access,
- Not all applications have hard RT-requirements
- IO-access is always part of the response loop (read data, process data, write data)





Motivations for a uniform data access : RT-Objectives

Mixed Environment

- RT and non-RT applications running on an RT capable VM.
- The same application and the same API implementation either running on a RT capable VM as well as on a standard VM.
- RT capable VM implementations that don't have extensions visible/relevant to the application.

Hard real-time is a niche which doesn't support significant implementation and maintenance efforts. As a consequence only minimal RT-Extensions should be necessary.



RT Extensions: Permanent Objects (1)

- **Permanent objects are a must**
 - **Performance:** Avoid the penalty of repeated object creation/destruction.
 - **Deterministic execution time:** Avoid the memory availability problem.
- **Deterministic execution time also requires a preemptible GC.**
 - **If the VM provides for a preemptible GC, then permanent objects may be just a feature of the application architecture,**
 - Requires a special implementation of the VM.
 - **If the VM knows about permanent objects, then preemptible GC is cheaper to implement**

Two approaches:

 - Type based separation: J Consortium, *Real-Time Core Extensions* (MLMT).
 - Dynamically enforced separation: SUN, *The Real-Time Specification for Java* (JSR-000001).

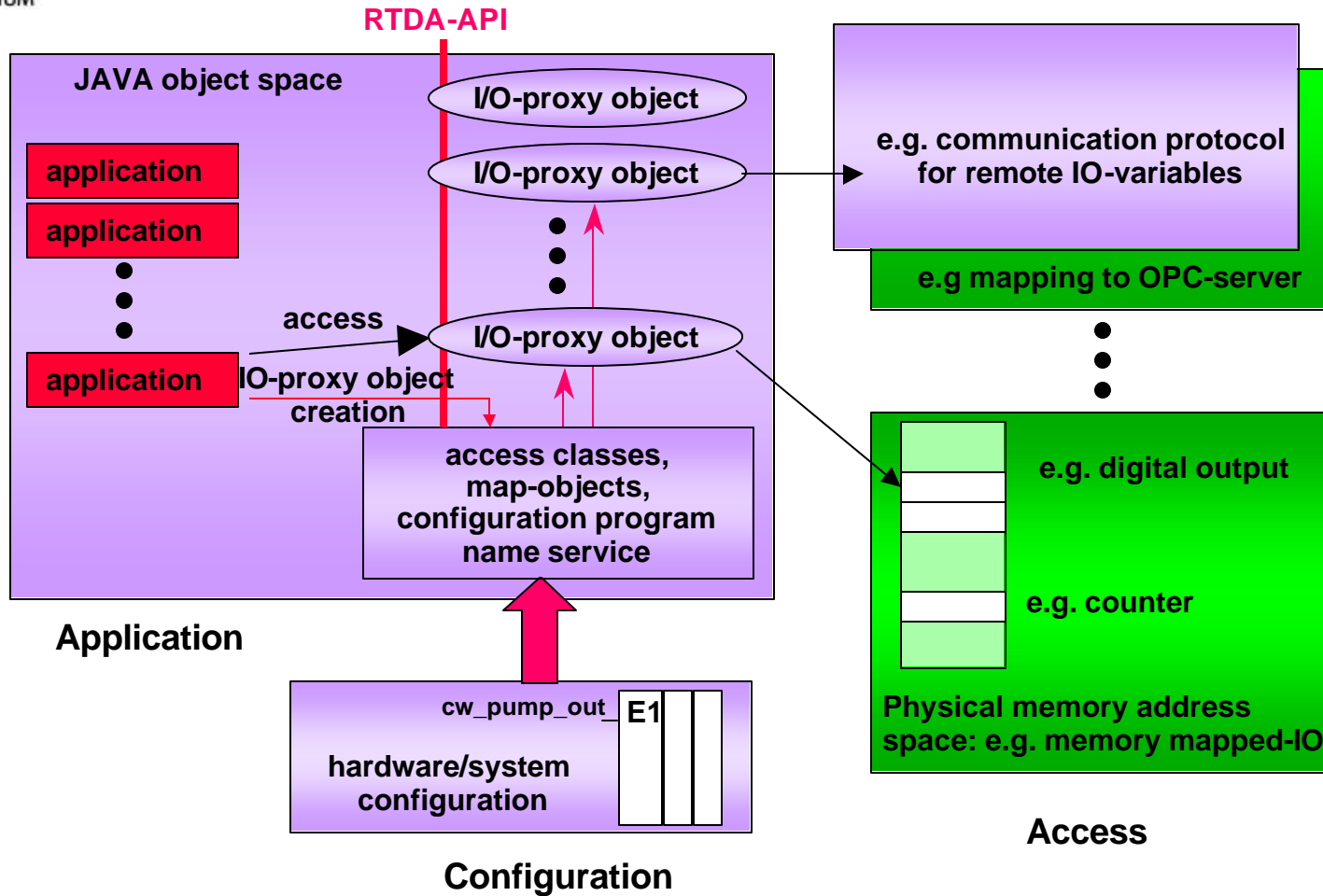


RT Extensions: Permanent Objects (2)

- **RTDA can be implemented on top of any VM** (mixed environment)
 - “NOOP”-implementation of the RT-provisions
- **RTDA can exploit the RT-capabilities of the underlying VM transparently.**
- **RTDA allows for cheap and efficient implementations of RT-capable VMs by allowing for**
 - optimized code
 - space control
 - efficient dynamically enforced separation

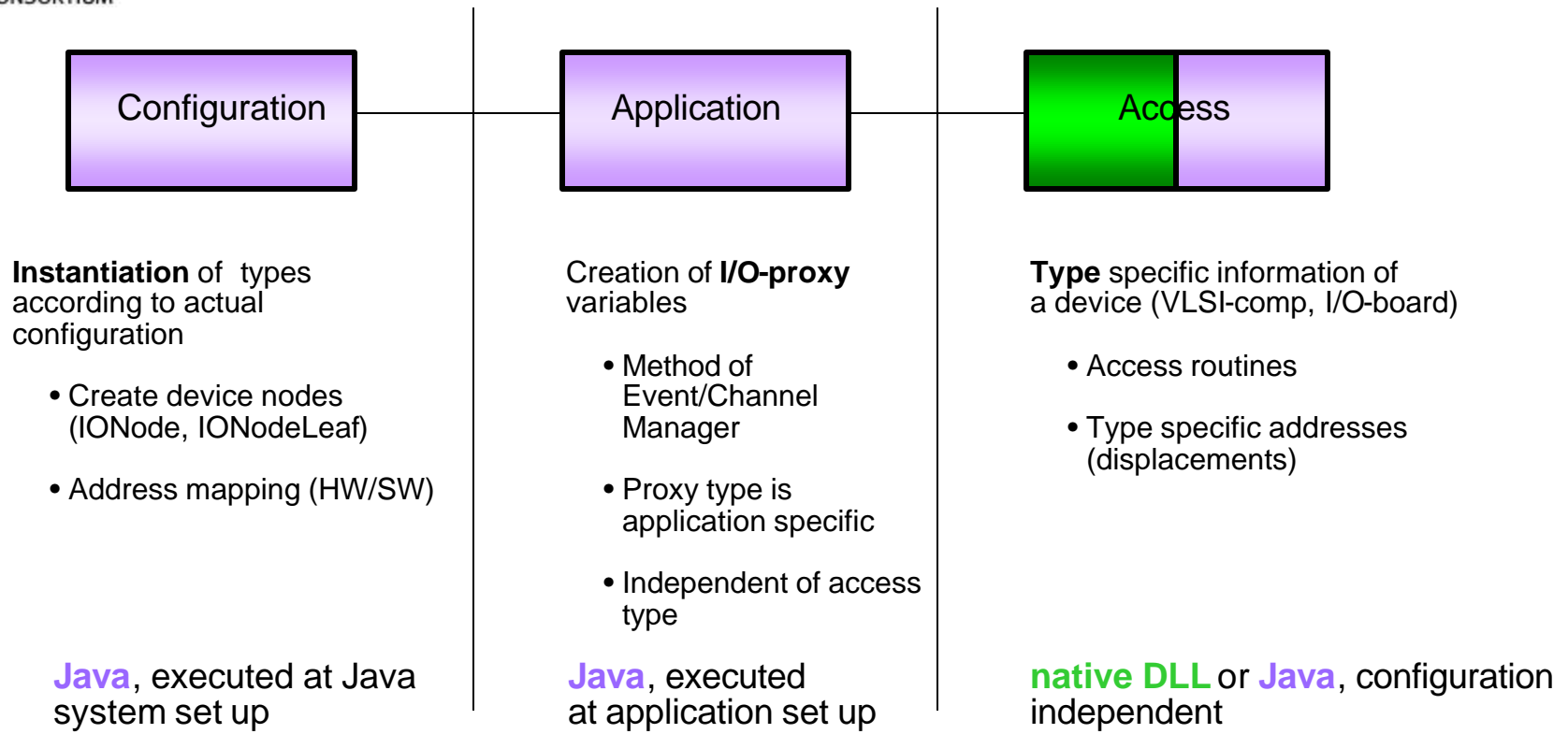


I/O Data Access API: The overall architecture





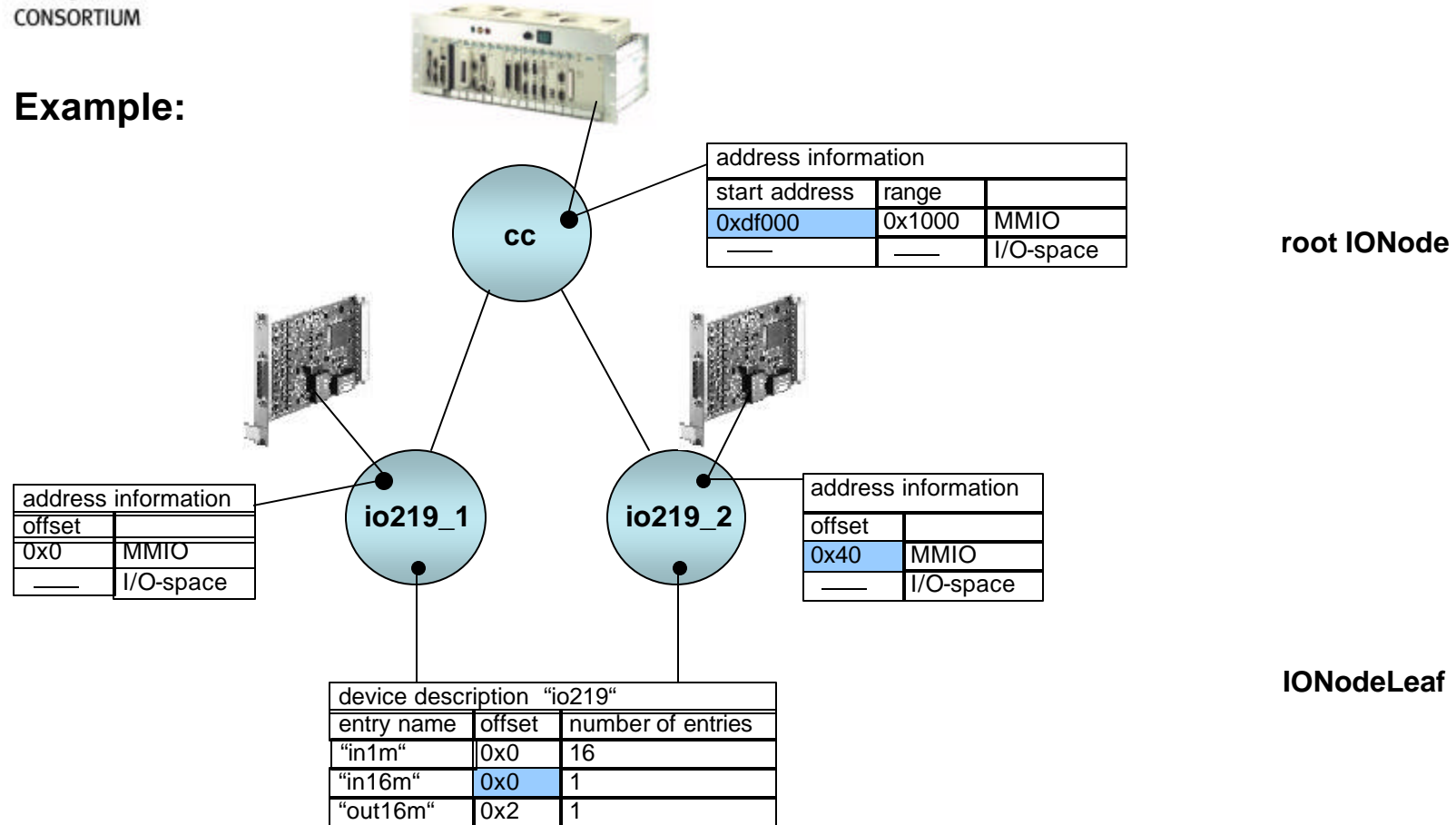
I/O Data Access: Overall Partitioning





Configuration Part: IONodes

Example:



e.g. physical name `"/cc/io219_2/in16m"` \Rightarrow address: `0xdf040`



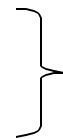
Configuration Part: Device Description

IIODeviceDescription: Type like information which describes all access specifics of a component (VLSI component, I/O-board, ...)

Example:

device description name: "IO219"

"in1m", 0x0, 16;
"in16m", 0x0;
"out16m", 0x2;



access entries (I/O-proxies and events):

entry name (may be chosen arbitrarily),
attributes (offset, number of entries, I/O-space,
access method index)

individual entry
offset

range entry
offset, number of entries

All device descriptions of a given I/O-system are typically described in a native DLL.



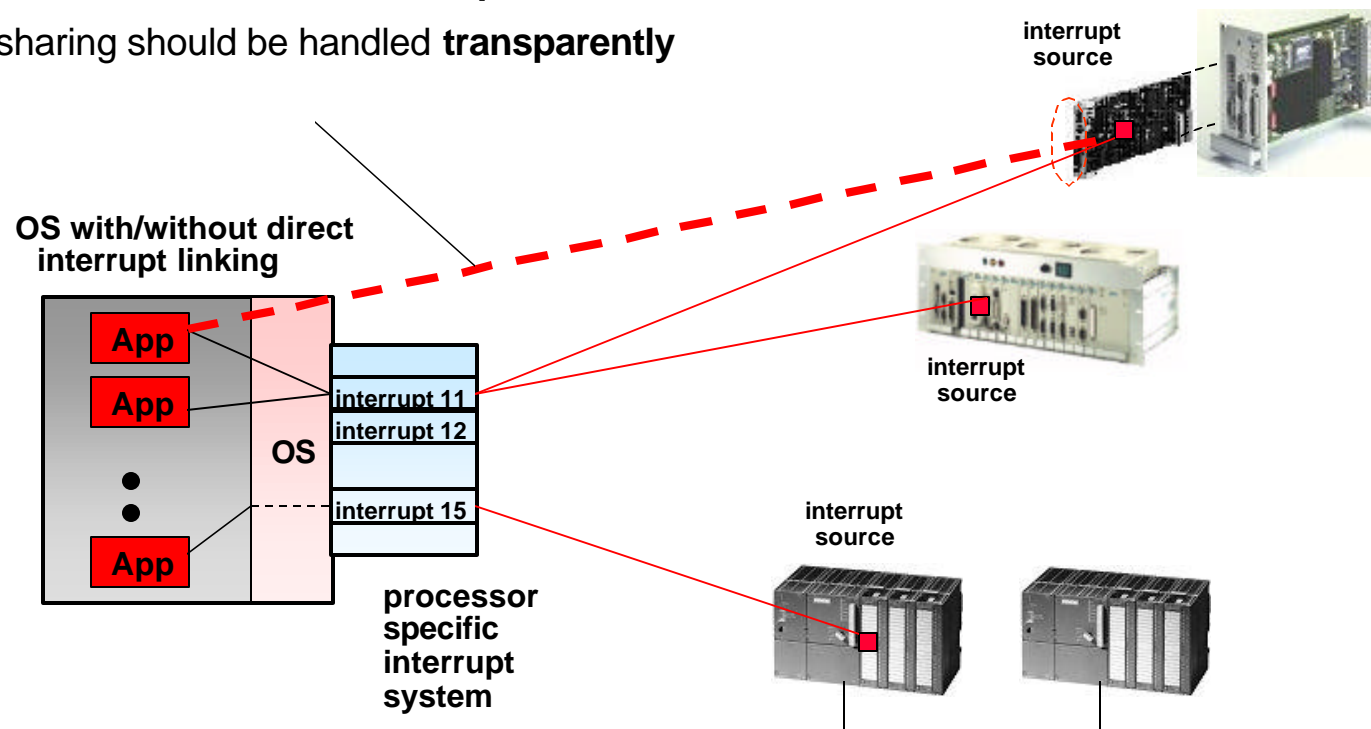
Configuration Part: Name Service

- Mapping of symbolic names to physical names
 - Configuration of name map tables at system start-up (optional)
 - Precondition for device independent applications
 - Entries are defined through String-pairs (symbolic/physical name)
- Registration of dynamically created resources
 - Precondition for parsing the pathnames (physical names)
 - Lookup service for dynamically created resources



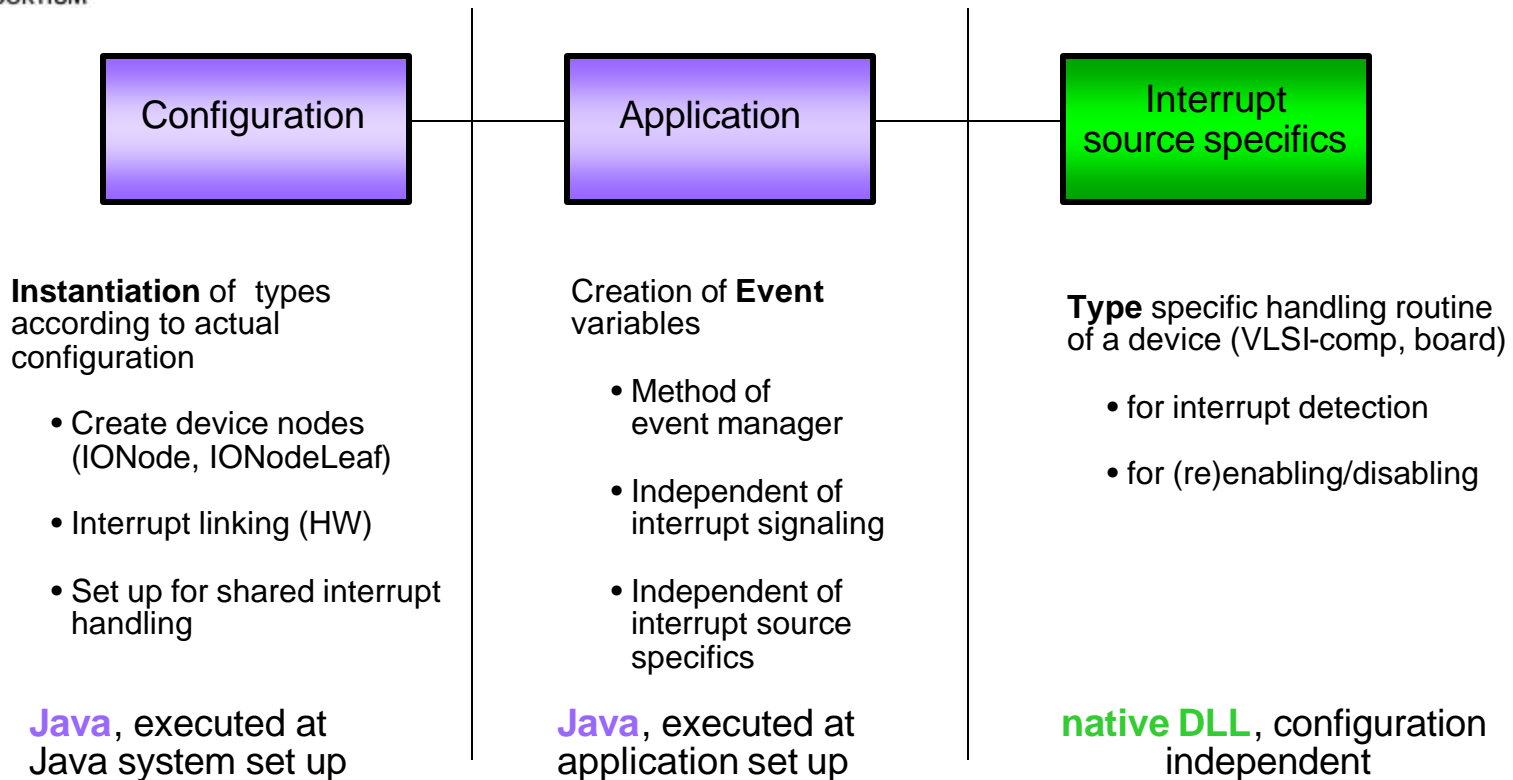
Interrupt Handling: Scenario

- An application cares about the **interrupt source**
- Interrupt sharing should be handled **transparently**



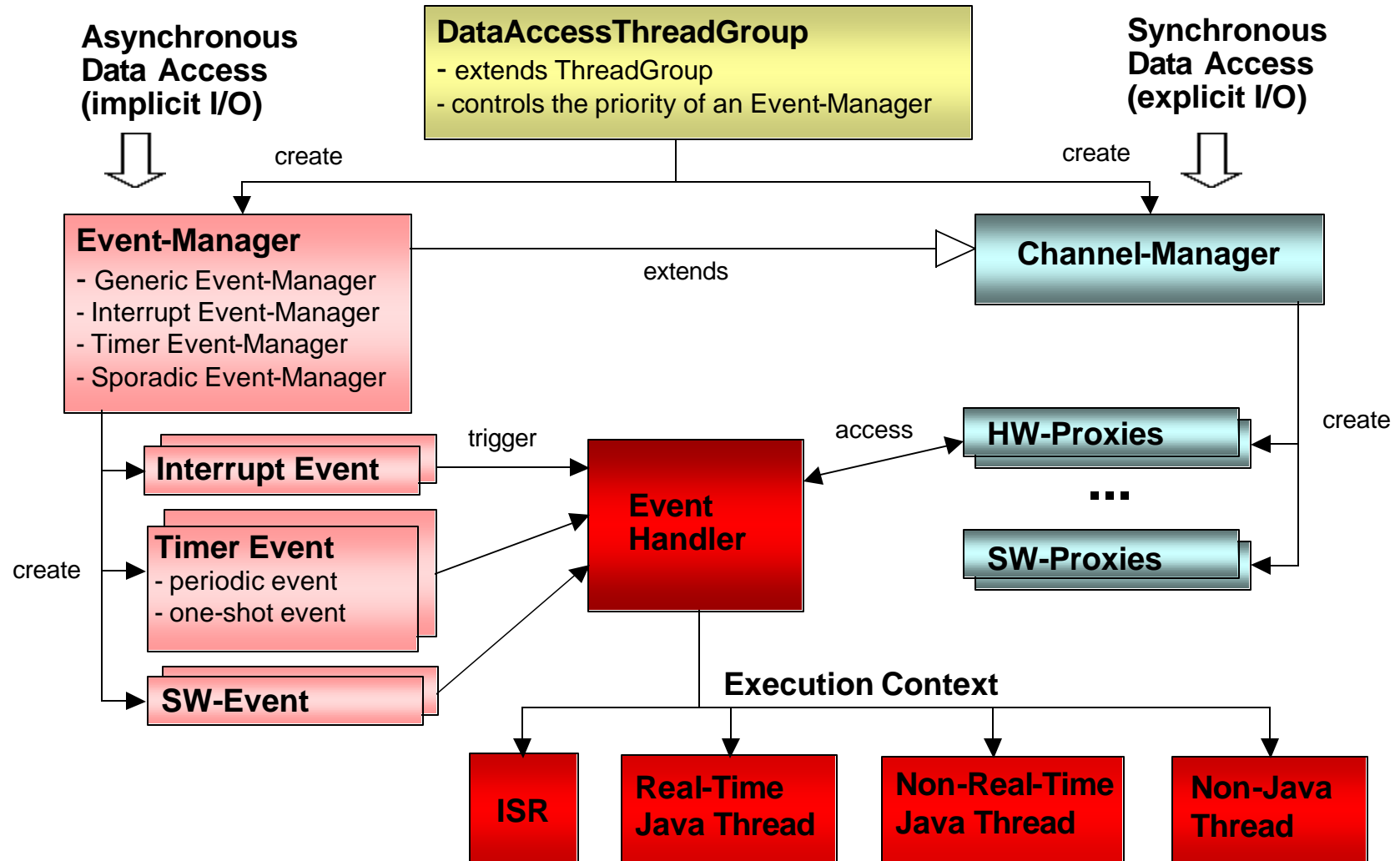


Interrupt Handling: Overall Partitioning





I/O Data Access: Dynamic Execution Model

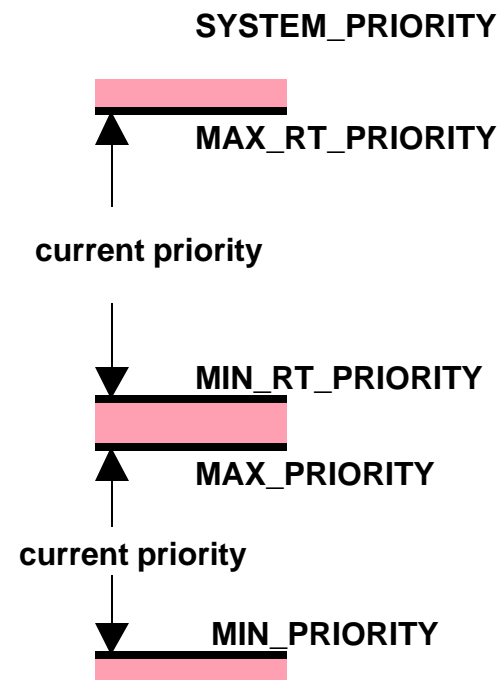




RT Extensions: Priorities

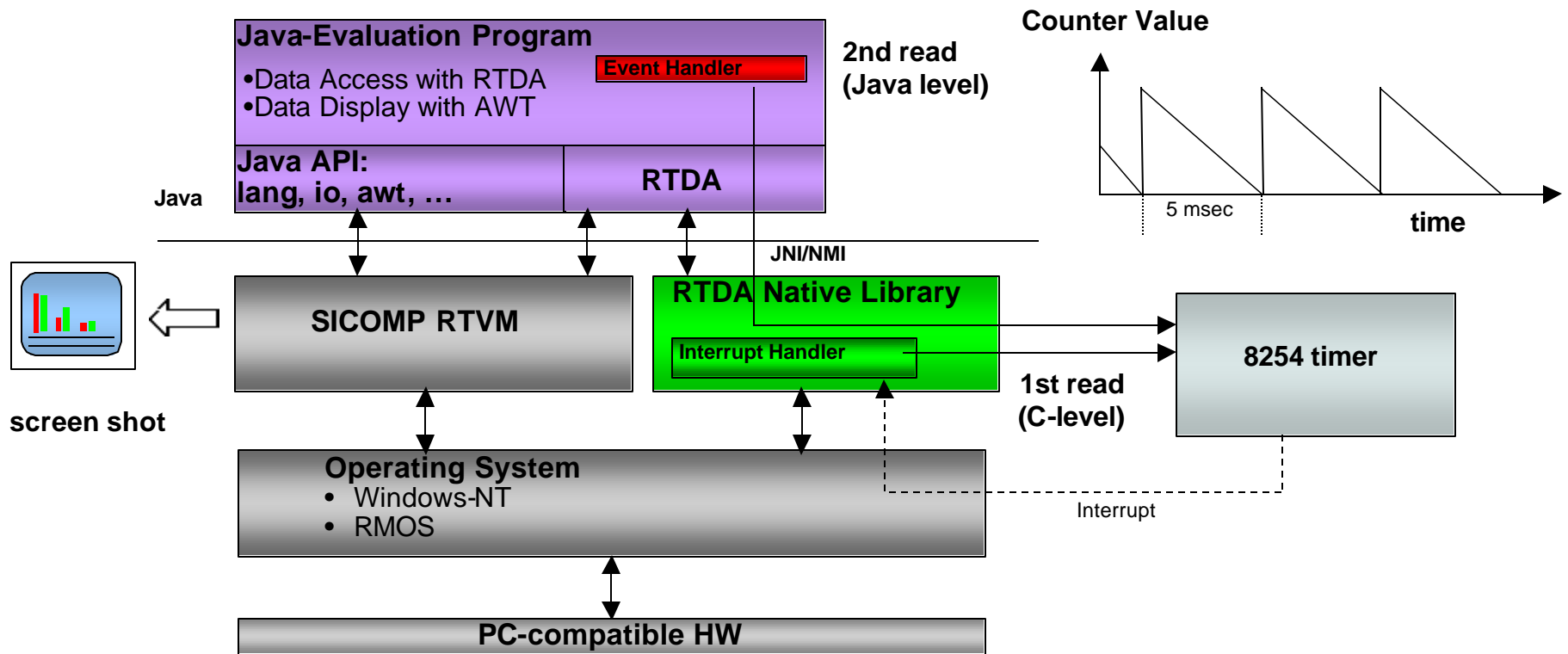
RT-Thread/Event Manager creation controlled by a ThreadGroup (DataAccessThreadGroup):

- Root instance (least frequent)
(only one place where RT gets visible)
- Regular thread class API can be used without violating compatibility
- Leaves room for thread groups with different priority range or for future additional types
e.g. time sliced
- Automatic mapping to highest thread priority if ISR based execution is not possible





Interrupt Response Time Evaluation: Configuration

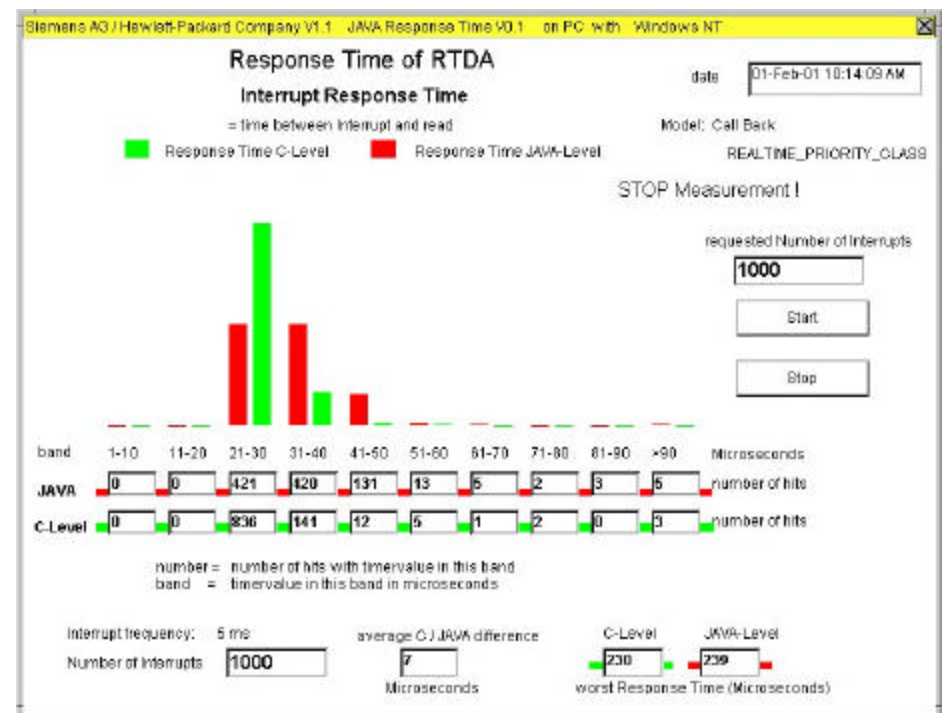
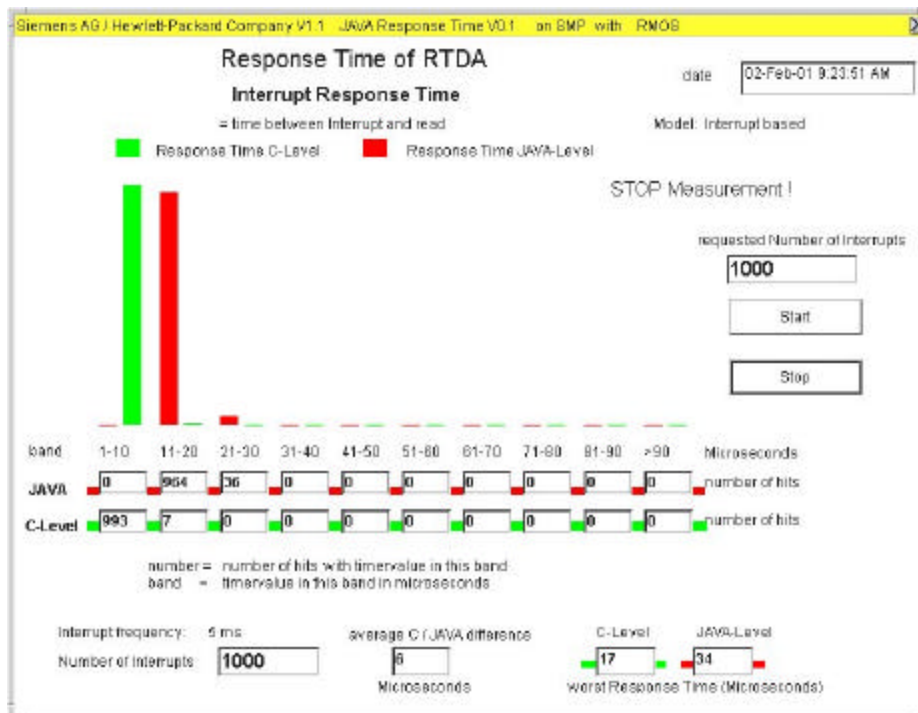




Interrupt Response Time Evaluation: SYSTEM_PRIORITY

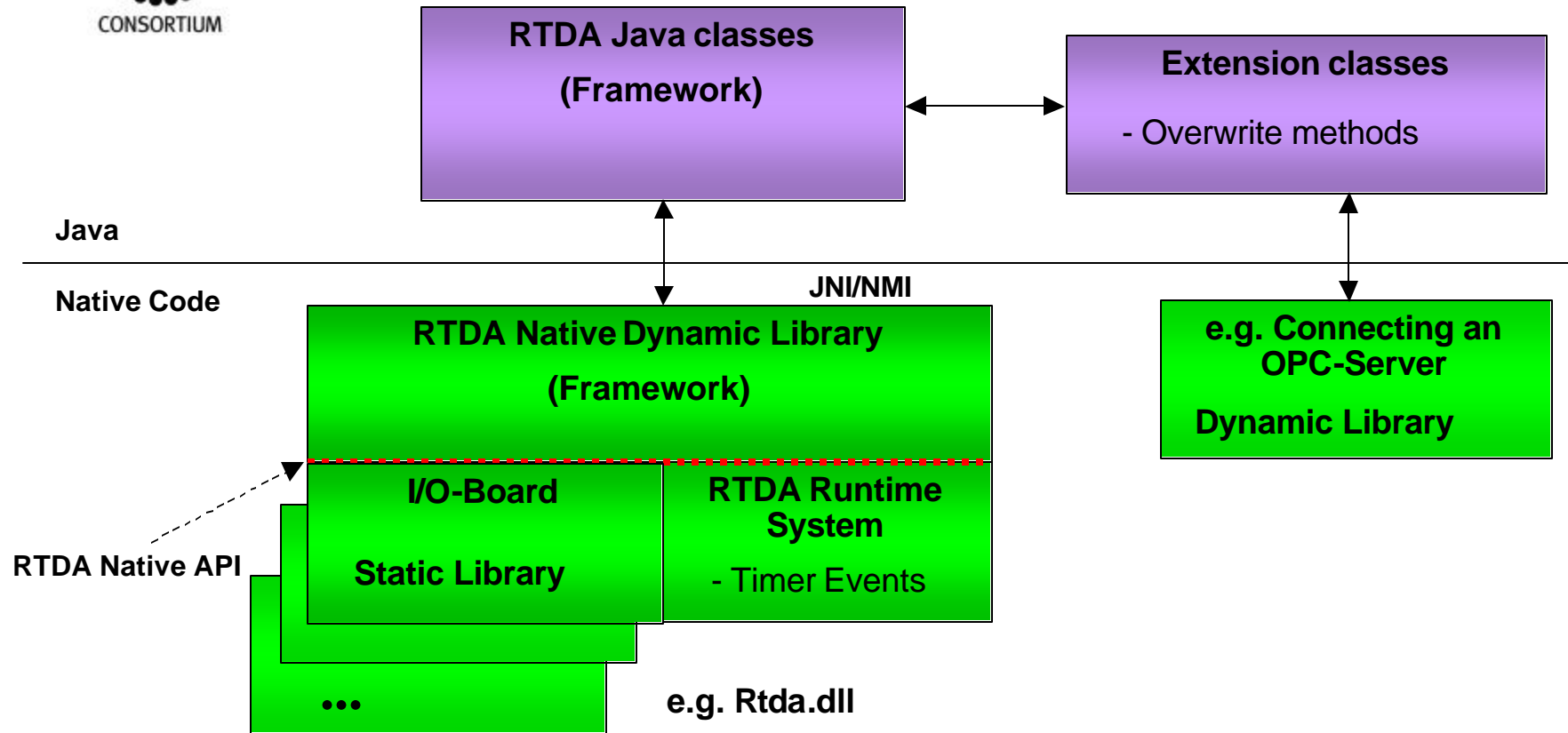
RMOS (300 Mhz)

Windows NT (400 MHz)

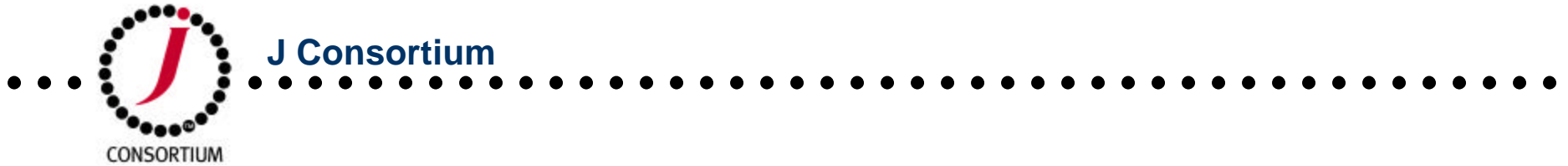




Extensibility: The RTDA Architecture (Prototype)



For adding new I/O boards there has no Java code to be changed.



For further information see...

www.j-consortium.com