About 15 years of Real-Time Java

M. Teresa Higuera-Toledano
Universidad Complutense de Madrid Ciudad Universitaria,
Madrid 28040, Spain

This research was supported by Consejería de Educación de la Comunidad de Madrid, Fondo Europeo de Desarrollo Regional (FEDER) and Fondo Social Europeo (FSE), through Research Program S2009/TIC-1468, and by Ministerio de Educación y Ciencia, through the research grant TIN2009-.07146.

Outline

- Introduction
- Real-Time Java Solutions
- The Real-Time Java Specification
- RTSJ-Based Solutions
- Java Components-based Solutions
- Distributed Real-Time Java Issues
- High-Integrity Systems
- Conclusions

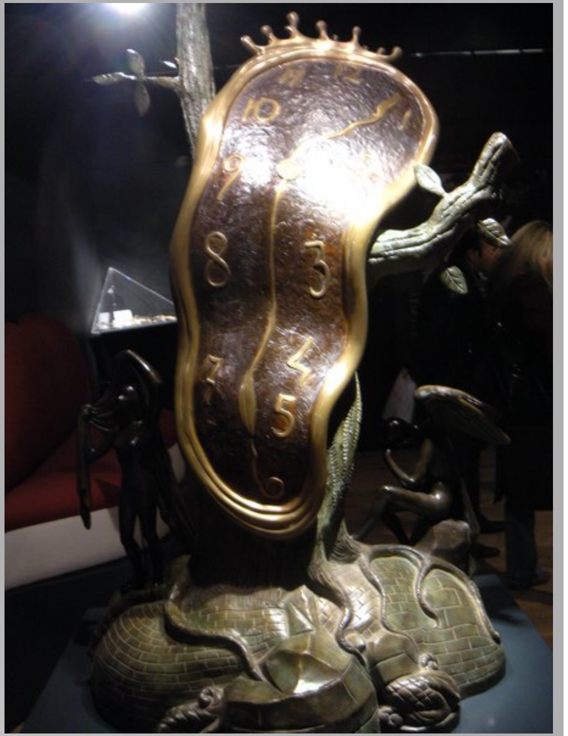
History

- 1995: Java was introduced for Sun Microsystems
- 1997: several research works focus on the limits of Java to execute real-time applications
- 1998: PERC
- 1999: the NIST document
- 2000: the first JSR (Java Specification Request) for RTSJ
- 2002: the JSR-50 for DRTSJ
- 2005: the JSR-282 to enhance RSTJ
- 2006: final release of RTSJ 1.0
- 2006: the JSR-302 for SCJ
- 2011: final draft review of SCJ
- 2011: taken again DRTS

Introduction

Solutions before the NIST document

- 1997-2000
- Real Time Java Threads (Tokuda): provides real task support and synchronization.
- PERC (Nielsen): provides an original API with atomic execution of code and resource negotiation.
- CSP and Transputers: deals with single and multi-processor environments.
- JavaOS: integrates real-time capabilities in a Java-based operating system.
- picoJava: runs the Java bytecodes as its native instruction set.



Real-Time Java Solutions

The NIST document

- Started on december 1999
- A standard Java extension for real-time applications
- API-based solution and profiles
- Two alternatives:
  - The Real-Time Core Extesion fpr the Java Platform (RT-Core)
  - The Real-Time Specification of Java (RTSJ)
    - RT-Core proposses modifications to the Java language, and it was not well accepted
- Profiles: distributed, safety critical, business critical …

Real-Time Java Solutions

Real-time Systems and Java

- The Real-time Specification of Java
- The Distributed Real-time Specification of Java
- The Certiable Safety-Critical Java

The Real-Time Specification of Java (RTSJ) JSR-1 and JSR-282

Problems

- Time values and clocks
- Accessing underlying hardware
- Scheduling
- Synchronization
- Asynchronous event handling
- Dynamic memory management
- Resource management

The Real-Time Specification of Java (RTSJ) JSR-1 and JSR-282

Problems

- Time values and clocks
- Accessing underlying hardware
- Scheduling
- Synchronization
- Asynchronous event handling
- Dynamic memory management
- Resource management

RTSJ solutions

- HighResolutionTime class
- RawMemoryAccess class
- RealtimeThread class
- Synchronized keyword
- AsyncEvent and AsyncEventHandler classes
- MemoryArea abstract class and RT-GC
- MemoryParameters and Scheduling Parameters

- Timesys RI
- OVM (Purdue)
- PERC (AONIX)
- Jamaica (AICAS)
- McKinack (SUN)
- Websphere (IBM)
- JOP
- JRate

Implementations

- To avoid the garbage collector we can only use ScopedMemory and ImmortalMemory.
  - objects within the heap or immortal cannot contain references to objects in scoped memory (RTSJ)
  - objects within a scoped region cannot contain external references to objects within a non-outer scoped memory (RTSJ)

- Programing with scoped regions is error-prone
- Still an open research issue in JSR-286

Memory Considerations

- Besides guaranteeing the functional behavior of a specific component, the composition must also guarantee that the communication, synchronization and timing properties of the components are time-analyzable.

- The development RT components which can be run on different HW platforms is complicate because different timing characteristics of different platforms.

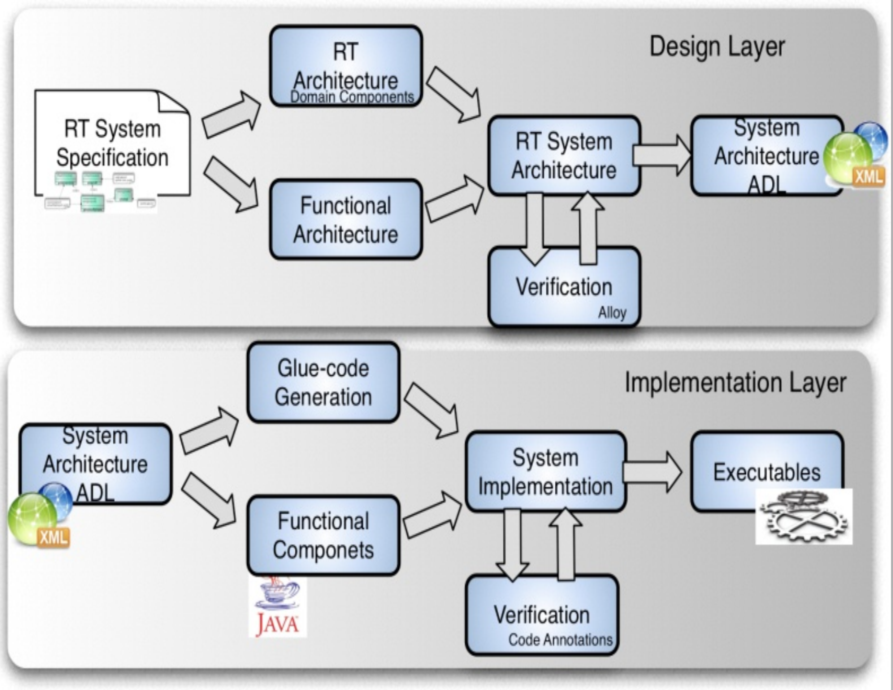
- A RT component should provide the following information:
  - Memory requirements –
  - WCET test cases - WCET for a particular processor family.
  - Dependencies – Describing dependencies on other components
  - Environment assumptions - in which the component operates, for example the processor family.

Real-time Components

RTSJ Components-based Solutions

- Soleil
  (INRIA)
- RTComposer
  (University of Pennsylvania)
- RT-OSGi
  (University of York)

Basic goal of the framework

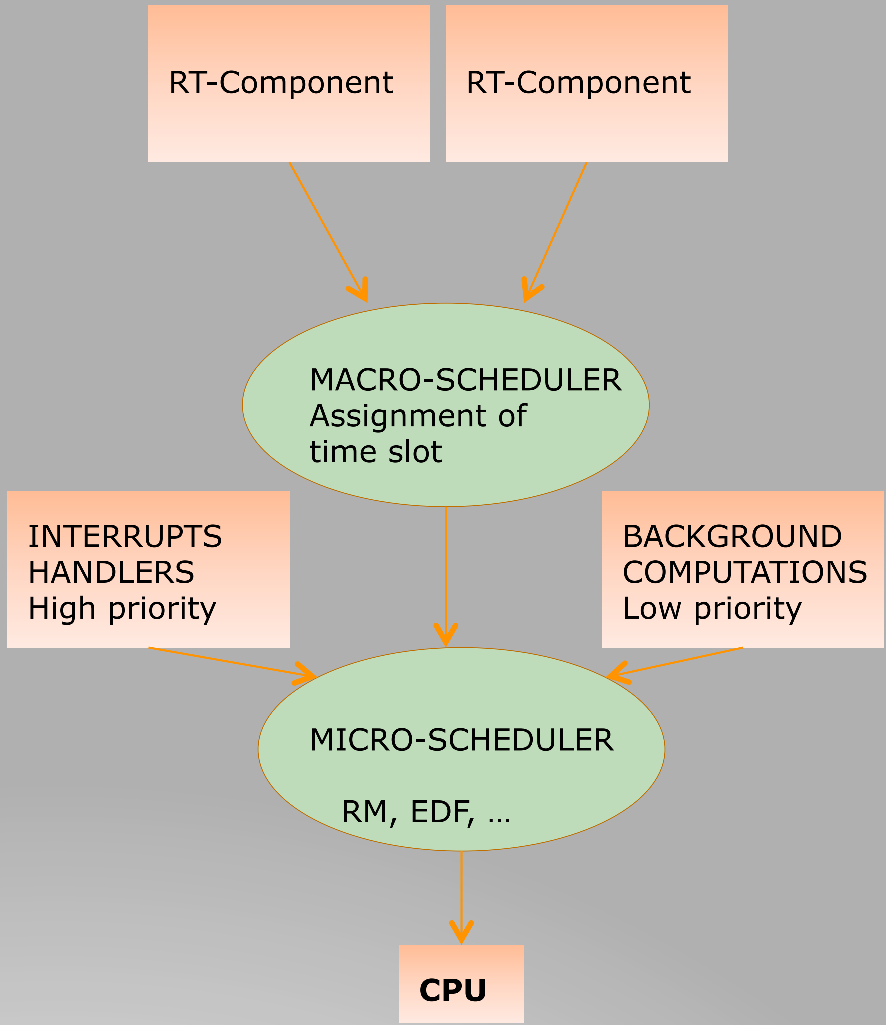


Soleil (INRIA)

- A systematic architecture design
- Automatic generation of RTSJ code
- The *ThreadDomain* component represent the RealTimeThread hierarchy (i.e., RealTimeThread and NoHeapRealTimeThread)
- The *MemoryArea* component represent the MemoryArea hierarchy (i.e., ImmortalMemoryArea, HeapMemory, and ScopedMemory)
- The functional architecture is obtained as a combination of The *ThreadDomain* and *MemoryArea* components.

Hierarchical scheduling approach

- Two scheduling levels:
  - A standard task scheduler is used for inter-slot scheduling
  - An automata-based scheduler is used for intra-slot scheduling



RTComposer (University of Pennsylvania)

Dynamic Configuration

- Combines OGSi and RTSJ real-time thread, priority-based scheduling and real-time GC
- Provides:
  - An admission control protocol.
  - A priority assignment approach supporting temporal isolation.
  - A hierarchical scheduling.
- The combination of these characteristics guarantees safety update of components
- Memory isolation has not been still addressed

RT-OSGi (University of York)

Problems

- RTSJ is focused on centralized
- Since 2000 inactive
- Programming model:
  - *networked* (asynchronous messages)
  - *control flow* (method invocation)
  - *data flow*(publish/subscribe)

The Distributed Real-Time Specification of Java (DSRTJ) JSR-50

The Distributed Real-Time Specification of Java (DSRTJ) JSR-50

Problems

- RTSJ is focused on centralized
- From 2000 inactive
- Programming model:
  - *networked* (asynchronous messages)
  - *control flow* (method invocation)
  - *data flow*(publish/subscribe)

DRTSJ extends RMI in RTSJ

- Distributed (multi-node) RTSJ
- Taken again in 2011
- To add end to end timelines:
  - focusses on control flow
  - RMI, events, thread transfer of control, and scheduling

Distributed Real-Time Java

- Profiles for RT-RMI
  (Polytechnic University of Madrid)
- Compadres
  (University of California)
- DREQUIEMI
  (Carlos III University)

Three different profiles:

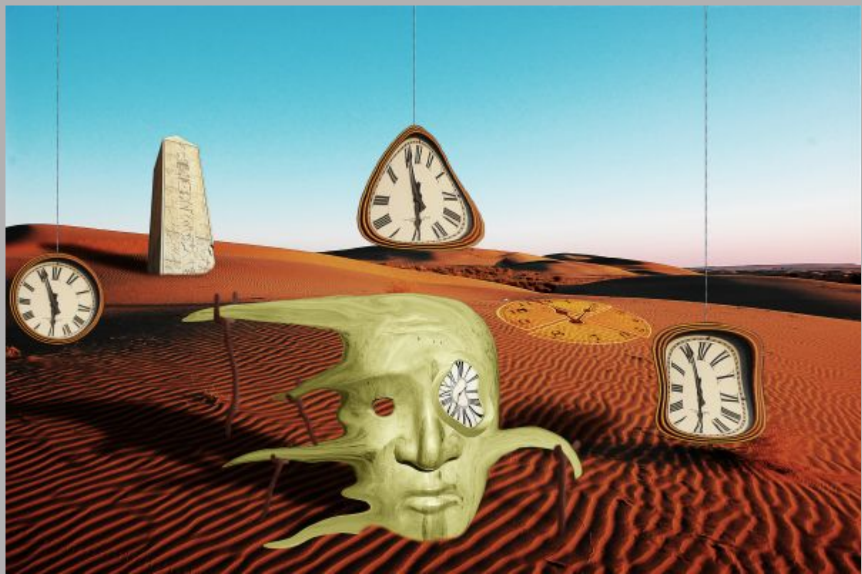
- RMI-HRT for safety critical systems, requires highly deterministic behavior. Deadlines misses can cost human lives or cause fatal errors (i.e., hard real-time systems)
- RMI-Quality of Service for efficient and robust system, which anomalous behavior can cause financial cost (i.e., soft real-time systems)
- OSGi-based solution for flexible business systems (e.g., multimedia systems, ambient intelligent). It considers RT-GC, and does not consider asynchronous interrupt exceptions, nor asynchronous event handling.

Profiles for RT-RMI (Polytechnic University of Madrid)

AComponent Middleware Framework

- Components for Distributed Real-Time Embedded RTSJ
- Components connected by ports that communicate through strongly-typed objects
- Abstracts away RTSJ memory management complexity
- Compiler that automatically generates the scoped memory architecture for components

Compadres (University of California)



A Middleware Framework

- It allows to support the level L1 od DRTSJ
- It incorporates some DRTSJ L2 elements
- It is based on RMI having influences from RT-CORBA
- It offers four services:
  - A stub/skeleton allowing remote object invocation
  - A distributed garbage collector
  - A naming service for white pages
  - A synch/event service for data-flow communication

DREQUIEMI (Carlos III University)

Problems

- Started on 2006
- Safety critical applications
- Validation:
  - standards DO-178B / ED-12B
  - formal models,schedulability analysis
- Requires transformation from bytecodes to target machine Programming model

The Safety Critical Java Specification (SCJS) JSR-302

The Safety Critical Java Specification (SCJS) JSR-302

Problems

- Started on 2006
- Safety critical applications
- Validation:
  - standards DO-178B / ED-12B
  - formal models,
  - schedulability analysis
- Requires transformation from bytecodes to target machine Programming model

A RTSJ Subset for critical system

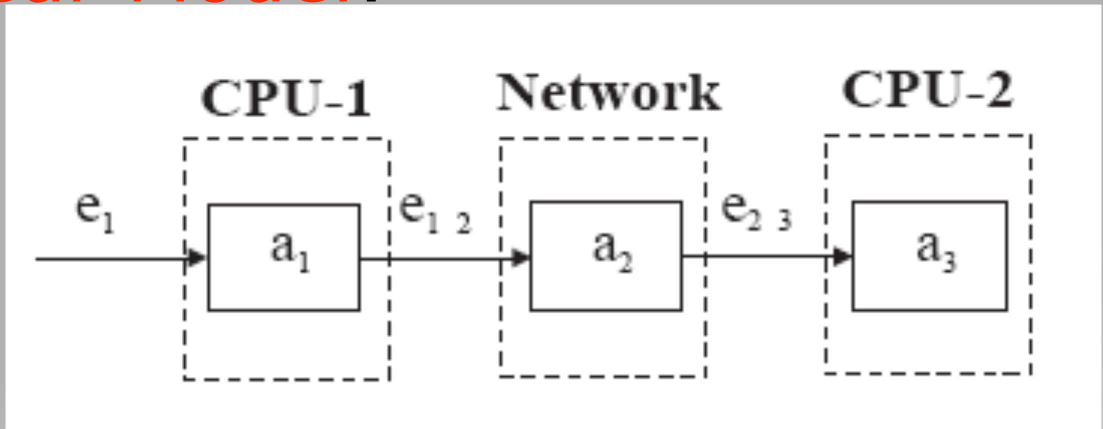
- Finished on 2011
- Minimal set of features:
  - static resource allocation and usage
  - minimal temporal conflicts
  - without dynamic loading
  - Without GC
- It is expected that this JSR will result in an ISO standard

High-Integrity Systems

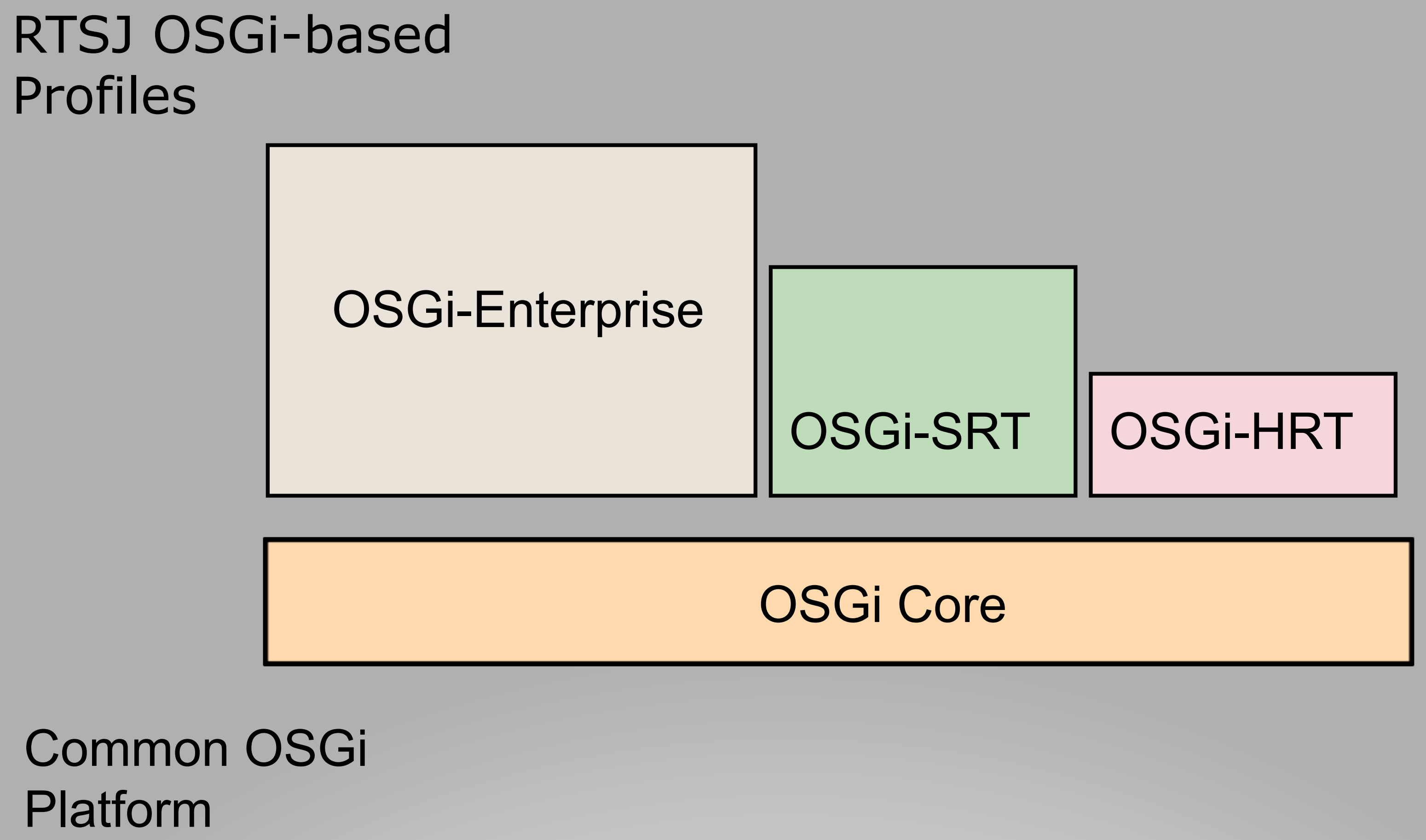
- RMI-HRT Profile (HIJA)
- PERC and OSGi

- Computational Model based on HRTJ:
  - based on pre-emptive priority-based scheduling
  - threads or event handlers (periodic or sporadic)
  - priority ceiling inheritance protocol
  - two phases: initialization and mission

- Linear Model:



RMI-HRT Profile (HIJA)



PERC and OSGi

- RTSJ is the standard Java extension adding real-time capabilities to the Java environment
- It introduces the MemoryArea class; an original mechanism that combines pre-allocates spaces with the GC.
  - Scoped memory present some difficulties regarding their use
- The DRSJ profile supports the development of distributed Java programs with real-time restrictions
- RT Embedded systems interact with the real-world
  - must be dynamically adaptive
  - must be capable of being modified and updated at run-time
- We give an overview of existing RTSJ components based solutions
- The SCJS profile supports the development of programs that must be certified. This specification includes annotations and rules to check statically the semantic program.

Summary and Conclusions