Jeopard

Java Environment for Parallel Realtime Development

Industry Use Cases

GMV

Information Society Technologies

Jeopard

- Java Environment for Parallel Real-Time Development
  - "Platform-independent software development interface for complex multi-core systems based on RTSJ and SCJ"

- FP7 ICT Project, 2008-2010
  - The Open Group, aicas, SYSGO, FZI, TUV, TUCN, DTU, UoY, Cassidian, RadioLabs, GMV

Terminology

- Partition/Partitioning
  - Time and Space container
- Application
  - Software running within a partition
- Module
  - Computer (= Processor + Memory + IO)
- IMA
  - Integrated Modular Avionics

Jeopard Architecture

- OR
  - Partitioning RTOS + RT-JVM executed within partitions
    - Access from Java to FPGA via "HW-Methods"
  - JOP
- Tools
  - Thread Monitor
  - Code Analyser
  - Concurrent Unit Testing
  - Schedulability Analysis

Use Case Objectives

- Tool Evaluation
- Validation with real-world applications
- Demonstration of feasibility

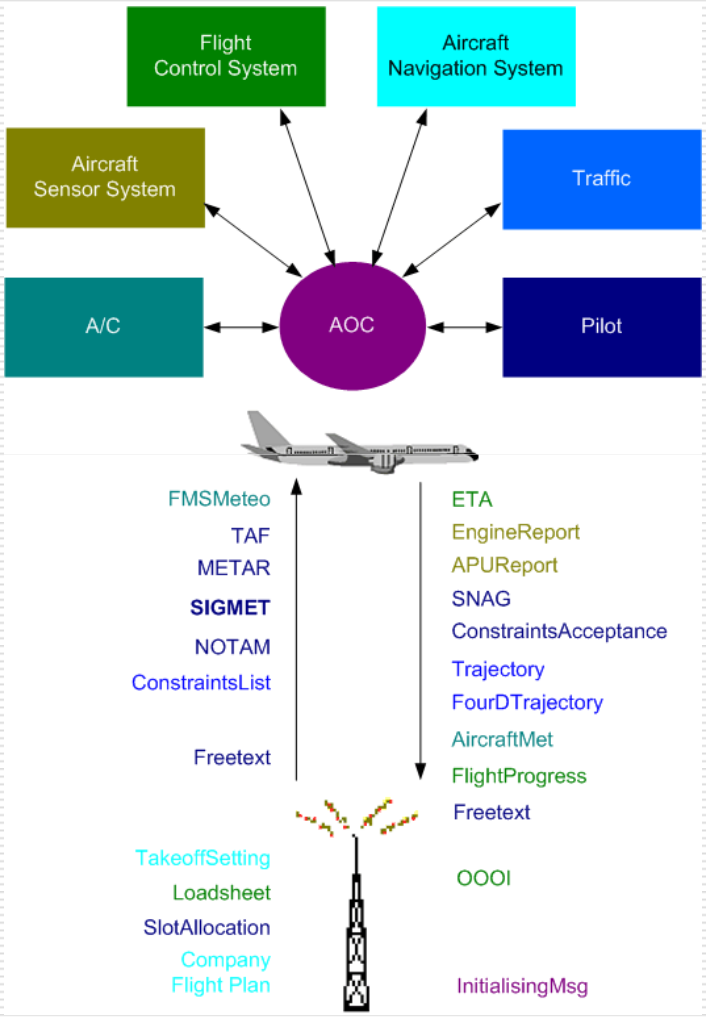
- Validation Approach
  - Verify applications on Jeopard agains real application requirements

Use Cases

- Three Use Cases
  - **Radar Use Case (EADS Cassidian)**
  - **Aviation Use Case (GMV)**
  - SW-Radio (RadioLabs)

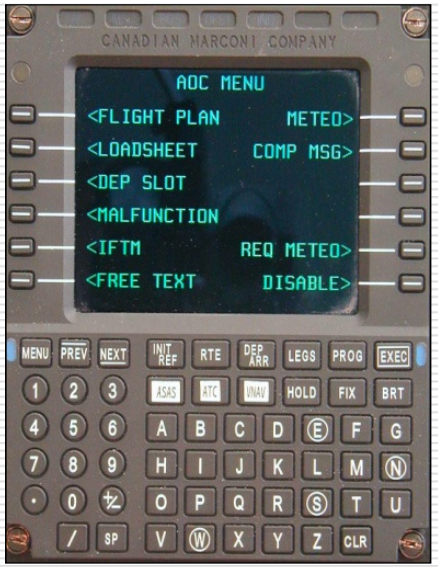
Avionics Use Case

- Airline Operational Center (AOC)
  - Router of Reports
    - Ground <-> Aircraft
    - Database <-> Pilot
    - Between on-board systems



Avionics Use Case

- AOC
  - Written in C
  - ARINC 653
  - According to DO-178B DAL C
  - 30 KLOC
  - Originally 3 Threads
    - 1 periodic thread
    - 1 sporadic thread
    - 1 background thread
  - Timing:
    - 120ms period
    - 30ms deadline



Why Java?

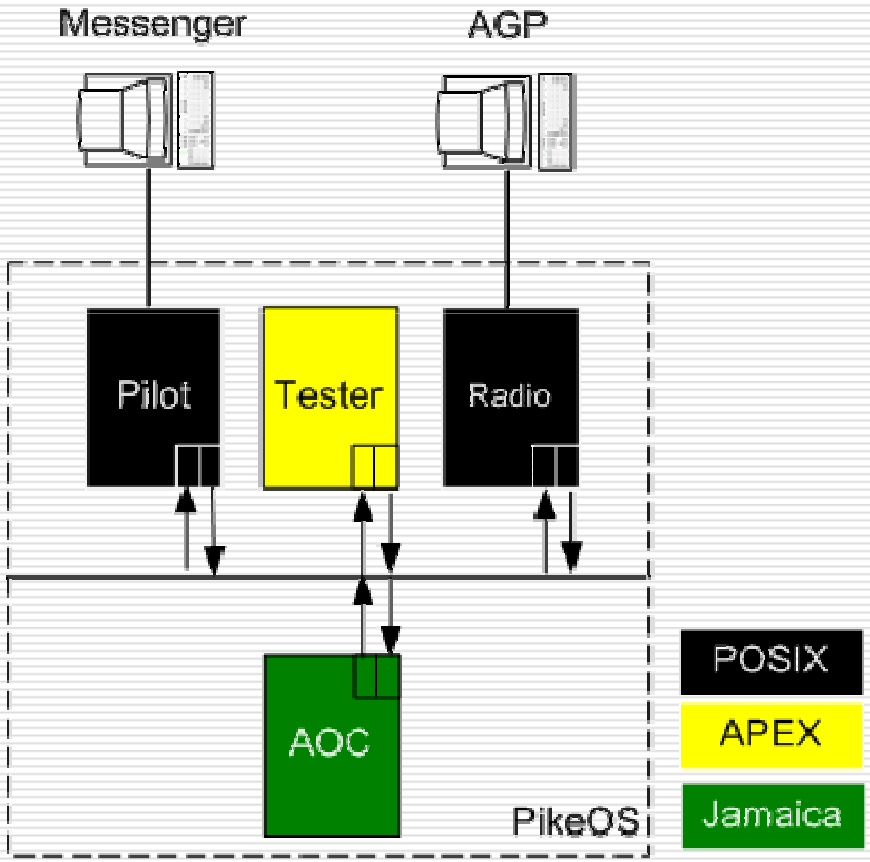
- Four arguments:
  - The "Prototyping Argument"
    - We prototype in Java and
    - want to gradually add non-functional requirements to come to the real thing
  - The "Complexity Argument"
    - A380 FMS: 2Mio LOC
  - The "Engineers Argument"
  - The "Alternative-to-C Argument"

Why Multicore?

- Overall Argument:
  - Reduction of weight & power consumption
- "High-Performance Modules"
  - Running 32-64 applications in partitioned system
- IMA in Hardware
  - Use CPUs to relax time-partitioning

The Demonstrator

- Main Module
  - QuadCore, Intel-based
- Communication simulated over Ethernet
- Reuse of original test drivers



The Java Exercise

- DIANA Project:
  - Use of Perc Pico, with memory annotations
  - Result:
    - "Certification-friendly" approach
    - Memory managment must be taken into account during application design
    - Adding memory annotations implies some re-factoring
    - "First we abstract the platform away with Java, then we bring it back with annotations"

The Java Exercise

- Jeopard with GC
  - Engineer: “Cool! It’s like Java!”
  - Manager: “Cool! Short Time-to-Market!”
  - But difficult to demonstrate that we never run out of memory?
    - A solution for this would be annotations!
  - So the solution in practice seems to be
    - Low criticality (C, D): Automatic GC
    - High criticality (A, B): Scope-based memory

The Java Exercise

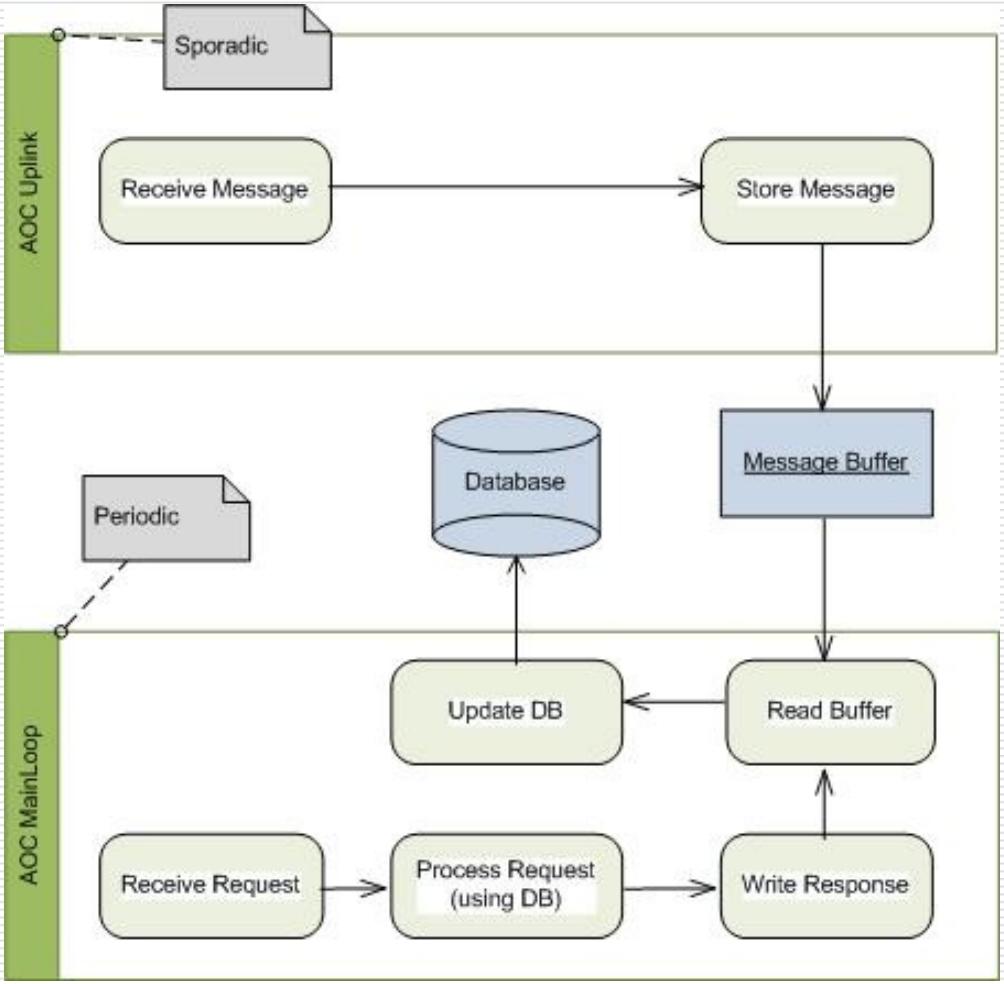
- No execution time issues C -> Java!
  - Deadlines were slightly relaxed (2ms more)
- But some libraries caused problems and had to be rewritten
  - Message Queues
  - RegEx Library

Multi-Core Exercise

- Particular Interest in this use case
  - Scaling:
  - Give clear usage domains for reports that can be processed with given
    - Processing power (CPUs)
    - With given upper-bound waiting time for pilot requests (Note: Each time the pilot navigates through the menu, a request is sent to the application!)

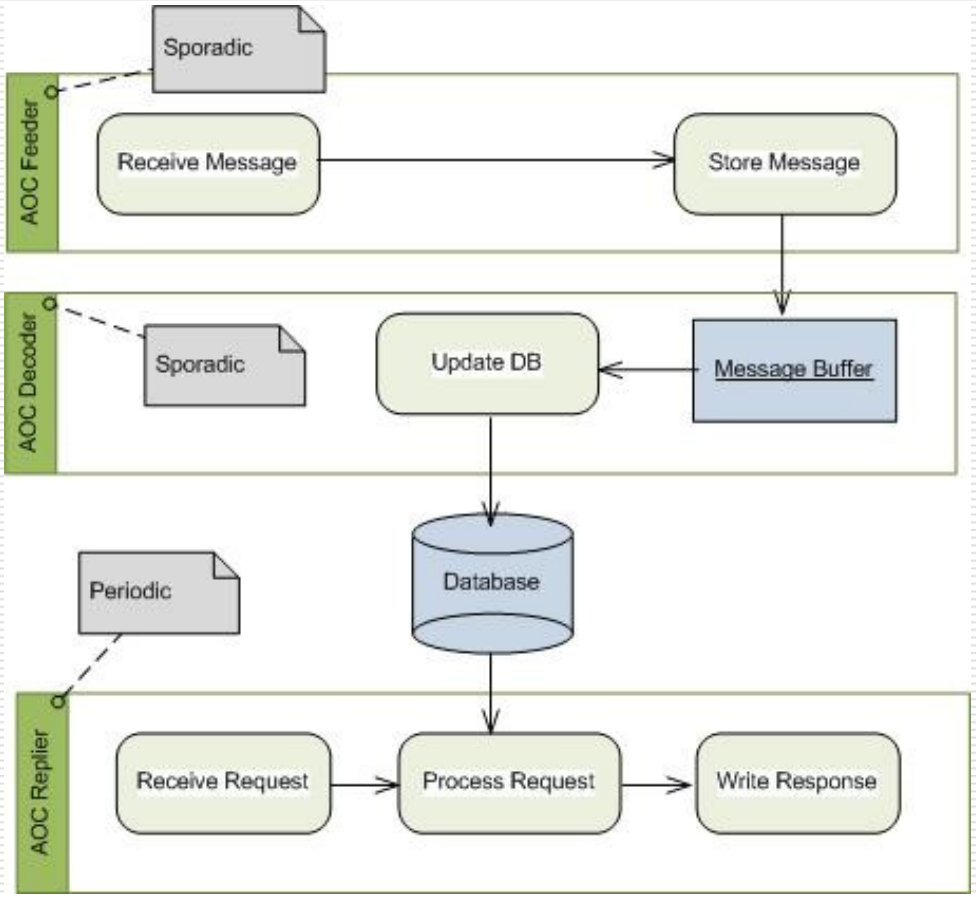
Multi-Core Exercise

- Original Design led to poor speed-up due to lock contention on database



Multi-Core Exercise

- New design with less locking
- Easier to parallelise message decoding
- Isolated Request Handler makes life easier!



Multi-Core Exercise

- Results:
  - The isolated request handler on one CPU makes life much easier!
  - Scaling:
    - With each CPU added, the number of reports can be increased by constant factor!
    - This is a very nice achievement!

Tool Support

- Mini Tool Chain:
  - **Veriflux** (static analyser) to find suspicious code (data races)
  - When we could not claim false positive, we used
    - **cJUnit** (Concurrent JUnit) to define test cases
      - Find different orders of execution to provoke different result

Conclusions

- Java
  - There are use cases for Java in Avionics!
  - There is justification for GC even in Avionics!
  - There is justification for memory annotations in critical applications.
- Multi-Core
  - We achieved precise scaling
  - Re-design was (of course?) necessary

Questions?

- ...