Embedded Computer Systems
ecs/f09

Jan Vitek

based on slides by Andy Wellings

Personal history

- PhD on the Seal calculus (mobile processes)
- Started on Real-time Java in 2001, in DARPA project. At the time, no real RTSJ implementation
- Developed the Ovm virtual machine framework, a clean-room, open source RT Java virtual machine
- Fall 2005, first flight test with Java on a plane
- Collaboration with IBM Research
- Fiji systems LLC

Duke's Choice Award

Technical Aims of the Course

- To understand the basic requirements of concurrent and real-time systems
- To understand how these requirements have influenced the design of Java and the Real-Time Specification for Java
- To be able to program advanced concurrent real-time Java systems
- To gain experience with embedded programming using actual hardware
- To get ideas for you own research

Course Contents

- Introduction to real-time processing
- Concurrent Programming in Java
- Real-time Specification for Java
- Advanced Topics
  - Real-time garbage collection
  - Practice with Embedded boards
  - Certification, Safety Critical Systems
  - WCET analysis

Workload

- ECS is a seminar class which will eventually become a regular Qual class
- YOU pick how much you want to get out of it (as a function of the effort you put in)

Workload I

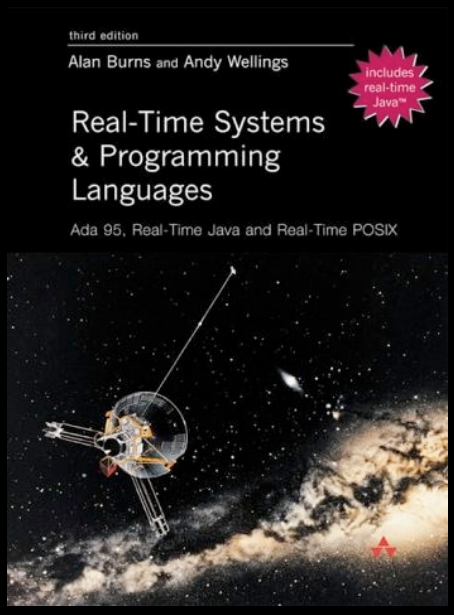
- 1 take-home midterm (20%)
- 1 in-class final (30%)
- 3 small programming assignments (10%)
- 3 optional programming assignments (+10%)
- Choose: either programming project or literature review (40%)

Workload II

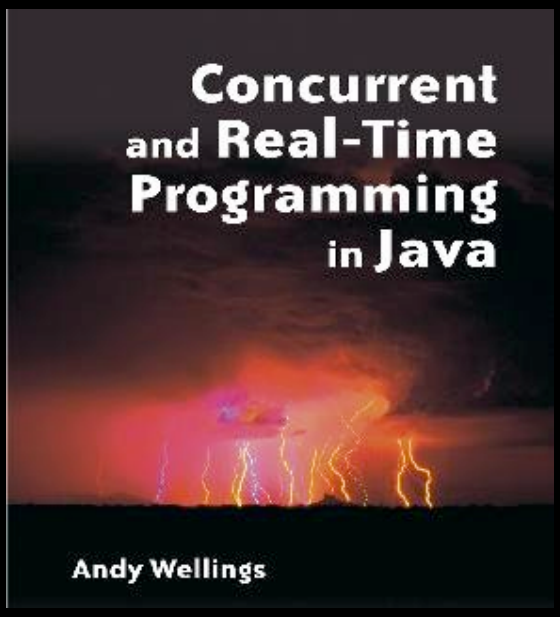
- 1 take-home midterm (20%)
- 1 in-class final (30%) + additional questions (50%)

Introduction to Concurrent and Real-time Processing

Books

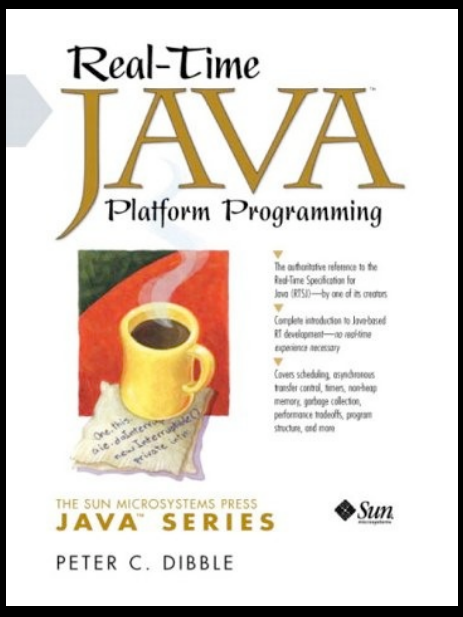


RTSJ Version 0.9

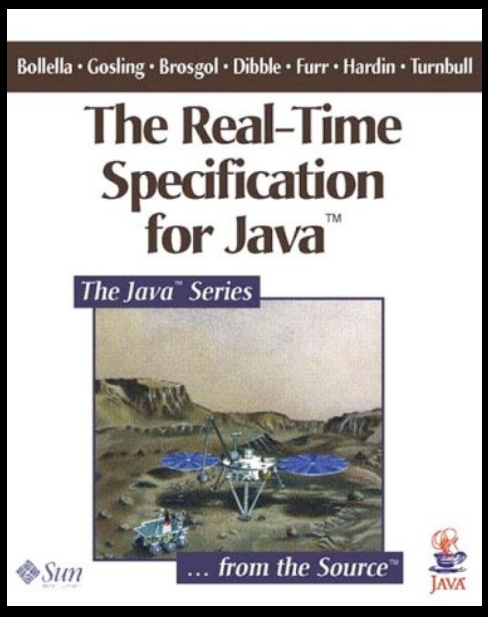


RTSJ Version 1.0.1

Other books



RTSJ Version 1.0

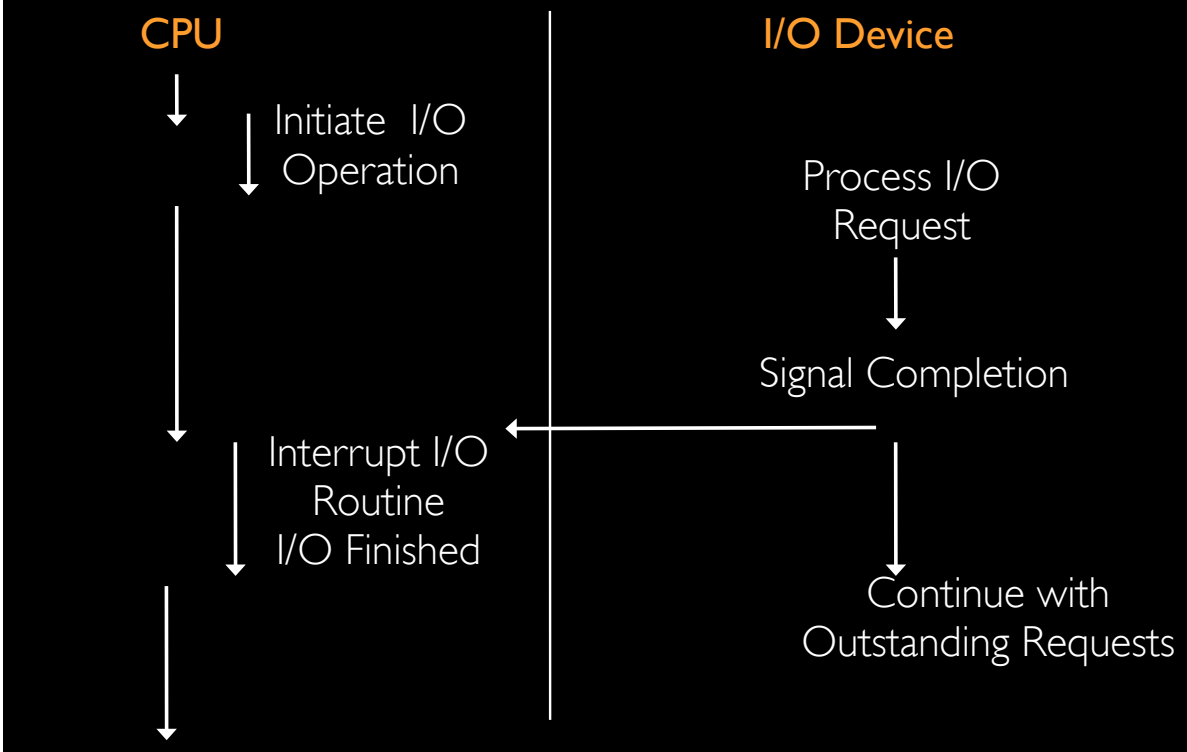


RTSJ Version 0.9

Concurrent Programming

- The name given to programming notation and techniques for expressing potential parallelism and solving the resulting synchronization and communication problems
- Implementation of parallelism is a topic in computer systems (hardware and software) that is essentially independent of concurrent programming
- Concurrent programming is important because it provides an abstract setting in which to study parallelism without getting bogged down in the implementation details
- A concurrent programming model is a set of abstractions exposed by the programming environment to specify and control concurrent activities

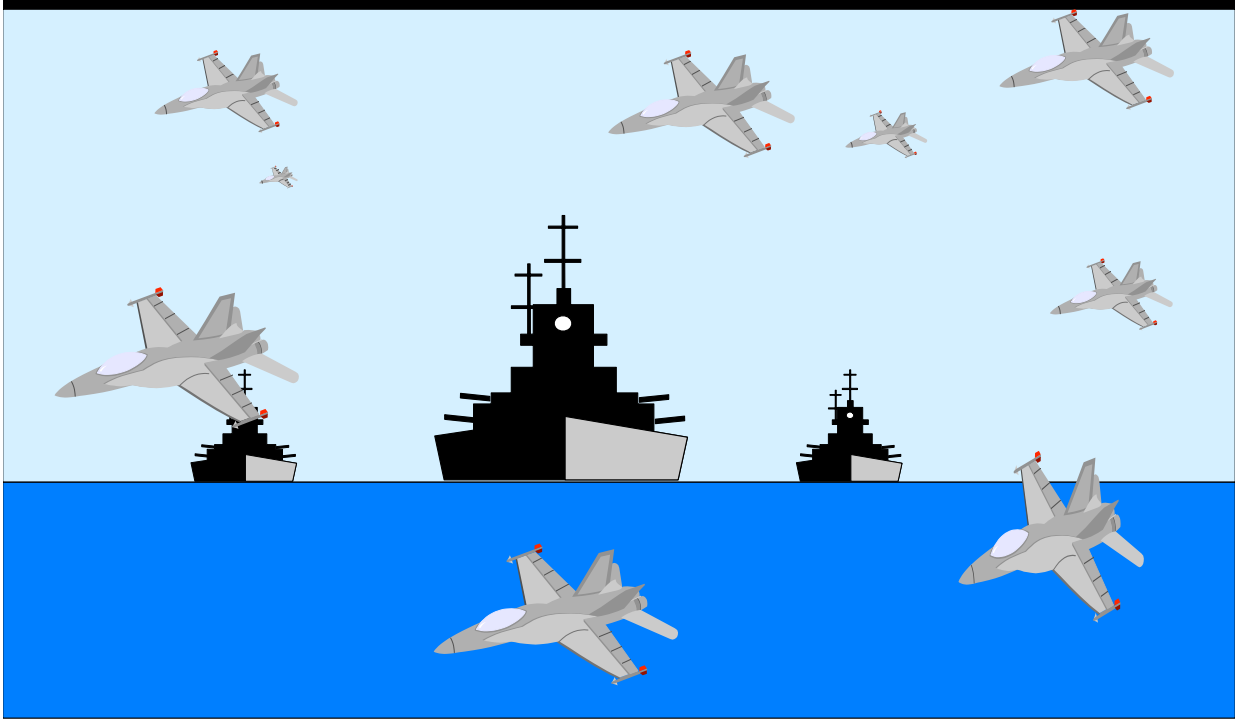
Parallelism Between CPU and I/O Devices



Why we need it

- To fully utilize computing resources
- To allow the expression of potential parallelism so that more than one computer can be used to solve the problem
- To model the parallelism in the real world
- Virtually all real-time systems are inherently concurrent — devices operate in parallel in the real world
- This is, perhaps, the main reason to use concurrency

Air Traffic Control



Why we need it

- Alternative: use sequential programming techniques
- We must construct the system as the cyclic execution of a program sequence to handle the various concurrent activities
- This complicates the task and involves considerations of structures irrelevant to the control of the activities in hand
- The resulting programs will be more obscure and inelegant
- Decomposition of the problem is more complex
- Parallel execution on >1 processors is more difficult to achieve
- The placement of code to deal with faults is more problematic

Terminology

- A concurrent program is a collection of autonomous sequential processes, executing (logically) in parallel
- Each process has a single thread of control
- The actual implementation (i.e. execution) of a collection of processes usually takes one of three forms.
  - Multiprogramming: processes multiplex execution on single processor
  - Multiprocessing: processes multiplex their executions on a multiprocessor system where there is access to shared memory
  - Distributed Processing: processes multiplex their executions on several processors which do not share memory

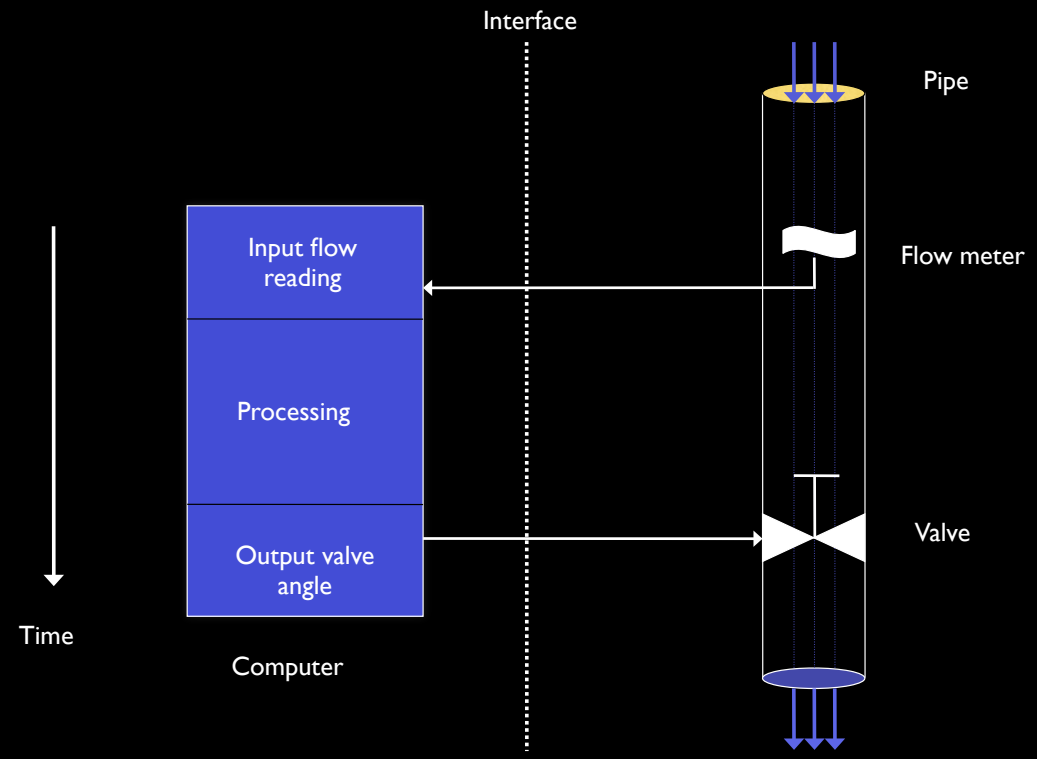
What is a real-time system?

- A real-time system is any information processing system which has to respond to externally generated input stimuli within a finite and specified period
  - *correctness depends not only on logical result but also time it is delivered*
  - *failure to respond is as bad as the wrong response!*
- The computer is a component in a larger engineering system => **Embedded Computer System**
- 99% of all processors are for the embedded systems market

Terminology

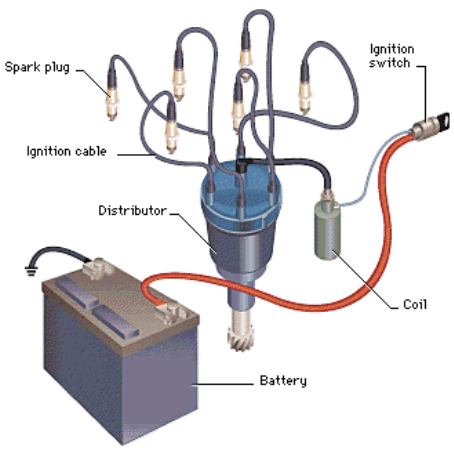
- **Hard real-time** — systems where it is absolutely imperative that responses occur within the required deadline. E.g. Flight control systems.
- **Soft real-time** — systems where deadlines are important but which will still function correctly if deadlines are occasionally missed. E.g. Data acquisition system.
- **Firm real-time** — systems which are soft real-time but in which there is no benefit from late delivery of service.
- A system may have all hard, soft and real real-time subsystems. Many systems may have a cost function associated with missing each deadline

A simple fluid control system

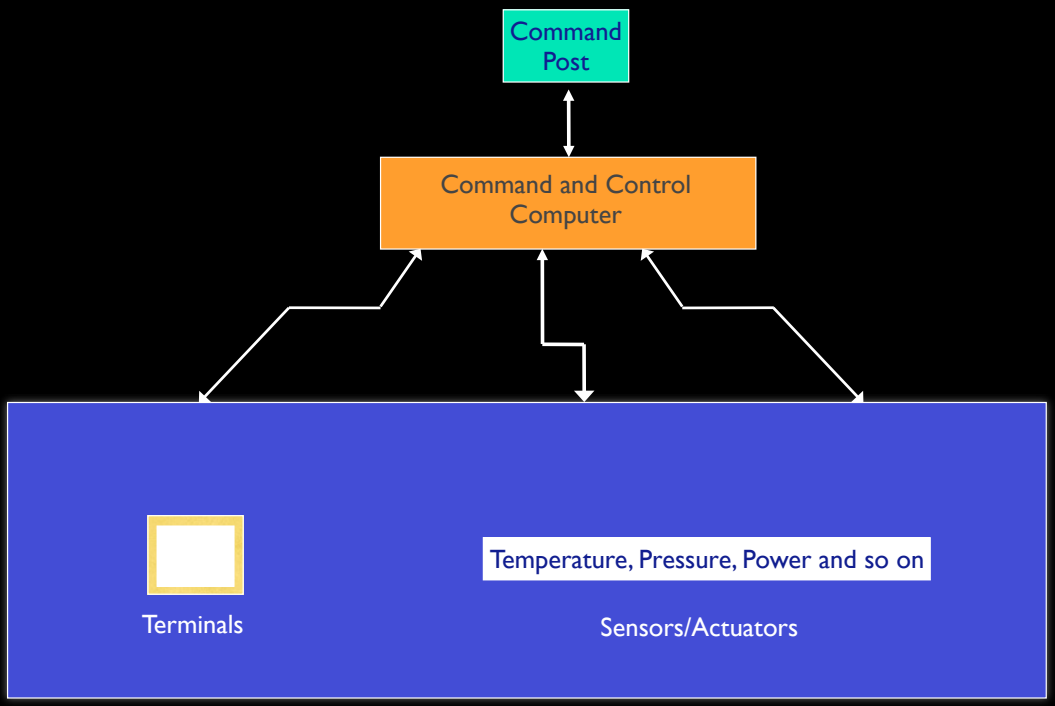


Distributor in Ignition System

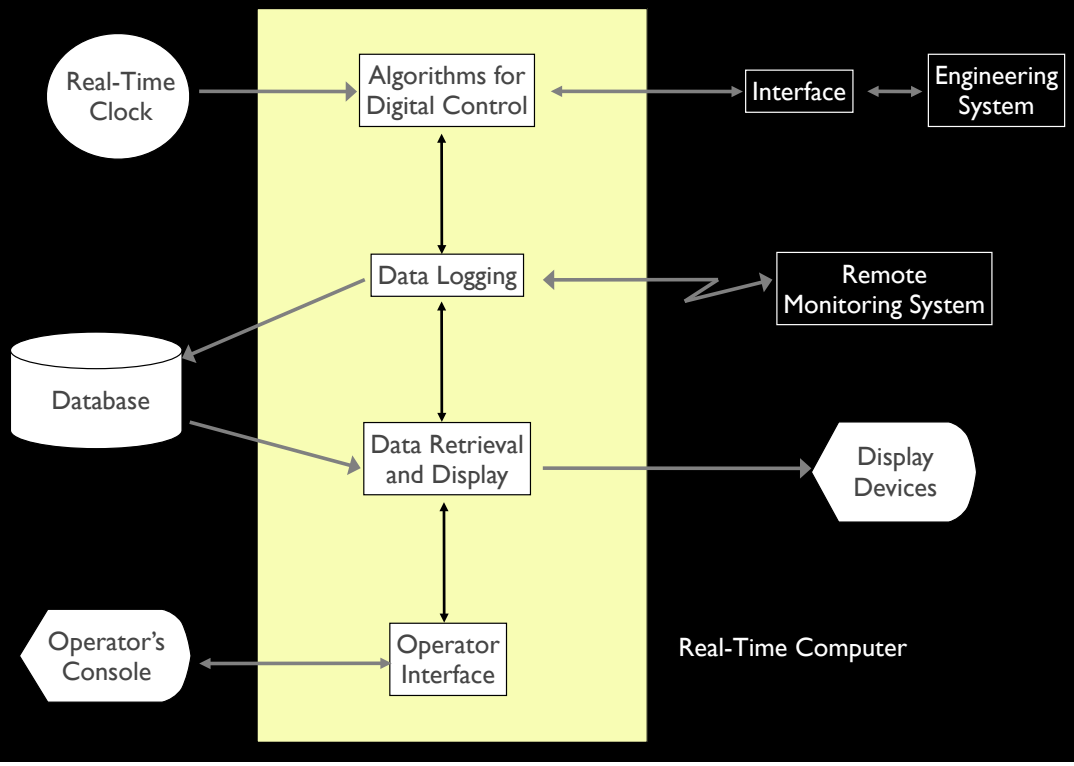
- Mechanical Distributor
  - Advance timing as function of load and speed
    - Vacuum advance—vacuum from intake manifold draws against a diaphragm under heavy loads
    - Centrifugal advance — weights swing out as engine speed increases
- Analog Computer
  - Same control laws as mechanical system
- Digital Computer
  - Equations
  - Look-up tables with interpolation
  - Digital filtering



A Command and Control System



A Typical Embedded System



Relevance of Embedded Systems

- "[…] estimate that each individual […] may unknowingly use more than 100 embedded computers daily"
- "The world market for embedded software will grow from about $1.6 billion in 2004 to $3.5 billion by 2009, at an average annual growth rate (AAGR) of 16%."
- "Embedded hardware growth will be at the aggregate rate of 14.2% to reach $78.7 billion in 2009, while embedded board revenues will increase by an aggregate rate of 10%"

http://www.bccresearch.com/comm/G229R.html, http://www.ecpe.vt.edu/news/ar03/embedded.html

Characteristics of a RTS

- Large and complex — vary from a few hundred lines of assembler or C to 20 million lines of Ada estimated for the Space Station Freedom
- Concurrent control of separate system components — devices operate in parallel in the real-world; better to model this parallelism by concurrent entities in the program
- Facilities to interact with special purpose hardware — need to be able to program devices in a reliable and abstract way

Characteristics of a RTS

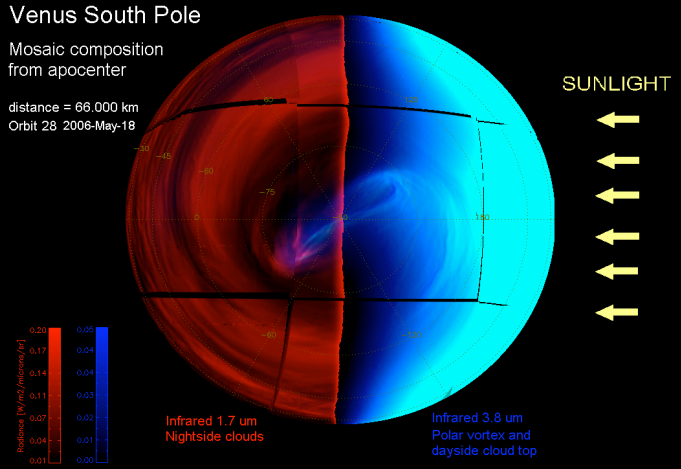
- Extreme reliability and safe — embedded systems typically control the environment in which they operate; failure to control can result in loss of life, damage to environment or economic loss
- Guaranteed response times — we need to be able to predict with confidence the worst case response times for systems; efficiency is important but predictability is essential

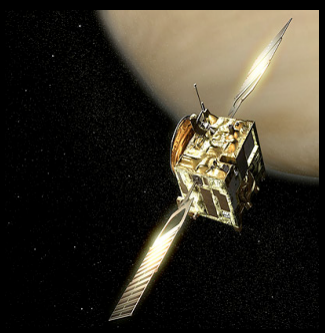
Real-time Programming Languages

- Assembly languages
- Sequential systems implementation languages
  - e.g. RTL/2, Coral 66, Jovial, C, C++.
- High-level concurrent languages. Impetus from the software crisis.
  - e.g. Ada, Chill, Modula-2, Mesa, Esterel, Java.
- We will focus on Java and the Real-Time Specification for Java
- See Burns, Wellings, *Real-Time Systems and Programming Languages*, for a discussion on languages and operating systems

RTEMS/LEON Embedded Platform

ESA Venus Express Mission



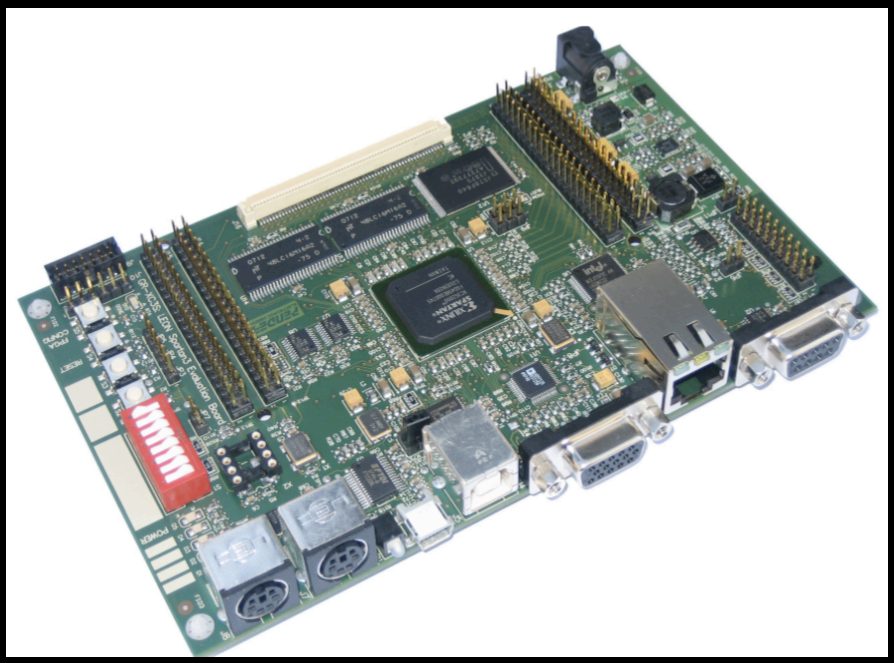


- Framing camera controlled by RTEMS/LEON
- Launched 2005, arrived 2006, still operating

NASA Dawn Mission

- Launched 2007, now (2009) 700+ days from Vesta Asteroids
- Framing camera with Xilinx FPGA runnign RTEMS/LEON2

Xilinx FPGA GR-XC3S-1500 Board



LEON3 Processor

- SPARC v8 instruction set architecture
  - 32-bit, big-endian, RISC inspired, 32-bit linear address space
  - 8 generic registers, register windows
  - MMU
  - FPU
- Instruction and data cache
- Scratch-pad RAM
- Pipe-line
- Hardware multiplication and division in FPU

Why care about Real-time?

- Real-time systems are inherently concurrent with multiple tasks of different priorities running on the same machine
- Most concurrent systems have implicit timeliness and responsiveness expectations
- There is much research in concurrent programming abstractions (e.g. transactional memory, data flow, actors,…)
- Yet, there has been, historically, little interest in Real-time from the programming language community and the real-time community tends to focus on scheduling and operating systems

Why care about Real-time?

- The **programming model** for most real-time systems is 'defined' as a function of the hardware, operating system, and libraries.
- Consequently real-time systems are not portable across platforms
- **Good news:** programming languages, such as Java and C#, are wrestling control from the lower layers of the stack and impose well-defined semantics (on threads, scheduling, synchronization, memory model)

What programming model?

- There are many dimensions:
  - Imperative vs. Functional
  - Shared memory vs. Message passing
  - Explicit lock-based synchronization vs. Higher-level abstractions (data-centric synchronization, transactional memory)
  - Time-triggered vs. synchronous / logic execution time
- And multiple languages, systems:
  - C, C++, Ada, SystemC, Assembler, Erlang, Esterel, Lustre, Giotto …

What programming model?

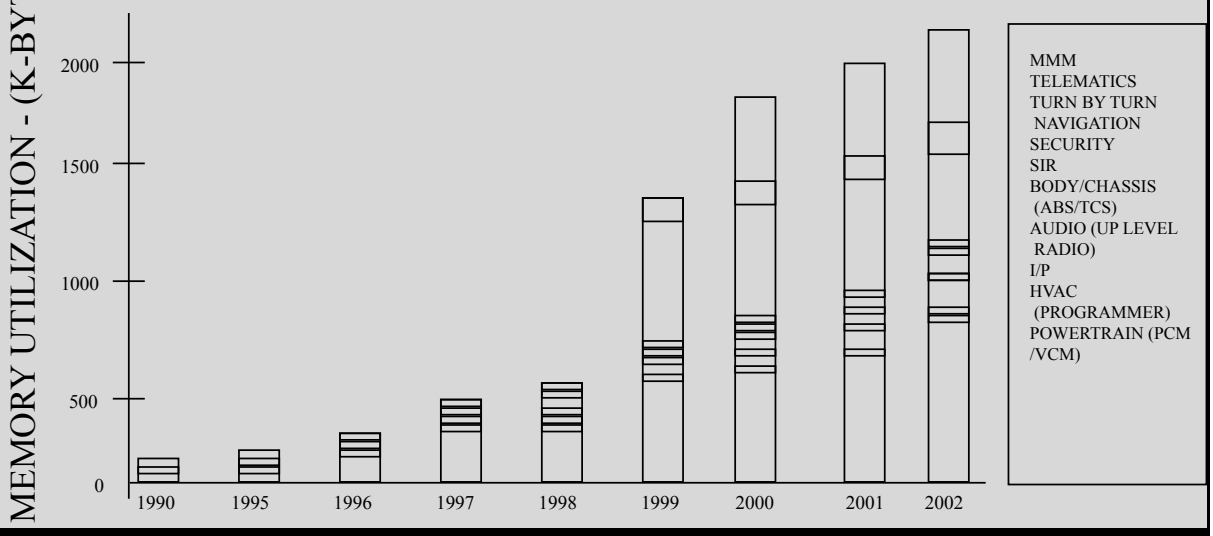
- "Real-time systems require fine grained control over resources and thus the language of choice is C, C++ or assembly"
- ...entails the software engineering drawbacks of low-level code
- Consider the following list of defects that have to be eradicated (c.f. "Diagnosing Medical Device Software Defects" Medical DeviceLink, May 2009):
  - Buffer overflow and underflow (does not occur in a HLL)
  - Null object dereference (checked exception in a HLL)
  - Uninitialized variable (does not occur in a HLL)
  - Inappropriate cast (all casts are checked in a HLL)
  - Division by zero (checked exception in a HLL)
  - Memory leaks (garbage collection in a HLL)

What programming model?

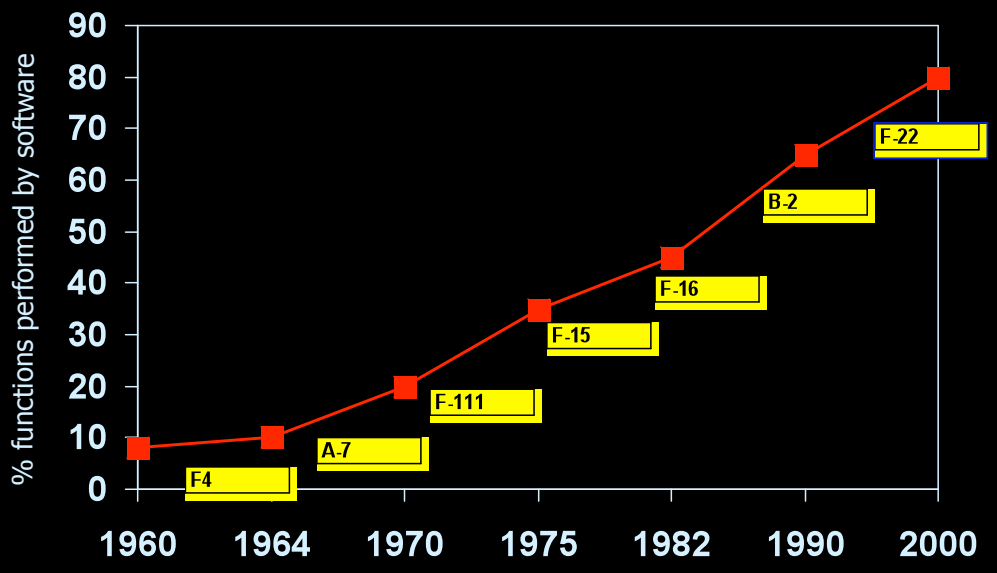
- Development time, code quality and certification are increasingly criteria. For instance in the automotive industry:
  - 90% of future innovation in the auto industry will be driven by electronics and software — *Volkswagen*
  - 80% of car electronics in the future will be software-based — *BMW*
  - 80% of our development time is spent on software — *JPL*
- Worst, software is often the source of missed project deadlines.

What programming model?

Growth in Embedded Vehicle Software



What programming model?



DoD Defense Science Board Task Force on Defense Software, November 2000

What programming model?

- Software and money
- Typical productivity
  - 5 Lines of Code (LoC) per person day ⇒ 1 kloc/person year
  - From requirements to end of module test
- Typical avionics "box"
  - 100 kloc ⇒ 100 person years of effort
  - $500M on a modern aircraft?

What programming model?

- The important metrics are thus
  - Reusability
  - Software quality
  - Development time

What programming model?

Java?

- Productivity

Software-intensive systems require high-level languages
C/C++ are less safe, and less portable
Ada is struggling for developers
Millions of Java programmers

- What about performance?

For hand-tuned code, Java is ~2x slower than C,
but large systems are often coded defensively
it is often easier to analyze Java code

- Is Java too dynamic?

Class loading and Just in time compilation need not be used
Indeed many real-time JVM are compiled to C and statically linked

Java?

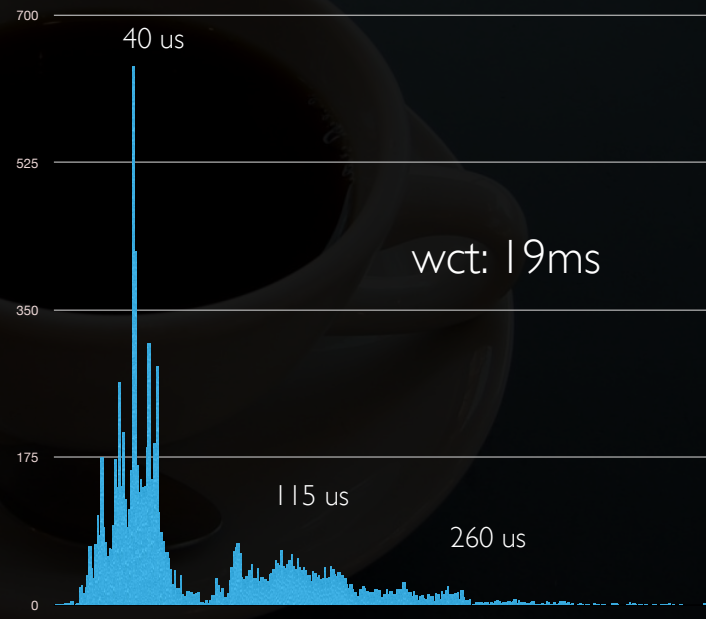
- Object-oriented programming helps software reuse
- Mature development environment and libraries
- Memory-safe high-level language
- Portable, little implementation-specific behavior
- Concurrency built-in, support for multiprocessors, memory model
- Garbage collected

Java?

- Predictable?
- *Not really.*

Call sleep(10ms) and get up 20 milis.sec. variability.

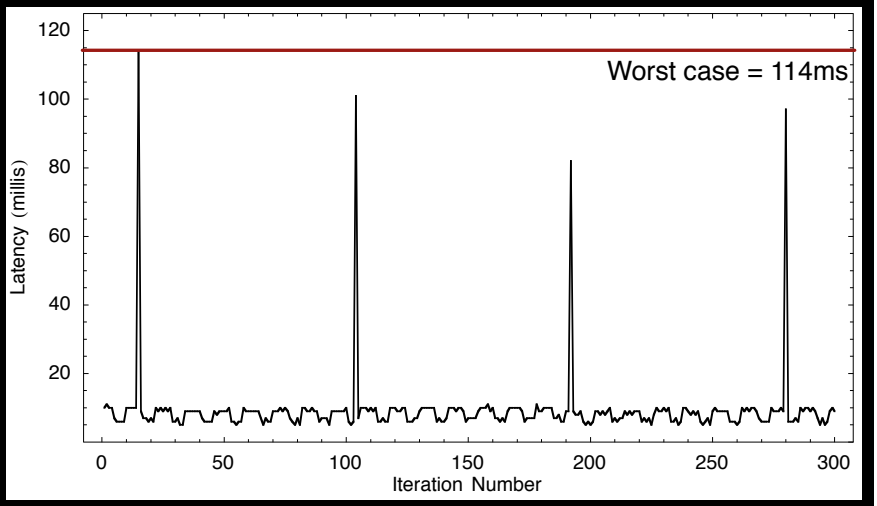
Hard real-time often requires microsecond accuracy.



Time scale

Java?

- Predictable?



- **Java Collision Detector** running at 20Hz.
  - *Bartlett's Mostly Copying Collector. Ovm. Pentium IV 1600 MHz, 512 MB RAM, Linux 2.6.14, GCC 3.4.4*
- GC pauses cause the collision detector to miss up to three deadlines… *and this is not a particularly hard problem should support KHz periods*

Java?

- Java for **hard** real-time?

RTSJ Programming Model

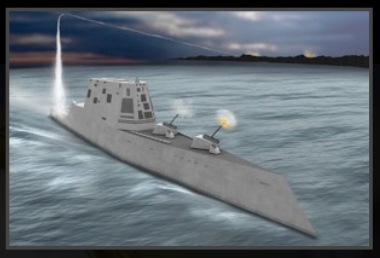
- Java-like:
  - Shared-memory,
  - lock-based synchronization,
  - first class threads.
- Main real-time additions:
  - Real-time threads + Real-time schedulers
  - Priority avoidance protocols: PIP or PCE
  - Region-based memory allocation (to avoid GC pauses)

RTSJ

- Commercial implementations:
  - PERCS (AONIX), JamaicaVM (AICAS) ahead-of-time compiler
  - McKinack (SUN) based on Hotspot, JIT, SMP, RTGC
  - Websphere (IBM) based on J9, ahead/JIT, SMP, RTGC

Real-time applications

- Shipboard computing
  - US navy Zumwalt-class Destroyer,
    Raytheon / IBM
    5 mio lines of Java code
    Real-time GC key part of system.
- Avionics
  - Zedasoft's Java flight simulator
  - IBM Scaneagle UAV
- Financial information systems



Case Study: ScanEagle

ScanEagle

ScanEagle

- Flight Software:
  - 953 Java classes, 6616 methods. Multiple Priority Processing:
    - High (20Hz) - Communicate with Flight Controls
    - Medium (5 Hz) - Computation of navigation data
    - Low (1 Hz) - Performance Computation
  - Embedded Planet 300 Mhz PPC, 256MB memory, Embedded Linux
- Java performed better than C++