CS557: Foundation of CPS

Introduction to ES, and CPS

XiangYang Li

Slides are developed based on slides by others.
Special thanks to S.-P. Ren, Rajkumar, Lee, etc
The Next Computing Revolution

• Mainframe computing (60’s – 70’s)
  Large computers to execute big data processing applications

• Desktop computing & Internet (80’s – 90’s)
  One computer at every desk to do business/personal activities

• Ubiquitous computing (00’s)
  Numerous computing devices in every place/person
  Millions for desktops vs. billions for embedded processors

• Cyber Physical Systems (10’s)
Introduction to Cyber-Physical Systems

Topics covered:

1. Real life applications of cyber-physical systems
2. General concepts about distributed real-time computing
3. Distributed systems modeling --- the Actor model
4. Modeling of time
5. Emerging cyber-physical systems
Reading Materials

- Jane Liu’s book chapter 1
- R. Kurki-Suonio, *Real time: further misconceptions (or half-truths) [real-time systems]; Computer, Volume 27, Issue 6, June 1994 Page(s):71 - 76
- Friedemann Mattern, *Virtual Time and Global States of Distributed Systems*, 1989
- NSF Workshops
- Proceedings of ICCPS
What are Distributed Real-Time Embedded (DRE) Systems?

• Distributed systems

• Real-Time systems

• Embedded systems

• DRE Systems
DISTRIBUTED SYSTEMS
Distributed Systems

- Consist of components that may necessarily be physically distributed.
- Consist of communicating processes on multiple processors and/or dedicated hardware connected by communication links.
  - Communications by wired line or wireless channels
- Motivation:
  - economical
    - 4 8-bit micro-controller may be cheaper than a 32-bit processor
  - multiple processors to handle multiple time-critical tasks
  - physically distributed
    - devices under control may be physically distributed.
EMBEDDED SYSTEMS
Embedded Systems

• Embedded systems
  ‣ employs a combination of hardware & software (a “computational engine”) to perform a specific function;
  ‣ is part of a larger system that may not be a “computer”;
  ‣ works in a reactive and time-constrained environment.

• Software is used for providing features and flexibility

• Hardware = {Processors, Memory,...} is used for performance (& sometimes security)
“Embedded Systems”

• More than 95% of all microprocessors are used in real-time embedded systems!

• High-end automobiles may have 100 microprocessors or more:
  ▸ seat belt check, dashboard, engine, comfort, security, operation, ABS, airbag

• They are everywhere

• number of embedded processors in home?
  ▸ 1999: estimated: 40-50
  ▸ 2008: about 150 embedded microprocessors around the home in various appliances, media players and other consumer electronic devices, plus there are another 40 or 50 in your car
  ▸ Estimated: be over 1,000 embedded devices per person by 2015.
Example: Car

- Physical Operating environment: road conditions, other cars
- Controlling system:
  - human driver (sensors?)
  - computer (sensors?)
- Controls:
  - accelerator, brake pedal, steering wheel
- Actuators:
  - wheels, engines, brakes
- Cruise control:
  - regulates speed of car by adjusting throttle
  - measures speed through device connected to drive shaft
  - hard real-time: drive shaft revolution events
  - soft real-time: driver inputs, throttle adjustments
Example: Automotive Telematics

• In 2005, 30 – 90 processors per car
  ▸ Engine control, Break system, Airbag deployment system, Windshield wiper, Door locks, Entertainment system
  ▸ The new Mercedes S-class has 63 microprocessors; a 1999 BMW 7-series has 65.

• Cars are sensors and actuators in V2V networks
  Active networked safety alerts
  Autonomous navigation

• Future Transportation Systems
  Incorporate both single person and mass transportation vehicles, air and ground transportations.
  Achieve efficiency, safety, stability using real-time control and optimization.
Some Other Examples
Embedded Systems

Gumstix Overo COM, a tiny, OMAP-based embedded computer-on-module with Wifi and Bluetooth.

Picture of the internals of an ADSL modem/router. A modern example of an embedded system. Labelled parts include a microprocessor (4), RAM (6), and flash memory (7).
Example: Health Care & Medicine

- National Health Information Network, Electronic Patient Record
- Home care: monitoring and control
  Pulse oximeters, blood glucose monitors, infusion pumps, accelerometers, …
- Operating Room of the Future
  Closed loop monitoring and control; multiple treatment stations, plug and play devices; robotic microsurgery
  System coordination challenge
- Progress in bioinformatics: gene, protein expression, systems biology, disease dynamics, control mechanisms
Example: Electric Power Grid

- **Current picture:**
  - Equipment protection devices trip locally, reactively
  - Cascading failure: August (US/Canada) and October (Europe), 2003

- **Better future?**
  - Real-time cooperative control of protection devices
  - Or -- self-healing -- (re-)aggregate islands of stable bulk power (protection, market motives)
  - Ubiquitous green technologies
  - Issue: standard operational control concerns exhibit wide-area characteristics (bulk power stability and quality, flow control, fault isolation)
  - Technology vectors: FACTS, PMUs
  - Context: market (timing?) behavior, power routing transactions, regulation

Images thanks to William H. Sanders, Bruce Krogh, and Marija Ilic
More Embedded Systems Examples

• Examples
  ▶ Consumer electronics, e.g., cameras, camcorders, ....
  ▶ Consumer products, e.g., washers, microwave ovens, ...
  ▶ Industrial process controllers & avionics/defense applications
  ▶ Computer/Communication products, e.g., printers, FAX machines, ...
  ▶ Emerging multimedia applications & consumer electronics
    • e.g., cellular phones, personal digital assistants, videoconferencing servers, interactive game boxes, TV set-top boxes, ...
    • Multimedia => Increasing computational demands, and increased reliance on VLSI, HW/SW integration.
Major Components in Embedded Systems

- Data acquisition and processing
- Communication
- System logic and control algorithm
- Interface
- Auxiliary units
  - display
  - storage
  - monitoring and protection
  - test and diagnosis.
Major Components in Real-Time Embedded Systems

A/D → Controller → D/A

Sensor → Plant → Actuator

Input: r(t)
Output: u(t)
Feedback: y(t)
Embedded Systems Characteristics

• Typical characteristics:
  ‣ perform a single or tightly knit set of functions;
    • (not usually "general purpose")
  ‣ increasingly high-performance & real-time constrained;
  ‣ power, cost and reliability are often important attributes that influence design;
Embedded Systems Design

- **Embedded systems design: Major subtasks**
  - **Modeling**
    - the system to be designed, and experimenting with algorithms involved;
  - **Refining (or “partitioning”)**
    - the function to be implemented into smaller, interacting pieces;
  - **HW-SW partitioning: Allocating**
    - elements in the refined model to either (1) HW units, or (2) SW running on custom hardware or a general microprocessor.
  - **Scheduling**
    - the times at which the functions are executed. This is important when several modules in the partition share some resources (e.g. a single hardware unit).
  - **Mapping (Implementing)**
    - a functional description into (1) software that runs on a processor or (2) a collection of custom, semi-custom, or commodity HW.
Embedded Systems

• Complicating factors in the design of embedded systems
  ▸ Many of the subtasks in design are intertwined.
    • Allocation depends on the partitioning, and scheduling presumes a certain allocation.

• Predicting the time for implementing the modules in hardware or software is not very easy, particularly for tasks that have not been performed before.
  ▸ If a particular module has been implemented earlier, or there is data about an almost similar design, then prediction can exploit this precedence. e.g.,
    • Building a processor (2- years?)
    • C/FORTRAN compiler for a standard architecture.

• Even then, details and personnel may change, causing perturbations in the actual time and resources consumed.
Embedded Systems

- Some Characteristics of Embedded Systems
  - Application specific
  - Digital signal processing in ECS
  - Reactive
  - Real-time
  - Distributed
Application Specific Embedded Systems

• Application specific processor design can be a significant component of some embedded systems.
  ▸ Advantages
    • Customization yields lower area, power, cost, ...
  ▸ Disadvantages
    • Higher HW/software development overhead
    • Hardware/software design, compilers, debuggers, ...
    • May result in delayed time to market!
Application Specific

• **System designed for a given application**
  - application is known a priori before the system design begins
  - system flexibility is important for **upgrades, product differentiation** and **design reuse**, usually achieved through limited system re-programmability

• **In practice, however, the application development takes place concurrently with ES design**
  - delayed partitioning of hardware and software is needed (though rarely achieved)
Reactivity in Embedded Systems

• Closed systems
  ‣ execution indeterminacy confined to one source
  ‣ causal relations are easily established.

• Open systems
  ‣ indeterminacy from multiple sources, not controllable or observable by the programmer
  ‣ not possible to infer causal relations.

• constraints are an important part of system functionality in building embedded computing systems.

• *Embedded is no longer dis-connected, remote or low performance:*
  ‣ *In fact, increasing applications are real-time.*
Real-Time Applications of ES

- **What is a real-time system?**
  A real-time system (defined by IEEE) is a system whose correctness includes its response time as well as its functional correctness.

- **A real-time system consists of tasks under deadline constraints**
  - notion of time typically is global and “physical”

- **Hard real-time versus soft real-time versus firm real-time systems**
  - **Hard real-time systems** are ones in which failure to meet a single deadline may lead to complete and catastrophic system failure.
  - **Soft real-time systems** are ones in which performance is degraded but not destroyed by failure to meet deadlines
  - **Firm real-time systems** are ones in which a few missed deadlines will not lead to a total failure, but missing more than a few may lead to complete and catastrophic failure

- **Hard real-time systems are more often embedded**
  - dedicated applications.
## Soft? Firm? Hard?

<table>
<thead>
<tr>
<th></th>
<th>Real-time classification</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ATM</strong></td>
<td><strong>S</strong></td>
<td></td>
</tr>
<tr>
<td>Avionics weapons delivery system in which pressing a button launches an air-to-air missile</td>
<td><strong>H</strong></td>
<td></td>
</tr>
<tr>
<td>Embedded navigation controller for autonomous robot weed killer</td>
<td><strong>F</strong></td>
<td></td>
</tr>
</tbody>
</table>
Embedded Systems Requirements

• Types of requirements imposed by embedded applications:
  ‣ R1 Functional requirements
  ‣ R2 Temporal requirements
  ‣ R3 Dependability requirements
Functional Requirements

• **Data Collection**
  ‣ Sensor requirements
  ‣ Signal conditioning
  ‣ Alarm monitoring

• **Direct Digital Control**
  ‣ Actuators

• **Man-Machine Interaction**
  ‣ informs the operator of the current state of the controlled object
  ‣ assists the operator in controlling the system.
Temporal Requirements

- Tasks may have deadlines
- Minimal latency jitter
  - Jitter is delay that varies over time
- Minimal error detection latency
- Timing requirements due to tight software control loops
- Human interface requirements.
Dependability Requirements

- **Reliability**
  - probability that a system will provide a specified service until time $t$, given that the system was operational at $t_0$.
  - $R(t) = \exp(-l(t-t_0))$ for a constant failure rate $l$ failures/hr.
  - inverse of the failure rate is called **Mean-Time-To-Failure (MTTF)** in hours.
  - highly reliable ~10^-9 failures/hour

- **Safety**
  - critical failure modes
  - certification

- **Maintainability**
  - MTTR in terms of repairs per hour, *Mean time* to recovery (*MTTR*) is the average time that a device will take to recover from any failure

- **Availability**
  - $A = \frac{MTTF}{MTTF + MTTR}$
  - *MTBF*: Mean time between failures describes the expected time between two consecutive failures for a *repairable system*. $MTBF = MTTF + MTTR$

- **Security**
Classification of Embedded Systems

• Multi-dimensional classifications:
  ▶ Hard versus software systems
  ▶ Fail-safe versus fail-operational systems
    • A fail-safe or fail-secure device is one that, in the event of failure, responds in a way that will cause no harm, or at least a minimum of harm, to other devices or danger to personnel.
    • Fail-operational systems continue to operate when their control systems fail. Examples of these include elevators, the gas thermostats in most home furnaces
  ▶ Guaranteed-response versus best-effort
  ▶ Resource-adequate versus resource-inadequate
  ▶ Event-triggered versus time-triggered.
CYBER PHYSICAL SYSTEMS
Cyber-Physical Systems

• Most of the computers in the world are components of cyber-physical systems
  ‣ Automotive
  ‣ Avionics/aerospace
  ‣ Industrial automation
  ‣ Telecommunications
  ‣ Consume electronics
  ‣ Intelligent homes
  ‣ Health and medical equipment

• Electronics will reach 53% of the cost by the end of the decade
Cyber-Physical Systems

- Embedded computers allow us to add capabilities to physical systems that we could not feasibly add in any other way.

- CPS **merges** computing and communication with physical processes and **mediate** the way we interact with the physical world.
  - Make the system safer and more efficient
  - Reduce the cost of building and operating these systems
  - Allow individual machines to work together to form complex systems that provide new capability

- CPS ? (= or ≠) computing and communication equipment + physical systems
  - Not add, but **merge**. It is about merging computing and networking with physical systems to create new capabilities and improve/maintain product quality
Cyber-Physical Systems

• Computation/information processing and physical processes tightly integrated that it is not possible to identify whether behavioral attributes are the result of computations, physical laws, or both working together.

• Functionality and salient system characteristics are merging through the interaction of physical and computational objects

• Computers, networks, devices and their environments in which they are embedded have interacting physical properties, consume resources, and contribute to the overall system behavior.
Cyber-Physical Systems

Traditionally

- **Information scientists**
  - Abstract away the physical environment in which the computation is performed
  - Fuzzy notion of requirements imposed by physical environment

- **Engineers**
  - Ignore physical properties of the embedded computing platforms
  - Computers are only devices executing algorithms

- **CPS design are often done in an ad hoc way, not repeatable**
  - Need science for this, 😊
Design of Embedded Control Systems

Traditional approach: *Separation of Concerns*

- **Control-theoretic design** of **continuous dynamic feedback loops**
  - ignore implementation details: mode switching, fault detection, real-time constraints, implementation platform, etc.

- **Event-based design** to supervise real-time control loops
  - ignore continuous dynamics: stability, transient response, parametric variations, etc.
Design of Embedded Control Systems

Traditional approach: *Separation of Concerns*

- **Control-theoretic design** of continuous dynamic feedback loops
  - ignore implementation details: mode switching, fault detection, real-time constraints, implementation platform, etc.

- **Event-based design** to supervise real-time control loops
  - ignore continuous dynamics: stability, transient response, parametric variations, etc.

*This works in most cases, BUT ...*
Demands from Emerging Applications

New challenges

• increasingly complex applications
  ‣ safety critical systems
  ‣ autonomy
  ‣ multi-agent

• increasingly complex solutions
  ‣ heterogeneous, distributed platforms
  ‣ sophisticated numerical control algorithms

• Implications
  ‣ engineering insight is inadequate
  ‣ testing-based V&V is insufficient
  ‣ move toward model-based design
# Tools for Design & Implementation of Embedded Control Systems

<table>
<thead>
<tr>
<th><strong>Models</strong></th>
<th><strong>Control Design:</strong> Continuous State</th>
<th><strong>Control Implementation:</strong> Discrete State/Events</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>differential equations, transfer functions, etc.</td>
<td>automata, Petri nets, statecharts, etc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Analytical Tools</strong></th>
<th>Lyapunov functions, eigenspace analysis, etc.</th>
<th>Boolean algebra, formal logics, recursion, etc.</th>
</tr>
</thead>
</table>

| **Software Tools**   | MATLAB, Matrixx, VisSim, etc.                 | SCADE, Statemate, SMV, SAT, etc.               |
Physical is central to CPS:

We need

- new cross-cutting paradigms
- new architectures

CPS will lead to

- more rapid transition of science/technology to critical applications
## Opportunities Created by Cyber-Physical Systems

Cyber-physical technology can be applied in a wide range of domains, offering numerous opportunities in products:

<table>
<thead>
<tr>
<th>Sectors</th>
<th>Opportunities</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transportation</strong></td>
<td>Aircraft that fly faster and further on less energy. Air traffic control systems that make more efficient use of airspace. Automobiles that are more capable and safer but use less energy.</td>
</tr>
<tr>
<td><strong>Defense</strong></td>
<td>More capable defense systems; defense systems that make better use of networked fleets of autonomous vehicles.</td>
</tr>
<tr>
<td><strong>Energy and Industrial Automation</strong></td>
<td>New and renewable energy sources. Homes, office, buildings and vehicles that are more energy efficient and cheaper to operate.</td>
</tr>
<tr>
<td><strong>Health and Biomedical</strong></td>
<td>In-home healthcare delivery. More capable biomedical devices for measuring health. New prosthetics for use within and outside the body. Networked biomedical systems that increase automation and extend the biomedical device beyond the body.</td>
</tr>
<tr>
<td><strong>Critical Infrastructure</strong></td>
<td>Highway systems that allow traffic to become denser while also operating more safely. A national power grid that is more reliable and efficient.</td>
</tr>
</tbody>
</table>
Shortcomings of the Current Science and Technology

• **Composition**
  ‣ **Compositionality**: system-level properties can be computed from local properties of components (subsystems)
  ‣ **Composability**: component properties are not changing as a result of interactions with other components
  ‣ Successful examples
    ‣ Digital logic in computer engineering
    ‣ Linear dynamics in control engineering
    ‣ Process algebra in modeling some aspects of distributed computing
    ‣ Etc.
  ‣ Common feature: property and semantic framework homogeneity
  ‣ What about Cyber-Physical Systems?
Shortcomings of the Current Science and Technology

• Composition for CPS?
  ▸ Inherently heterogeneous
  ▸ Subject to a wide range of physical requirements
  ▸ Subject to system-level requirements
  ▸ Design views may not be orthogonal, i.e., separation of concerns may not work

• System integration?
  ▸ Current technology cannot provide predictability for partially compositional properties
Shortcomings of the Current Science and Technology

- **Design automation**
  - CPS products consist multiple types of physical systems and multiple models of computation and communication.
  - Ad hoc, application specific, build tools for specific applications
  - Do not scale
  - Cannot be generalized
Shortcomings of the Current Science and Technology

- **Certification**
  - Key for safety critical systems
  - Currently, standards, procedures and tests do not scale and incomplete
  - Ideal: compositional certification
    - Works well for physical systems
    - Breaks down for CPS (why?)
Shortcomings of the Current Science and Technology

• Security and privacy
  ▸ CPS opens up new threads: physical systems can now be attacked through cyberspace and vice versa
  ▸ Physical systems are parsimony:
    • failures are independent,
    • Multiple, simultaneous failures are unlikely
  ▸ CPSs are reverse parsimony
    • Cyberattacks can introduce simultaneous or precisely coordinated failures
    • Software replication can cause systematic failures
Sept. 27, 2007 CNN Headline News

• “U.S. power grid vulnerable to cyber attacks”
• “Study: Mouse click could plunge city into darkness”
• “Coordinated attacks could cause widespread damage to electric infrastructure that could take months to fix”
• “Researchers who lunched an experimental cyber attack caused a generator to self-destruct, alarming the government and electrical industry about what might happen if such an attack were carried out on a larger scale”
Science and Technology Challenges

- A new system science foundation and new technology infrastructure that will *merge* fundamental concepts from each and *inject* new ideas of its own.
  - Realign abstraction layers in design flow
  - Semantic foundations for composing heterogeneous models and modeling languages
  - New views of compositionality in heterogeneous systems
  - Technology for achieving predictability of systems with limited compositional properties
  - Foundation for system integration
  - Compositional certification
  - Agile design automation
  - Open architectures for CPS
  - Architectures and tools for reliable, resilient CPSs
Systems Theoretic Approach to Engineering Design
Systems Theoretic Approach to Engineering Design

Outline

- Engineering Design Process
- Design of Feedback Control Systems
- Design of Software Systems
- Cyber-Physical Systems
- Case study
Engineering Design Process

• The design of any engineering system (embedded or otherwise) is a recursive process.
  ▷ Design must satisfy a formal specification on system’s performance
    • The specification may not initially achievable
      – Unrealistic customer’s desires
      – Current technological limitation
  ▷ Prototypes are based on simplified set of assumptions and process model
    • Refine to more realistically represent the actual physical system
    • Refinement may introduce complexities
    • New complexity may violate the specification
    • Exhaustive testing for compliance with the specifications.
Engineering Design Process

Graphical representation of the design process
Engineering Design Process

• Specification: a formal description of what the proposed engineering system is to accomplish

• Process model: formal description of the system that will be built to meet that specification

  Does the process model satisfy the formal specification?

• Verification: a formal analysis method that answers the above question
Engineering Design Process

Does the process model satisfy the formal specification?

- Verification: a formal analysis method that answers the above question
  - Yes: refinement
    - Construct more detailed model of the physical process
    - Construct more detailed specification
  - No: synthesis
    - Construct another process/specification to be appended or composed with the earlier version
Engineering Design Process

Does the process model satisfy the formal specification?

- **Verification**: a formal analysis method that answers the above question
  - Why need to modify specification?
    - Desired functionality or behavior may not be attainable
    - Unrealistic assumptions about the state of technology
    - Problem with the process
  - Verification can help to uncover useful clues to guide the refinement or synthesis
  - Re-verify the refinement or synthesis

*Recursive process. When do we stop?*
Design of Feedback Control System

- \( u(\cdot) : R \rightarrow R \) output of the controller
- \( y(\cdot) : R \rightarrow R \) output of the plant
- \( w(\cdot) : R \rightarrow R \) observation noise
- \( r(\cdot) : R \rightarrow R \) commanded reference
- \( e(\cdot) : R \rightarrow R \) input to the controller, \( e(t) = r(t) - (y(t) + w(t)) \)
Design of Feedback Control System

- Controller and plant processes can be modeled by frequency response functions
- Plant
  - Gain magnitude function: \(|P(\cdot)| : \mathbb{R} \rightarrow \mathbb{R}|$
  - Argument function: \(\text{arg}(P(\cdot)) : \mathbb{R} \rightarrow \mathbb{R}\)
Design of Feedback Control System

- **Plant**
  - $u(t) = \cos(w_0t)$
  - $y(t) = |P(w_0)|\cos(w_0t + \text{arg}(P(w_0)))$
  - $|P(w_0)|$: amplitude
  - $\text{arg}(P(w_0))$: phase difference between input and output sinusoids
  - For *linear systems*, these frequency response functions are sufficient to completely characterize the output of the plant to any periodic input signal
Design of Feedback Control System

- **Controller**
  - Gain magnitude function: $|C(\cdot)| : \mathbb{R} \rightarrow \mathbb{R}$
  - Argument function: $\arg(C(\cdot)) : \mathbb{R} \rightarrow \mathbb{R}$
  - Similar frequency response functions
Design of Feedback Control System

- Controller and plant (loop frequency responses)
  - Gain magnitude function: $|L(w)| = |C(w)||P(w)|$
  - Argument: $\text{arg}(L(w)) = \text{arg}(C(w)) + \text{arg}(P(w))$
The basis of a control system design procedure known as *loopshaping*
Design of Feedback Control System

- The basis of a control system design procedure known as *loopshaping*
  - Low frequency interval: when $|L(w)| > > 1$
  - High frequency interval: when $|L(w)| << 1$
  - Crossover frequency $w$: when $|L(w)| = 1$
Design of Feedback Control System

- Desired loop function for the actual closed-loop system
  - Low frequency region: when $|L(w)| > |W(w)|$
  - High frequency region: when $|L(w)| < |W(w)|$
  - Transition region: $|L(w)|$ has 20dB/decade slope

where
- Specification $|W(w)|$
- Design $|L(w)|$
Design of Feedback Control System

- Loopshaping and specification
Design of Software System

Correctness of a program

• For terminating program
  ‣ Return correct value

• What about programs that never terminate, such as OS?
  ‣ Temporal notion of correctness: temporal ordering of events
  ‣ Formal model of the process
    ‣ Transition system
Design of Software System

- Formal model of the process

```c
int sem, x;
proc1 {
    while (sem != i) {
        wait(sem == 0);
        sem = i;
        pause(a);
    }
    /*start critical section*/
    x = x + 1;
    /*end critical section*/
    sem = 0;
}
```
Design of Software System

- Formal model of the process
Design of Software System

- **Formal model of the process**

  - $T_1 = (Q_1, \rightarrow_1, Q_{01})$
    - $Q_1 = \{q1, q2, q3, q4, q5\}$
    - $\rightarrow_1 = \{(q1, q2), (q2, q2), (q2, q3), (q3, q4), (q4, q2), (q4, q5), (q5, q1)\}$
    - $Q_{01} = \{q1\}$

  - $T_2 = (Q_2, \rightarrow_2, Q_{02})$
    - $Q_2 = \{p1, p2, p3, p4, p5\}$
    - $\rightarrow_2 = \{(p1, p2), (p2, p2), (p2, p3), (p3, p4), (p4, p2), (p4, p5), (p5, p1)\}$
    - $Q_{02} = \{p1\}$

- Mutual exclusion property must hold in composed system
  - Cartesian product of the subprocesses state sets and transition relations
  - $T = (Q_1 \times Q_2, \rightarrow, Q_{01} \times Q_{02})$
Design of Software System

• Formal model of the process
  ‣ Mutual exclusion property must hold in composed system
    • Cartesian product of the subprocesses state sets and transition relations
    • \( T = (Q_1 \times Q_2 \rightarrow, Q_{01} \times Q_{02}) \)

• Given the transition system, \( T \), for the composed process, we define an execution of this process as a sequence \( r : \mathbb{Z}^+ \rightarrow Q_1 \times Q_2 \) of states such that
  1. \( r[0] \in Q_{01} \times Q_{02} \) (we start in the initial sets of both processes)
  2. \( r[k] \rightarrow r[k+1] \) for all \( k = 0, 1, \ldots \),

• \( T \) satisfies the mutual exclusion property if and only if
  ‣ for all executions, \( r[\cdot] : \rightarrow Q_1 \times Q_2 \), of the transition system \( T \), there is no \( k \in \mathbb{Z}^+ \) such that \( r[k] = (q5, p5) \).
Design of Software System

• Automata-theoretic approaches are commonly used in CPS

• In general, these specifications are logical formulas constraining the temporal ordering of events with program execution
Cyber-Physical Systems
Cyber-Physical System

- The orientation angles for the $i$th robot are governed by:

$$\dot{\theta}_i(t) = -\theta_i(t) + u_i(t) + w_i(t)$$

- $u(t)$ is a control signal generated by the cyber part of the system
  $$u_i(t) = K_i(r_i - \theta_i(t))$$
- $w(t)$ is an external disturbance

- **Computer**
  - Supervisor:
    - logical directives: where to go
    - Variables: discrete
  - Controller
    - Translate discrete logical directives to continuous voltage to control the motor
    - Variable: dense
Cyber-Physical System

• **Design approach: Separation-of-Concern**
  ▸ Construct an interface between the supervisor and controller
    • Logical directives
    • Family of movements associated with these directives
  ▸ Maintain the logical abstraction implied by this interface is essential for the correct operation of the system
  ▸ Maintain this interface is trivial if there are sufficient computational resource
  ▸ In practice, the cyber process has finite resources
    • Minimizing the utilization of cyber-resources may create problems if there is any uncertainty in the system
Case Study: Ariane 5 Failure

What happened on June 4, 1996 when the Ariane 5 was lunched?

- Weather condition was good
- Visibility condition was acceptable
- The flight was lunched normally, but 37 seconds later …
- Direct cost $370 million
Case Study: Ariane 5 Failure

Flight data show:

- nominal behavior of the launcher up to H0 + 36 seconds;
- failure of the back-up Inertial Reference System followed immediately by failure of the active Inertial Reference System;
- swivelling into the extreme position of the nozzles of the two solid boosters and, slightly later, of the Vulcain engine, causing the launcher to veer abruptly;
- self-destruction of the launcher correctly triggered by rupture of the links between the solid boosters and the core stage.

Anomalies observed

- Gradual development of variations in the hydraulic pressure of actuators of the main engine nozzle at H0+22 seconds
- These variations had a frequency of approximately 10Hz.
Case Study: Ariane 5 Failure

Ariane 5 design

- In general, flight control system of the Ariane 5 is of standard design.
- The design of Ariane 5 is practically the same as that of an SRI used in Ariane 4 from software perspective.
- The early part of the trajectory of Ariane 5 differs from Ariane 4.
- The attitude of the launcher and its movements in space are measured by an Inertial Reference system (SRI).
- SRI has its internal computer in which angles and velocities are calculated based on sensor data from its inertial platform.
Case Study: Ariane 5 Failure

Ariane 5 design

- The data from the SRI are transmitted through data bus to on board computer.
- The on-board computer (OBC), executes the flight program and controls the nozzles of the solid boosters and the cryogenic Vulcain engine, via servo valves and hydraulic actuators.
- For fault tolerant purpose
  - Two SRI operates in parallel, with one active, and the other in “hot” standby
  - Two OBC
  - Some other units are duplicated as well
Case Study: Ariane 5 Failure

Chain of Technical Events:

SRI $\rightarrow$ (data) OCB $\rightarrow$ (control) nozzles and engine

64-bit to 16 bit floating point conversion, converted value larger than 16 bit could represent $\rightarrow$ operand error $\rightarrow$ Software exception $\rightarrow$ declare failure $\rightarrow$ diagnostic bit $\rightarrow$ interpreted as flight data

SRI (backup) cease to function for the same reason $\rightarrow$ OCB domed
Case Study: Ariane 5 Failure

Ariane 5 design

• software specifications from the Ariane 4, were reused in Ariane 5 but its flight path was different and beyond the range for which the code had been reused.
• The disintegration occurred due to software exception of On board computer.
• The software exception was caused during conversion of 64 bit floating point to 16 bit signed integer value.
Case Study: Ariane 5 Failure

Comments on the Failure Scenario

- The primary cause for failure scenario are operand error when converting the horizontal bias variable BH.
- Lack of protection of this conversion which caused the SRI computer to stop.
- Not all conversions were protected was due to maximum 80% workload constraint for SRI computer (in the failure scenario, 4 out of 7 operands were protected, why not all – obscure)
- No trajectory data were used to analyze the behavior of the unprotected variables
Case Study: Ariane 5 Failure

Comments on the Failure Scenario

• Incorrect software specification: in the event of any kind of exception, the system specification stated that: the failure should be indicated on the databus, the failure context should be stored in an EEPROM memory, and finally, the SRI processor should be shut down.

• Hardware failure and software failure should be handled differently
  ▶ Random hardware failure can be easily handled by a backup
  ▶ Software failure, sometimes degraded performance is more preferred than simple shutdown
Case Study: Ariane 5 Failure

Comments on the Failure Scenario

• Ariane 5 inherited a lot from Ariane 4, even some of the stuff were not applicable to Ariane 5.
  ▸ “Unless proven necessary, it was not wise to make changes in software which worked well on Ariane 4”
Case Study: Ariane 5 Failure

Comments on the Failure Scenario

- Software is an expression of a highly detailed design and does not fail in the same sense as a mechanical system.
- Software is flexible and expressive and thus encourages highly demanding requirements, which in turn lead to complex implementation which are difficult to access.
- Critical software must be identified at a very detailed level, exceptional behavior must be confined, and a reasonable back-up policy must take software failures into account.
Case Study: Ariane 5 Failure

Comments on possible other weaknesses

• The review has covered following areas
  ‣ The design of the electrical system
  ‣ Embedded on board software in subsystem other than the Inertial frame of reference system
  ‣ The on board computer and the flight program software
Case Study: Ariane 5 Failure

Comments on testing and qualification procedures

- Equipment qualification
- Software qualification (On-Board Computer software)
- Stage integration
- System validation tests
Case Study: Ariane 5 Failure

Comments on testing and qualification procedures

• Test!
• Try to write code so that it cannot fail.
• Don't allow errors or exceptions to propagate in an uncontrolled manner
• Reused code still needs to be tested.
• Test!
Case Study: Ariane 5 Failure

Recommendations

• No software function should run during flight unless it is necessary
• Prepare a test facility including as much real equipment as technically feasible, inject realistic input data, and perform complete, closed-loop, system testing. Complete simulations must take place before any mission. A high test coverage has to be obtained.
• Do not allow any sensor, such as the inertial reference system, to stop sending best effort data.
Case Study: Ariane 5 Failure

Recommendations

- Organize, for each item of equipment incorporating software, a specific software qualification review. The Industrial Architect shall take part in these reviews and report on complete system testing performed with the equipment. All restrictions on use of the equipment shall be made explicit for the Review Board. Make all critical software a Configuration Controlled Item (CCI).
Case Study: Ariane 5 Failure

Recommendations

• **Review all flight software (including embedded software), and in particular:**
  ‣ Identify all implicit assumptions made by the code and its justification documents on the values of quantities provided by the equipment. Check these assumptions against the restrictions on use of the equipment.
  ‣ Verify the range of values taken by any internal or communication variables in the software.
  ‣ Solutions to potential problems in the on-board computer software, paying particular attention to on-board computer switch over, shall be proposed by the project team and reviewed by a group of external experts, who shall report to the on-board computer Qualification Board.
Case Study: Ariane 5 Failure

Recommendations

- Wherever technically feasible, consider confining exceptions to tasks and devise backup capabilities.
- Provide more data to the telemetry upon failure of any component, so that recovering equipment will be less essential.
- Reconsider the definition of critical components, taking failures of software origin into account (particularly single point failures).
- Include external (to the project) participants when reviewing specifications, code and justification documents. Make sure that these reviews consider the substance of arguments, rather than check that verifications have been made.
Case Study: Ariane 5 Failure

Recommendations

- Review the test coverage of existing equipment and extend it where it is deemed necessary.
- Give the justification documents the same attention as code. Improve the technique for keeping code and its justifications consistent.
- Set up a team that will prepare the procedure for qualifying software, propose stringent rules for confirming such qualification, and ascertain that specification, verification and testing of software are of a consistently high quality in the Ariane 5 programme. Including external RAMS experts is to be considered.