An Outline of a Course on Safety-Critical Embedded Systems

H. Kopetz
TU Vienna
October 2003

Why a Course on *Safety-Critical Systems*?

- The number of safety-critical computer systems is on the increase (e.g., X-by-wire, Nuclear, Medical, etc.).
- State of the Art Design is demanded by society:
  Otherwise *Contributory Negligence*
- Precise specification and assessment of all assumptions is a requirement--Integration of diverse viewpoints.
- Every Design Decision must be substantiated by rational arguments.
- Rare-event behavior at the focus of interest--arguments must stand up versus the 10^{-9} failures/hour challenge.
- Justification vis-à-vis a Certification Agency

Challenging topic for an advanced academic course.

Some of the Challenges

System Perspectives

- The 10^{-9} Challenge
- The Process of Abstracting
- Physical Hardware Faults
- Design Faults
- Human Failures

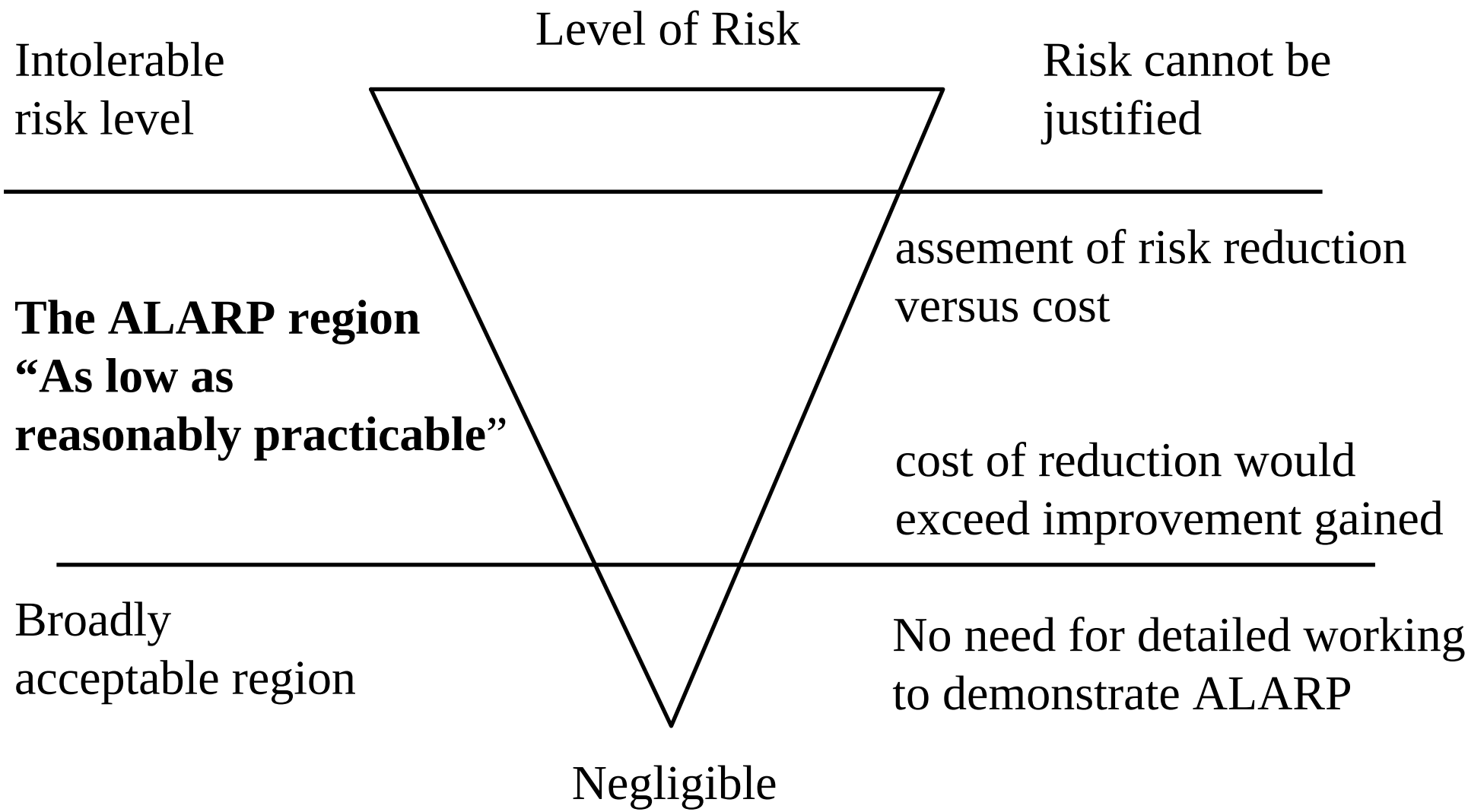
The 10^{-9} Challenge

- **The system as a whole must be more reliable than any one of its components**: *e.g., System Dependability 1 FIT--Component dependability 1000 FIT (1 FIT: 1 failure in 10^9 hours)*
- Architecture **must support fault-tolerance** to mask component failures
- System as a whole is **not testable** to the required level of dependability.
- The safety argument is based on a **combination** of **experimental evidence** about the expected failure modes and failures rates of **fault-containment regions (FCR)** and a **formal dependability model** that depicts the system structure from the point of view of dependability.

The Process of Abstracting

- The behavior of a safety-critical computer system must be explainable by a hierarchically structured set of behavioral models, each one of them of a cognitive complexity that can be handled by the human mind.
- Establish a **clear relationship between the behavioral model and the dependability model** at such a high level of abstraction that the analysis of the dependability model becomes tractable.
- From the dependability point of view, the future unit of hardware failure is considered to be a complete chip.

ALARP--Technology vs. Economy vs. Legal



Example: Nuclear Plant Safety

There is a limit for an intolerable risk above which the operation of a nuclear plant would not be allowed. Risk reduction is sought as long as it is economically reasonable (ALARP).

The ALARP principle rests on the assumption that marginal improvements in safety can be compared with the associated marginal costs. The method rests on probabilistic analysis.

There is a difficulty with software, because software failure occurrences are hard to predict.

Zero Failure Rate Software?

- Is the claim of “zero failure-rate software” *achievable* and *assessable?*
- If the “zero failure-rate software route” is taken, than the first software failure invalidates the argument.
- Experience has shown that it is highly probable that software (and even hardware) is not free of design faults.
- Scientifically based statements^{-5}/hour.

Panel Discussion at WORDS 03 in Capri

Research Challenges for the ORDS (Object-Orientend Real-Time Dependable Systems**) Community in the Coming Years?**

- Integration difficult, because every community defines its own basic concepts
- Communication and Education difficult, because there is no standard set of definitions for fundamental concepts (*compare physics)*
- More Basic Research badly needed in order to achieve a unification of concepts and mechanisms (*e.g.*, *DSOS*)

Course Organisation

- Start with establishing *common concepts*--DSOS Conceptual Model.
- Course is organized along *design principles*.
- Students must read the relevant literature before the respective lecture. Course material consists of a list of selected references--no textbook available.
- Students are expected to actively contribute towards the objective of the course.

Design Principles

1. Regard the Safety Case as a Design Driver
2. Start with a Precise Specification of the Design Hypotheses
3. Ensure Fault-Containment and Error Containment
4. Establish a Consistent Notion of Time and State
5. Partition the System along well-specified LIFs
6. Make Certain that Components Fail Independently
7. Follow the Self-Confidence Principle
8. Hide the Fault-Tolerance Mechanisms
9. Design for Diagnosis
10. Create an Intuitive and Forgiving Man-Machine Interface
11. Record Every Single Anomaly
12. Provide a Never Give-Up Strategy

Regard the Safety Case as a Design Driver

- A safety case is a set of documented arguments in order to convince experts in the field (e.g., a certification authority) that the provided system as a whole is safe to deploy in a given environment.
- The safety case, which considers the system as **whole**, determines the criticality of the computer system and analyses the impact of the computer-system failure modes on the safety of the application.
- The distributed computer system should be structured such that the required experimental evidence can be collected with reasonable effort and that the dependability models that are needed to arrive at the system-level safety are tractable.
- The safety case should be regarded as a design driver since it establishes the critical failure modes of the computer system.

Start with a Precise Specification of the Design Hypotheses

The design hypotheses is a statement about the assumptions that are made in the design of the system. Of particular importance for safety critical real-time systems is the fault-hypotheses: a statement about the number and types of faults that the system is expected to tolerate:

- Determine the Fault-Containment Regions (FCR): *A fault-containment region (FCR)* is the set of subsystems that share one or more common resources and that can be affected by a single fault.
- Specification of the Failure Modes of the FCRs and their Probabilities
- Be aware of Scenarios that are not covered by the Fault-Hypothesis

Fault Isolation

- The immediate consequences of a fault must be isolated to within a well-defined region, the *fault containment region.*
- Fault-Containment Regions must fail independently.
- Consider spatial proximity.
- Design Faults?

Ensure Error Containment

In a distributed computer system the consequences of a fault, the ensuing error, can propagate outside the originating FCR either by an erroneous message or by an erroneous output action of the faulty node to the environment that is under the node's control.

- A propagated error **invalidates** the independence assumption.
- The error detector must be in different FCR than the faulty unit.
- Distinguish between architecture-based and application-based error detection
- Distinguish between error detection in the time-domain and error detection in the value domain.

Establish a Consistent Notion of Time and State

A system-wide consistent notion of a discrete time is a prerequisite for a consistent notion of state, since the notion of *state* is introduced in order to separate the *past* from the *future*:

"The state enables the determination of a future output solely on the basis of the future input and the state the system is in. In other word, the state enables a "decoupling" of the past from the present and future. The state embodies all past history of a system. Knowing the state "supplants" knowledge of the past. Apparently, for this role to be meaningful, the notion of past and future must be relevant for the system considered." (Taken from Mesarovic, Abstract System Theory, p.45)

Fault-masking by voting requires a consistent notion of state in distributed Fault Containment Regions (FCRs).

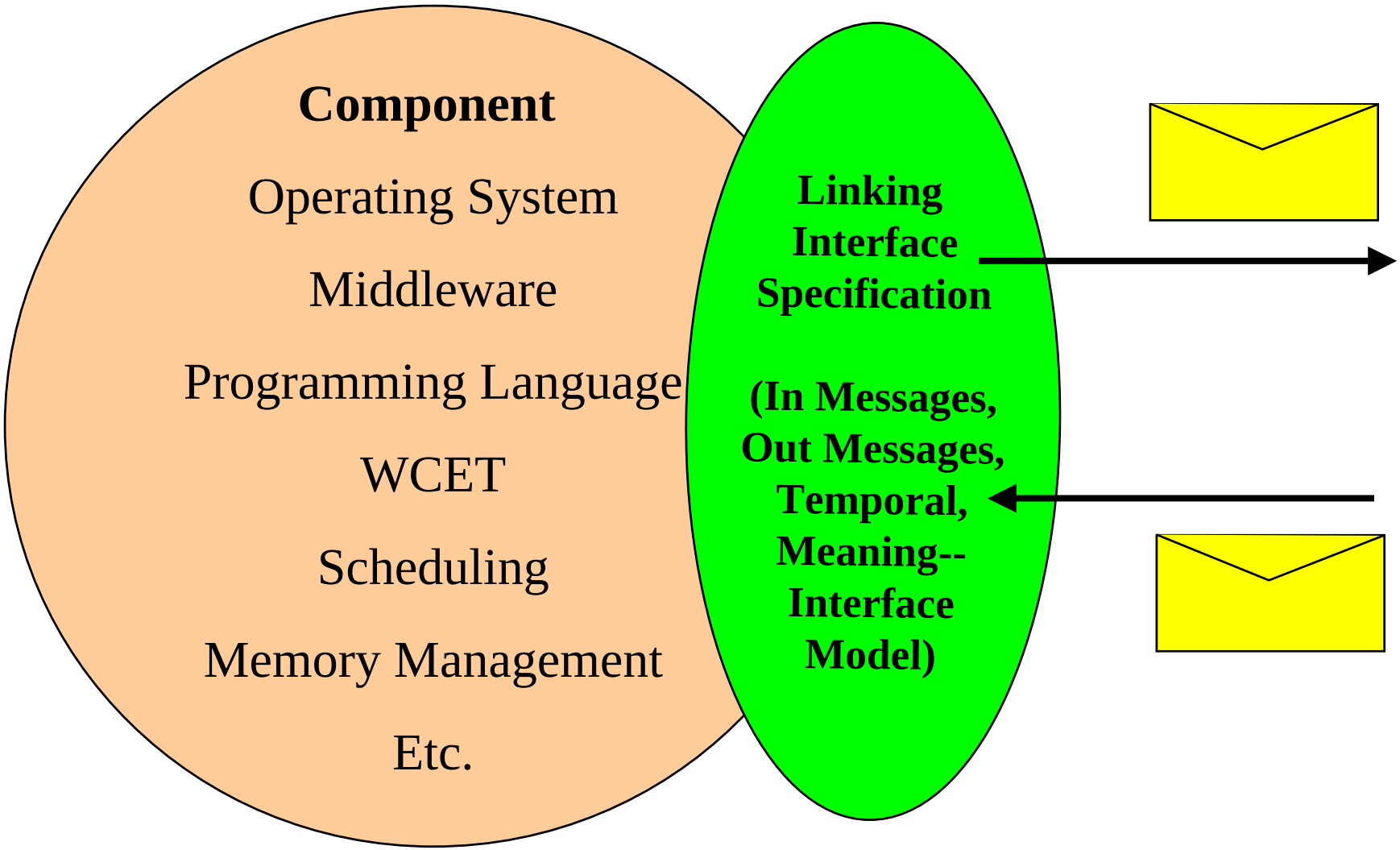
Partition the System along well-specified LIFs

"Divide and Conquer" is a well-proven method to master complexity.

A linking interface (LIF) is an interface of a component that is used in order to integrate the component into a system-of-components.

- We have identified only two different types LIFs:
  - time sensitive LIFs and
  - not time sensitive LIFs
- Within an architecture, all LIFs of a given type should have the same generic structure
- Avoid concurrency at the LIF level

The LIF Specification hides the Implementation



Make Certain that Components Fail Independently

Any dependence of FCR failures must be reflected in the dependability model--a challenging task!

Independence is a system property. Independence of FCRs can be compromised by

- Shared physical resources (hardware, power supply, time-base, etc.)
- External faults (EMI, heat, shock, spatial proximity)
- Design
- Flow of erroneous messages

Conclusion

- Safety-critical computer systems are penetrating into many applications.
- Some design decision require a delicate balance between technical, economic and legal parameters.
- Managers must trust the recommendations made by *technical experts*.
- Society has the right to expect that our academic institutions provide the proper training in the state of the art for these technical experts.