NPS-CS-07-006

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

Proceedings of the
First IEEE International Workshop on Safety of Systems

By

James Bret Michael
John Hauraz
Zachary Pace

Approved for public release; distribution is unlimited.

Prepared for: Center for National Software Studies
NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

Summary of Position Papers

Selected Issues in Computer Systems Safety: Position Paper, *Andrew J. Kornecki, Janusz Zalewski*

This paper addresses the role of software in system safety, where the application of computers or programmable devices may put the users or public at risk.

To make significant progress inventing and innovating in the area of safety assessment and assurance there will need to be a corresponding level of funding to similar to steps taken a few years ago to sponsor security research.

The way the present authors see progress made possible in the next 5-10 years is via a significant coordinated effort of respective government agencies and industrial sectors. It should be made clear to the decision makers that if cost minimization will continue to be an essential factor in safety-related industries, then we may soon experience the kind of failures which were caused not so long ago by breaches in security.

Subject Introduction, *Archibald McKinlay*

This introduction is background for three papers which require a similar introduction:

- Hooking into Systems Engineering
- Systems Safety Engineering HR
- Systems Safety in new Architecture and Technologies

Unlike Systems Safety Engineering, little has been done to incorporate software requirements and risk management into Systems Engineering. There needs to be a holistic systems integration approach to the updating of Systems Engineering to re-integrate Systems Safety Engineering, Systems Assurance and Security, and Software Engineering. The DoD has efforts in-work to update the Systems Engineering Plan (SEP) but it will be a year more before it is finished.

Each added discipline required a change in the typical engineer's abilities, education and experience. The advancing technologies must be viewed in the same model. When a safe system is taken and simply attached to the Internet for monitoring the safety risk changes, and changes in ways that are not obvious to the traditional Software Safety or Systems Safety Engineer. The technologies are changing so fast that systems are being built right now without the updated training, education, or toolkit being available because neither the chip nor the interface existed at the project's start.

Systems Engineering must return to the roots of risk management and use that to maintain focus in prioritizing tasks in all schedules, meetings and budgets. Like Systems Safety was made to absorb occupational and then environmental tasks, so also must Systems Engineering reconnect to its many children. All children must coordinate through and with Systems Engineering.

Table of Contents

Selected Issues in Computer Systems Safety: Position Paper

Andrew J. Kornecki
Dept. of Computer & Software Engineering
Embry-Riddle Aeronautical University
Daytona Beach, FL 32114-3900, USA
kornecka@erau.edu

Janusz Zalewski
Computer Science Department
Florida Gulf Coast University
Fort Myers, FL 33965-6565, USA
zalewski@fgcu.edu

Abstract

The position paper presents the authors' views on the critical issues in safety of computer systems and software. It is based on selected results from several studies the authors have done for various government agencies, private companies and professional societies. Main limitations and challenges in designing computer systems for safety are discussed.

1. Introduction

System safety is a very broad term and books have been written on various aspects of safety analysis and safety assurance [1,2]. In this position paper, we are focusing in particular on various aspects of computer safety, especially the role of software in system safety, where the application of computers or programmable devices may put the users or public at risk. The authors' experience comes mostly from research related to aviation, air transportation and space, but partially also from research on medical, automotive and nuclear devices and technologies. However they by no means claim that the treatment of the subject is complete and exhaustive.

In a broader sense, to evaluate safety of a computer product, especially the software product that is used in a safety critical system, one has to take a closer look at a product itself, but also at the way it has been developed, as well as at the way the tools for developing this product have been created. This logic is illustrated in Figure 1, and is very different from the traditional approaches to system safety, where the analysis is limited only to the product and the related application environment.

The examples come from the recent study on the assessment of software development tools for safety-critical real-time systems conducted for the Federal Aviation Administration (FAA) [3]. Modern commercial development tools are typically complex suites combining multiple functionalities. Considering tool complexity, the quality of support materials is often marginal. Unless developers become expertly proficient with the tool, reliance on it may lead to ignorance of tool functionality, complacency and thus compromise the safety of developed system.

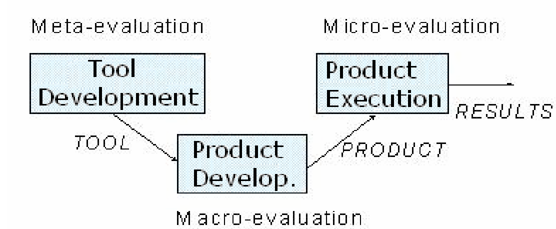


Fig. 1. Context for Evaluating Computer Products.

2. Limitations and Knowledge Barriers

What are the three fundamental limitations and knowledge barriers for safety of systems today?

From the computer use and software standpoint, there are several issues that obstruct progress in dealing with safety. The most important among them seem to be the following:

1) Limited understanding of computers and software by safety engineers and, vice versa, limited understanding of safety issues by computer and software engineers.
2) Very confusing state of safety standards and guidelines, and proliferation of sometimes contradicting guidelines. This situation results in the sheer number of documents the safety critical system developers must be aware of.
3) Lack of well-defined, measurable safety metrics is another fundamental limitation to progress in safety assurance.

Our studies based on the safety related software guidelines in civil aviation DO-178B [4] indicated that the criteria used in this and other safety related standards do not include solid theoretical underpinnings to be used as measures of metrics for safety. This is a significant impediment in product qualification and certification [5].

3. Research Challenges

What are the three most important research challenges?

As it stands right now, even agreeing on the state of the art and practice in computer and software safety research would be difficult. One important step forward would be to produce a document defining the *body of knowledge* in computer system safety, similar to the one produced for security [6]. This would help establish the common ground, from which further steps could be possibly defined. The challenges that researchers are facing in this respect, come from at least the following:

1) Lack of specific data typically available from industrial projects, since the industry does not share this type of data due to the competitive advantage.
2) Common-off-the-shelf components (COTS), both hardware and software, are going to be increasingly used in safety critical systems, but very few studies have been done how to approach their safety assessment.
3) New technologies will proliferate, both in hardware, such as high speed databuses [7], and software, such as automatic code generation [8], for the analysis of which new research methods and approaches will have to be created.

From the perspective of our studies, a critical issue for vendors and government agencies was the necessity of certification based on solid experimental data. However, the qualification data collected from experiments constitute a component of the certification package and are highly proprietary. This situation puts researchers in a very disadvantageous position. Some relevant discussions how to address this and similar issues, have recently taken place at the Tools Forum [8].

4. Promising Innovations

What are promising innovations and abstractions for building future high-confidence safety systems?

It is extremely difficult to determine, which specific techniques or technologies are the most innovative or make the best promise, mostly because their suitability and usefulness have to be proved over time and a range of applications. However, a few essential directions in innovation can be mentioned [9]:

1) Improvement of quality and trustworthiness of products and tools via advances in verification and validation, possibly via the application of formal approaches, such as model checking, has been already in a view of researches for some two decades and is still making a promise.
2) Design diversity as an essential technique in improving computer and software safety has been used successfully for years and will remain to be used as one of the most effective safety techniques thus far.
3) Several newer technologies emerged over the recent years, of which we mention only two: model based development and active safety systems.
4) Present authors' own research based on the concepts of a safety shell [10] and Bayesian belief networks [11] has also a potential to improve safety in an array of applications.

It seems that a significant progress to develop new innovative technologies for safety assessment and assurance may not be possible without some major concentrated effort towards funding respective research. This should be an effort similar to steps taken a few years ago to sponsor security research. The scale of funding should be such that development of innovative solutions would be truly possible. For comparison, it is worthwhile mentioning that the European Commission has recently provided over €3M of funding for a joint university-industry project on active system safety [12].

5. Possible Milestones and Conclusion

What are possible milestones for the next 5-to-10 years?

The way the present authors see progress made possible in the next 5-10 years is via a significant coordinated effort of respective government agencies and industrial sectors, driven by the following three factors:

1) Setting priorities in research directions, for example to define and verify measurable safety metrics.
2) Establishing educational preferences to design and implement changes in the computing curricula as well as by offering respective training for safety engineers.
3) Enforcing qualification and certification processes, so that industry would become better aware how their respective products and activities will undergo thorough but transparent assessment.

Certainly, all this requires a significant increase in the level of funding, which may not be possible without decisive legislative actions. It should be made clear to the decision makers that if cost minimization will continue to be an essential factor in safety related industries, then we may soon experience the kind of failures which were caused not so long ago by breaches in security.

Acknowledgments

This project was supported in part by the Aviation Airworthiness Center of Excellence (AACE) under contract DTFA-0301C00048 sponsored by the Federal Aviation Administration (FAA). Findings contained herein are not necessarily those of the FAA. J. Zalewski acknowledges additional support from the Florida Space Grant Consortium under Grant No. UCF01-E000029751

References

[1] Redmill F. (Ed.), *Dependability of Critical Computer Systems*, Vol. 1 & 2, Elsevier Applied Science, London, 1988/89

[2] Leveson N.G., *Safeware – System Safety and Computers*, Addison-Wesley, Reading, Mass., 1995

[3] Kornecki A.J., J. Zalewski, Experimental Evaluation of Software Development Tools for Safety-Critical Real-Time Systems, *Innovations in Systems and Software Engineering – A NASA Journal*, Vol. 1, pp. 176-188, 2005

[4] RTCA, *Software Considerations in Airborne Systems and Equipment Certification*, Report RTCA/DO-178B, Washington, DC, 1992

[5] Kornecki A., J. Zalewski, The Qualification of Software Development Tools from the DO-178B Certification Perspective, *CrossTalk – The Journal of Defense Software Engineering*, Vol. 19, No. 4, pp. 19-22, April 2006

[6] Redwine, Jr., S.T (Ed.), *Secure Software Assurance: A Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software*, Draft Version 0.9. U.S. Departments of Homeland Security and Defense, January 2006

[7] Kornecki A., J. Zalewski, J. Sosnowski, D. Trawczynski, A Study on Avionics and Automotive Databus Safety Evaluation, *The Archives of Transport*, Vo. 17, No. 3-4, pp. 107-132, 2005

[8] Software Tools Forum, Embry-Riddle Aeronautical University, Daytona Beach, FL, May 18-19, 2004, URL: http://www.erau.edu/db/campus/softwaretoolsforum.html

[9] Zalewski J., W. Ehrenberger, F. Saglietti, J. Gorski, A. Kornecki, Safety of Computer Control Systems: Challenges and Results in Software Development, *Annual Reviews in Control*, Vol. 27, pp. 23-37, 2003

[10] Sahraoui A.E.K., E. Anderson, J. van Katwijk, J. Zalewski, Formal Specification of a Safety Shell in Real-Time Control Practice, *Proc. 25th IFAC/IFIP Workshop on Real-Time Programming*, Mallorca, Spain, May 15-19, 2000, pp. 117-123

[11] Zalewski J., A.J. Kornecki, H. Pfister, Numerical Assessment of Software Development Tools in Safety-Critical Systems Using Bayesian Belief Networks, *Proc. Int'l Multiconference on Computer Science and Information Technology*, Wisła, Poland, November 6-10, 2006, pp. 433-442.

[12] ONBASS – An Onboard Active Safety System, URL: http://www.onbass.org and http://ec.europa.eu/research/aeronautics/projects/article_3704_en.html

Authors' Bios

Dr. Andrew J. Kornecki is a Professor at the Dept. of Computer and Software Engineering, Embry Riddle Aeronautical University. He has over twenty years of research and teaching experience in areas of real-time computer systems. He contributed to research on intelligent simulation training systems, safety-critical software systems, and served as a visiting researcher with the FAA. He has been conducting industrial training on real-time safety-critical software in medical and aviation industries and for the FAA Certification Services. Recently he has been engaged in work on certification issues and assessment of development tools for real-time safety-critical systems. He is currently, with Dr. Zalewski, conducting a study on tool qualification for complex electronic hardware, sponsored by the FAA.

Dr. Janusz Zalewski is a Professor of Computer Science and Engineering at Florida Gulf Coast University. Before taking a university position, he worked for various nuclear research labs, including the Data Acquisition Group of Superconducting Super Collider and Computer Safety and Reliability Center of Lawrence Livermore National Laboratory. He also worked on projects and consulted for a number of private companies, including Lockheed Martin, Harris, and Boeing. He served as a Chairman of IFIP Working Group 5.4 on Industrial Software Quality and of an IFAC TC on Safety of Computer Control Systems. His major research interests include safety-related real-time computer systems. He currently works with Dr. Kornecki on a study for the FAA on tool qualification for complex electronic hardware.

Selected Issues in Computer System Safety

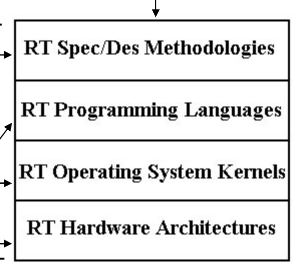
Andrew J. Kornecki
Embry Riddle Aeronautical University
Daytona Beach, FL 32114, USA
`kornecka@erau.edu`
`http://faculty.erau.edu/korn/`

Janusz Zalewski
Florida Gulf Coast University
Fort Myers, FL 33965, USA
`zalewski@fgcu.edu`
`http://www.fgcu.edu/zalewski/`

IFAC Summer School in Prague 2005
Control, Computing and Communication

Lecture Outline

1) Fundamental Concepts
2) Design Principles & Architectures
3) System Safety
4) Real-Time Programming
5) Databus Safety
6) Case Studies & Summary



© 2005 by Andrew Kornecki and Janusz Zalewski

System Safety: Dependability

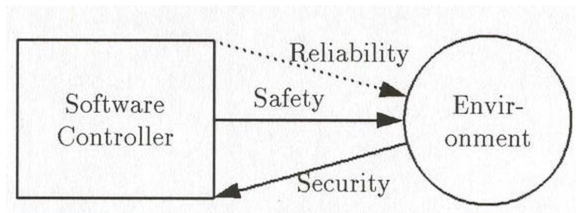
- **Dependability** is the property of the system that justifies reliance on its services
  {more on the topic: "Dependability: Basic Concepts and Terminology", Edited by Laprie, J.-C., Springer Verlag, 1992, ISBN: 3-211-82296-8}
- Dependability is encapsulation of the following properties/abilities *{adapted from Laprie}:*
  - **Reliability** - probability to function correctly over a given period of time
  - **Security** - ability to prevent unauthorized access and system damage
  - **Safety** - ability of not harming people and not cause property damage

System Safety: Dependability

<u>Dependability involves:</u>

- **Attributes** - the metrics for evaluation of system services (safety, reliability, security, availability, integrity, maintainability, confidentiality, etc.)
- **Impairments** - causes or results of lack of dependability (error, fault, failure)
- **Means** - the methods used to deliver dependable services (fault prevention, removal, detection, tolerance)

System Safety: Dependability



Relationships among Reliability, Safety and Security Attributes

System Safety: Dependability

Reliability – failure does not lead to severe consequences (high risk) to the environment or computer system, nevertheless improving the failure rate is of principal concern

Safety – failure leads to severe consequences (high risk) to the environment (and possibly to computer)

Security – failure leads to severe consequences to the computer system (and possibly to the environment)

System Safety: Dependability

Example of dependability issues in a car embedded control software:

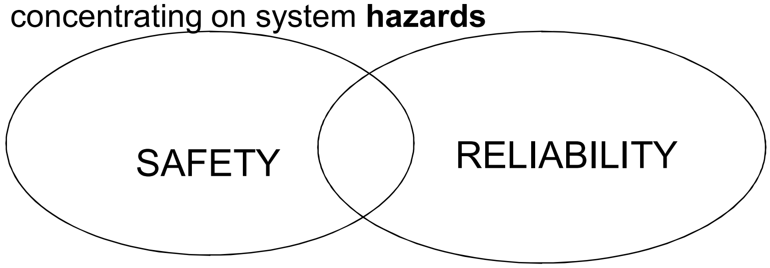
Reliability – ignition control, cruise control, fuel gauge, odometer, etc.

Safety – air bag, seat belts control, anti-lock brakes, etc.

Security – door locks, alarms, etc.

System Safety: Dependability

- **Reliability** involves **bottom-up** activities focusing on system **failures**
- **Safety** involves **top-down** approach concentrating on system **hazards**



fail-safe state defined
(reliability is secondary)

no safety analysis
(reliability assessment only)

System Safety: Basic Terms

- **Safety** is a characteristic of a system ensuring that it will not endanger human life, property or environment
- **Safety-critical software system** is a software intensive system involved in assuring that safety of equipment or plant it is interfacing with is not compromised
- Software Safety is achieved by implementing **features** and **procedures** ensuring that a product performs predictably under normal and abnormal conditions so the likelihood of unplanned events is minimized and their consequences are controlled and contained

System Safety: Basic Terms

- **Hazard** is the capability of the system to harm the people, destroy the property or environment
- Nature of the hazard defines the way how it works and how it can be controlled *(radiation, electric shock, mechanical break)*
- The hazard is a **potential** danger to do harm during the system operation
- The **actual** occurrences of hazards are **incidents** and **accidents**

System Safety: Basic Terms

- The role of safety features and procedures is to ensure safety preventing incidents and accidents
- **Incident** is an occurrence of a situation that could result in a severe consequences (in terms of loss of life or property) but it was prevented or the situation was kept under control
- **Accident** is an unplanned event or series of events that results in death, injury, environmental or material damage
- Both **incidents** and **accidents** are exemplifying **safety violation**

IFAC Summer School in Prague 2005
Control, Computing and Communication

System Safety: Basic Terms

- **Severity** of hazard describes consequences of potential accident (in terms of the human lives or monetary value)
- **Likelihood** of a hazard defines how often can we expect the hazard to occur (in terms of how many times per time unit)
- **Risk** is the combined measure of **severity** and **likelihood** of a hazard – *likelihood of hazard leading to an accident (combined with hazard severity)*

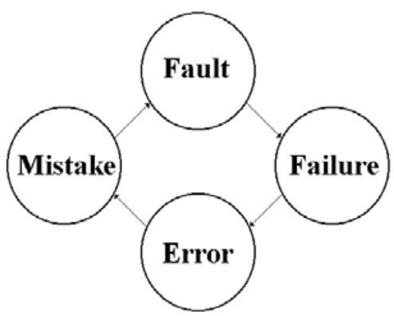
Prague, Czech Republic, June 27th – July 1st, 2005
© 2005 by Andrew Kornecki and Janusz Zalewski

System Safety: Basic Terms

- **Mistakes** are made by people (specification, design, coding, manufacturing, etc.)
- **Fault** is an internal defect within hardware or software caused by a **mistake**, component imperfection, or external disturbance (or inability of a function to perform a required action)
- **Failure** is an external view of the system, showing its inability to perform required functions
- **Error** is the difference between computer (observed, measured) value and the true (specified or theoretically correct) value (it’s a manifestation of a **failure**)

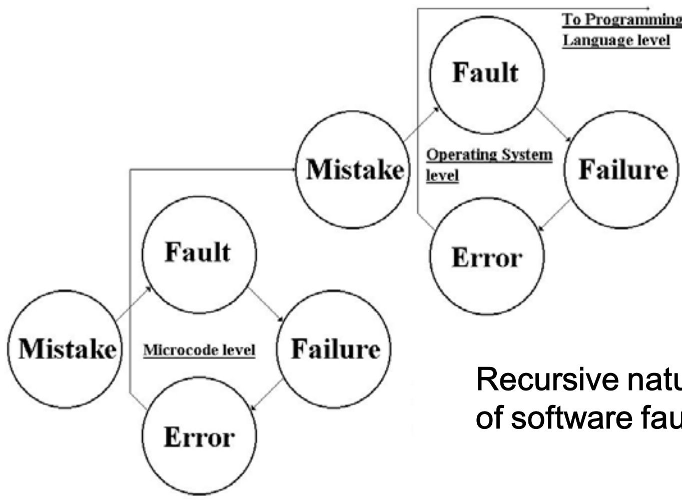
IFAC Summer School in Prague 2005
Control, Computing and Communication

System Safety: Basic Terms



Fault propagation cycle

System Safety: Basic Terms



Recursive nature of software faults

IFAC Summer School in Prague 2005
Control, Computing and Communication

System Safety: Basic Terms

- **Mistakes** made by people (*"errare humanum est"*) are the primary reasons that "something went wrong"
- **Failure** is when the system fails to perform its required function in the operational phase
- Failure can be caused by:
  - **user** makes ***how-to-use*** **mistake**
  - **fault** (or **defect**) within the **hardware**
  - **fault** (or **bug**) within the **software**

NOTE: Keep in mind that safety may be compromised when no failure occurs.

Prague, Czech Republic, June 27th – July 1st, 2005
© 2005 by Andrew Kornecki and Janusz Zalewski

System Safety: What went wrong?

- **How-to-use mistake** is more likely to happen because:
  - of **user mistake** during analysis or training
  - the **product** is **imperfect** (too complex, difficult to use, poor diagnostics)
  - there is a **fault within the software**
- **Software bug** (or imperfect product) is due to the **developer's mistake**
- **Hardware defect** is due to:
  - **designer's mistake**
  - **manufacturer's mistake**

System Safety: What went wrong?

- Environmental and operating conditions (disabling interrupts may lead to failing an interrupt driven safety critical function)
- Logic control by Real Time Executive (order of processing may impact the failure conditions)
- System function calls (detailed understanding their operations and side-effects is critical)
- System resources (implicit use of memory in stack ops)
- Timing (deadlines, jitter, or drift may prove dangerous)
- Software architecture (choice of representation may impact safety)

System Safety: What went wrong?

- Most of development methods do not provide guarantee that timing constraints be met, thus verification of timing requirements is carried out after writing the code
- Such approach can be costly because of late fault detection and need to re-write the code for speed
- Defining timing requirements can be ambiguous, thus notations allowing formally analyze or animate the system model are recommended
- Safety requirements must be traceable through the progression of the product artifacts *(requirements => design => code => operation)*

IFAC Summer School in Prague 2005
Control, Computing and Communication

System Safety: Techniques

Safety Techniques basically fall into two broad categories:

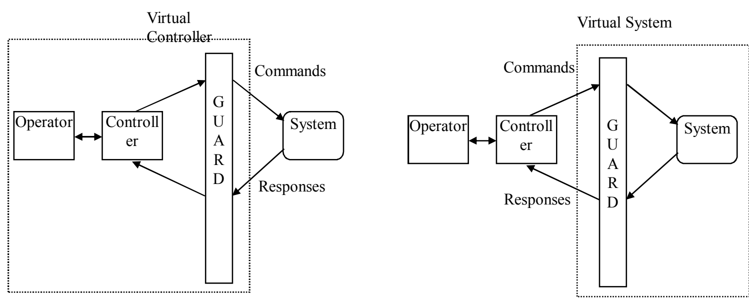
- Design Techniques (to improve the *product*)
- Process Techniques (to improve the *process*)

System Safety: Design Techniques

Major architectural safety techniques:

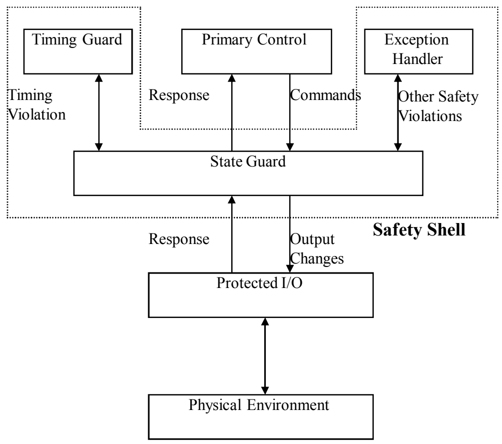
- **redundancy** - using multiple components to carry the same task
- **diversity** - two components (channels, systems) to carry the same task are based on different technologies

System Safety: Design Techniques



Principle of a Safety Shell

System Safety: Design Techniques



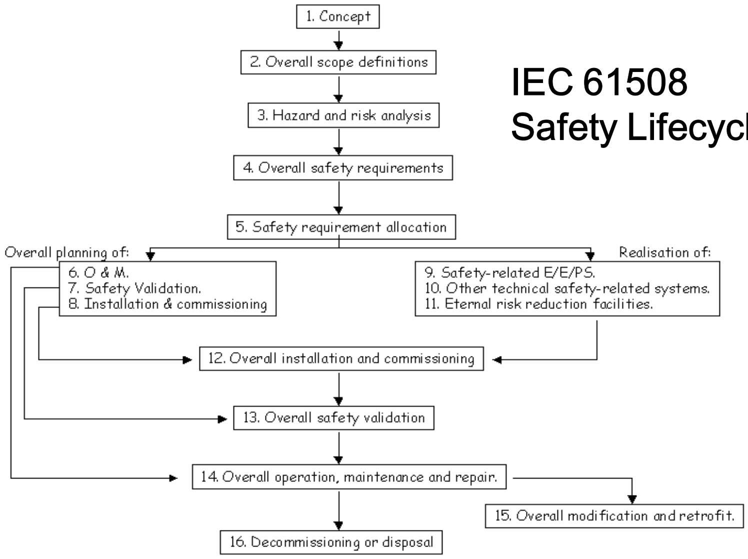
Guards Incorporated into a Safety Shell

System Safety: Process Techniques

Safety Lifecycle according to IEC 61508:

- IEC 61508 is a standard for the life-cycle management of Instrumented Protection Systems – it formalizes a risk-based approach to establishing target Safety Integrity Levels (SIL) and assessing if systems meet these targets
- **IEC 61508**: *"The necessary activities involving safety-related systems, occurring during a period of time that starts at the concept phase of a project and finishes when any safety-related systems are no longer available for use"*

System Safety: Process Techniques



IEC 61508
Safety Lifecycle

System Safety: Process Techniques

- Hazard Operability Analysis (HAZOP)
- Failure Mode and Effect Analysis (FMEA)
- Failure Mode and Effect Criticality Analysis (FMECA)
- Fault Tree Analysis (FTA)
- Event Tree Analysis (ETA)
- Common Mode Failure Analysis (CMF)
- Cause Consequence Diagrams (CCD)
- Petri Nets

Real-Time Programming

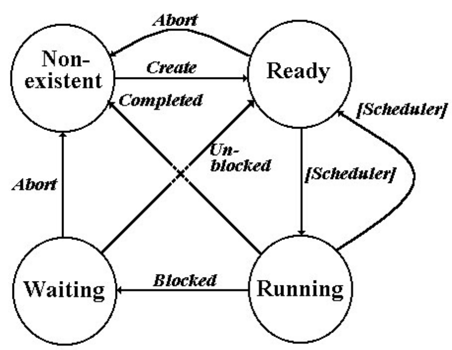
Basic Concepts

- Concurrency and basic program properties
- Programming language features
- Real-Time kernel features
- Timing issues in real-time programs
- Practical aspects of real-time scheduling
- Board Support Packages & Device Drivers

Real-Time Programming: Concurrency

- **TASK -** a unit of concurrency executing sequentially itself, designed to fulfill a specific system function**,** typically defined by:
  - ***event*** *- environmental or internal stimulus occurring at a time point requiring response*
  - ***activity*** *- a set of operations responding to the event requiring time*
- Task can be implemented as:
  - **PROCESS -** a virtual computing environment set up to run as an application program (contains its own data, code, context, & resources)
  - **THREAD -** a sequence of instructions executed within the context of a process

Real-Time Programming: Concurrency



States of Concurrent Tasks

Real-Time Programming: Concurrency

Concurrency Terms

- Synchronization
- Critical Section
- Mutual Exclusion
- Reentrancy
- Deadlock
- Pre-emption
- Safety and Liveness
- Scheduling

Real-Time Programming: Concurrency

```
task body SLICER is
   ME : ID := GET_ID;
begin
   PUT_LINE("Task " & ID'IMAGE(ME) & " up & running");
   loop
      -- delay 0.0;
      COUNTS(ME) := COUNTS(ME) + 1;
   end loop;
exception
   when others =>
      PUT_LINE("Exception in slicer " & ID'IMAGE(ME));
end SLICER;
```

10-task example of concurrent execution

IFAC Summer School in Prague 2005
Control, Computing and Communication

Real-Time Programming: Concurrency

10-task Slicing Example (null Main)

Case 1:
delay 0.0 -- inactive

```
Task 1 up & running
```

Case 2:
delay 0.0 -- active

```
Task 1 up & running
Task 2 up & running
...
Task 10 up & running
```

Real-Time Programming: Concurrency

Case 3:
delay 0.0 -- inactive
Exceptions active

```
Task 1 up & running
Exception in task 1
...
Task 10 up & runing
Exception in task 10
```

Case 4:
delay 0.0 -- active
Exceptions active

```
Task 1 up & running
...
Task 10 up & running
Exception in task 1
...
Exception in task 10
```


Real-Time Programming: Concurrency

<u>10-task Slicing Example (Main killing children)</u>

Case 5:
delay 0.0 -- inactive
prio Main < prio Tasks'

```
Task 1 up & running
```

Case 6:
delay 0.0 -- inactive
prio Main > prio Tasks'

```
Main waiting 10 sec
Task 1 up & running
Main killing off ch.
Task 1 count is 16M
Task 2 count is 0
...
Task 10 count is 0
```

Real-Time Programming: Concurrency

Case 7:
delay 0.0 -- active
prio Main > prio Tasks'

```
Main waiting 10 sec
Task 1 up & running
...
Task 10 up & running
Main killing off ch.
Task 1 count is 78K
Task 2 count is 78K
...
Task 10 count is 78K
```

Case 8:
delay 0.0 -- active
prio Main < prio Tasks'

```
Task 1 up & running
```


Real-Time Programming: Languages

- An informal scan of the real-time (embedded, dedicated, safety-critical) market reveals:
  - 30% assembly and legacy languages
  - 30% Ada
  - 30% C/C++
  - 10% other (100+ other languages)
- C and Ada are the most commonly used languages in civil aviation today
- C++ is gaining popularity, but its usage is still limited

Real-Time Programming: Languages

FEATURE	Ada	C/C++	Java
Memory Management	automatic	manual	garbage collected
Run-Time Efficiency	high	high	medium
Run-Time Predictability	high*	OS dependent	low
Concurrency Control	language features	OS specific	language library

IFAC Summer School in Prague 2005
Control, Computing and Communication

Real-Time Programming: Languages

Language Safety Features:

- Formally defined syntax; Block structure
- Strong typing; Wild (unstructured) jumps
- Memory overwrites; Memory exhaustion
- Dangling pointers; Variable initialization
- Model of floating-point arithmetic
- Exception handling; Reentrancy
- Separate compilation with cross-checking
- Temporal predictability of loops

Efforts include: SPARK, MISRA-C, PEARL.

Real-Time Programming: Languages

Strong Typing – strict application od data type checking rules to prfevent misuse of variables and data

```
int a:
float x;        // and x are of different types
a = x;          // formally, this is incorrect,
                // but C/C++ may allow it
a = (int)x;     // mode appropriate coding
```

Fortran allows implicit declarations
C/C++ and Java allow implicit type casting
Ada requires explicit type casting

Real-Time Programming: Languages

Exception – an unexpected situation that may cause a program to crash.

Examples: division by 0, overflow, reference to nonexisting object (memory, device), I/O error, etc.

Exception handling – to provide facilities within a language to neutralize consequences of exceptions.

Real-Time Programming: Languages

Usual sequence of actions in exception handling:

1. Exception handler included in a program.
2. Exception raised during program execution.
3. Control is transferred to exception handler.
4. Handler executes and exits to the surrounding block.

```
x := a/b;    -- What if b=0?
…
exception
   when CONSTRAINT_ERROR => x=MaxInt;
end;
```


Real-Time Programming: Languages

Unstructured jump (wild jump) – a program jump which is not controlled by the programmer.

Unstructured jumps are most likely to occur in case statement or their equivalents.
Examples include:
- incomplete coverage of cases (missing default/others)
- erroneous exit from cases (missing break)
- incomplete if/else pairs.

Real-Time Programming: Languages

C/C++ pair setjpm() and longjmp():

int setjump(jmp_buf env)

// Saves state info in env for use by longjmp

void longjmp(jmp_buf env, int v)

// Restores the state saved by setjmp

IFAC Summer School in Prague 2005
Control, Computing and Communication

Real-Time Programming: Languages

Memory overwrite – an uncontrolled access to arbitrary memory locaations.

May be cause by: erroneous pointers, out-of-bounds array indexes, dynamic allocation.

The programmer must remember that a pointer is not just an address; it is an address of a data item of a certain type.

Real-Time Programming: Languages

```
// Check memory space available
int * arr , j=0;
for ( ;  ; ) {
    j++;
    arr = (int *)malloc(TEN_K);
    printf("%d ", j);
}
```


Real-Time Programming: Languages

<u>Reentrancy</u> – subprogram property that allows it to be executed by multiple callers at the same time.

Need for reentrancy is typical in multithreaded programs. Therefore library routines are usually indicated MT-safe or not.

Real-Time Programming: Languages

Other language aspects:

- variable initialization
- order of evaluation vs. operator precedence
- spawning processes via fork
- killing concurrent units.

IFAC Summer School in Prague 2005
Control, Computing and Communication

Real-Time Programming: Languages

```
// Both should be avoided
x = i++ + a[i];
x = (i++) + a[i];

// What is the result, and why?
int i = 0;
i = i+++i;
```

Real-Time Programming: Languages

```
// When if has both branches
// executed simultaneously!
if (fork()) {
    ... // some code
}
else {
    ... // other code
}
```

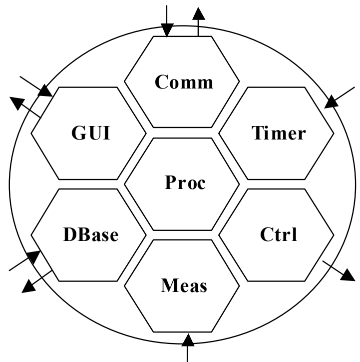

IFAC Summer School in Prague 2005
Control, Computing and Communication

Real-Time Programming: Languages

Beware of problems with
destroying concurrent units:
- Ada tasks via abort
- Aunix processes via kill()
- threads vi acancellation.

Prague, Czech Republic, June 27th – July 1st, 2005
© 2005 by Andrew Kornecki and Janusz Zalewski

Real-Time Programming: Languages



Observation: Why Java, as the first programming language in common use, included GUI and Networking as part of the language? Do LabVIEW and MATLAB show similar trend?

Real-Time Programming: RT Kernel

Concept of RTOS/Kernel Operation:

- Strong distinction between internal system operations and the user tasks
- RTOS kernel does not participate in the priority scheme - it operates in the **hardware context**
- Peripheral interrupts handled by extensions to the kernel (device drivers) which also function outside normal application task prioritization

Real-Time Programming: RT Kernel

Concept of RTOS/Kernel Operation:

- User tasks communicate with the kernel and perform most I/O through entry points or calls into the drivers - I/O is processed outside the user application context
- Modern RTOS uses threaded micro-kernel with fast response and options for handling interrupts at the system priority level

Real-Time Programming: RT Kernel

<u>Basic Terminology</u>

- **EVENT** – a result of an externally or internally generated occurrence handled by the processor
- **LATENCY** - time required to recognize and start responding to an event
- **RESPONSE TIME** - time interval between presentation of an input *(stimulus)* and the appearance of the associated output *(response)*
- **DEADLINE** - a time point before which a specific event must occur (e.g. the task must complete the execution)

Real-Time Programming: RT Kernel

Basic Terminology

- **INTERRUPT LATENCY** – the time interval between the occurrence of an external event and the start of the first instruction of the interrupt service routine
- **INTERRUPT LATENCY INVOLVES**: hardware logic processing, interrupt disable time, handling higher hardware priority interrupts, switching to handler code (saves, etc.)

IFAC Summer School in Prague 2005
Control, Computing and Communication

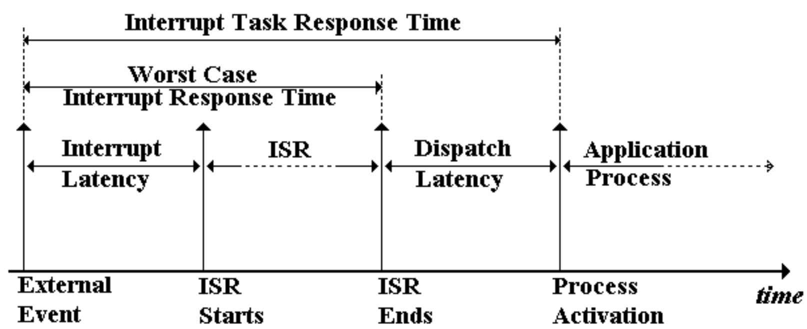
Real-Time Programming: RT Kernel

Basic Terminology

- **DISPATCH LATENCY** – the time interval between the end of interrupt handler code and the first instruction of the process activated (made runnable) by this interrupt.
- **DISPATCH LATENCY INVOLVES**: OS decision time to reschedule (non-preemptive kernel state), context switch time, return from system call.

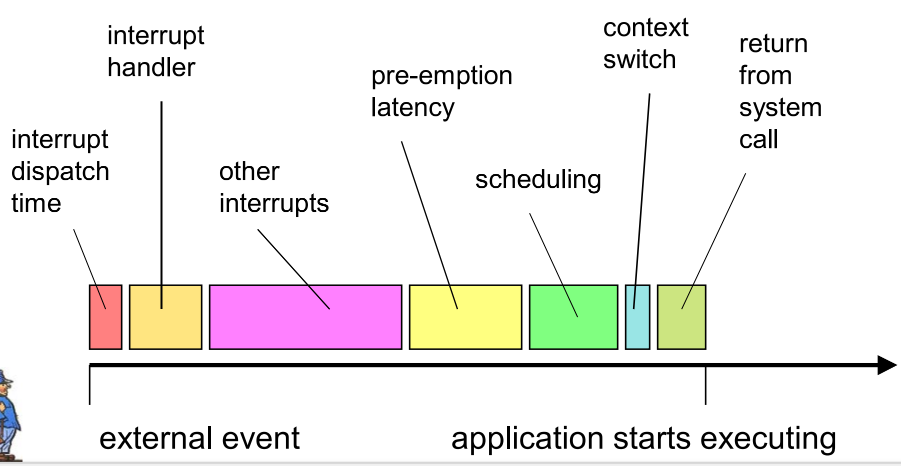
Prague, Czech Republic, June 27th – July 1st, 2005
© 2005 by Andrew Kornecki and Janusz Zalewski

Real-Time Programming: RT Kernel



Real-Time Programming: RT Kernel

Contributions to Interrupt Task Response Time



Real-Time Programming: RT Kernel

Kernel Responsiveness Involves:

- **INTERRUPT LATENCY**
- **TASK DISPATCH LATENCY**
- **(WORST CASE) INTERRUPT RESPONSE TIME**
  (Interrupt Latency + Worst case Execution of the Interrupt Handler + Interrupt Exit Overhead)
- **INTERRUPT TASK RESPONSE TIME**
  (Interrupt Response Time + Dispatch Latency)

Real-Time Programming: RT Kernel

Schedulability and Determinism

- **SCHEDULABILITY** - a property of a set of tasks ensuring that all tasks will meet their respective deadlines
- **PREDICTABILITY** - the property of meeting the temporal determinism criteria
- **TEMPORAL DETERMINISM -** the situation in which timing properties of the system are known (or bounded) for each set of inputs

Real-Time Programming: RT Kernel

Topics important but not covered here:

- **Real-Time Scheduling**

  „What Every Engineer Needs to Know about Rate-Monotonic Scheduling"

  IN: Advanced Multimicroprocessor Bus Architectures, IEEE Computer Society Press, 1995, pp. 321-335,

  and Real-Time Magazine, Issue 1/95, pp. 6-24
- **Device Drivers**

  „Teaching Device Drivers Technology in a Real-Time Systems Curriculum"

  IN: Real-Time Systems Education III, IEEE Computer Society Press, 1999, pp. 42-48

  and at http://www.wrs.com/univ/html/featurevol4.html

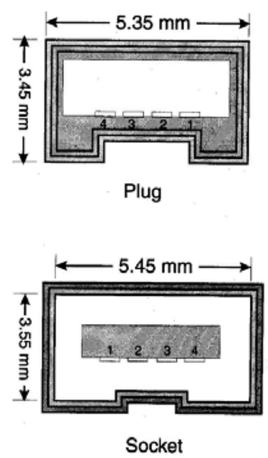
IFAC Summer School in Prague 2005
Control, Computing and Communication

Databus Safety

Databus Characteristics

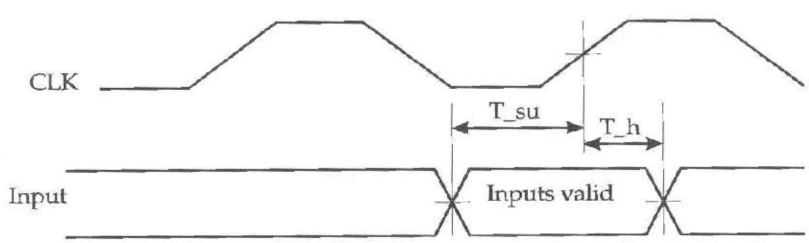
- **Mechanical** properties concern bus wiring, connectors, their pinout, and module design and dimensions
- **Electrical** (or optical) properties are related to signal levels and their dynamics to carry information, including electromagnetic characteristics
- **Logical** properties concern the protocol of exchanging information over a bus

Databus Safety



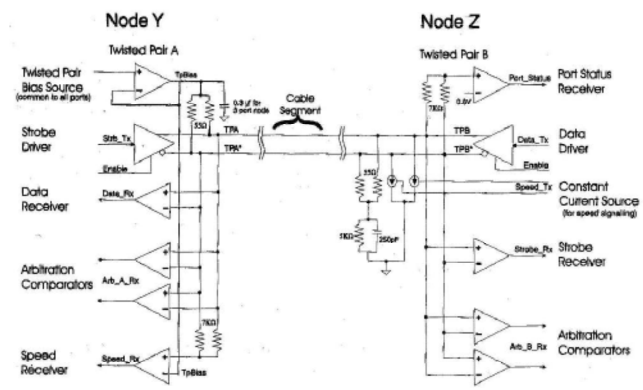
Example of mechanical properties of the connector for FireWire bus.

Databus Safety



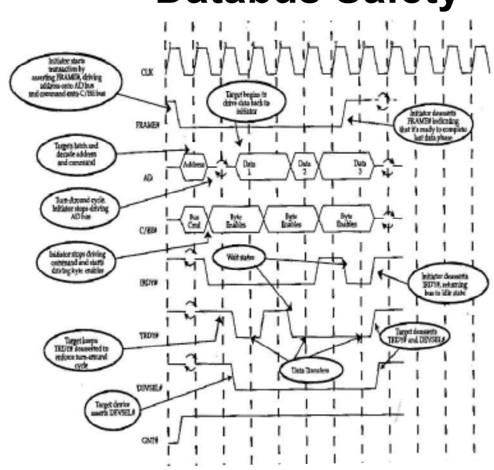
Example of electrical properties of the PCI bus input signals (T_su – setup time, 7-12 ns; T_h – hold time).

Databus Safety



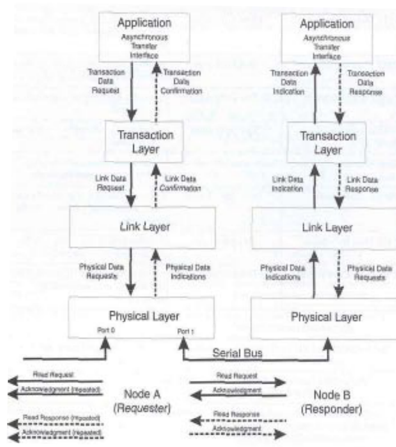
Electrical interface between two FireWire nodes.

Databus Safety



Example of electrical properties and low-level bus protocol for the PCI bus Read Transaction.

Databus Safety



Example of logical properties of the bus for FireWire Asynchronous READ Transaction.

Databus Safety

Specifics of the Bus Protocol:

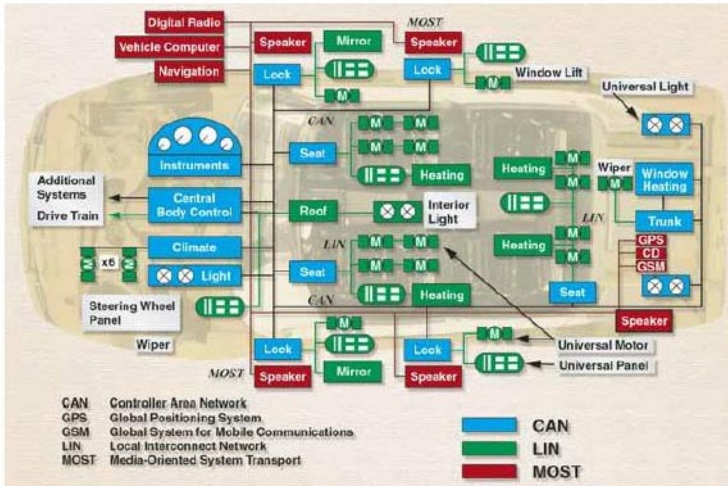
- ***Bus arbitration***

  competing for bus access
- ***Data transfer***

  how devices exchange data once they obtain bus access
- ***Fault handling***

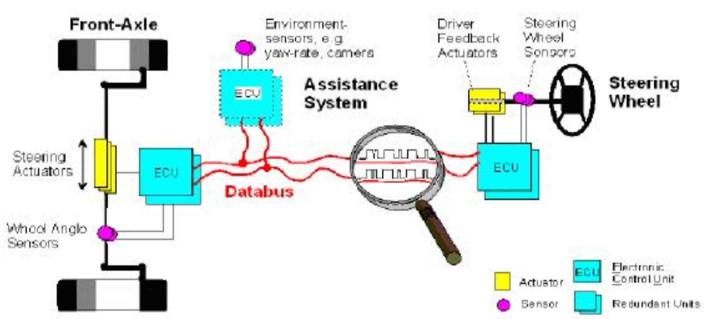
  dealing with bus errors

Databus Safety



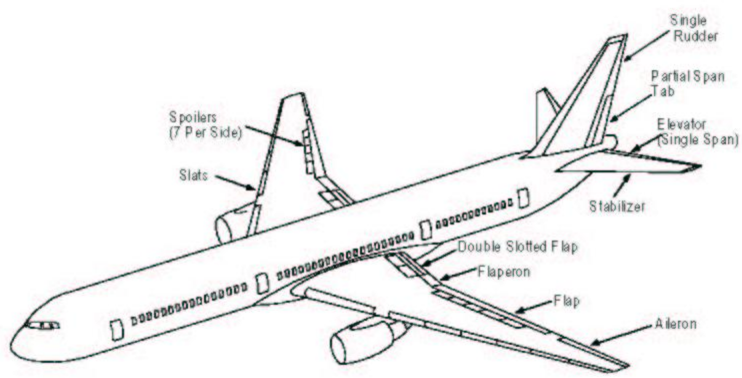
An Example of Modern Vehicle Network (Leen 2002)

Databus Safety



Steer-by-Wire System (Waern 2003)

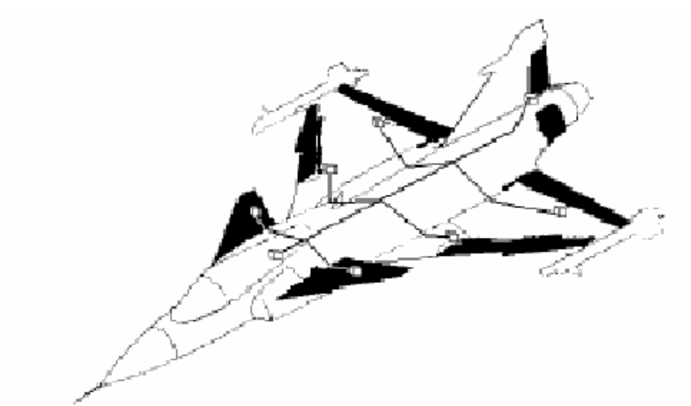
Databus Safety



Distributed Flight Control System for Boeing 777 Aircraft

IFAC Summer School in Prague 2005
Control, Computing and Communication

Databus Safety



Distributed Flight Control System for JAS 39 Gripen Aircraft (Johansson 2003)

Databus Safety

Databus	Type	Architecture	Medium	Rate	Encoding
Arinc 429	serial unidir.	single master	2 wires	100kb/s	RTZ bipolar
MIL1553	serial bi-dir.	single master	twist pairs	1 Mb/s	biphase Manch.
Arinc 629	serial bi-dir.	multi master	twist pairs	2 Mb/s	Manchester II
Arinc 659	serial bi-dir.	quad redund	twist pairs	30MHz	biphase Manch.
FlexRay	serial bi-dir.	fault-tolerant	optic/wire	10Mb/s	undefined
CAN	serial bi-dir.	multi-master	twist pairs	1 Mb/s	NRZ + bit stuff
TTP/C	serial bi-dir.	dbl redund	twist pairs	25Mb/s	MFM
IEEE1394	serial	d-chain/tree	twist pairs	400Mb/s	LVDS
Safe-Wire	serial bi-dir.	master-slave	twist pairs	200 kb/s	3-level
SpaceWire	serial bi-dir.	master-slave	2 wires	> 2Mb/s	undefined

IFAC Summer School in Prague 2005
Control, Computing and Communication

Databus Safety

Risk Assessment Process

1) Multicriteria-based Safety Assessment
2) Hazard Analysis
3) Failure Mode Analysis

Prague, Czech Republic, June 27th – July 1st, 2005
© 2005 by Andrew Kornecki and Janusz Zalewski

Databus Safety

Criterion	Selected Evaluation Factors
Safety	Availability and reliability; Partitioning; Failure detection; Common cause/mode failures; Bus expansion strategy; Reconfigurability; Redundancy management
Data Integrity	Maximum error rate; Error recovery; Load analysis; Bus capacity; Security
Performance	Operating speed; Schedulability of messages; System interoperability; Bus length and max. load; Retry capability; Bandwidth; Data latency; Transmission overheads
EMC	Switching speed; Pulse rise and fall times; Wiring; Shielding effectiveness; Lightning/radiation immunity
Design Assur.	Compliance with standards (such as DO-254/DO-178B)
V&V	Examples: functionality testing, system testing, failure management, degraded mode operation
Configuration Management	Examples: change control, compliance with standards, documentation, interface control, system analysis, etc.
Continued Airworthiness	Lifetime issues, such as physical degradation, in-service modifications and repairs, impact analysis. (Rierson/Lewis, 2003)

Databus Safety

Failure Mode	Description
Invalid Messages	Messages sent across the bus Contain invalid data
Non-Responsive	An anticipated response to a message does not occur or return in time
Babbling	Communication among nodes Is blocked or interrupted by uncontrolled data Stream
Conflict of Node Adrs	More than one node has the same identification (Debouk et al. , 2003)

Databus Safety

Potential Hazard	Possible Mitigation
Loss of Power	**Dual power system (including battery, wires and connectors)**
Loss of Communicat'n	**Dual communication system**
Loss of Steering	**Backup system; Reduced functionality Redundant system; Steer by braking active safety system**
Loss of Braking	**Backup system; Reduced functionality redundant System; Brake by steering active safety system**
Loss of Electronic Throttle	**Backup system; Reduced functionality redundant system**
Loss of Actuators	**Backup actuators; Red. performance actuator**
Loss of Sensors (recording driver cmds)	**Backup sensors; Red. performance sensor** (Chau et al. , 2003)

Databus Safety

Bus Experiments

- Plain simulation for well developed databus networked configurations VME/Raceway
- Actual data transfer experiments with a modern bus FireWire
- Simulation and real experiments for routing in Bluetooth
- Improving Real-Time Characteristic of the Ethernet

Databus Safety

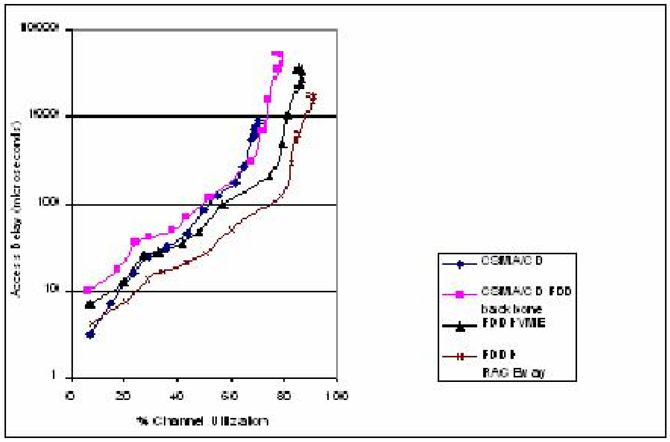
Bus Parameters

- Bus response – access delay vs. bus load
- Bus throughput - data transfer rate vs. packet size
- Interconnect formation and routing
- Predictability of packet transmission time

Databus Safety

Access delay vs. bus load:
When bus load increases,
how does it impact access delay?

Databus Safety

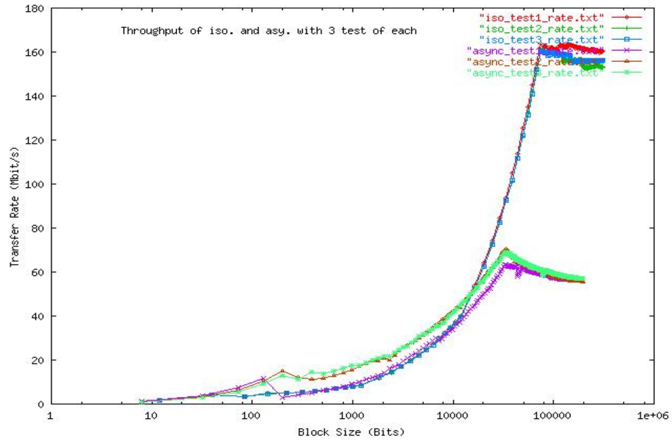


Server Access Delay for 64B Packets

Databus Safety

<u>Bus throughput:</u>
When packet size increases,
how does it impact transfer rate?

Databus Safety

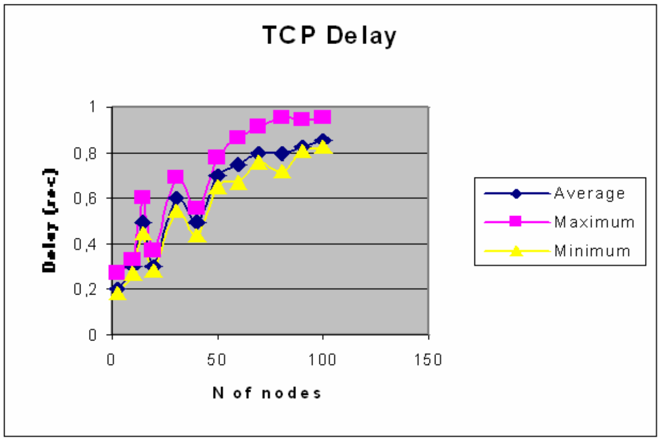


IEEE 1394 Throughput over a Raw Driver for Asynchronous and Isochronous Modes

Databus Safety

<u>Interconnect formation and routing:</u>
When nodes are being added,
how does it impact access delay and data transfer rate?

Databus Safety



Bluetooth TCP Delay for Increasing Number of Nodes

Databus Safety

<u>Deterministic Ethernet:</u>

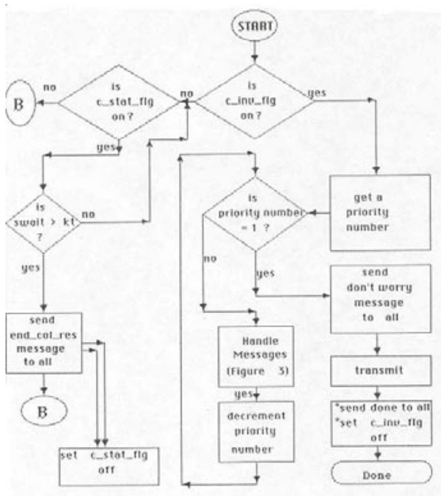
Can Ethernet be made predictable without modification of its CSMA/CD protocol?

Each node is assigned a priority and two flags:

- collision status flag, c_stat_flag (collision resolution in progress)

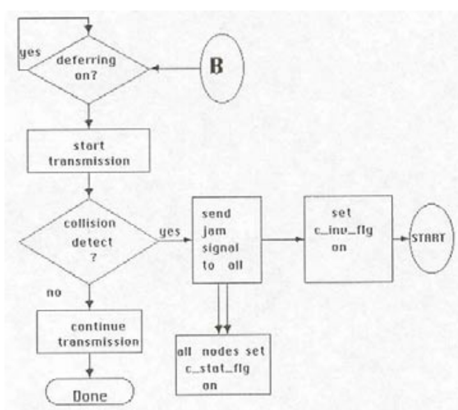
- collision involved flag, c_inv_flag (node was involved in the collision)

Databus Safety



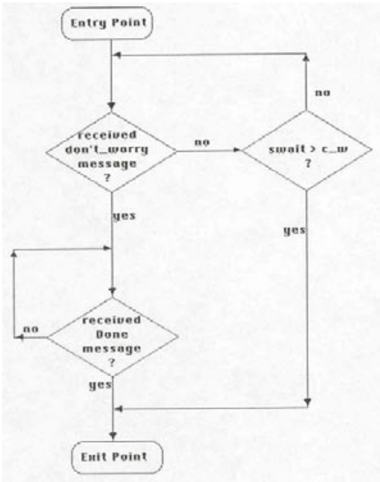
Principle of a deterministic Ethernet protocol.

Databus Safety



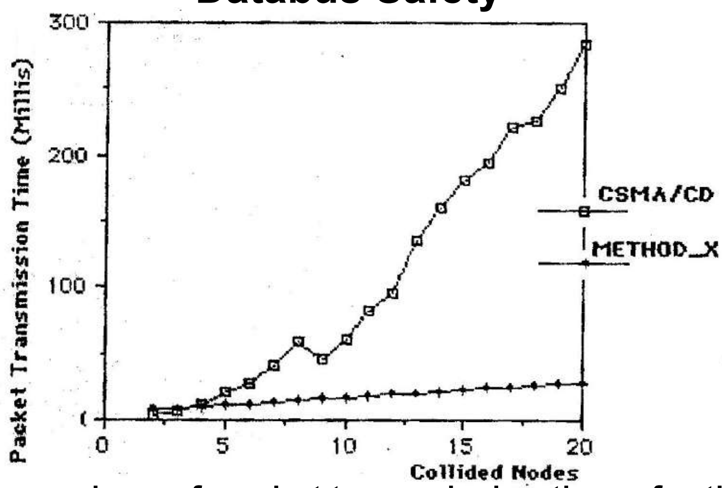
Behavior of a regular CSMA/CD node.

Databus Safety



Handling messages block of the protocol.

Databus Safety



Comparison of packet transmission times for the classic CSMA/CD and the extended protocol.

Databus Safety

Databus Safety

- **New area with ongoing research**
- **Risk assessment methods essential as a starting point**
- **Definition of critical parameters**
- **Experimentation needed**

OK! Let's go for a beer!!!