

The Perfectly Designed Chaos - or How Can We Stop the Running Amok of an Imperfect Software on a Faulty Hardware

András Pataricza, Technical University Budapest, Hungary

András Pataricza is associate professor for Computer Science at the Technical University in Budapest. His research interests are in fault-tolerant systems. In this area he cooperates with researchers at Universities in Germany and France.

January 27, 1997

INFOGEM Conference

A. Pataricza

The Perfectly Designed Chaos

or

How Can We STOP

the Amok Run of an Imperfect Software on a Faulty Hardware

András Pataricza

Technical University of Budapest

H-1502 Budapest

E-mail: pataric@mmt.bme.hu

Web: www.mmt.bme.hu/~pataric

The perfectly designed chaosThe Perfectly Designed Chaos or How Can We STOP the Amok Run of an Imperfect Software

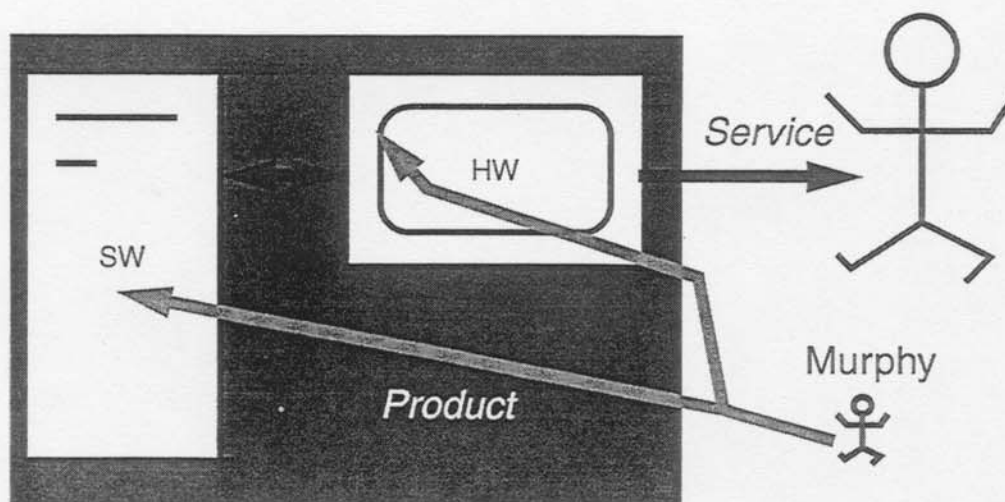
c:\users\pataric\chaos\link

January 27, 1997

INFOGEM Conference

A. Pataricza

Paradox approaches:

The designer offers: Quality of *product* (?)The user wants: Quality of *services*

The perfectly designed chaos

Paradox approaches:

2

January 27, 1997

INFOGEM Conference

A. Pataricza

Importance of Dependability:

Is the story of computer applications a SUCCESS or a HORROR story?

Some NEGATIVE experiences collected from a *very* long list:

- USA: approximately 4 billion \$/year damage/year
- Several accidents:
 - radiation therapy
 - airplane
 - Ariane 5
- Collapse of stock exchange, phone and banking systems
- Administration:
 - invoice on autopsy (vivisection?)
- Internet

The perfectly designed chaos

Importance of Dependability:

3

c:\users\peter\cultural.fmk

January 27, 1997

INFOGEM Conference

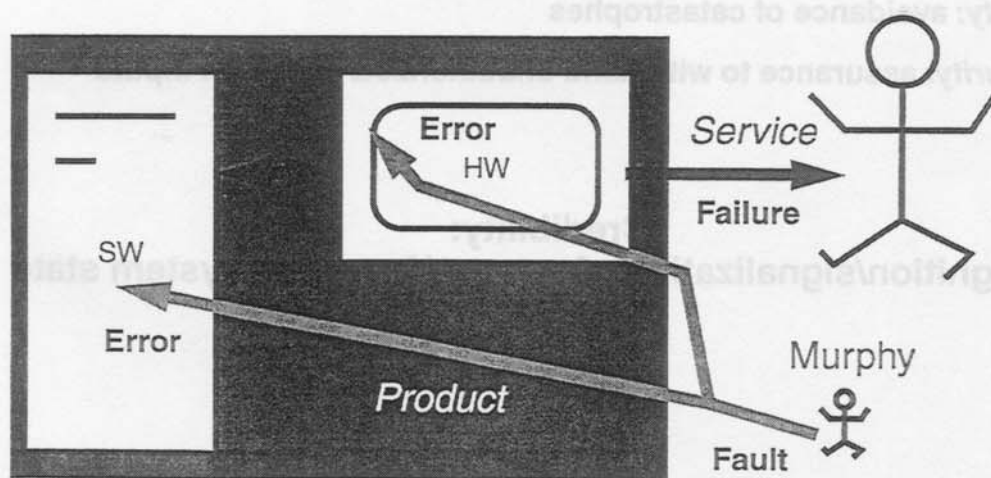
A. Pataricza

Dependability

“The *thrustworthiness* of a computer system such that *reliance* can be *justifiably* placed on the *services* it delivers”

(IFIP WG 10.4 / Laprie)

Fault \Rightarrow Error \Rightarrow Failure



The perfectly designed chaos

Dependability

4

c. *Supercalarchaeal* form

January 27, 1997

INFOGEM Conference

A. Pataricza

Means for dependability

Phase	Fault handling		Implementation	Target
Before service	Prevention (= no faults, please)	Avoidance	Design & manufacturing technology	Fault
		Removal	Testing	
During service	Robustness (=Russian roulette)		Self-confidence	?
	Tolerance (= proper service in spite of faults)		Redundancy - time - performance - hardware - information	Failure
	Forecasting (= be ready for faults)		- data acquisition prediction	Fault

The perfectly designed chaos

Means for dependability

5

c:\users\pataric\harcal.mak

January 27, 1997

INFOGEM Conference

A. Pataricza

Dependability attributes

IEC 1069-5

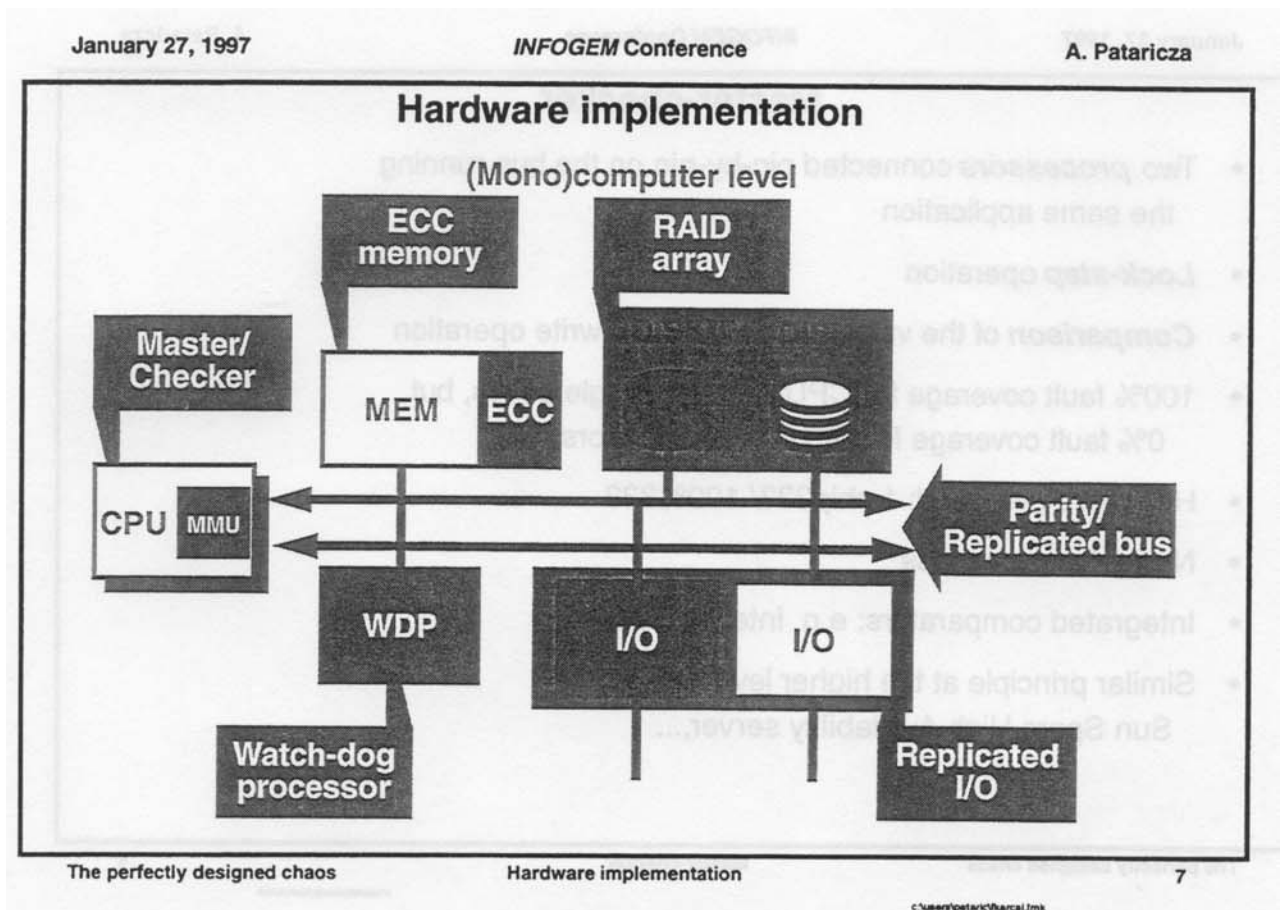
- **Reliability:** correct operation for a defined period of time
- **Availability:** ability in a correct operational state for a defined period of time
- **Safety:** avoidance of catastrophes
- **Security:** assurance to withstand unauthorized/ incorrect inputs

Credibility:
Recognition/signalization of correct/ incorrect system state

The perfectly designed chaos

Dependability attributes

6



January 27, 1997 INFOGEM Conference A. Pataricza

Standard solutions

CPU-MMU

- Illegal opcode
- Arithmetic errors
- Bus error
- Memory address range access rights (fetch, read, read/write) checks only task level \Rightarrow no fine granular checks (the entire address range of a task is uniformly mapped)
- Similar object/function oriented implementation, like index checks not (really) supported (need for user - supervisor - user mode context switching + MMU prg.)
- Fault tolerance/ latency?
- Supervisor mode unprotected

The perfectly designed chaos Standard solutions 8

January 27, 1997

INFOGEM Conference

A. Pataricza

Master-checker

- Two **processors** connected pin-by-pin on the bus running the same application
- **Lock-step** operation
- **Comparison** of the values at each signal write operation
- 100% fault coverage for CPU-**internal** single errors, but 0% fault coverage for CPU-**external** errors
- Hardware overhead: 1 chip???/ 100%???
- No performance loss
- Integrated comparators: e.g. Intel 585.995
- Similar principle at the higher levels of HW:
Sun Sparc High Availability server,...

The perfectly designed chaos

Master-checker

9

c:\users\pataric\local\mk

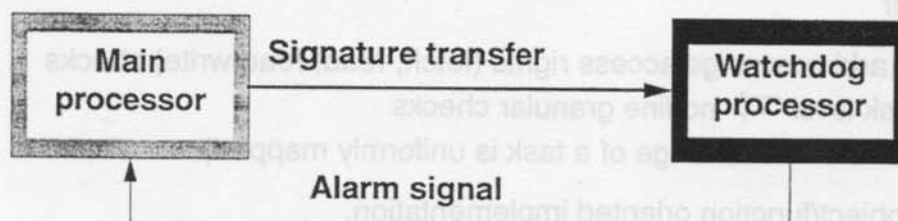
January 27, 1997

INFOGEM Conference

A. Pataricza

Watch-dog processor

- Simple co-processor checking the **control-flow** of the application program
- Signatures (**numerical labels**) assigned to the (high-level) **instructions** at precompile time
- Signatures **transferred** to the WDP at run time

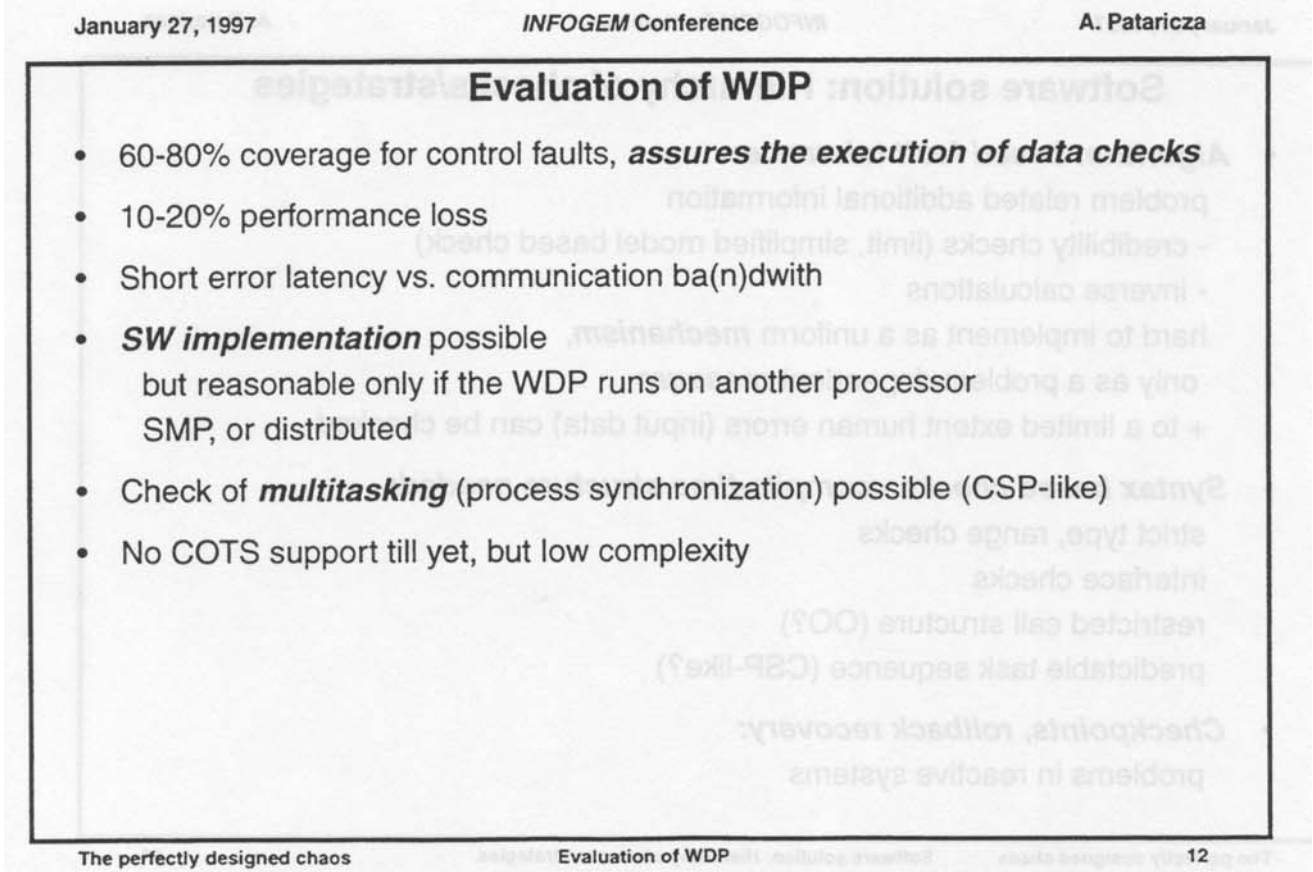
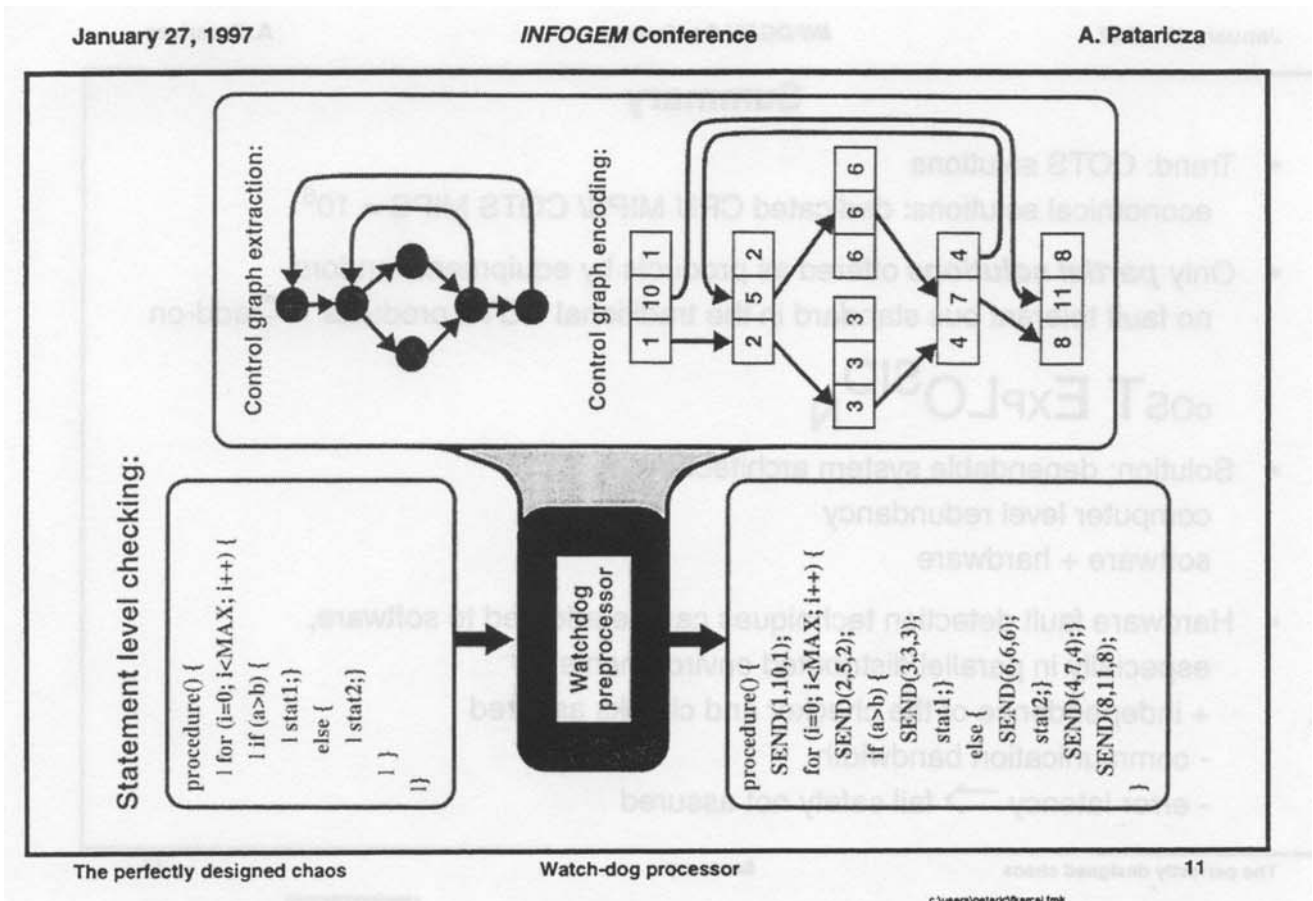


- Syntactic check of the label sequence but: no data dependencies
E.g.: IF-THEN-ELSE:
both THEN branch and ELSE branch accepted
selection unchecked

The perfectly designed chaos

Watch-dog processor

10



January 27, 1997

INFOGEM Conference

A. Pataricza

Summary

- Trend: COTS solutions
economical solutions: dedicated CPU MIPS/ COTS MIPS = 10^6
- Only **partial solutions** offered as products by equipment vendors
no fault tolerant bus standard in the traditional COTS products \Rightarrow add-on

cOST EXPLOSION

- Solution: dependable system architecture
computer level redundancy
software + hardware
- Hardware fault detection techniques can be adopted to software,
especially in parallel/distributed environments:
 - + independence of the checker and checks assured
 - communication bandwidth
 - error latency \Rightarrow fail safety not assured

The perfectly designed chaos

Summary

13

January 27, 1997

INFOGEM Conference

A. Pataricza

Software solution: Hierarchy of checks/strategies

- **Algorithm based fault tolerance:**
problem related additional information
 - credibility checks (limit, simplified model based check)
 - inverse calculations
 hard to implement as a uniform **mechanism**,
only as a problem dependent **measure**
+ to a limited extent human errors (input data) can be checked
- **Syntax based checks - compile-time structure needed:**
strict type, range checks
interface checks
restricted call structure (OO?)
predictable task sequence (CSP-like?)
- **Checkpoints, rollback recovery:**
problems in reactive systems

The perfectly designed chaos

Software solution: Hierarchy of checks/strategies

14

January 27, 1997

INFOGEM Conference

A. Pataricza

- **Redundant data structures:**
 - double linking of lists
 - checksum-like protection of operations and data structures
 - ("total" in financial tables, parity-like protected matrix operations)
- **Fault tolerant elementary operations:**
 - atomic transaction processing
 - fault tolerant commit protocols
- Uniform interface: **exception handling**
- **Validation:**
 - huge cardinality of the candidate faults \Rightarrow statistical methods
 - simulated fault injection (radiation, bus signal or software)
 - no validated fault model on the effect of the transient faults

The perfectly designed chaos

Software solution: Hierarchy of checks/strategies

15

c:\users\pataric\chaos\fmk

January 27, 1997

INFOGEM Conference

A. Pataricza

Open problems

- Predictable dependability requires (nearly) **deterministic** control flow
- Hard to solve problems:
 - dependable real-time systems (**responsive** systems)
 - distributed, reactive applications
- **Implementation** techniques
 - performance \Leftrightarrow determinism (load depending task migration)
 - run-time determined control flow (pointer programming)
 - self-modification (LISP)
- Checking of the **completeness** of the checks and reactions
 - do all failures of every volatile operations have an exception handler?
- **Human** intelligence/unintelligence
 - security
 - operator errors

The perfectly designed chaos

Open problems

16

January 27, 1997

INFOGEM Conference

A. Pataricza

CAD, CASE and all other CA

Basic idea:

any formal method can be used for **dependability modelling**, if

- a/ the rough **structure** remains **unaltered**
- b/ the description of the **elements** is **extended** by

- **local effects of faults**
- **effects of erroneous input data on the state and output of the elements**

Typical formalisms used in CASE/ HW-SW Co-Design

- **Data-flow notation** (activity charts)
- **Finite state machine** (state diagrams / statecharts)
- Function-structure correlation
 - CASE: Classes, inheritance
 - HW Block diagram

The perfectly designed chaos

CAD, CASE and all other CA

17

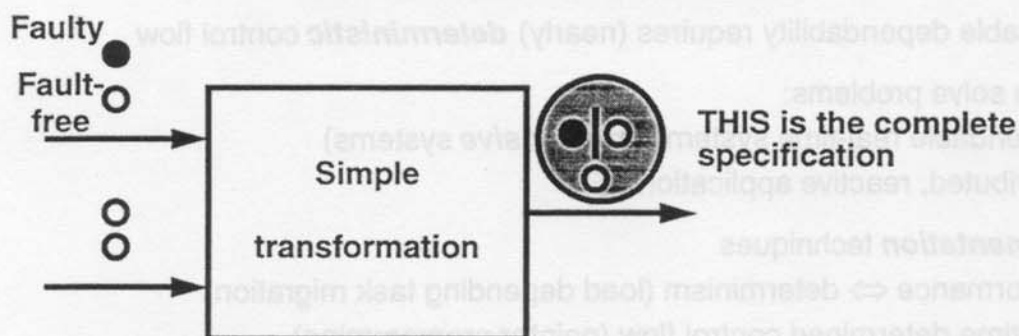
c:\user\pataric\patacal.tmk

January 27, 1997

INFOGEM Conference

A. Pataricza

Example: Data-flow notation



- Qualitative data classes:
 - simplest: GOOD/ FAULTY
 - function: GOOD/ DATA ERROR/CTRL ERROR/
 - severity of the faults: GOOD/ FAULTY/ CATASTROPHIC
 - etc.
- Model simplification: non-deterministic behavior

The perfectly designed chaos

Example: Data-flow notation

18

Conclusion:

Quality Control assures, that it works fine, if it works

**Dependability assures, that it works not so very fine, but a little bit
if it does not work**

Can we really stop a high-tech car with a defective ABS on the ice???

Let be a depressed schizoid, in order to sleep well!