TECHNISCHE UNIVERSITÄT WIEN

TTTech

Dependable Computer Systems

Part 2: Basic Concepts and Taxonomy

Contents

- The Basic Concepts
- The Threats to Dependability and Security (Fault – Error – Failure)
- The Means to Attain Dependability
- Error Recovery and Redundancy
- On the Importance of the Specification

The Basic Concepts

Dependability Definitions

Original: Dependability is the ability to deliver service that can justifiably be trusted.

Alternate: Dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable.

System Function, Behavior, Structure, and Service

A **system** is an entity that interacts with other entities, i.e., other systems.

For a particular system A, the sum of all the other systems system A is interacting with is referred to as the **environment** of system A.

The **system boundary** is the common frontier between a system and its environment.

System Function, Behavior, Structure, and Service (cont.)

The **function** of a system is what the system is intended to do.

The function is described in the **functional specification**.

The **behavior** of a system is what the system does to implement its function and is described by a sequence of states.

System Function, Behavior, Structure, and Service (cont.)

The **structure** of a system is what enables it to generate the behavior.

In terms of a structure, a system is composed of **components** bound together to interact.

Components are systems which can be *composed of other components*.

Alternatively, a component is said to be **atomic**, in case the inner structure of the component is of no interest.

System Function, Behavior, Structure, and Service (cont.)

A system is the **provider** of a **service** to one or many **users**.

Users are, again, systems.

The **service interface** between the provider and the one or many users is the respective part of the provider's system boundary.

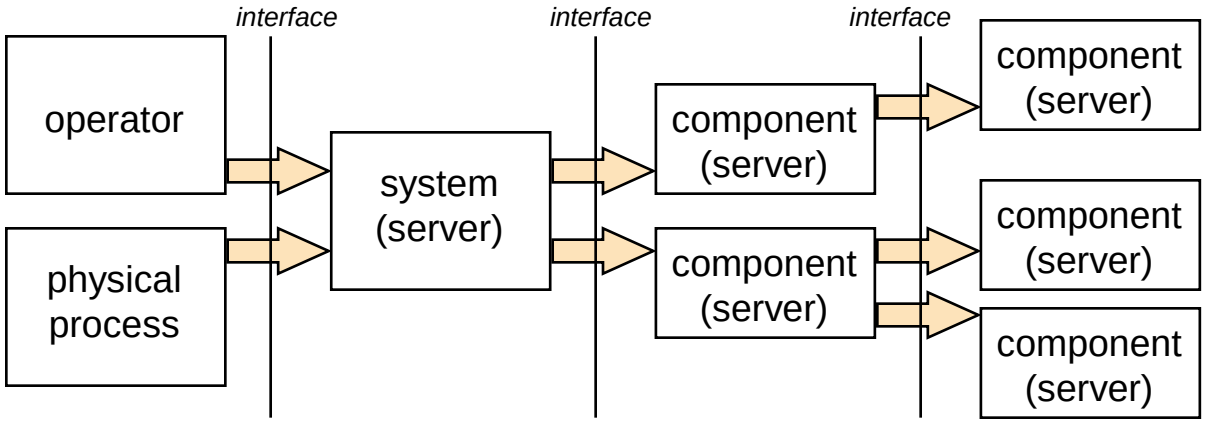
The part of the provider's total state that is perceivable at the service interface is its **external state**. The remaining part is its **internal state**.

The interface of the user at which the user receives the service is the **use interface**.

System Function, Behavior, Structure, and Service (cont.)

Recursive nature of the *depends* (⇨) relation

- service users depend on the services provided by the system (server)



Definitions of Failure – Error – Fault

Correct service is delivered when the service implements the system function.

A **(service) failure** is an event that occurs when the delivered service deviates from correct service.

- Thus, a *failure is a transition* from correct service to incorrect service.

The different kinds of incorrect service delivery are referred to as the **failure mode** and these modes are ranked according to **failure severity**.

Definitions of Failure – Error – Fault (cont.)

A service failure means that at least some external state of the provider service deviates from the correct state.

This deviation is called the **error** (i.e., a deviation from the current state from the correct state).

The adjudged or hypothesized cause of an error is called a **fault**.

Attributes of Dependability

Reliability: continuity of correct service.

Availability: readiness for correct service.

Maintainability: ability to undergo modifications and repairs.

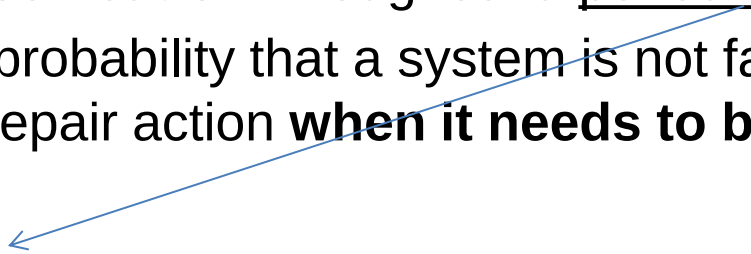
Safety: absence of catastrophic consequences on the user(s) and the environment.

Integrity: absence of improper system alterations. (see security)

Reliability vs. Availability

Reliability is the probability that the system will conform to its **functional specification** throughout a period of duration t.

Availability is the probability that a system is not failed or undergoing a repair action **when it needs to be used**.



→ "period of duration t" < "time when the system needs to be used"

→ Given the *need to use time* of a system, availability is the percentage of the *need to use time* for which the system will conform to its functional specification.

→ Availability is a function of reliability and maintainability.

Reliability vs. Availability (cont.)

Factory automatization:

- the computer has to assure proficient manufacturing
- availability is most important parameter
- reliability is not that important

Satellite:

- once put into operation there is no possibility for maintenance
- mission reliability is most important parameter

Reliability vs. Safety

Reliability is the probability that the system will conform to its **functional specification** throughout a period of duration t.

Safety is the probability that the system will not exhibit **specific undesired behaviors** throughout a period of duration t.

- → In general, not all deviations from the functional specification imply specific undesired behaviors.
- → When for a given system reliability equals safety then we call the system **safety-critical**.

Reliability vs. Safety (cont.)

Railway signalling:

- red signal is a safe system state
- safe system state is unreliable
- safety ≠ reliability

Fly-by-wire airplane control:

- after take off there is no safe (non-functional) system state
- safety ~ reliability
  (degraded modes of operation are possible)

Often there is a conflict between safety and reliability

Attributes of Security

Confidentiality: the absence of unauthorized disclosure of information.

+ **Integrity** (as before)
+ **Availability** (as before)

The Threats to Dependability and Security (Fault – Error – Failure)

Life Cycle of a System

- Development Phase, including
  - initial system conception
  - system design, development, verification, and validation
- Use Phase, including
  - service delivery
  - service outage (service not available)
  - service shutdown (service not needed)
  - maintenance

Faults

Recap:

The adjudged or hypothesized cause of an error is called a **fault**.

Faults – eight elementary fault classes

Classification into **eight elementary fault classes**:

- Phase of creation or occurrence (development vs. use phase)
- System boundaries (internal vs. external)
- Phenomenological cause (natural vs. human-made)
- Dimension (hardware vs. software)
- Objective (malicious vs. non-malicious)
- Intent (deliberate vs. non-deliberate)
- Capability (accident vs. incompetence)
- Persistence (permanent vs. transient)

Example Faults: Software Flaws

Software flaws (may) have the following aspects (in red):

- Phase of creation or occurrence (development vs. use phase)
- System boundaries (internal vs. external)
- Phenomenological cause (natural vs. human-made)
- Dimension (hardware vs. software)
- Objective (malicious vs. non-malicious)
- Intent (deliberate vs. non-deliberate)
- Capability (accident vs. incompetence)
- Persistence (permanent vs. transient)

Faults – combined fault classes

- A particular fault will typically fall into multiple of the eight elementary fault classes.
- Since three of the elementary fault classes are of particular importance, we use them to derive combined fault classes:
  - Phase of creation or occurrence (development vs. use phase) → Development Faults
  - System boundaries (internal vs. external) → Interaction faults
  - Dimension (hardware vs. software) → Physical faults

Failures

Recap:

A **(service) failure** is an event that occurs when the delivered service deviates from correct service.

- Thus, a *failure is a transition* from correct service to incorrect service.

Failure Mode Classification – Overview

- Domain:
  - content, early timing failure, late timing failure, halt failure, erratic failure
- Detectability:
  - signaled failures, unsignaled failures
- Consistency:
  - consistent failure, inconsistent failure
- Consequences:
  - minor failure, ..., catastrophic failure

Failure Mode Classification – Domain

- Content
- Early timing failure
- Late timing failure
- Halt failure
  - the external state becomes constant, i.e., system activity is no longer perceptible to the users
  - silent failure mode is a special kind of halt failure in that no service at all is delivered
- Erratic failure
  - not a halt failure, e.g., a babbling idiot failure

Failure Mode Classification – Consistency

When there are more than one users of a service.

- Consistent failure:
  - All users experience the same incorrect service.
- Inconsistent failure
  - Different users experience different incorrect services.

Failure Mode Classification – Consequences, e.g., Aircraft

Minor: 10E-5 per flight hour or greater

no significant reduction of aeroplane safety, a slight reduction in the safety margin

Major: between 10E-5 and 10E-7

significant reduction in safety margins or functional capabilities, significant increase in crew workload or discomfort for occupants

Hazardous: between 10E-7 and 10E-9

large reduction in safety margins or functional capabilities, causes serious or fatal injury to a relatively small number of occupants

Catastrophic: less than 10E-9

these failure conditions would prevent the continued safe flight and landing of the aircraft

Error

Recap:

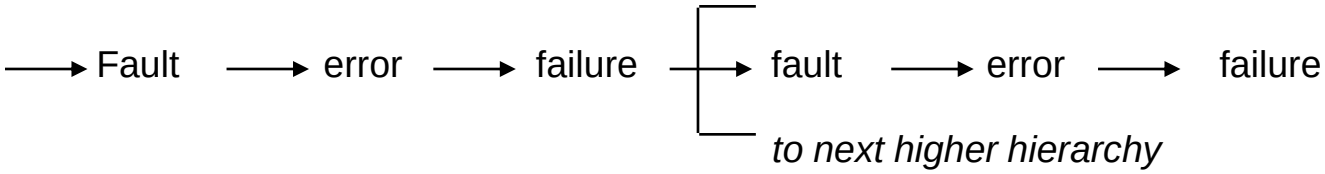
A service failure means that at least some external state of the provider service deviates from the correct state.

This deviation is called the **error** (i.e., a deviation from the current state from the correct state).

Error

- An error is detected if its presence is indicated by an error message or error signal.
- Errors that are present but not detected are *latent* errors.
- Whether or not an error actually leads to a failure depends on the following facts:
  - the system composition and the existence of redundancy (intentional or unintentional redundancy)
  - the system activity after the introduction of an error (the error may get overwritten)
  - the definition of a failure by the user's viewpoint

Fault – Error – Failure Chain



fault → error

- a fault which has not been activated by the computation process is *dormant*
- a fault is *active* when it produces an error

error → failure

- an error is *latent* when it has not been recognized
- an error is *detected* by a detection algorithm/mechanism

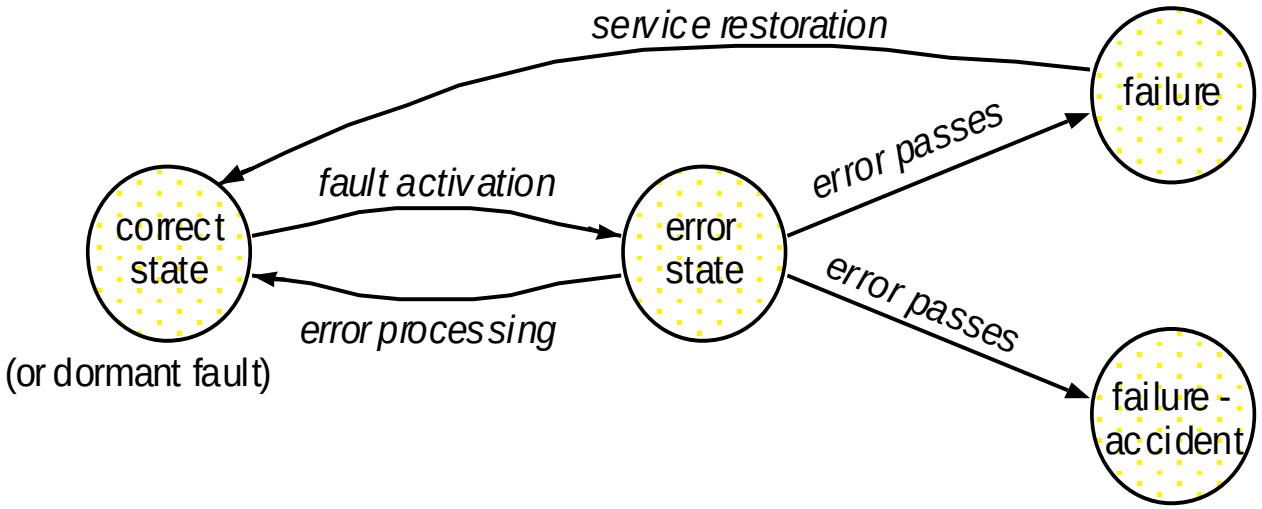
failure → fault

- a failure occurs when an error “passes through” and affects the service delivered
- a failure results in a fault for the system which contains or interacts with the component

Examples for fault/failure chain

- **Program error (software):**
  - a dormant *fault* in the written software (instruction or data)
  - upon activation the fault becomes active and produces an *error* (system state)
  - if the erroneous data affects the delivered service, a *failure* occurs
- **Electromagnetic interference (hardware):**
  - leads to *faulty* input value (either digital or analog)
  - by reading the input the fault becomes active and produces an *error*
  - if the erroneous input value is processed and becomes visible at the interface a *failure* occurs

Fault/failure state transition chart



The Means to Attain Dependability

Means to Attain Dependability and Security

Fault prevention: means to prevent the occurrence or introduction of faults.

Fault tolerance: means to avoid service failures in the presence of faults.

Fault removal: means to reduce the number and severity of faults.

Fault forecasting: means to estimate the present number, the future incidence, and the likely consequences of faults.

Fault prevention

- **hardware components:**
  - environment modifications (temperature)
  - quality changes, use “better” components
  - component integration level, higher integration
  - derating, reduction of electrical, thermal, mechanical, and other environmental stresses
- **software components:**
  - software engineering methodologies
  - OOD and OO languages
  - design rules
  - CASE tools
  - formal methods

Fault removal

- **verification:**
  to check, whether the system adheres to the specification.
  - Static analysis: inspections, walk-throughs, data flow analysis, complexity analysis, compiler checks, correctness proofs, petri net models, finite state automata.
  - Dynamic Analysis: testing, black-box, white-box, conformance, fault-finding, functional, timeliness, structural, deterministic, random or statistical
- **diagnosis:**
  diagnosing the fault which prevented the verification from succeeding
- **correction:**
  perform corrective actions to remove the fault $\Rightarrow$ regression verification

Fault forecasting

- performing an evaluation of the system with respect to faults
- evaluation of aspects such as:
  - reliability
  - availability
  - maintainability
  - safety
- see chapter “Fault-tolerance and modelling”

Fault tolerance

There are four phases, which, taken together, provide the general means by which faults are prevent from leading to system failures.

- **error detection:**
  errors are the manifestations of faults, which need to be detected to act upon
- **damage confinement and assessment:**
  before any attempt is made to deal with the detected error, it is necessary to assess and confine the extent of system state damage

Fault tolerance (cont.)

- **error recovery:**
  error recovery is used to transform the currently erroneous system state into
  a well defined error-free system state
- **fault treatment and continued service:**
  even if the error-free system state has been recovered it is often necessary to perform further actions to prevent the fault from being activated again

Error Recovery and Redundancy

Error recovery

There are two possibilities to transform the currently erroneous system state into an error-free system state:

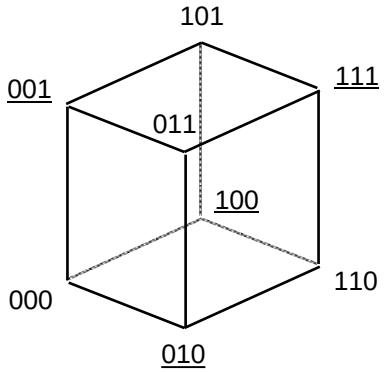
- **Backward recovery:**
  - system state is reset to a previously store error-free system state
  - re-execution of failed processing sequence
  - typical for data base systems
    (it is not possible to predict valid system states)
- **Forward recovery:**
  - system state is set to a new error-free system state
  - typical for real-time systems with period processing patterns
    (it *is* possible to predict valid system states)

Redundancy

A system requires some kind of redundancy to tolerate faults. This redundancy can be implemented in three different domains:

- **Domain of information:**
  redundant information e.g. error correcting codes, robust data structures
- **Domain of space:**
  replication of components, e.g. 2 CPU's, UPS (uninterruptable power supply)
- **Domain of time:**
  replication of computations, e.g. calculate results by same (or different) algorithm a second time, sending messages more than once

Fault-tolerance in the domain of information



3 bit code, $d = 1$

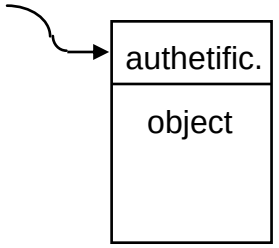
- **error correcting codes:**
  - for all error correcting codes (ECC)

$$(2t + p + 1) \leq d$$

d .. Hamming distance of code
t .. number of single bit errors to be tolerated
p .. number of additional errors that can be detected

Fault-tolerance in the domain of information (cont.)

- **robust data structures:**
  - store the number of elements
  - redundant pointers
    (e.g. double linked chains with status)
  - status or type information
    (e.g. authenticated objects)
  - checksum or CRC
- **application specific knowledge**

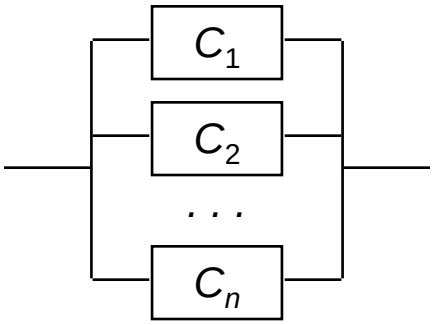


pointer to authentificated object

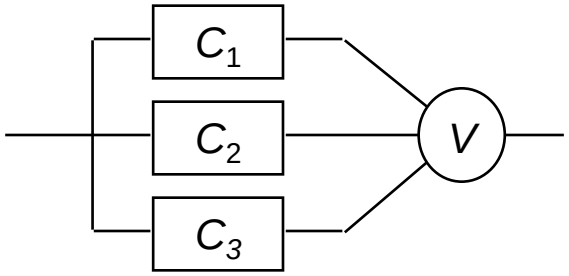
Fault-tolerance in the domain of space

- **active redundancy**

– parallel fail-silent components



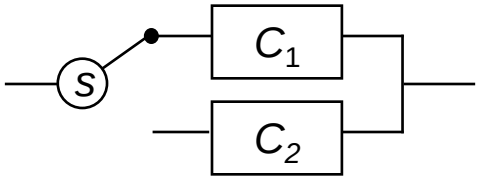
– voting, triple modular redundancy (TMR)



Fault-tolerance in the domain of space (cont.)

passive or standby redundancy

- hot standby:
  standby component is operating
- cold standby:
  standby components starts only in case of a failure



Fault-tolerance in the domain of time

Allows tolerance of temporary faults

- **multiple calculation:**
  - a function is calculated n times with the same inputs
  - the result is checked by an acceptance test
  - or the multiple results are voted
- **sending messages multiple times:**
  - message transmission is repeated n times
  - retransmission only in case of failures (positive acknowledge retransmit PAR)
  - retransmission always n times (reduces temporal uncertainty for real-time systems)

On the Importance of the Specification

Specification

The definition of all dependability attributes is based on specifications. A *good* specification must be:

- exact
- consistent
- complete
- authoritative

Importance of specification

> Together with the analysis of possible behavior and its consequences, system specification is the most difficult part of building a dependable system.

Specification (cont.)

Multiple levels of specifications

To consider the different aspects and attributes of dependable systems, usually different levels of specifications exists.

An example

level	*specification*
functional	“all commands have to be carried out correctly”
reliability	“either correct commands or warning indicator”
safety	“recorded info may not be corrupt”

The underground train

The underground train

- an electronically controlled underground train had the following buttons:
  - to open and close doors
  - to start the train
- it was specified that “the train only may start if and only if the start button is pressed and all doors are closed”
- a driver blocked the start train button by means of a tooth pick to start the train immediately if the doors were closed

The underground train (cont.)

What happened?

- one day a door was blocked and the driver went back to close the door, and of course, the train left the station without the driver

What went wrong?

- it was the drivers fault to block the start button with a tooth pick
- but it was also a specification fault since the correct specification should have read: “the train only may start if and only if the start button changes it state to start and all doors are closed”
- in that example it made a big difference whether *state* or *event*-semantics are implemented