UNIVERSITY OF NEWCASTLE

Facing Up to Faults

(v.2.0.1)

Brian Randell

The Menu

- On Dependability Concepts
- On Fault Assumptions
- On System Structure

On Dependability Concepts

- A system **failure** occurs when the delivered service deviates from fulfilling the system function, the latter being what the system *is aimed at*.
- An **error** is that part of the system state which is *liable to lead to subsequent failure:* an error affecting the service is an indication that a failure occurs or has occurred. The *adjudged or hypothesised cause* of an error is a **fault.**

(Note: errors do not necessarily lead to failures; component failures are not necessarily faults to the surrounding system)

The Failure/Fault/Error “Chain”

- A failure occurs when an error “passes through” the system-user interface and affects the service delivered by the system – a system of course being composed of components which are themselves systems. Thus the manifestation of failures, faults and errors follows a “fundamental chain”:

. . . → failure → fault → error → failure → fault →. . .

i.e.

. . . → event → cause → state → event → cause → . . .

Dependability - the “standard” definition

Dependability is usually defined as that property of a computer system such that *reliance can justifiably be placed on the service* it delivers. (The service delivered by a system is its behaviour *as it is perceptible* by its user(s); a user is another system (human or physical) which *interacts* with the former.)

Dependability > Correctness

- The four basic dependability technologies are
  - fault prevention (rigorous design),
  - fault removal (verification & validation)
  - fault tolerance
  - fault forecasting (system evaluation)
- The effective combination of the first three is crucial - reliance on any one - or even two - of them is in general insufficient to achieve dependability, even just for software, leave alone systems
- And the fourth, being the means of assessing progress towards achieving adequate dependability, is equally vital, in order to demonstrate this achievement

UNIVERSITY OF
NEWCASTLE

The Role of Judgement

A given system, operating in some particular environment (a wider system), may fail in the sense that some other system makes, or could in principle have made, a *judgement* that the activity or inactivity of the given system constitutes **failure.**

The concept of **dependability** can then be more simply defined as: "the quality or characteristic of being dependable", where the adjective "**dependable**" is attributed to a system whose failures are *judged* sufficiently rare or insignificant.

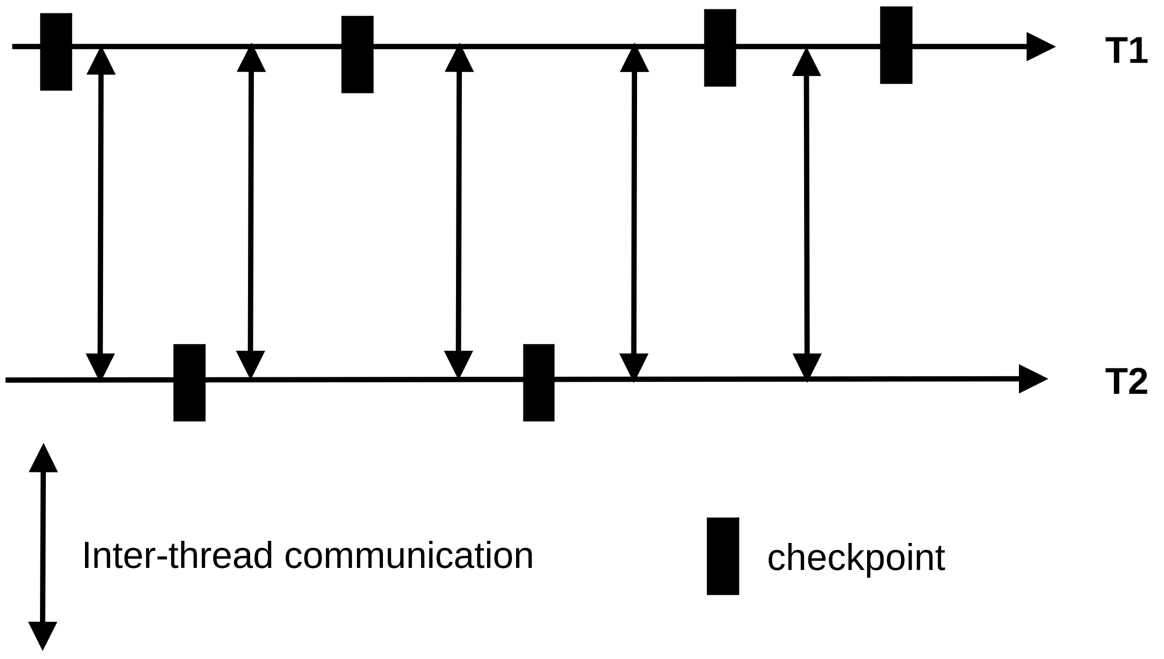
Concepts & Terminology

- Note the generality of the definitions of fault,error, failure and dependability, and their wide applicability
- What matters are concepts, rather than terminology
- Differing research communities (reliability, safety, survivability, security, etc.,) use differing terminology, and definitions, unfortunately
- But what is critical is a fully general notion of failure, and of the three *different* concepts: fault, error, failure
- (to deal properly with the complexities (and realities) of failure-prone components, being assembled together in possibly incorrect ways, so resulting in failure-prone systems.)

On Fault Assumptions

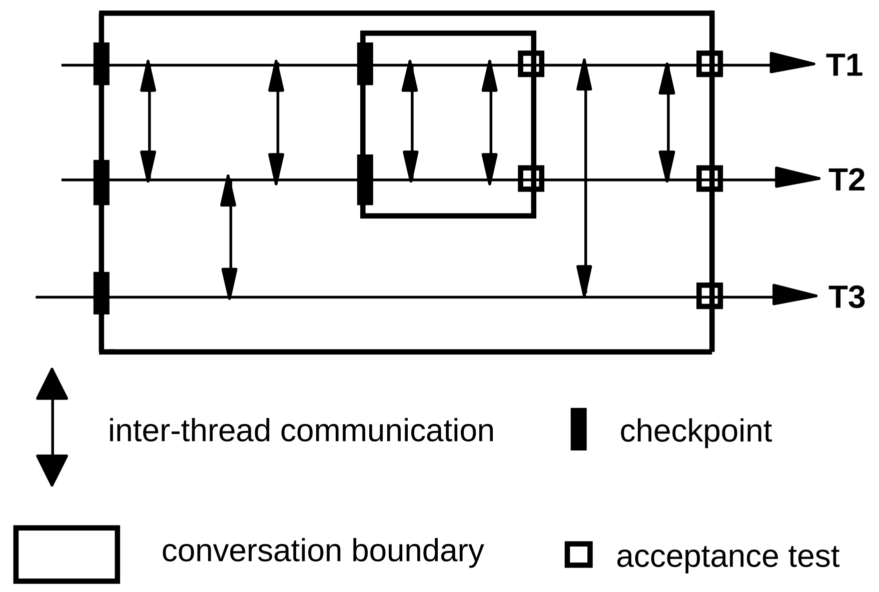
- Regarding the nature and likelihood of faults
  - and the effectiveness of fault masking - possibly obviating the need for error recovery
- Regarding the ability to validate inputs and ouputs
  - and the practicality of various types of error recovery
- All these assumptions greatly influence the system designer’s task
  - including that of the designer of the facilities and processes used for system design
- Their careful identification is one of the most crucial aspects of system design

Fault Assumptions - the possible "domino effect"



The possibility of this effect depends critically on validation assumptions

A "solution"
- the nested conversation structure



But conversations deal only with co-operative, not competitive concurrency - Hence Newcastle's work on "Coordinated Atomic Actions":

On Structure

“The price of reliability is utter simplicity - and this is a price that major software manufacturers find too high to afford!” - Hoare

On Structure

“The price of reliability is utter simplicity - and this is a price that major software manufacturers find too high to afford!” - Hoare

But

“Everything should be made as simple as possible, but not simpler” - Einstein

On Structure

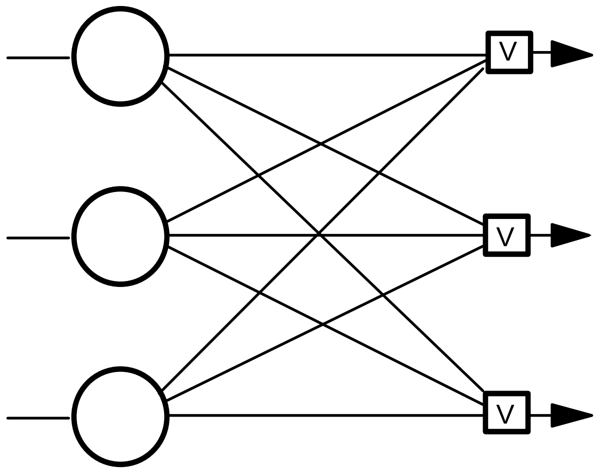
“The price of reliability is utter simplicity - and this is a price that major software manufacturers find too high to afford!” - Hoare

But

“Everything should be made as simple as possible, but not simpler” - Einstein

- Good system structuring allows one to deal with the added complexity that result from more realistic fault assumptions - its quality is measured by its:
  - coupling and cohesion (for performance)
  - **strength** (for dependability)

Structural Strength - e.g. in Triple Modular Redundancy



A strongly-structured system is one in which the structuring exists in the actual system, not just its description or design, and helps to limit the impact of faults

Structure and Redundancy

- The basic idea underlying all techniques aimed at achieving high dependability is that of "*consistency-checking of useful redundancy*"
  - It underlies <u>all</u> forms of validation, from program verification and code inspection to debugging,
  - and <u>all</u> forms of fault tolerance, (including in hardware, software, bureaucracies, and socio-technical systems)
- Equally fundamental and closely-related is the use of system (in particular program) <u>structuring techniques</u>.
  - Important for complexity reduction (i.e. understandability), and code re-use, but also – if retained in the operational system – for error detection and for limiting error propagation.

Structure for Dependability

- Exception Handling - in programming languages, and at higher system levels (e.g. in workflow languages)
    - this is a form of retained structuring that aids the provision of coherent methods of error recovery, and the production of systems which can when necessary "degrade gracefully"
- Software Architecture - e.g. "design patterns", and in particular techniques for constructing systems out of components and stylized connectors
    - these facilitate not just the system design and evolution, but also run-time error detection and confinement.
- Multi-level Architectures - the use of multiple representations of a system, at successively lower levels of abstraction. Ideally, such levels of abstraction are employed not just at design time, but instead are retained during operation.
    - they aid system adaptation, and enable consistency checking at each level, and between levels.

By Way of Summary: it is vital -

- To have a concept which is associated with a fully general notion of failure - not limited just to particular types, causes or consequences of failure
- To use separate terms for the three essentially different concepts: “fault”, “error” and “failure”
- To understand the “fundamental chain”:

  . . . → failure → fault → error → failure → fault →. . .

  - in order to deal with situations involving complex badly-specified systems, with uncertain boundaries, where judgements as to possible causes or consequences of failure are difficult, and provisions for preventing (possibly deliberate) faults from causing failures are likely to be fallible, i.e. with reality!
- And to pay careful attention to the use and retention of structure and redundancy

  - for purposes of complexity control, error containment, and system evolution
- As a basis for a coherent and comprehensive approach to dealing with the possibility of failure, in both system design and operation

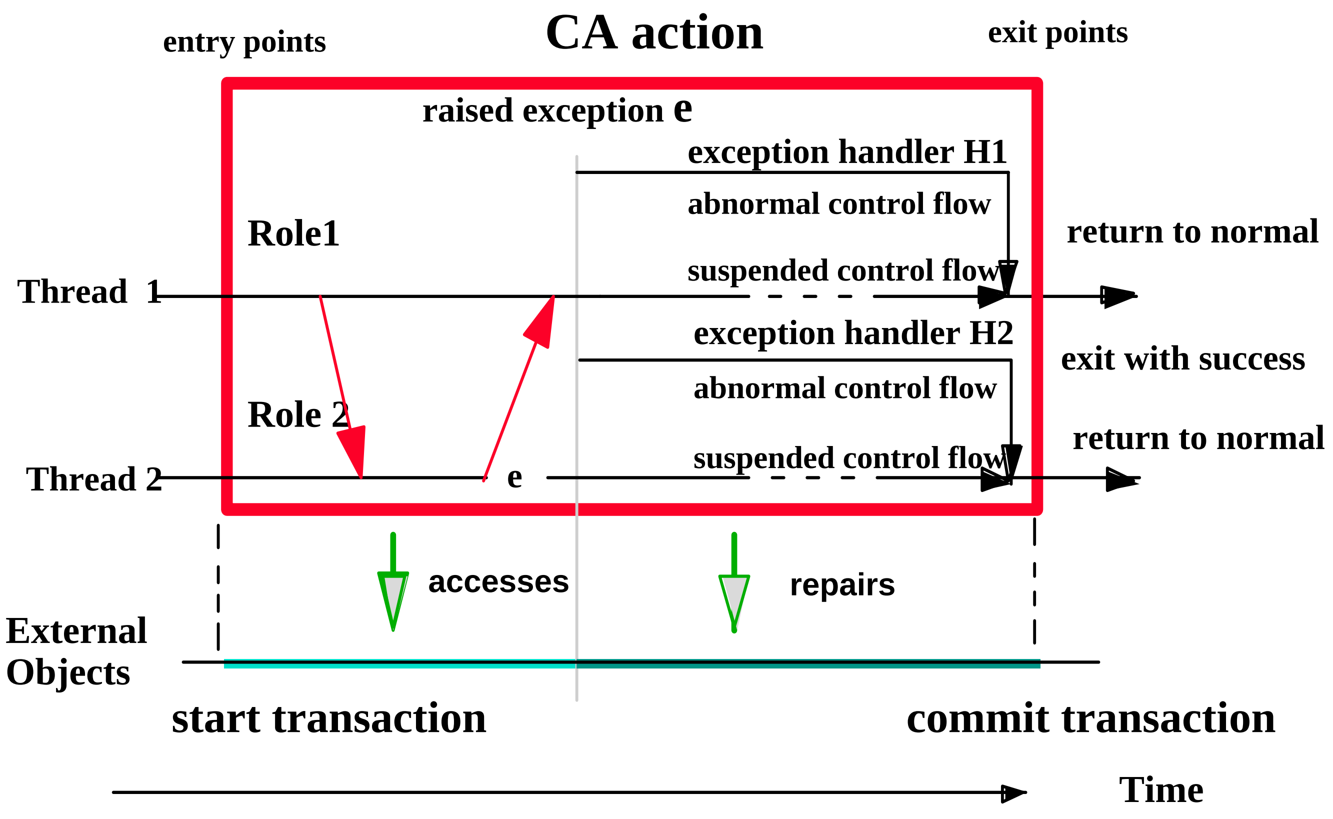
UNIVERSITY OF NEWCASTLE

Co-ordinated Atomic Actions

- A mechanism/protocol for (forward and/or backward) error recovery for systems and their environments in the presence of both cooperative and competitive concurrency.
- In effect a programming discipline for nested multi-threaded transactions with very general exception handling provisions
  - To cooperate in a CA action a group of concurrent threads must come together to perform the roles of the action collectively. They enter and leave the action in real or virtual synchrony
  - Inside a CA action, roles can be involved in (nested CA actions.
  - If an error is detected inside a CA action, recovery measures must be invoked co-operatively, by all the roles, in order to reach some mutually consistent conclusion (success, exception, or failure)
  - External objects, which are in effect being competed for by the CA action, must behave atomically with respect to other CA actions and threads so that they cannot be used as an implicit means of "smuggling" information into or out of a CA action.

http://homepages.cs.ncl.ac.uk/alexander.romanovsky/home.formal/caa.html

A Co-ordinated Atomic Action



entry points
CA action
exit points
raised exception e
exception handler H1
abnormal control flow
Role1
return to normal
suspended control flow
Thread 1
exception handler H2
exit with success
abnormal control flow
Role 2
return to normal
suspended control flow
Thread 2
e
accesses
repairs
External Objects
start transaction
commit transaction
Time