Architecting Software Fault Tolerance and Beyond

Rogério de Lemos
University of Kent, UK

Roadmap

- On architecting…
- Architecting dependability
  - motivation and contextualisation
  - architectural abstractions for fault tolerance
  - process of integrating verification and validation
- Architecting resilience
  - how to deal with change and uncertainty
  - self-adaptable software systems
  - adaptable error detection
  - "no failure assumptions"
- Summary and future work

On Architecting

- "Architecting"
  - Eberhardt Rechting. *Systems Architecting, Creating & Building Complex Systems*. 1991.
  - the process of creating and building architectures
    - the architect influences the whole development process
  - the art and science of creating and building complex systems
    - scoping, structuring and certifying
- Architecting dependable systems
  - http://www.cs.kent.ac.uk/~rdl/ADSFuture/

Motivation and Contextualisation

Two perspectives when considering dependability at the architectural level

- from the dependability perspective
  - reason about faults earlier rather than later
- from the software engineering perspective
  - acceptance of faults rather than their avoidance

Emerging applications that justify the approach

- trustworthy systems from untrustworthy components
  - off-the-shelf-components (OTS), system-of-systems, legacy systems, critical infrastructures

Motivation and Contextualisation

Fault tolerance at the architectural level

- isn't too early to consider failure behaviours?
  - faults don't respect phases of development or levels of abstraction
- why introduce unnecessary complexity?
  - fault tolerance doesn't come for free
    - structuring the complexity associated with redundancies
- wouldn't it be more effective to build a fault free system?
  - yes, but…
  - *optimists* vs *realists*

Architectural Abstractions for Fault Tolerance

- Architectural patterns, styles or abstractions
  - basic structural and behavioural building blocks
- Not patterns because of design patterns
  - but beyond architectural tactics
- Can be considered as extensions to architectural styles
  - e.g., P2P architectural style
  - ***architectural style*** is a set of design rules
    - the kinds of components and connectors
    - local or global constraints on the composition
  - not called as such, just to avoid confusion…

Architectural Abstractions for Fault Tolerance

- **Implicit redundancy** - the normal and abnormal specifications can be entangled
  - idealised fault-tolerant architectural element (iFTE)
    - based on exception handling
- **Explicit redundancy** - structuring of redundancy is part of the actual system
  - an inherent aspect of strongly-structured systems
    - restrict the impact of faults
  - halt-on-failure architectural element (HoFE)
    - assumes crash failure semantics
- Selection depends on
  - failure assumptions and availability of resources

Abstraction-Based Development Approach

Development approach based on abstractions

- instantiate formal abstractions into components and connectors of the software architecture
  - prevent the introduction of design faults
- automatic verification of architectural models
  - identify and remove design faults
- generation of architectural-based test cases
  - identify and remove implementation faults related to the architectural design

University of Kent

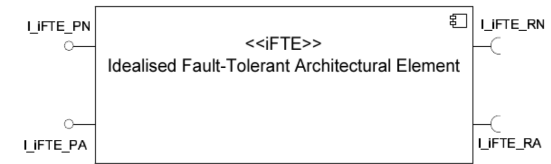
Idealised Fault-Tolerant Architectural Element (iFTE)

An architectural solution based on **exception handling**

- components need to collaborate for handling certain failure scenarios
- configurations that allow the propagation of exceptions
  - controlled error propagation
- based on the **idealised fault-tolerant component**
  - provided and required services, and interface and failure exceptions
  - separation between normal and abnormal behaviour

University of Kent

Idealised Fault-Tolerant Architectural Element (iFTE)



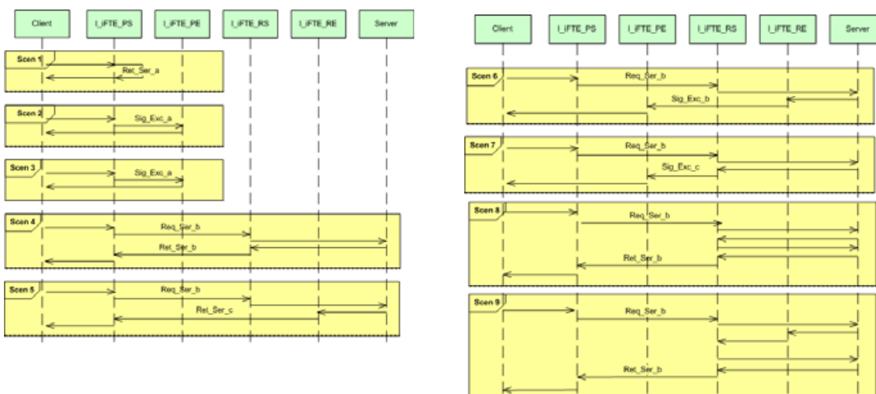

```
system ifte_abstraction
  features
    I_iFTE_PN_i: in event data port  Service;
    I_iFTE_PN_o: out event data port  Service;
    I_iFTE_PA_o: out event data port  Exception;
    I_iFTE_RN_i: in event data port  Service;
    I_iFTE_RN_o: out event data port  Service;
    I_iFTE_RA_i: in event data port  Exception;
  flows
    Ret_Ser_a: flow path I_iFTE_PN_i -> I_iFTE_PN_o;
    Sig_Exc_a: flow path I_iFTE_PN_i -> I_iFTE_PA_o;
    Req_Ser_b: flow path I_iFTE_PN_i -> I_iFTE_RN_o;
    Ret_Ser_b: flow path I_iFTE_RN_i -> I_iFTE_PN_o;
    Req_Ser_c: flow path I_iFTE_RN_i -> I_iFTE_RN_o;
    Sig_Exc_b: flow path I_iFTE_RN_i -> I_iFTE_PA_o;
    Req_Ser_d: flow path I_iFTE_RA_i -> I_iFTE_RN_o;
    Ret_Ser_c: flow path I_iFTE_RA_i -> I_iFTE_PN_o;
    Sig_Exc_c: flow path I_iFTE_RA_i -> I_iFTE_PA_o;
end ifte_abstraction;
```


University of Kent

Idealised Fault-Tolerant Architectural Element (iFTE)

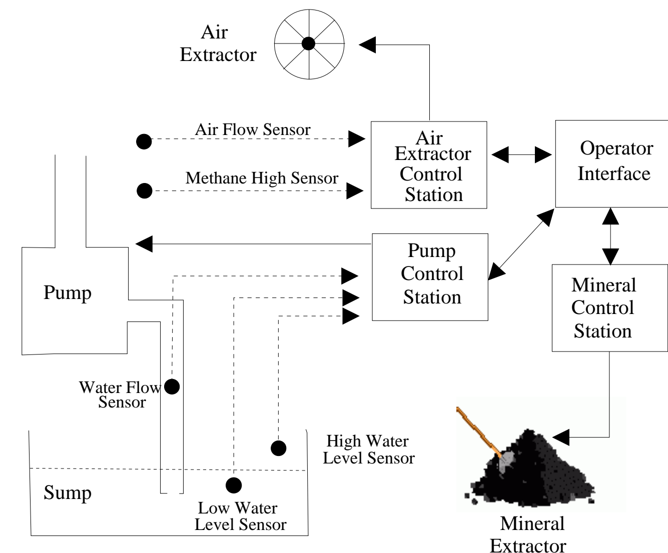
Behavioural scenarios

- constraint the behaviour of the iFTE



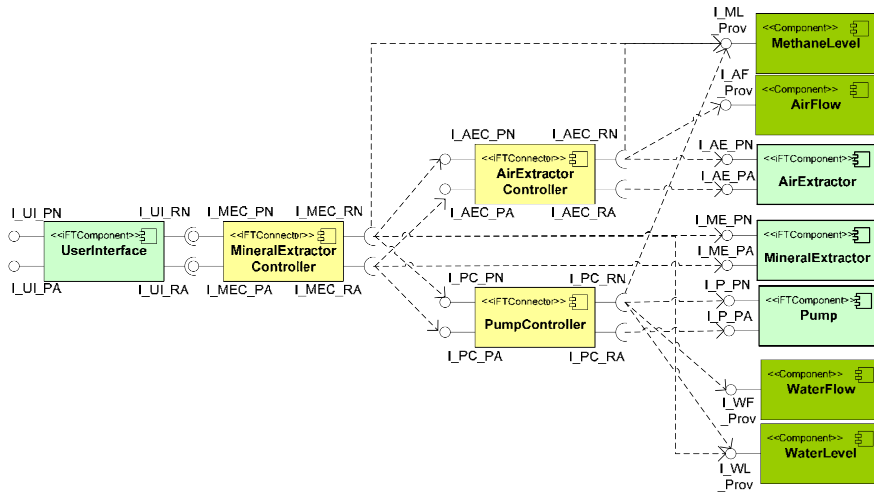
University of Kent

Mining Control System



University of Kent

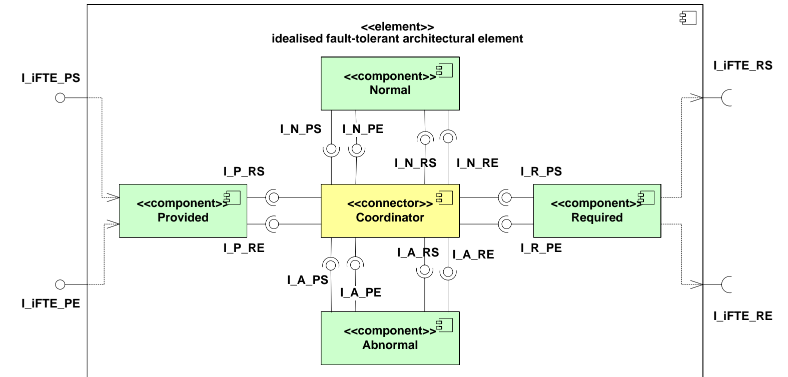
Mining Control System



University of Kent

Idealised Fault-Tolerant Architectural Element (iFTE)

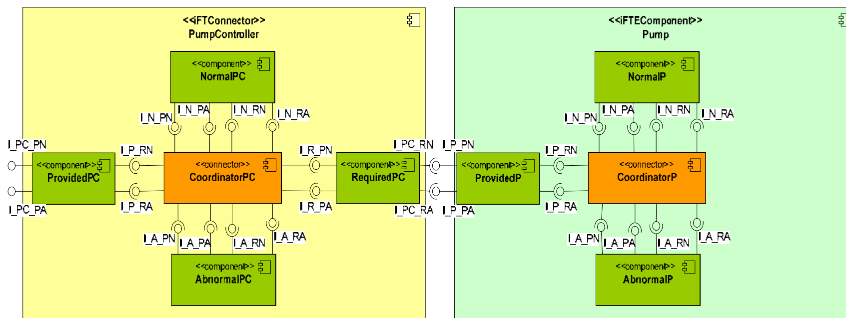
- iFTE detailed design
  - iFTE internal structure follows recursively the iFTE abstraction



University of Kent

Mining Control System

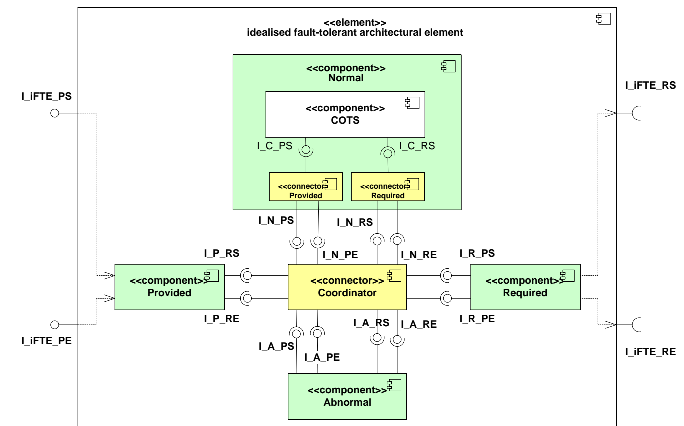
Detailed iFTE-based software architecture



University of Kent

Idealised Fault-Tolerant Architectural Element (iFTE)

- iFTE detailed design
  - iFTE applied to COTS



University of Kent

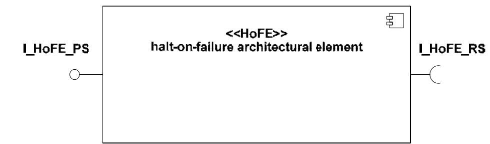
Halt-on-Failure Architectural Element (HoFE)

An architectural solution based on explicit redundancies

- assumes crash-failure semantics
  - either provides the service as specified or fails silent
- error detection by comparison (assuming a single fault)
  - to detect 1 error needs 2 redundant nodes
  - to tolerate 1 fault needs 2 HoFEs

University of Kent

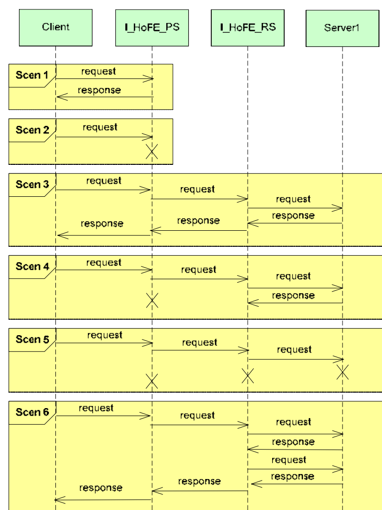
Halt-on-Failure Architectural Element (HoFE)




```
system hofe_abstraction
  features
    I_HoFE_PS_i: in event data port  Service;
    I_HoFE_PS_o: out event data port  Service;
    I_HoFE_RS_i: in event data port  Service;
    I_HoFE_RS_o: out event data port  Service;
flows
    Ret_Ser_a: flow path I_HoFE_PS_i -> I_HoFE_PS_o;
    Req_Ser_b: flow path I_HoFE_PS_i -> I_HoFE_RS_o;
    Ret_Ser_b: flow path I_HoFE_RS_i -> I_HoFE_PS_o;
    Req_Ser_c: flow path I_HoFE_RS_i -> I_HoFE_RS_o;
end hofe_abstraction;
```

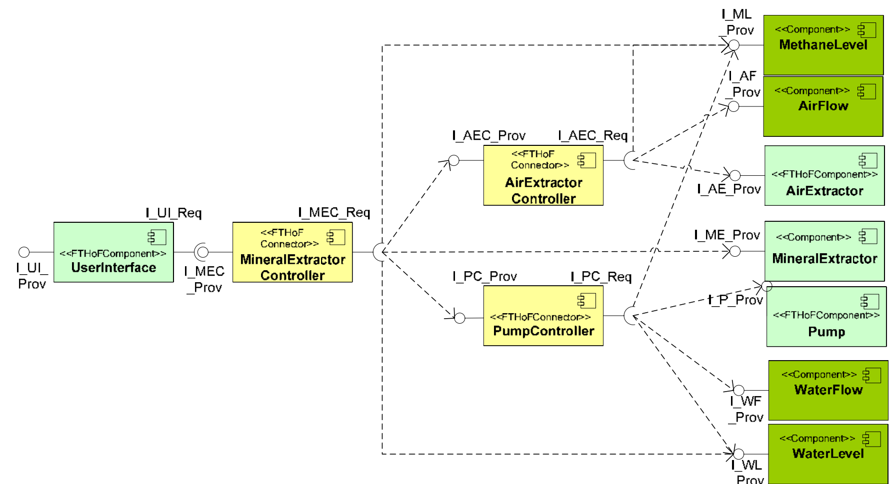

University of Kent

Halt-on-Failure Architectural Element (HoFE)



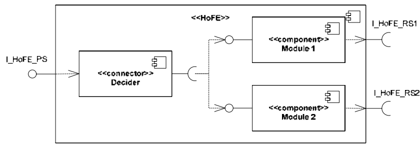
University of Kent

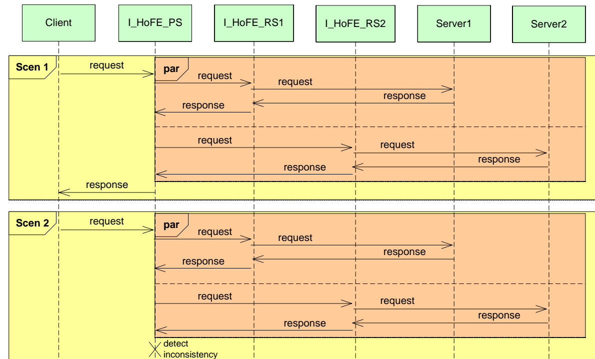
Mining Control System



University of Kent

Halt-on-Failure Architectural Element (HoFE)

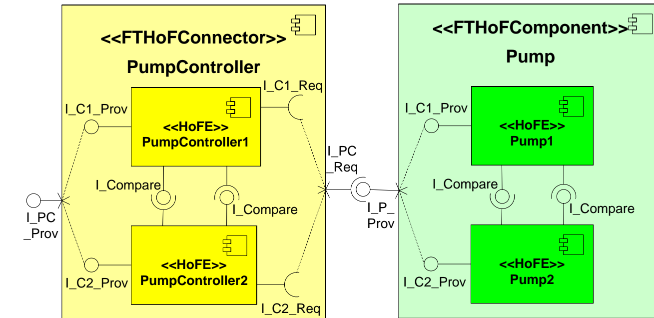




University of Kent

Mining Control System

Detailed HoFE-based software architecture



University of Kent

iFTE and HoFE as Architectural Abstractions

Main features of iFTE and HoFE architectural abstractions

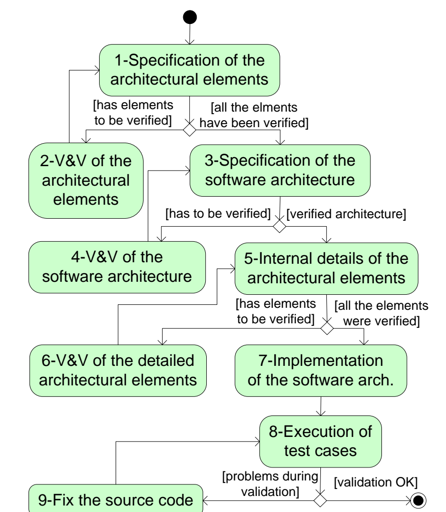
- enable the provision of fault tolerance
- described by their interfaces and behavioural scenarios
- allow to compose a system uniformly
  - eases integration of components and scaling
- can be used at two levels of abstraction
  - abstract and detailed design
- can be specified with other non-iFTEs or non-HoFEs

University of Kent

Rigorous Development

iFTE and HoFE are first class modelling entities

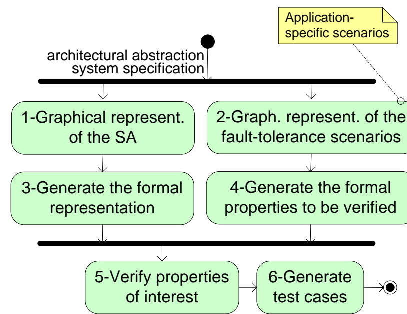
- formal specification of architecture elements and configurations
  - structural and behavioural
- verification of architecture elements and configurations
- validation by generating architectural-based test cases
  - integration and robustness



Rigorous Development – V&V

iFTE and HoFE are first class modelling entities

- formal specification of architecture elements and configurations
  - structural and behavioural
- verification of architecture elements and configurations
- validation by generating architectural-based test cases
  - integration and robustness



Architectural Representation

- Specification of the software architecture in UML
  - UML component diagram representing the structure of the software system
  - UML sequence diagrams representing the architectural scenarios related to fault tolerance
- Automatic model transformation from UML (XMI files) to B-Method and CSP

Formal Representation

Architectural specification uses both B-Method and CSP

- B-Method
  - represents the structural information about the architecture
    - architectural elements, their interfaces and respective operations and exceptions
      - events related to requests and responses of operations
    - architectural configurations
  - represents exception types
    - internal, external (interface, failure, propagated)
  - represents relations between exception types
    - e.g., exception conversions

Formal Representation

Architectural specification uses both B-Method and CSP

- CSP
  - facilitates the representation of complex ordering of events
    - behavioural scenarios
    - complex exception propagation rules
  - describes the sequence of events associated with each architectural element
  - how these events are related at the architectural level

University of Kent

Verification

Verification of iFTE- and HoFE-based architectures

- architectural elements
  - consistency against the architectural abstraction
- architectural configuration
  - composition rules of the architectural abstraction

University of Kent

Verification

Architectural verification uses ProB and the goals are

- integrity consistency
  - syntactical analysis of the B-Method machines
- architectural elements scenarios violations
  - violations on the specified scenarios
- architectural configuration scenarios violations
  - violation of specific scenarios from the composition of architectural elements
- application requirements
  - application specific scenarios

University of Kent

Architecture-Based System Validation

Two techniques for validating a software system against its architectural representation

- **integration testing**
  - identification of mismatches between architectural elements
- **robustness testing**
  - complement functional testing
  - guarantee that the software behaviour is acceptable
    - internal or external failures
    - stressful environmental conditions

University of Kent

Summary – Architecting Dependability

Architecting software fault tolerance using abstractions

- architectural abstractions
  - idealised fault-tolerant architectural element (iFTE)
    - implicit redundancy and based on exception handling
  - halt-on-failure architectural element (HoFE)
    - explicit redundancy and based on the crash failure semantics
- rigorous development method
  - automatic verification of architectural models
  - generation of architectural-based test cases

University of Kent

Architecting Resilience

- …beyond design faults
  - **change** instead of faults
    - environment, requirements or system (e.g., faults or QoS)
- **Resilience** according to *Laprie*
  - persistence of service delivery that can justifiably be trusted, when facing changes
- It is not only change
  - **uncertainty** associated with change
  - **uncertainty** how the system reacts to change
- Provision of self-adaptability

University of Kent

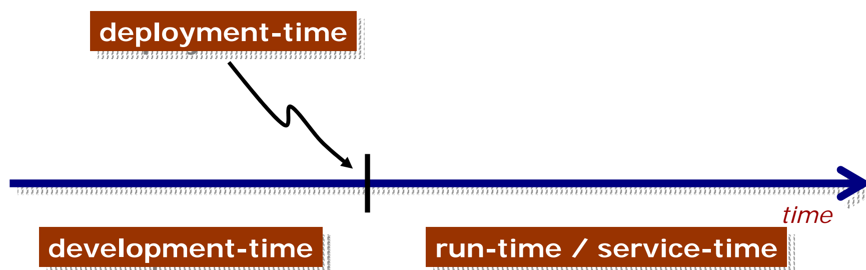
Self-Adaptive Systems

- **Self-adaptive system** is able to modify its behaviour according to environmental and system changes
  - it must continuously monitor changes and react accordingly
- Two dichotomies that need come together
  - centralised vs. decentralised (or self-organised)
    - top down vs. bottom up
  - process-driven vs. data-driven
    - based on models for the purpose of obtaining assurances

University of Kent

Software Engineering Process

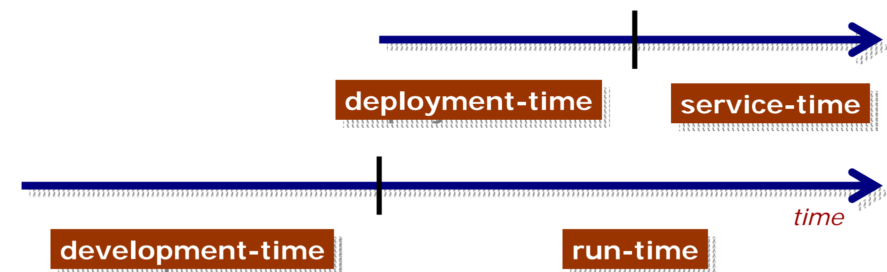
Traditional software engineering approaches



University of Kent

Software Engineering Process

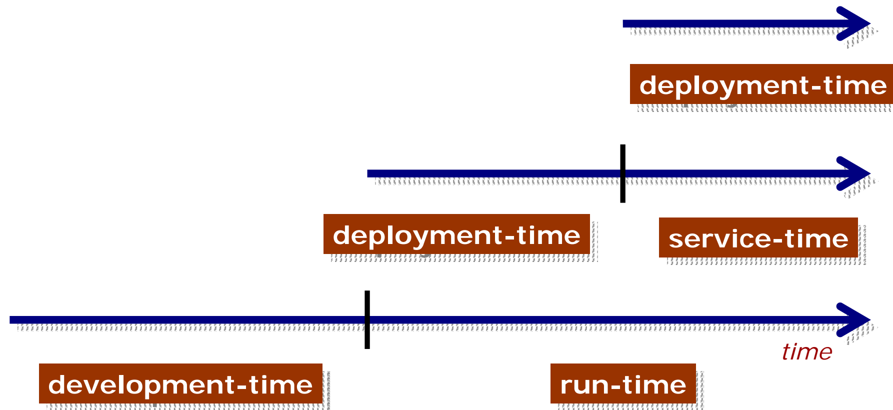
Software engineering for self-adaptive systems



University of Kent

Software Engineering Process

Software engineering for self-adaptive systems



University of Kent

Software Engineering Processes

- **Development-time process**
  - emphasis at the initial phases of development
- **Deployment-time process**
  - goal refinement
  - automatic generation of code
    - e.g., from architectural designs
    - e.g., mismatch resolution and tolerance
  - built-in self-testing
- **Decision-making process**
  - no humans in the loop during run-time
    - e.g., architectural design decisions

University of Kent

Classification of Self-Adaptive Software Systems

- Four groups of modelling dimensions
  - representation of **goals** considering system self-adaptation
    - evolution, flexibility, duration, multiplicity, and dependency
  - **change** the trigger of self-adaptation
    - source, type, frequency, and anticipation
  - **mechanisms** that handle self-adaptation
    - what is the reaction of the system towards change
    - type, autonomy, organisation, scope, duration, timeliness, and triggering
  - **effects** of self-adaptation upon the system
    - what is the impact of self-adaptation upon the system
    - overheads, predictability, criticality, resilience

University of Kent

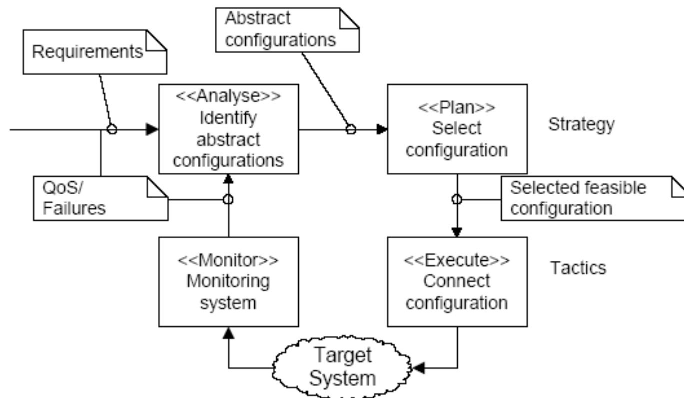
Resilience Challenges

- Modelling dimensions
  - understanding what are self-adaptive systems
- Requirements
  - new requirements language for dealing with uncertainty
  - transformation from requirements to architecture
- Assurances
  - agile run-time assurance in terms of architectural models
- Feedback control loops
  - explicit architectural representation

University of Kent

Feedback Control Loops

- Automatic generation of workflows for architectural reconfiguration



University of Kent

Self-Adaptability in Support of Resilience

Two examples how self-adaptability can support resilience

- Data-driven solution
  - improving the availability of self-service terminals
  - adaptable error detection
- Self-organisation
  - dependable and secure distributed storage system
  - relaxing failure assumptions

University of Kent

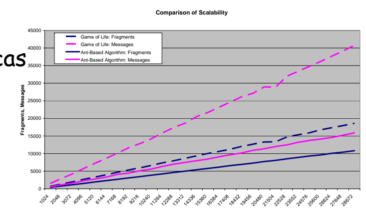
Vaccination and Adaptability as a Means for Protecting Systems against "Unknown Unknowns"

- Failure assumptions of evolving systems may change
- For improving system availability (reduce maintenance)
  - increase error detection coverage
- Adaptable error detection mechanisms
  - detection of errors that were not known during the design-time of systems
- Immune inspired adaptable error detection
  - *vaccination* and *adaptability* analogies of the immune system

University of Kent

Dependable and Secure Distributed Storage System for Ad Hoc Networks

- A self-organizing storage system
  - fragmentation-redundancy-scattering (FRS)
  - swarm of fragments replicas
- Two different algorithms
  - fragments are autonomous agents that manage replicas locally
  - the number of fragment replicas is an emergent property
    - depends on the parameters of the local algorithm
  - arbitrary node failures and hostile nodes
- Simulations have shown
  - stable population of fragment replicas
    - network instability
    - malicious information destruction
  - scalable solution (28K nodes)



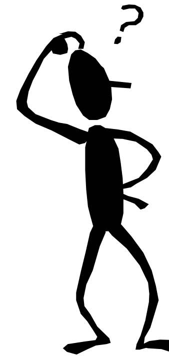
University of Kent

Summary – Architecting Resilience

- Self-adaptable software system going through a revival lately
  - there has been a lot of work on architectural resilience
    - dynamic architectural reconfiguration
  - change has become the main focus
    - not restricted to architectural design
- Incorporation of self-adaptable capabilities
  - increases system complexity
    - like redundancies in fault tolerance
  - aims to increase resilience in systems

University of Kent

Questions?



Thanks to

- Patrick Brito, Cecília Rubira, Eliane Martins, and Regina Moraes
- Jesper Andersson, Sam Malek, and Danny Weyns
- Jon Timmis, and Modupe Ayara
- Rudi Ball, James Grant, Jon So, and Vicki Spurrett