Software Architecture & Dependability

Valérie Issarny

INRIA

Joint work with Apostolos Zarras, Christos Kloukinas, Ferda Tartanoglou

Outline

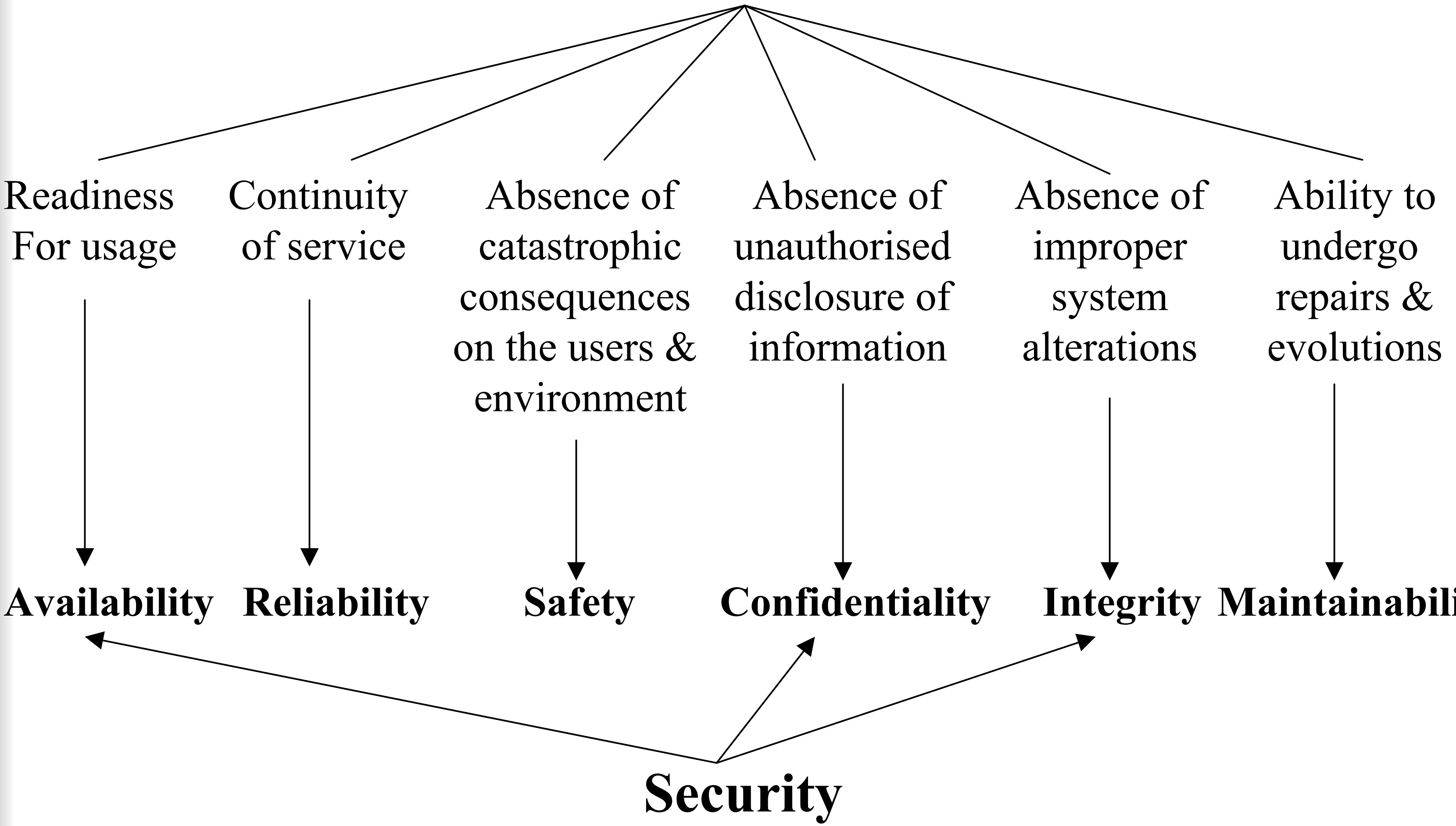
- Dependability concepts
- SA and dependability analysis
- Automated dependability analysis of SA
- Supporting environment
- Supporting the overall development of dependable systems
- Conclusion

Base System Properties

- Functionality
  – System's functional specification
- Usability
- Performance
- Cost
- Dependability

Dependability

Ability to deliver a service that can justifiably be trusted



Absence of unauthorized access to, or handling of, system state

Dependability Provision

*Ability to avoid **failures** that are more frequent or more severe and outage durations that are longer, than is acceptable to the users*

… Failures $\xrightarrow{\text{Cause}}$ **Faults** $\xrightarrow{\text{Activation}}$ **Errors** $\xrightarrow{\text{Propagation}}$ **Failures …**

Faults: Adjudged or hypothesized cause of an **error**

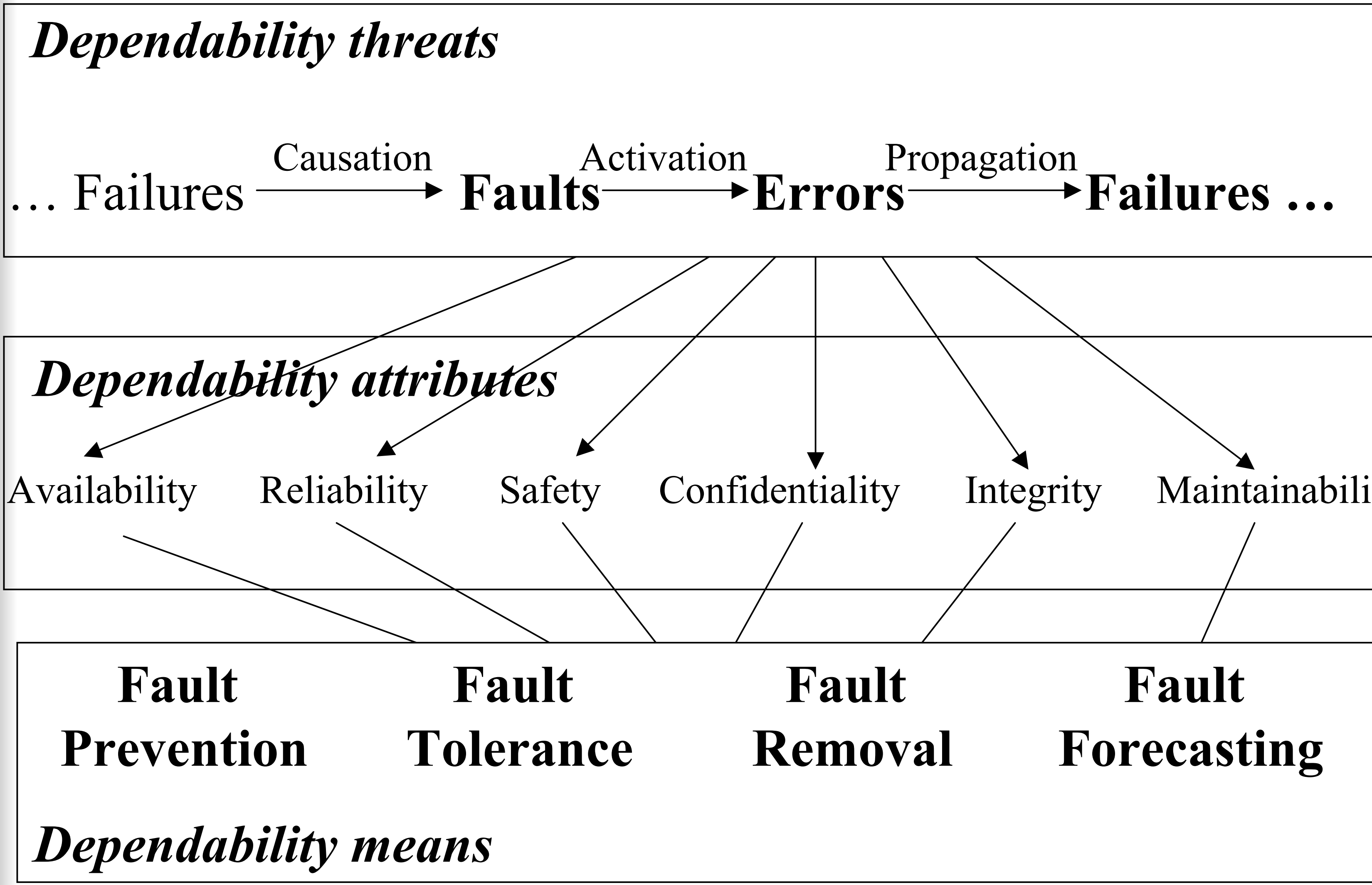
Errors: Part of the system state that *may* cause a subsequent failure

Failures: *Deviation* of the delivered service from correct service, i.e., implementing the system's function

System does not comply with the specification

Incorrect specificatio

Dependability Definitions



Dependability Threats

- **Fault**
  - **Active** when produces an error
    - Internal fault that was previously dormant and is activated by computation process or environmental conditions
    - External fault
  - **Dormant** otherwise

Activation

- **Error**
  - Propagated by the computation process
  - Due to:
    - Activation of an internal fault
    - Occurrence of a physical operational fault
    - Propagation of an error from another system

Propagation

- **Failure**
  - Due to error propagated to the service interface and unacceptably alters the service delivery by the system
  - Causes permanent or transient fault in the system that contains the component, and for the other system(s) that interact with the given system

Dependability Threats - Faults

- Major fault classes
  - Physical faults, design faults, interaction faults
- Phase of creation/occurrence
  - Design *vs* operational
- System boundaries
  - Internal *vs* external
- Dimension
  - Hardware *vs* software
- Phenomenological cause
  - Natural *vs* human-made
- Intention
  - Accidental or deliberate without malice *vs* malicious
- Persistence
  - Permanent *vs* transient

Dependability Threats - Failures

- Symptoms
  - False alarm
  - Degraded service
  - Safe shutdown
  - Signalled failure
  - Crash failure
  - Unsignalled failure
  - Byzantine failure
- Consequences
  - From minor to catastrophic failures
- Domain
  - Value *vs* timing failures

Dependability Threats - Errors

- Detection
  - Latent
  - Detected
- Multiplicity
  - Single
  - Multiple related

Dependability Means – Fault Prevention

- *Preventing the occurrence or introduction of faults*
  - Quality control techniques
    - Rigorous design rules
    - Software
      - Structured programming
      - Information hiding
      - Modularization
      - ➢ Benefit of SA-based development

Dependability Means – Fault Tolerance

- *Delivering correct service in the presence of faults*
  - **Error detection**
    - **Concurrent error detection** during service delivery
    - **Preemptive error detection** when service delivery is suspended

Dependability Means – Fault Tolerance (2)

- **System recovery**
  - Transforms a system state that contains one or more errors and (possibly) faults into a state without detected errors and faults that can be activated again
  - **Error handling** that eliminates errors from the system state
    - **Rollback** based on checkpoint
    - **Compensation** based on redundancy
      - **Fault masking** when systematic
    - **Rollforward** based on new state
  - **Fault handling** that prevents located faults from being activated again
    - **Fault diagnosis** that identifies and records the cause(s) of error(s)
    - **Fault isolation** that performs physical or logical exclusion of the faulty components
    - **System reconfiguration** that reassigns tasks among non-faulty components
    - **System reinitialisation** that enacts the reconfiguration

Dependability Means – Fault Tolerance (3)

- **Fail-controlled systems**
  - **Dependability requirements set the system's specific mode of failures**
    - **Halting failures only**
      - **Fail-halt**
      - **Fail-silent**
    - **Minor failures**
      - **Fail-safe**

Dependability Means – Fault Removal

- *Reducing the number or severity of faults*
  - Verification
    - Static
      - Static analysis
      - Theorem proving
      - Model checking
    - Dynamic
      - Symbolic execution
      - Testing
  - Diagnosis
  - Correction

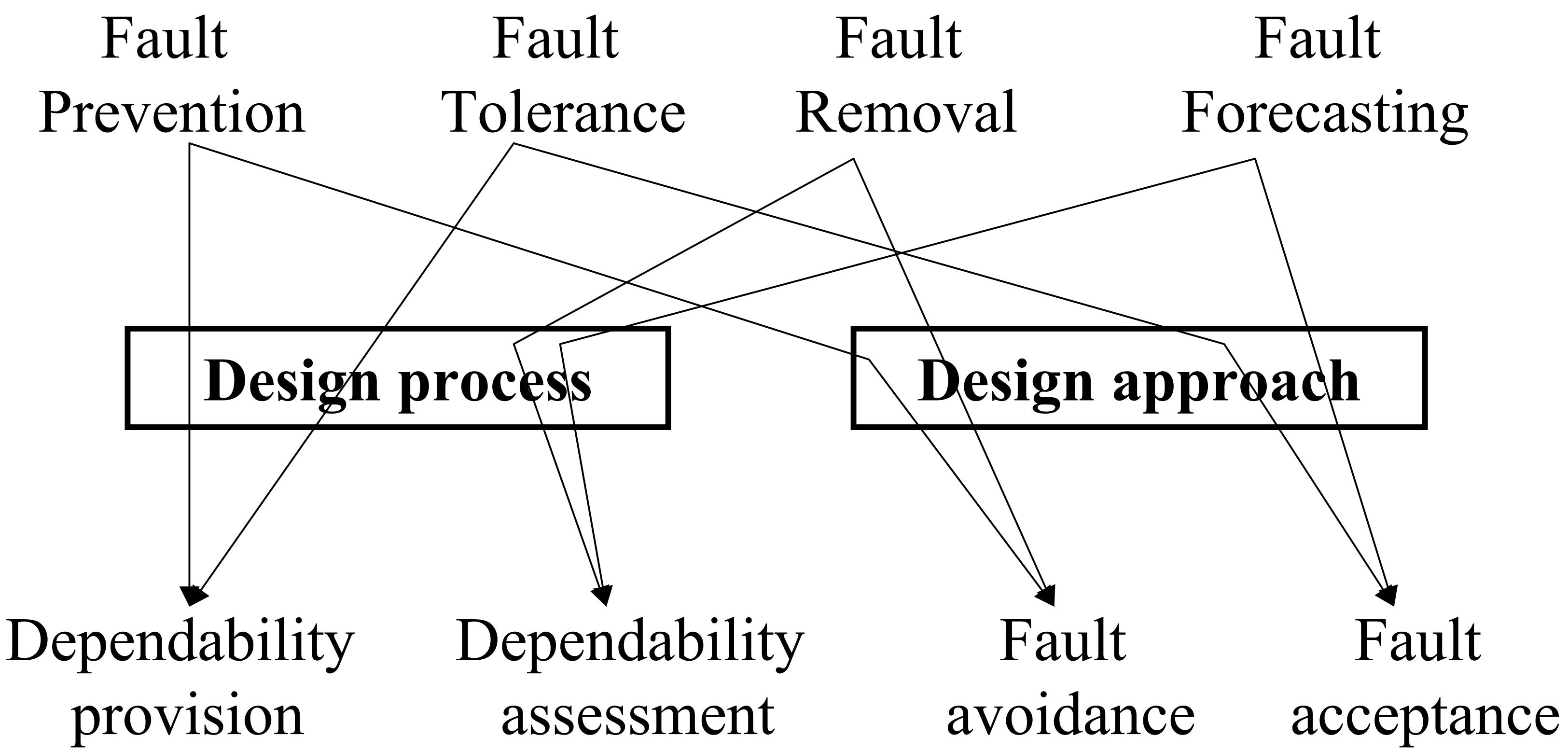
Dependability Means – Fault Forecasting

- *Estimating the present number, the future incidence, and the likely consequences of faults*
  - Qualitative/ordinal evaluation
    - Failure mode and effect analysis
    - Reliability block diagrams
    - Fault trees
  - Quantitative/probabilistic evaluation
    - Markov chains
    - Stochastic Petri nets
    - Reliability block diagrams
    - Fault trees

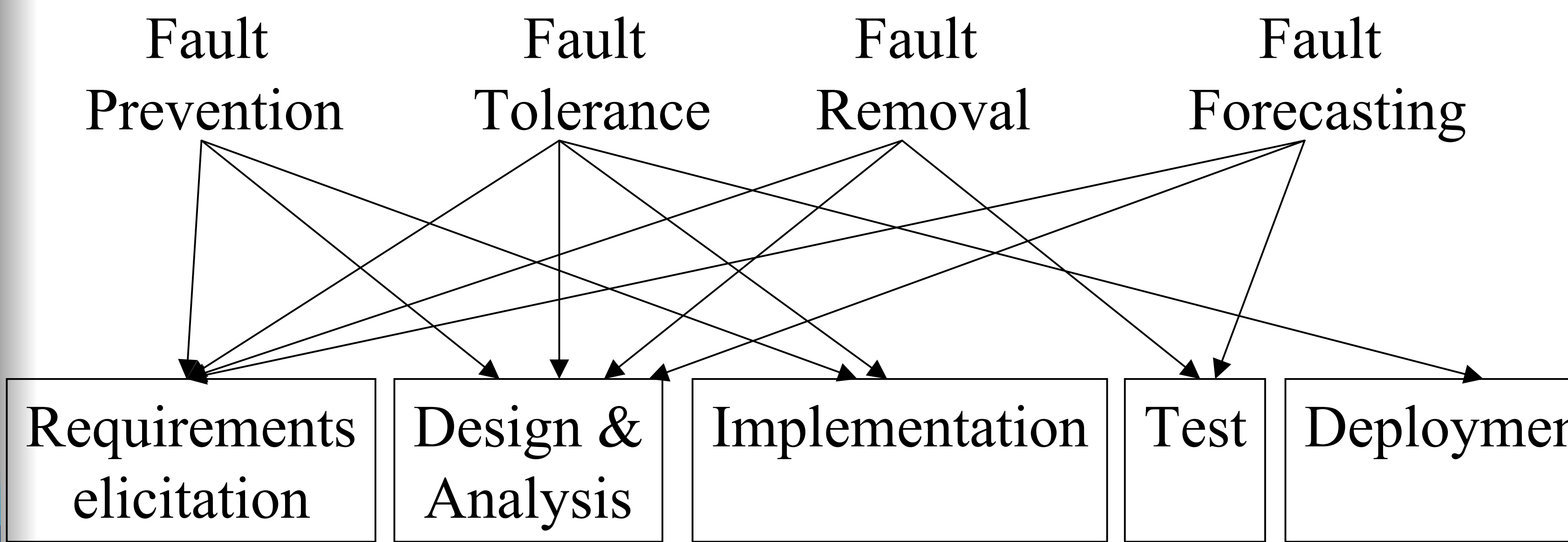
Dependability Attributes - Assessment

- **Reliability**
  - *Measure of the continuous delivery of correct service (time to failure)*
- **Availability**
  - *Measure of the delivery of correct service wrt alternation of correct and incorrect service*
- **Maintainability**
  - *Measure of the time to service restoration since the last failure occurrence (measure of the continuous delivery of incorrect service)*
- **Safety**
  - Extension of reliability: reliability wrt catastrophic failure with safe state as the grouping of the sates of correct service and incorrect service due to non-catrastrophic failure
  - *Measure of continuous safeness (time to catastrophic failure)*

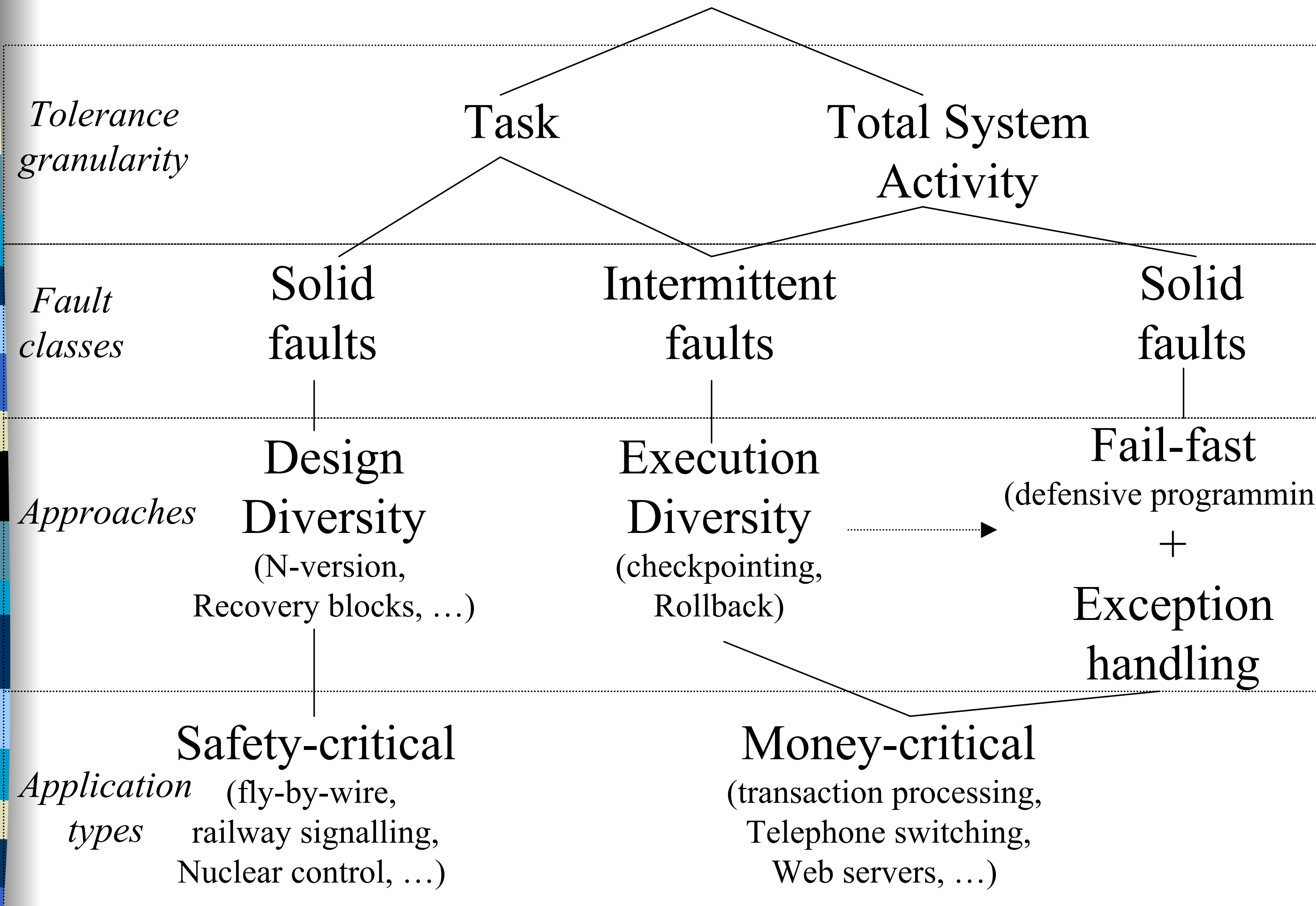
Dependability Means and System Design



Dependability Means & System Development



Software Fault Tolerance



Impact of Dependability on SA Design

- Dependability provision & assessment
  - Rigorous design
    - Fault prevention
    - Fault tolerance via integration of dedicated components
    - Fault removal
  - Dependability analysis
    - Evaluating dependability attributes
      - Fault removal
      - Fault forecasting

Outline

- Dependability concepts
- SA and dependability analysis
- Automated dependability analysis of SA
- Supporting environment
- Supporting the overall development of dependable systems
- Conclusion

Context

- Nowadays, business industry and society tend to place increasing **dependence** on their systems
  - Systems consist of numerous disparate and autonomous component systems
- Systems users have ever-increasing non-functional requirements on the quality of the systems
  - Performance, reliability, availability, etc.

Context (2)

- Assessing systems quality against users' requirements involves performing quality analysis during the lifetime of the systems
- Quality analysis, is not a new challenge:
  - Analytic, simulation, measurement-based techniques do exist
  - Models of system behavior serve as input to those techniques (Markov-chains, Petri-nets, queuing-nets etc.)
  - Values of the quality attributes are produced by solving, simulating the models

Context (3)

But...

- Developing good quality models in general is not an easy task and **requires experience in formal methods**
- Everyday developers:
  - Are generally very experienced in using ADLs and OO methods for designing and developing a system
  - Are not keen on using formal methods for verifying/assessing/evaluating the system

Context (4)

Hence...

- The ideal would be to provide systems developers with an environment that:
  - **Enables the design of Dependable Systems (DS) architectures**
  - **Provides tool support that facilitates the specification of formal models for DS quality analysis**
- The previous are our main directions towards a ADL-based developer-oriented environment for DS quality analysis

Modeling DS Architectures using ADL

- Component
  - Unit of data or computation
  - Characterized by
    - Interface, type, properties
- Connector
  - Interaction protocol
  - Characterized by
    - Interface, type, properties
- Configuration
  - Assembly of components and connectors
  - Possibly constrained

ADL and Dependability Analysis

- Base approach [Klein *et al*. 99]
  - Attribute-Based Architectural Style (ABAS) that provides modeling support for the analysis of a particular quality attribute
- Dependability-oriented ABAS
  - Quality attribute measures
    - Measure associated with dependability attributes
  - Quality attribute stimuli
    - Failures affecting the value of the dependability attributes
  - Quality attribute properties
    - Architectural properties affecting the value of the dependability attributes, e.g., faults, redundancy
  - Quality attribute models
    - Models that relate the above elements for measuring dependability attributes, e.g., state space model

Need for Automated Analysis

- Base approach [Kazman *et al*. 00]
  - Architecture Tradeoff Analysis Method (ATAM) that couples ABAS with a scenario for analyzing quality attributes
  - Expertise in formal models required
  - Development of quality analysis models takes about 25% of the time
- Automated generation of quality attribute models from architectural descriptions
  - ABAS for automated dependability analysis

Outline

- Dependability concepts
- SA and dependability analysis
- Automated dependability analysis of SA
- Supporting environment
- Conclusion

Automated Dependability Analysis of SA

- ADL support
  - Scenario/collaboration
  - Specifying dependability measures, stimuli and properties associated with architectural elements
- Dependability models
  - Block diagrams
  - State space models
- Automated generation of state space models

Dependability Measures

- Reliability measure
  - Probability that the system provides correct service for a given time period
- Availability measure
  - Probability that the system provides correct service at a given moment in time
- Safety measure
  - Probability that there will be no catastrophic failure for a given period of time

Dependability Stimuli: Specification of Failures

Features	Range	Architectural element
Domain	Time/Value	Component/ Connector/ Node
Perception	Consistent/ Inconsistent	

Dependability Properties: Specification of Faults

Features	Range	Architectural element
Nature	Intention/Accident	Component/
Phase	Design/Operational	Connector/
Causes	Physical/Human	Node
Boundaries	Internal/External	
Persistence	Permanent/Temporary	
Arrival-rate	Real	
Active-to-benign	Real	
Benign-to-active	Real	
disappearance	Real	

Dependability Properties: Specification of Redundancy

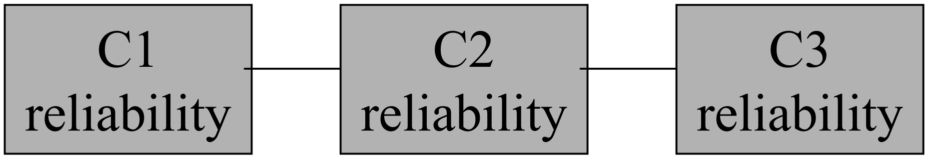
Features	Range	Architectural element
Error-detection	Vote/Comparison/ Acceptance	Component
Execution	Parallel/Sequential	
Confidence	Absolute/Relative	
Service-delivery	Continuous/Suspended	
No-comp-faults	Integer	
No-node-faults	Integer	

Automated Dependability Analysis of SA

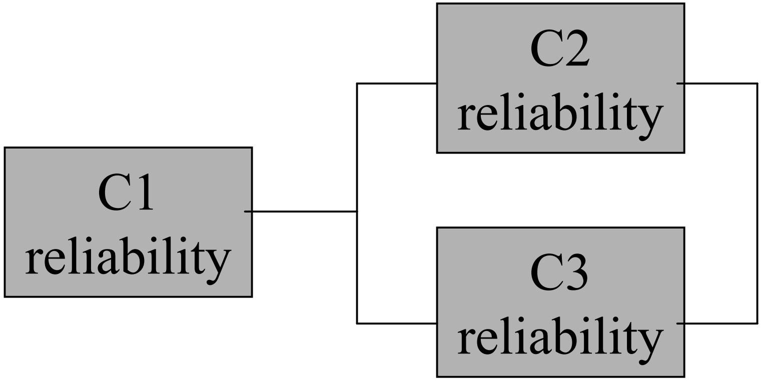
- ADL support
  - Scenario/collaboration
  - Specifying dependability measures, stimuli and properties associated with architectural elements
- Dependability models
  - Block diagrams
  - State space models
- Automated generation of state space models

Block Diagrams

- Graphical representation of constraint-to-succeed
  - Serial: C1 ^ C2 ^C3



  - M-out-of-N connections: C1 ^ (C2 v C3)



Block Diagrams – Reliability Evaluation

- Serial connection
  - BD.reliability = P(C1^C2^C3)
  - P(C1^C2^C3) = C1.reliability*C2.reliability*C3.reliability
- M-out-of-N connections
  - BD.reliability = P(C1 ^ (C2 v C3))
  - P(C1 ^ (C2 v C3)) =

    C1.reliability*C2.reliability +

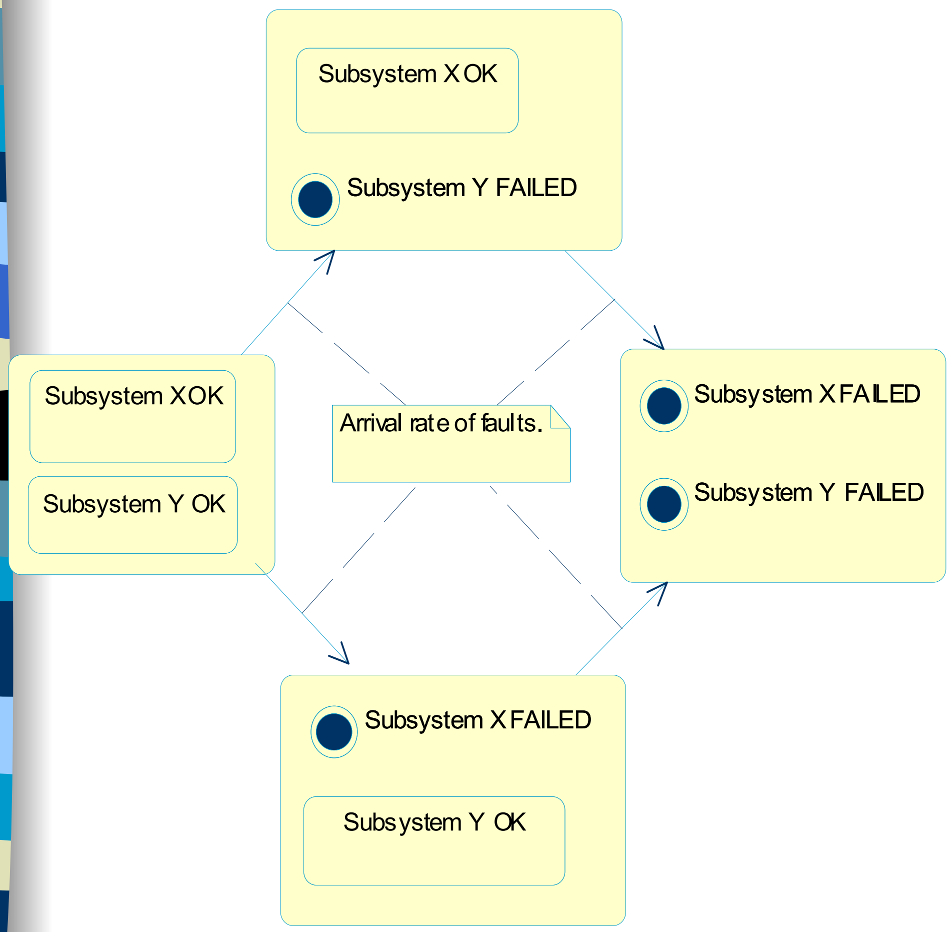
    C1.reliability*C3.reliability –

    C1.reliability*C2.reliability*C3.reliability

Block Diagrams – Limitation

- Based on static descriptions of the components needed for correct service provisioning
- Does not account for dynamic aspects
  - Transient/intermittent faults
  - Repairable systems
- Does not account for dependent failures

State Space Models



- A set of transitions between states of the system
- A state represents a situation where either the system works correctly, o not (i.e. a *death state*)
- A state of the system depends on the state of it constituent elements.
- A state is constrained by the range of all possible situations that may occur

State Space Models – Dependability Evaluation

- The specification of large state-space models is often too complex and error-prone
- Instead of all possible state transitions specify:
  - The state range of the system
  - Transition rules between sets of states of the system
  - The initial state of the system
  - A death state constraint
- An existing algorithm can be applied to generate a complete state space model
  - Start from the initial state
  - Apply recursively the transition rules:
    - During a recursive step, produce a transition to a state derived from the initial one
    - If the resulting state is a death state, end the recursion

Automated Dependability Analysis of SA

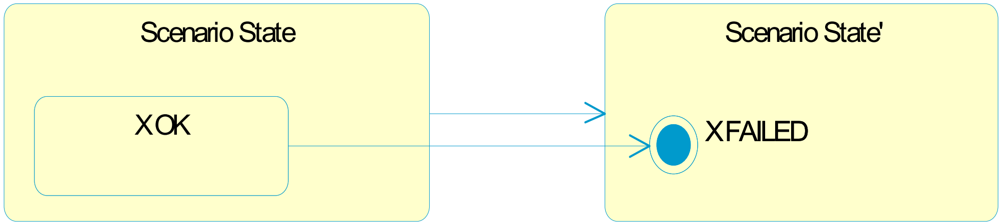
- ADL support
  - Scenario/collaboration
  - Specifying dependability measures, stimuli and properties associated with architectural elements
- Dependability models
  - Block diagrams
  - State space models
- Automated generation of state space models

Generating the State Range for a Scenario

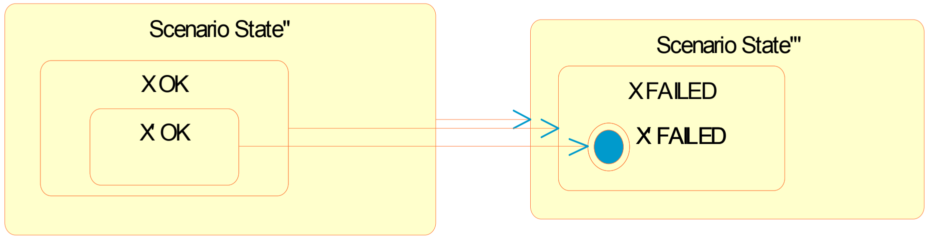
- The state of a collaboration is composed of:
  - The states of the component, connector instances used
  - The state of nodes on top of which the component instances execute
  - If a component is composite, its state is composed of the states of the constituent elements

- The range of states for a component/connector/node depends on the kind of faults that may cause failures and the kind of redundancy used.
  - For permanent faults, a component/connector/node may be OPERATIONAL/FAILED.
  - For intermittent faults, a component/connector/node may be OPERATIONAL/FAILED-ACTIVE/FAILED-BENIGN.

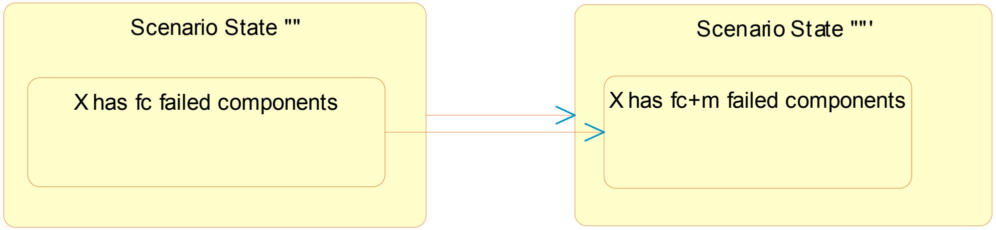
Generating the Transition Rules – Pattern for Permanent Fault



- Primitive component



- Composite component



- Redundancy schema

Generating the Initial State and Death State Constraints

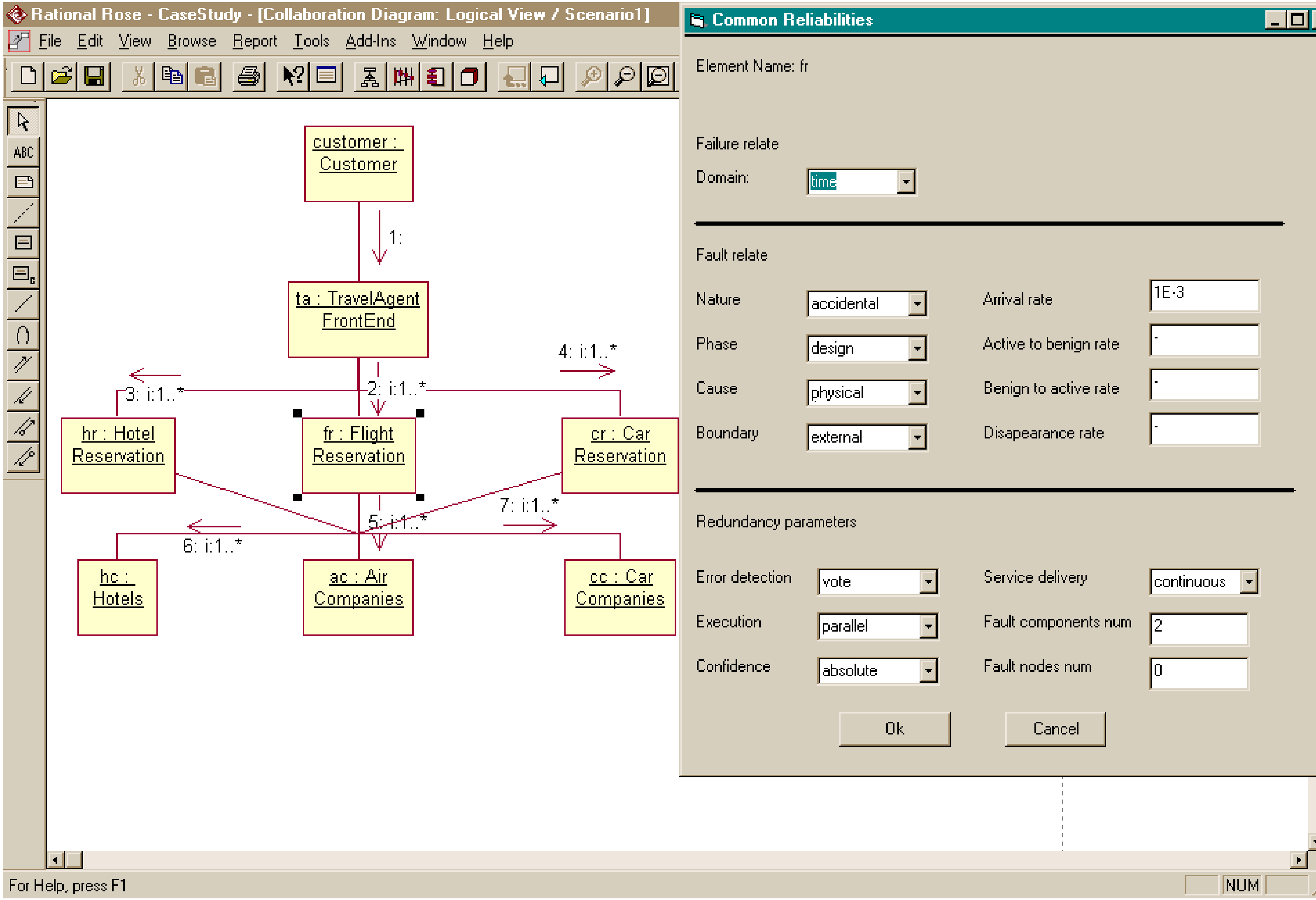
Generate the initial state definition…

- The initial state is a state where all of the elements used in the scenario are operational

Generate the death state constraint…

- The death state constraint consists of the disjunction of base predicates.
- Each predicate defines the death state constraint for an individual element used:
  - For a component, connector, or a node; states that the element is in a FAILED state
  - For a redundancy schema; is the disjunction of two predicates:
    - The first states that the number of failed redundant component instance is greater than the number component faults that can be tolerate
    - The second states that the number o failed redundant nodes is greater tha the number of node faults that can b tolerated

Example



Rational Rose - CaseStudy - [Collaboration Diagram: Logical View / Scenario1]
File Edit View Browse Report Tools Add-Ins Window Help
customer : Customer
1:
ta : TravelAgent FrontEnd
3: i:1..*
2: i:1..*
4: i:1..*
hr : Hotel Reservation
fr : Flight Reservation
cr : Car Reservation
5: i:1..*
6: i:1..*
7: i:1..*
hc : Hotels
ac : Air Companies
cc : Car Companies
For Help, press F1
NUM
Common Reliabilities
Element Name: fr
Failure relate
Domain: time
Fault relate
Nature accidental
Phase design
Cause physical
Boundary external
Arrival rate 1E-3
Active to benign rate
Benign to active rate
Disapearance rate
Redundancy parameters
Error detection vote
Execution parallel
Confidence absolute
Service delivery continuous
Fault components num 2
Fault nodes num 0
Ok
Cancel

Example – Reliability Properties

Components

- May fail due to permanent design faults.
- The fault arrival rates of the legacy systems is much smaller compared to the fault arrival rates of the new components.
- Add-hoc redundancy is present.
  - From n legacies, we need at least 1 answer to be successful.
- NVP redundancy is used for the realization of the TA components.

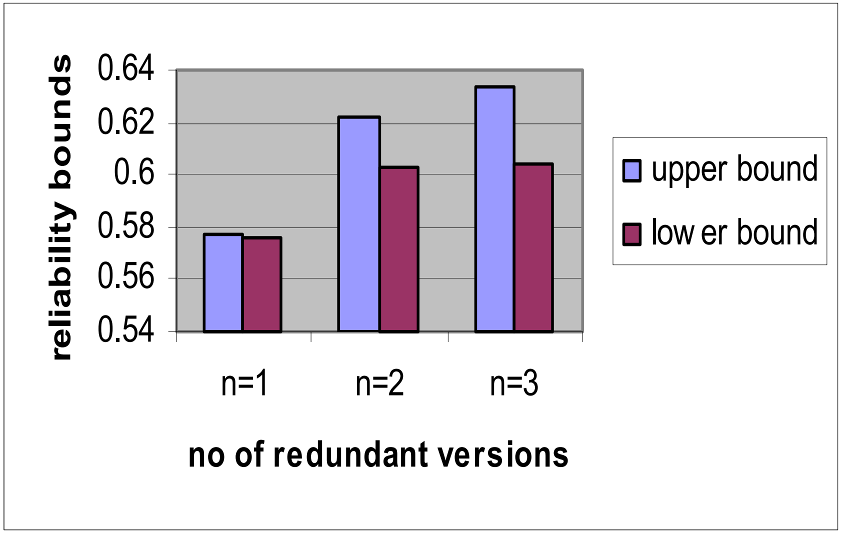
Nodes

- May also fail due to permanent faults.
- The arrival rates are much smaller than those of components.

Connectors

- May fail due to temporary faults that disappear with a certain rate.
- The fault arrival rate of the RPC connector is systems is much less compared to the one of the HTTP connector.

Example – Reliability Evaluation



- Reliability gets better while increasing the number of different versions realizing DS components.
- Improvement is not spectacular.
  - HTTP connectors are the main sources causing the reliability measure to have small values.
- However, the cost of realizing multiple versions is small.

Outline

- Dependability concepts
- SA and dependability analysis
- Automated dependability analysis of SA
- Supporting environment
- Supporting the overall development of dependable systems
- Conclusion

Modeling DS Architectures

- Two approaches quite popular in various different communities
  - Architecture Description Languages
  - OO methods like OMT, and recently UML
- ADLs offer notations enabling the rigorous specification of the structure and behavior of DS
  - Components/Connectors/Configurations.
  - Tools that ease the analysis and the construction of DS architectures.
  - ADLs are most popular in the academic community.
  - Industrials prefer using OO notations for the specification of system architectures.

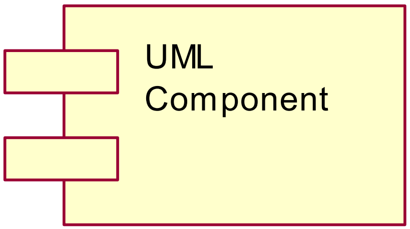
Modeling DS Architectures (2)

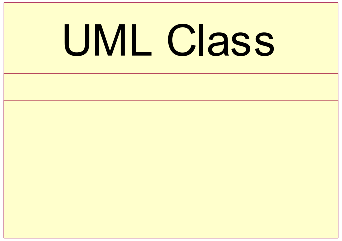
- UML is becoming an industrial standard notation for the definition of a family of languages (i.e., UML profiles) for modeling software.
  - Basic concern regarding the imprecision of the semantics of UML.
  - To increase the impact of ADLs in the real world, and to decrease the ambiguity of UML, **we propose an ADL defined in relation to standard UML elements.**

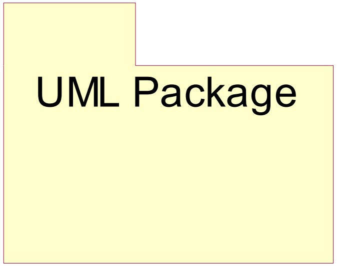
Extensible UML-based ADL

- Identify standard UML element(s), whose semantics are close to the ones needed for the specification of ADL components, connectors and configurations.
- If the semantics of the identified element(s) do not exactly match the ones needed for the specification of components, connectors, and configurations, extend them properly and define a corresponding UML stereotype(s).
- A UML stereotype is a UML element whose base class is a standard UML element. Moreover, a stereotype is associated with additional constraints and semantics.

ADL – Component Definition

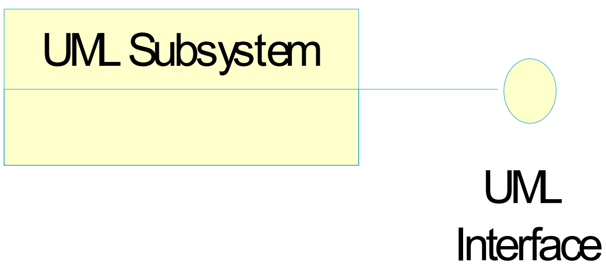






- UML Component: too concrete, i.e., an executable software module.
- Does not directly support the hierarchical composition of systems. A class may be composite but a class definition does not include the interrelationships among the constituent classes.
- Can not be instantiated, or associated with other packages.

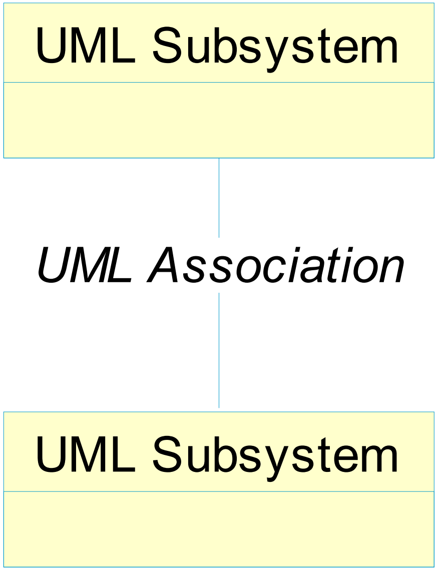
ADL – Component Definition (2)



- Subtype of both the Package and the Classifier elements
  - Can be instantiated
  - Can be Associated
  - It can contain other subsystems
  - It can contain models showing the relationships among the constituent subsystems

We define an ADL component as a stereotyped UML Subsystem, that may provide and require standard UML interfaces.

ADL – Connector Definition



- Connectors represent the protocols through which components may interact
- UML Association is a natural choice to model connectors in UML
  - A connector role corresponds to an association end

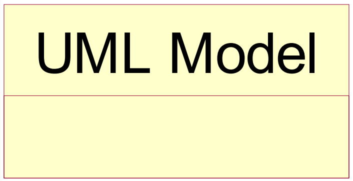
We define a connector as a stereotyped UML association characterized by a non-empty set of interfaces, named "Interfaces", representing the specific parts of components' functionality playing the roles.

ADL – Connector Definition (2)

- In practice, connectors are built from architectural elements, including components and more primitive connectors
  - E.g. A CORBA connector is typically made out of a base ORB connector and components representing CORBA services
- UML associations can not be composed of other model elements
- However, UML provides a semantic element called Refinement for defining the mapping between an element X and an element Y derived by X

To support the hierarchical composition of connectors, we define a stereotype, whose base class is the standard UML Refinement element and is used to define the mapping between a connector and a composite component that realizes the connector

ADL – Configuration Definition



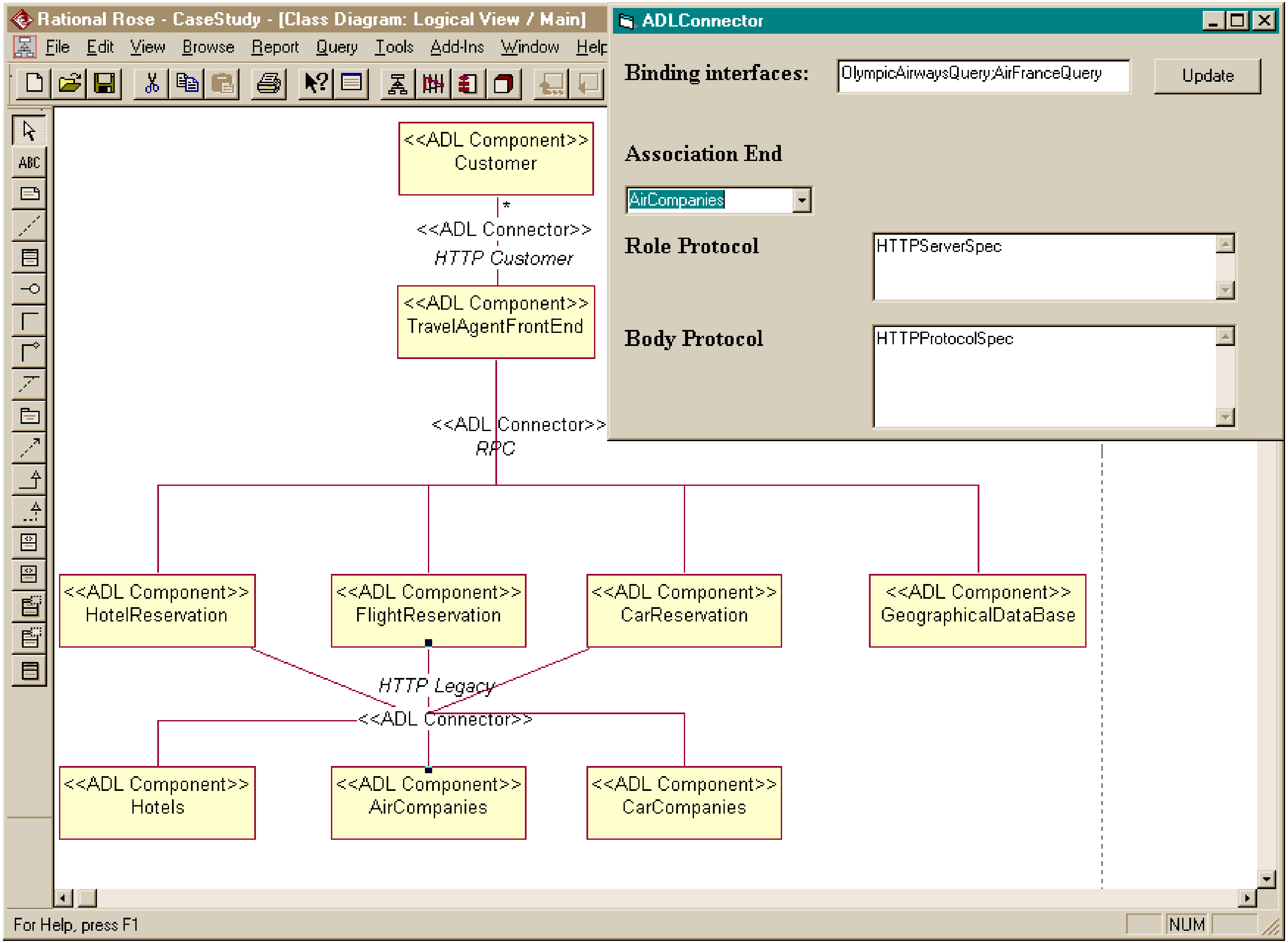
- An assembly of components and connectors
- In UML, the assembly of model elements is specified by a model

We define a configuration as a UML model, consisting of a containment hierarchy where the top-most package is a composite ADL component

Design Tools

- Rational Rose tool for the graphical specification of software architectures.
  - Implemented an add-in that eases the specification of architectural descriptions using the stereotypes discussed so far.
  - Use of an existing add-in to generate XML textual specs form ADL specs. XML specs serve as input to other tools integrated in our environment.
  - Implemented in OCAML a verifier of OCL constraints.

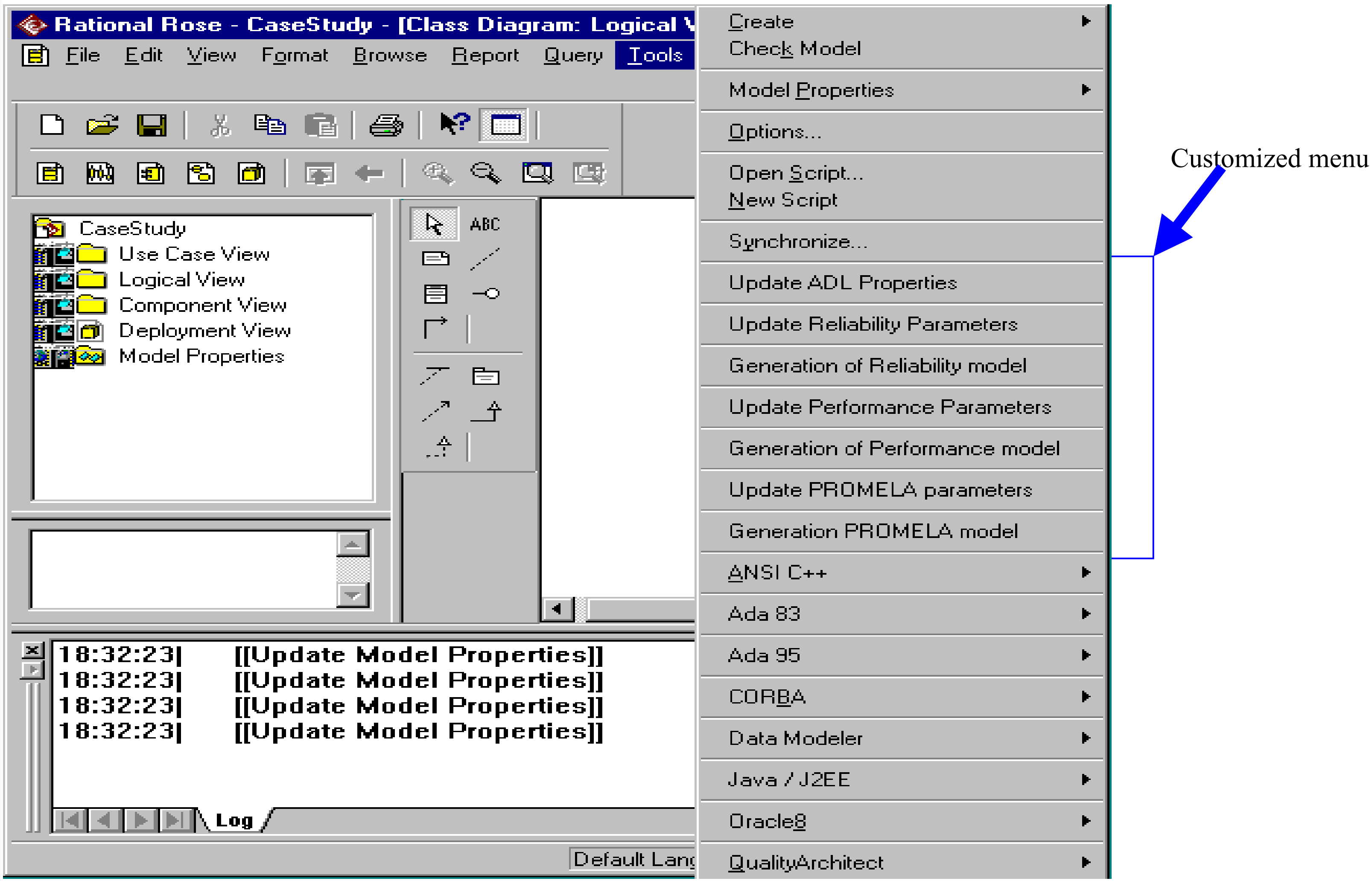
Example: The TA Case Study



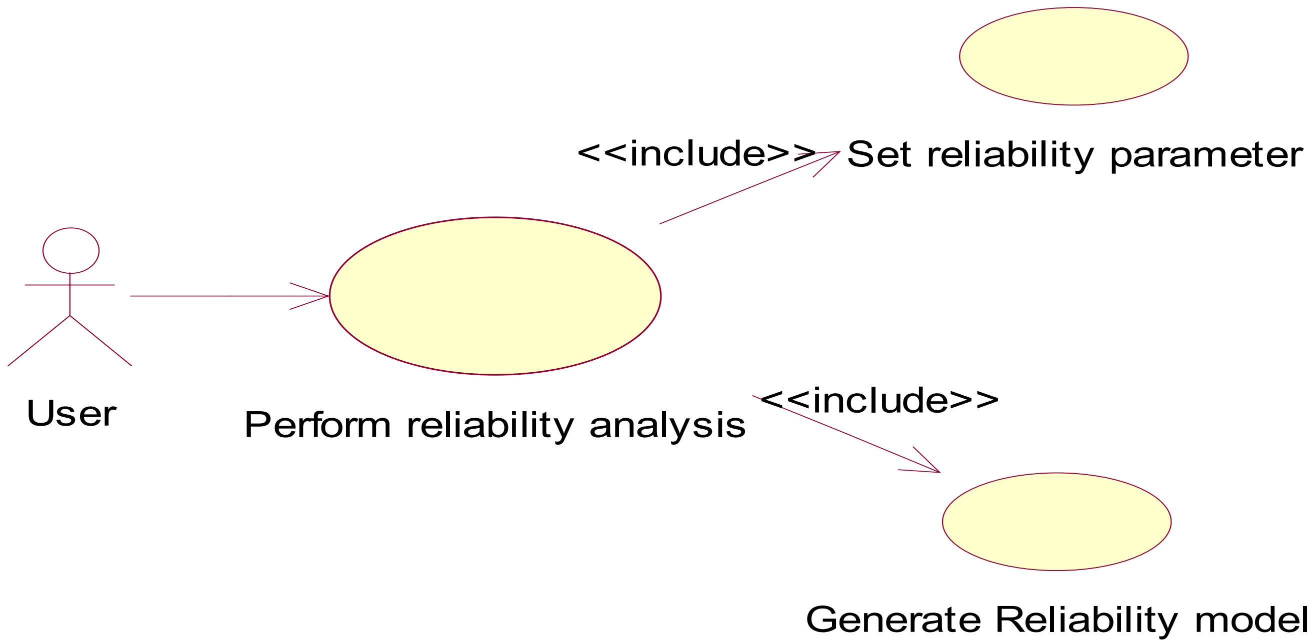
Dependability Analysis

- ADL customization for the specification of dependability measures/stimuli/properties
- Dependability analysis tools
  - Automated generation of dependability models from architectural descriptions
    - The procedure for generating state space models from architectural specifications can be used with any reliability analysis tool accepting as input a state space model.
  - Evaluation using SURE/ASSIST
    - Calculates reliability bounds given a state space model
      - Comes from NASA
      - Available for free
      - Highly rated compared to other reliability analysis tools

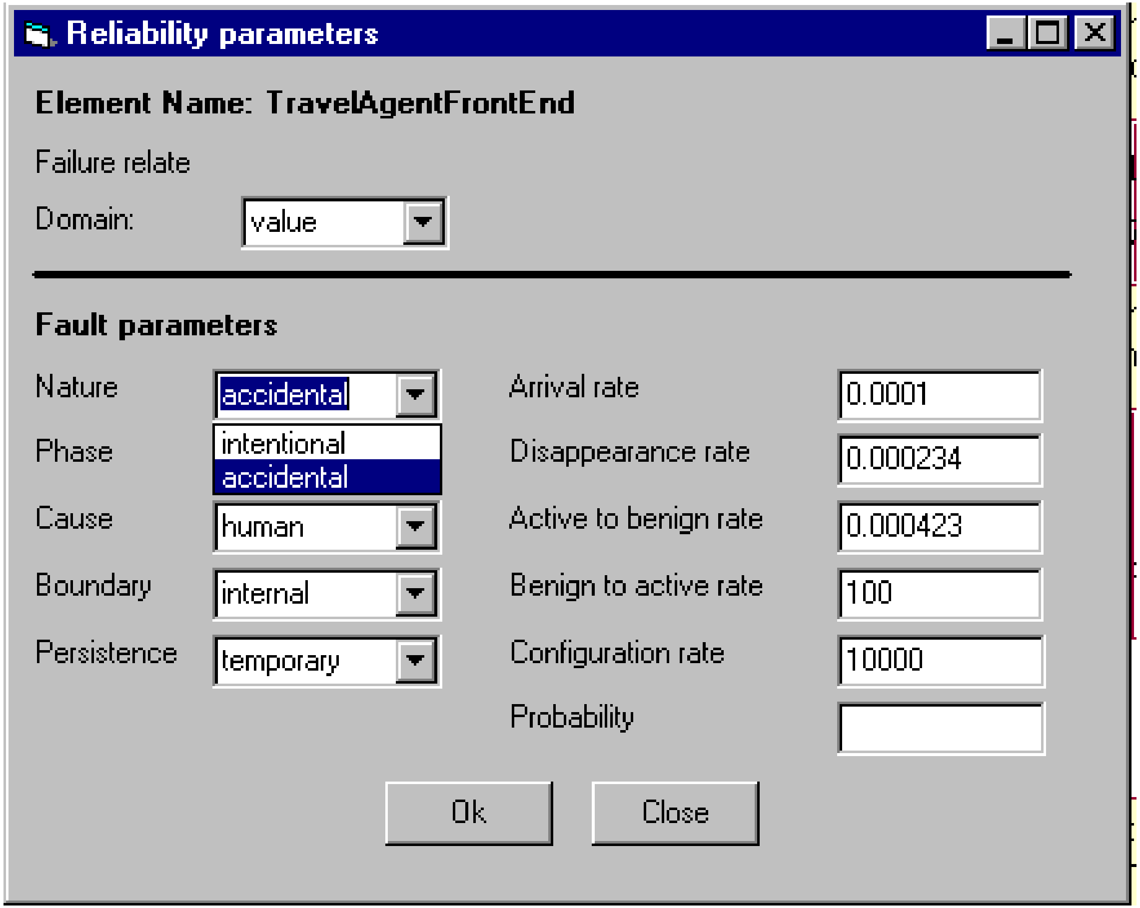
Dependability Analysis Tools



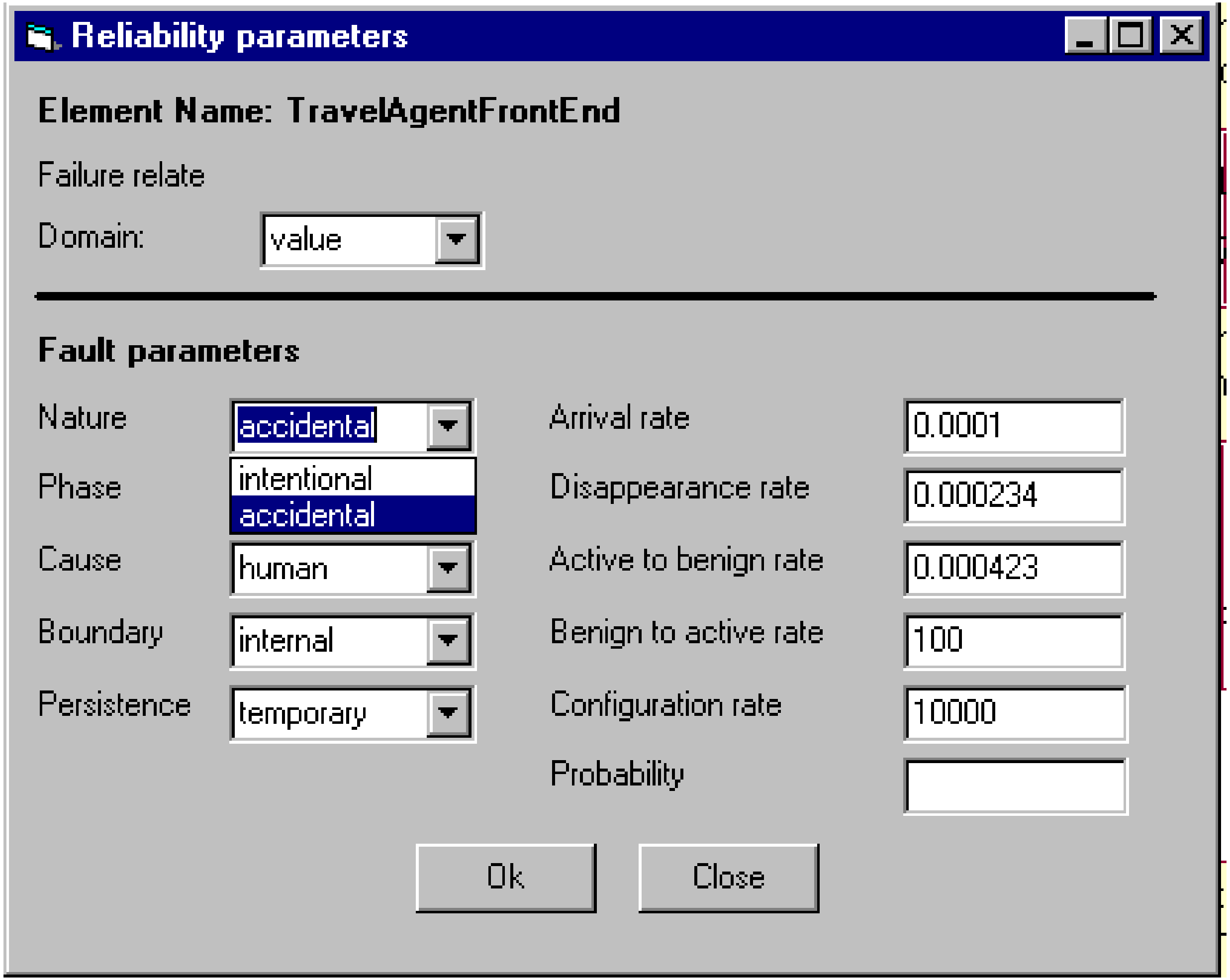
Add-ins for Quality Analysis



Setting Reliability Parameters



Setting Reliability Parameters (2)



Setting Reliability Parameters (3)

Accepted Fault Types

List of accepted faults :

Nature		Origin						Persistence		Usual labelling
		Phenomenological cause		System Boundaries		Phase of creation				
Accidental Faults	Intentional Faults	Physical faults	Human faults	Internal	External	Design faults	Operational faults	Permanent Faults	Temporary Faults	
X		X		X			X	X		*Physical faults*
X		X			X		X	X		
X		X			X		X		X	*Transient faults*
X		X		X			X		X	*Intermittent faults*
X			X	X		X			X	
X			X	X		X		X		*Design faults*
X			X		X		X		X	*Interaction faults*

Errornous Elements:

hr
fr
cr
ac
cc
tacustomer_Con

Close Window

Outline

- Dependability concepts
- SA and dependability analysis
- Automated dependability analysis of SA
- Supporting environment
- Supporting the overall development of dependable systems
- Conclusion

Architecture-based Development of Complex Software Systems

- **Benefits *wrt* systems robustness**
  - Methods and tools supporting analysis, and the mappings of architectures to their implementations
- Focus is on the standard behaviour of the software systems

Supporting the Development of Dependable Systems

- **Crucial to account for the occurrence of failures in architecture-based development**
  - Application-transparent fault tolerance using middleware infrastructures
    - Provide base services for managing failure detection & error recovery
    - ➢ Customized middleware architectures *wrt* composed services

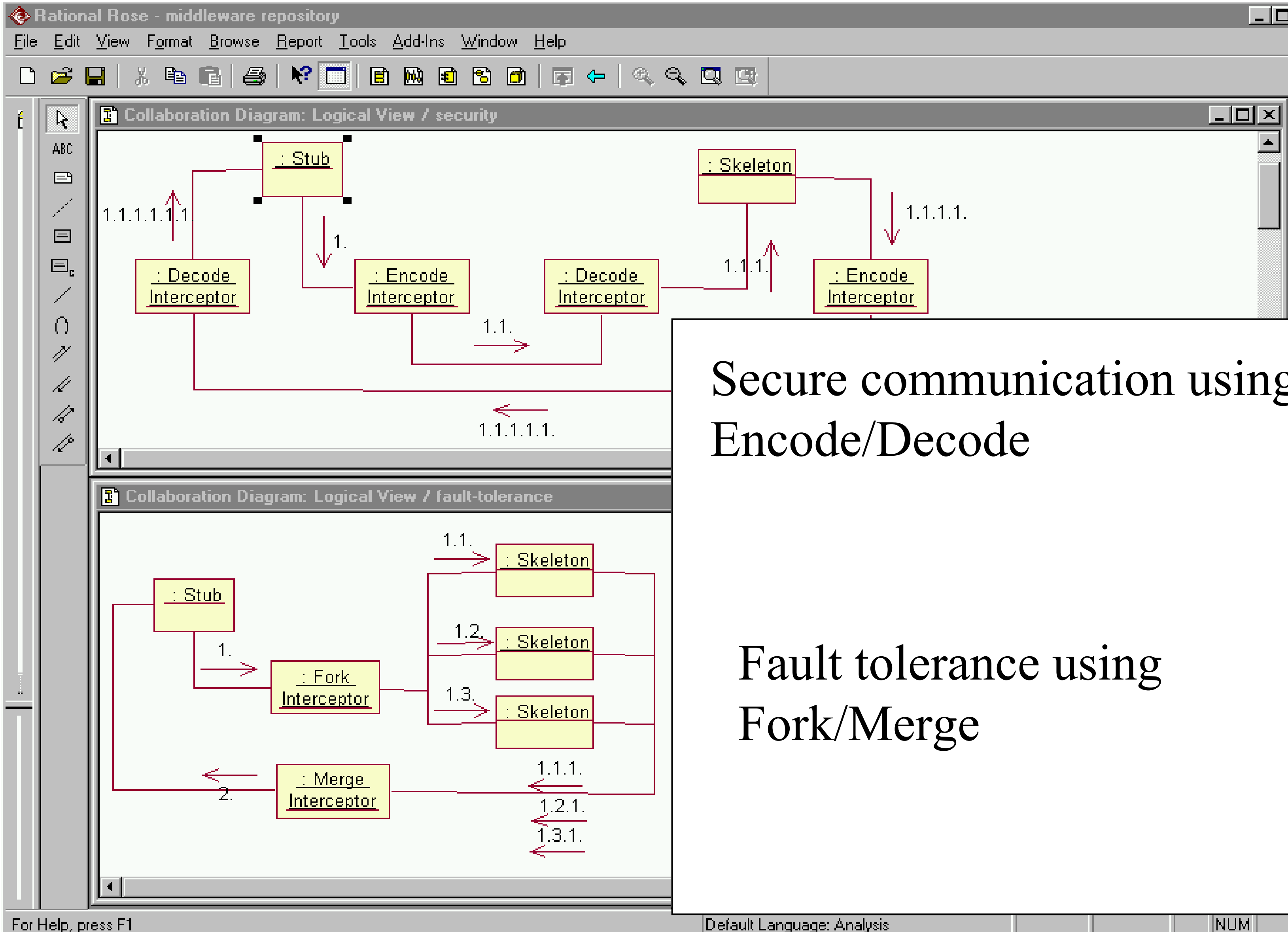
Aiding the Development of Middleware Architectures

- **Middleware infrastructures**
  - Customized composition of services through component-based middleware containers
  - Still, there is the need of supporting the development of containers
    - Right composition of services
    - Achieved quality

Systematic Composition of Middleware Architectures

- **A supporting environment** [CACM 06/02]
  - ADL for modeling middleware architectures
    - Repository of architectural descriptions of middleware infrastructures
  - Automated support for:
    - Composing middleware services
    - Analyzing the quality of composed architectures

Example



Secure communication using Encode/Decode

Fault tolerance using Fork/Merge

Composing Middleware Services

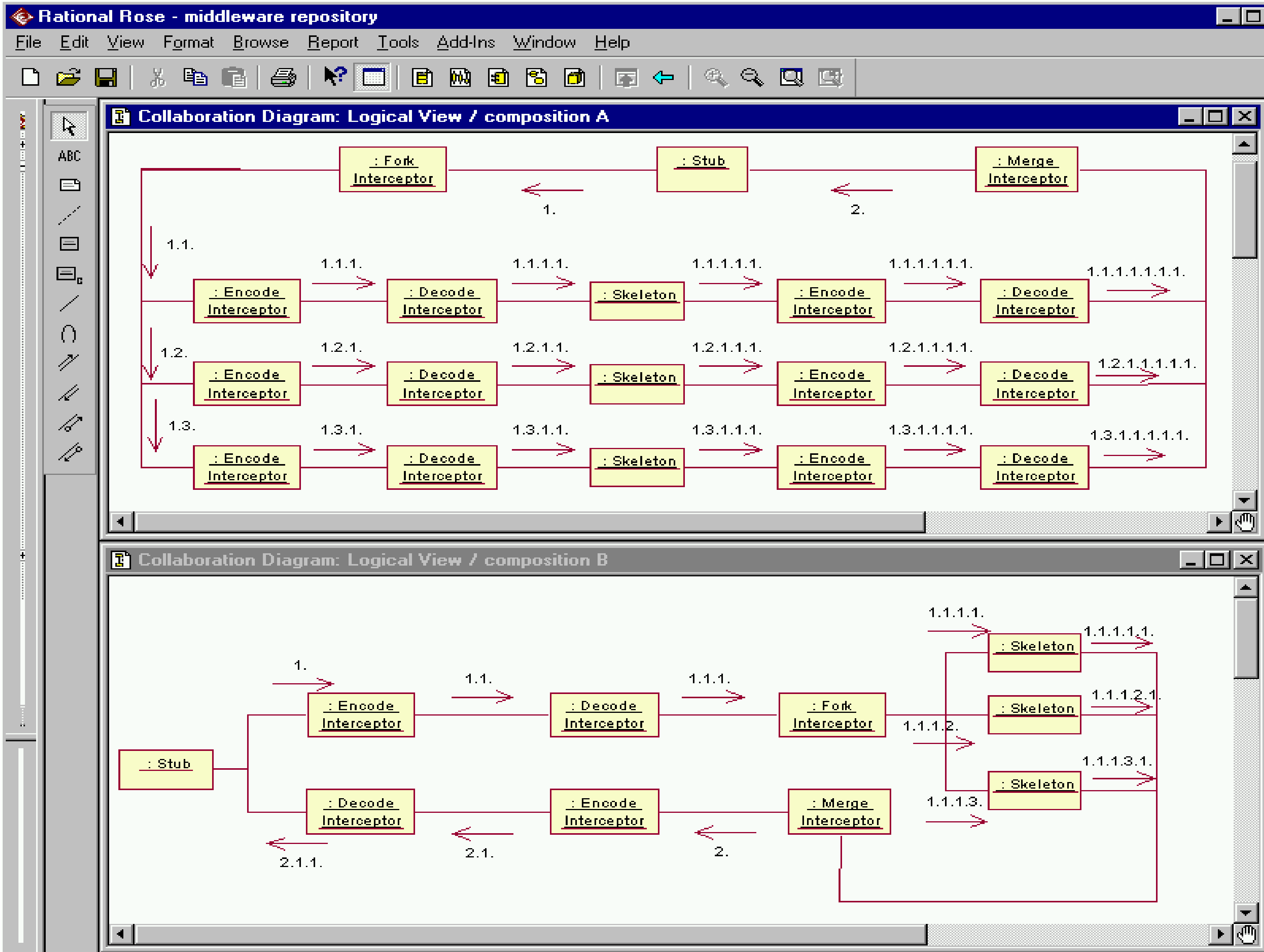
- **Approaches to architecture composition**
  - Horizontal = parallel composition [Qian *et al.*, 95]
    - Secure communication // multi-cast communication
  - Serial composition for linear architectures [Steffen & Beec 97]
    - FT architecture is not linear
  - Explicit interposition [Spitznagel & Garlan, 01]

➢ Need for an automatic solution to identify valid interpositions of components

Automating Composition

- **Solution** [WICSA'01]
  - Composition through model checking
  - Constrain composition through structure
- **Additional benefits**
  - Allows identifying unexpected compositions
  - Allows understanding interaction of qualities

Example



Rational Rose - middleware repository
File Edit View Format Browse Report Tools Add-Ins Window Help
Collaboration Diagram: Logical View / composition A
: Fork Interceptor
: Stub
: Merge Interceptor
1.
2.
1.1.
1.2.
1.3.
: Encode Interceptor
: Decode Interceptor
: Skeleton
1.1.1.
1.1.1.1.
1.1.1.1.1.
1.1.1.1.1.1.
1.1.1.1.1.1.1.
1.2.1.
1.2.1.1.
1.2.1.1.1.
1.2.1.1.1.1.
1.2.1.1.1.1.1.
1.3.1.
1.3.1.1.
1.3.1.1.1.
1.3.1.1.1.1.
1.3.1.1.1.1.1.
Collaboration Diagram: Logical View / composition B
1.1.1.1.
1.1.1.1.1.
1.1.1.2.1.
1.1.1.2.
1.1.1.3.1.
1.1.1.3.
1.1.1.
1.1.
2.1.1.
2.1.

Assessment

- **Making systems dependable is eased by middleware infrastructures**
  - Infrastructures offer base supporting services
  - Service composition may be automated
- **But…**
  - Allows only for backward error recovery and cannot cope with all failures
- Need for complementary application-specific forward error recovery
  - Exception handling as it is the most general mechanism

Architecture-based Exception Handling

- **Exception handling mechanisms**
  - Serves implementing the system's exceptional specification (definition of exceptions & handlers)
  - Relies on some model (e.g., termination, resumption)
- **Existing mechanisms are for handling exceptions within components**

➢ What about exception handling requiring changes to the architecture [HICSS'01]

Base Solutions to Architectural Exception Handling

- **Exception handling within ADL**
  - Limited to the specification of signalled/handled exceptions within the definition of component/connector interfaces
  - Behavioural specification would further improve correctness checking
    - Pre/post as supported by Inscape [Perry, 89]
    - ➢ Issue of taking into account the exception handling model

Base Solutions to Architectural Exception Handling (2)

- **Dynamic reconfiguration**
  - Determined at runtime
    - Reconfiguration manager
    - Possibly constrained based on invariant on the system structure
  - Fixed at design time
    - Specified in the architecture description (e.g., Durra [Barbacci *et al*., 93])
    - Independent of exception handling

Exception Handling Model

- **Exception handling within components and connectors**
  - Let exceptions flow among the architectural elements according to the embedding architectural style
- **Exception handling at the architecture level**
  - To enable changing the running configuration

Impact on Architecture Description

- **Support for internal exception handling**
  - Specification of exceptions raised/handled by the elements
- **Support for architectural exception handling**
  - Definition of configuration exceptions and associated handlers using the ADL
    - Keep abstract the description of architectures for the sake of analysis and synthesis
  - Mapping to implementation using a service for dynamic reconfiguration

Assessment

- **Architecture-based development can aid in the construction of dependable systems**
  - Application-transparent fault tolerance: systematic aid in the design of customized middleware architectures
  - Application-specific fault tolerance: support for exception handling at the architectural level
- **But…**
  - Existing support is mainly aimed at closed systems

➢ Need solutions for open systems

Towards Dependable Open Systems

- **Issues in the development of open systems**
  - Composition of autonomous systems
  - Highly dynamic systems
    - Mobility,
    - Evolution,
    - …

Towards Dependable Open Systems

- **Ongoing work**
  - Architecting open systems with mobile nodes
    - Design and analysis of dynamically composed systems
    - Supporting middleware infrastructure
  - Fault-tolerance mechanisms for autonomous systems
    - “Dependability in the Web Services Architecture” [SRDS’03]

Outline

- Dependability concepts
- SA and dependability analysis
- Automated dependability analysis of SA
- Supporting environment
- Supporting the overall development of dependable systems
- Conclusion

Conclusion

- The approach to dependability analysis
  - The design and realization of our environment is guided by the needs of its current and potential users.
    - Simplification of certain extremely important and inevitable development activities related to the quality analysis and assurance of the DS.
  - Overall environment provides support for both qualitative and quantitative analyses
- The prototype...
  - Has been used for evaluating
    - TA case study
    - Workflow-based information systems

Conclusion (2)

- Further assisting the development of dependable systems
  - Integrating fault tolerance means in the SA design
    - Synthesizing middleware architectures towards enforcing dependability
      - Modeling middleware architectures
      - Composing middleware architectures
    - Architecture-based exception handling
      - Architecture changes in the presence of faults
      - Associated runtime support
  - Developing dependable, open systems remains an open issue

Bibliography

- Fundamental concepts of dependability
  - A. Avizienis, J-C. Laprie & B. Randell, TR UCLA-CSD-010028, LAAS-CNRS-01-145,Newcastle-CS-TR-739
- ATAM / ABAS
  - Kazman *et al*., Annals of SE, 9:5, 2000
  - Klein *et al*., Proc. WICSA-1, 1999
- Dependability models
  - G. Myers, Software reliability, Wiley and Sons, 1976
  - R. Glass, Software reliability, Prentice-Hall, 1979
  - NASA tech report, 1995
- SA-based development of DS
  - http://www-rocq.inria.fr/arles/work/Env.html