Coordinated Atomic Actions: 10 years after

Alexander Romanovsky

School of Computing Science
University of Newcastle upon Tyne

alexander.romanovsky@newcastle.ac.uk

UNIVERSITY OF NEWCASTLE

1. Aims of the Talk

Error recovery is the crucial step of fault tolerance which follows error detection and error diagnosis.

This talk is about the past, ongoing and future work on an error recovery scheme.

The aims of the talk are

- to briefly introduce the concept of Coordinated Atomic (CA) actions developed in the mid-90s
- to present the relevant work completed in the past few years and to report on the ongoing work
- to try and understand where it is all going
- to help people working in the area to coordinate their efforts

UNIVERSITY OF NEWCASTLE

1. Aims of the Talk

PDCS 2

Design for Validation - DeVa (EC, 1996-1999)

DeVa
Design for Validation
Esprit Long Term Research Project No. 20072

Dependable Systems of Systems - DSoS (EC, 2000-2003)

Rigorous Open Development Environment for Complex Systems - RODIN (EC, 2004-2007)

Rigorous Stepwise Development of Complex Fault Tolerant Distributed Systems: from Architectural Description to Java Implementation - CORRECT (Luxembourg, 2004-2007)

2. Fault Tolerance and Error Recovery

Error recovery is typically more complex than the normal system activity.

In many systems more than 50-70% of the resources are dedicated to detecting and dealing with abnormal situations. Dealing with abnormalities is becoming an every day issue.

Dealing with faults is one of the major *sources of system complexity*, and, as experience shows, one of the major *sources of system faults and failures* (e.g. an ICSE 2006 paper - 2-3 bugs per 1 KLoC in *mis*using exception handling).

The time of ad hoc fault tolerance is long gone. We are all working on disciplined and systematic fault tolerance. We need specialised fault tolerance mechanisms suitable for specific

- application domains
- development paradigms
- fault assumptions
- execution environments
- application requirements

UNIVERSITY OF NEWCASTLE

2. Fault Tolerance: Application Level

Application level fault tolerance plays a defining role in making systems dependable.

Hardware faults are no longer the predominant threat for many applications due to

- an increase in hardware quality and a reduction in hardware cost for many applications (e.g. hardware replication is cheap)
- a dramatic rise in software complexity and volume
- the involvement of new actors: non-professional users, multiple organisations, critical infrastructures
- the growing complexity of the environment in which systems operate

These applications include a wide range of safety-, life-, business- and money-critical systems - cf the reports by J.-C. Laprie (1999) and J. Knight (2005).

As a result, the *application software* needs to deal with a broad variety of faults: integration mismatches, design bugs, mistakes by non-professional users, potentially harmful changes in the system and environment, malicious faults, etc.

UNIVERSITY OF NEWCASTLE

2. Fault Tolerance: System Structuring

Fault tolerance hugely benefits from good system structuring.

These two birds, can and should be killed with one stone:

- reducing system complexity
- achieving system fault tolerance

Fault tolerance needs error isolation to define exactly which part of the system to recover, and to prevent errors from unlimited propagation.

Structuring units

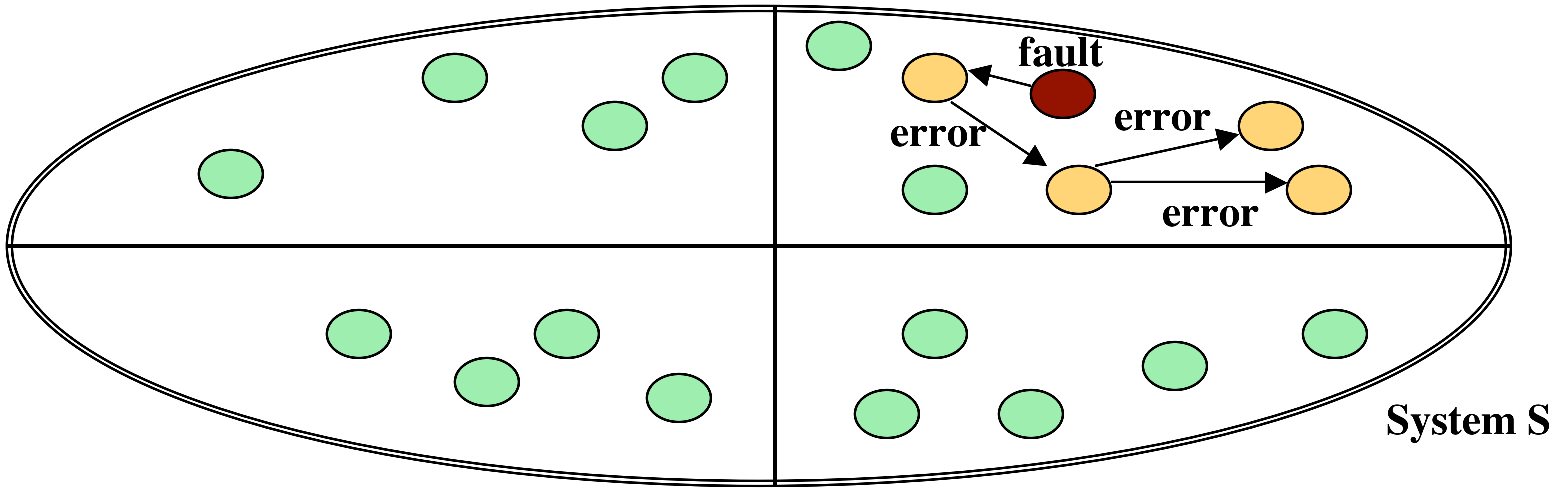
- are units of error confinement and recovery
- encapsulate states and behaviour
- make it easier to reason about the system.

UNIVERSITY OF NEWCASTLE

2. Fault Tolerance: System Structuring

Fortresses, submarine and nuclear power station structures, buildings, spy networks/underground cells as well as the human body *isolate and confine* propagation of water, fire, disease, enemies, radiation and information *to localise damage and recovery*.

Erroneous information does not cross the border of the structuring unit.



UNIVERSITY OF NEWCASTLE

2. Fault Tolerance: Forward and Backward Error Recovery

Forward error recovery: the system is returned to an error-free state by applying corrections to the damaged state. Such an approach demands some understanding of the errors.

Backward error recovery: the system is recovered to a previous error-free state. No knowledge of the errors in the system state is required.

Forward and backward error recovery techniques are complementary:

- **Forward error recovery allows efficient handling of the expected exception conditions**
- **Backward error recovery provides a general strategy which will cope with faults the designer did not (or chose not to) anticipate.**

Exception handling is the most general and effective means for tolerating faults of the widest possible range at the application level.

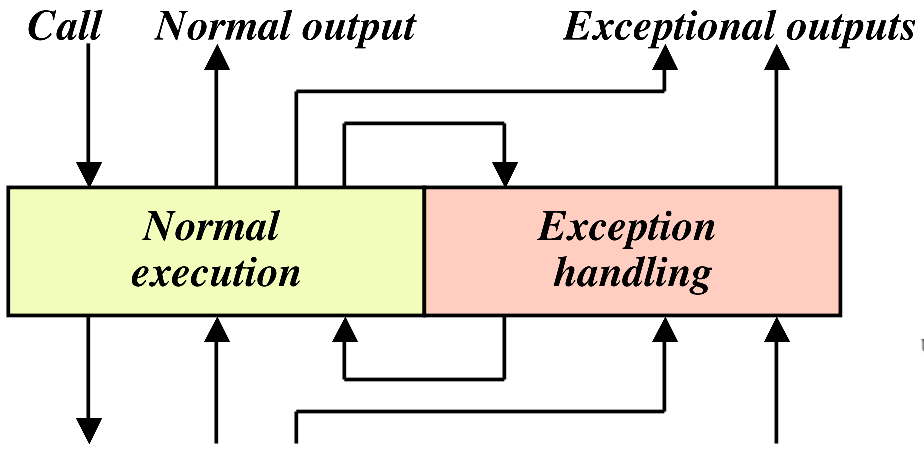
UNIVERSITY OF NEWCASTLE

2. Fault Tolerance: Application Level Exception Handling

EH:

- defines nested fault-tolerant units
- encapsulates internal recovery inside the units
- separates normal and abnormal behaviour
- introduces and separates normal and abnormal flows of control
- separates normal and abnormal outcomes of the unit
- defines the rules of how normal and exceptional activities are related
- allows recovery to be systematically developed and associated with the units to be executed in the same context

Program unit (e.g. object, process, block, module, method, class, component, etc.)



3. CA Actions: Concurrent/Distributed Systems

Exception handling and the provision of fault tolerance are more difficult in concurrent/distributed systems than in sequential programs.

Exception propagation in concurrent programs may not simply go through a chain of nested callers, e.g. an exception may need to be propagated to the members of a group of cooperating activities.

Distribution further complicates the cooperation and coordination of multiple concurrent activities.

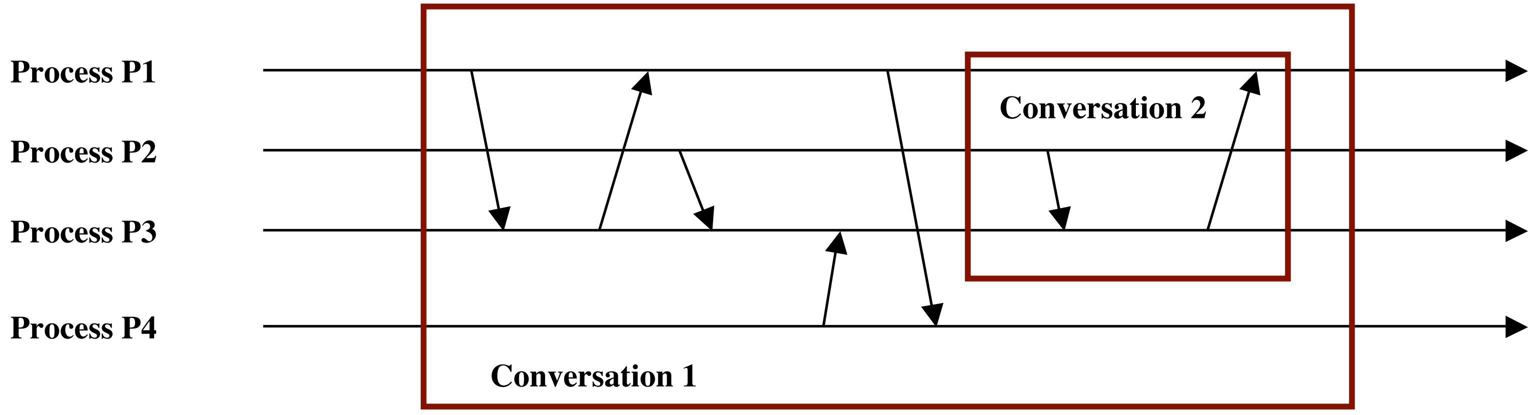
Damage confinement and assessment become more difficult in systems involving complex interactions among concurrent activities.

UNIVERSITY OF NEWCASTLE

3. CA Actions: ACID Transactions and Atomic Actions

ACID (atomicity, consistency, isolation, durability) transactions, which can be nested and multi-threaded, provide concurrency control and backward recovery for competing activities that are sharing external resources. *Recovery of resources.*

Conversations (M. Melliar-Smith and B. Randell) provide coordination and backward recovery for cooperating activities (processes, objects, threads, etc.), but do not support shared external resources. *Recovery of processes.*



UNIVERSITY OF NEWCASTLE

3. CA Actions: ACID Transactions and Atomic Actions

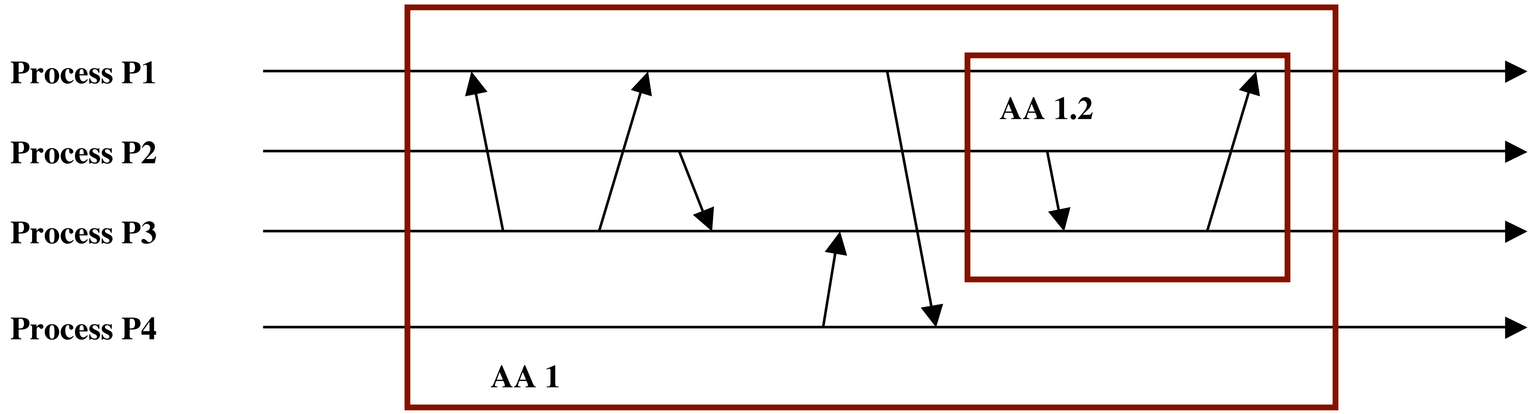
Atomic actions (R. Campbell and B. Randell) allow for exception handling in a system consisting of cooperating activities - when an exception occurs, every process in the action has to switch to an appropriate handler, so that cooperative forward error recovery is performed.

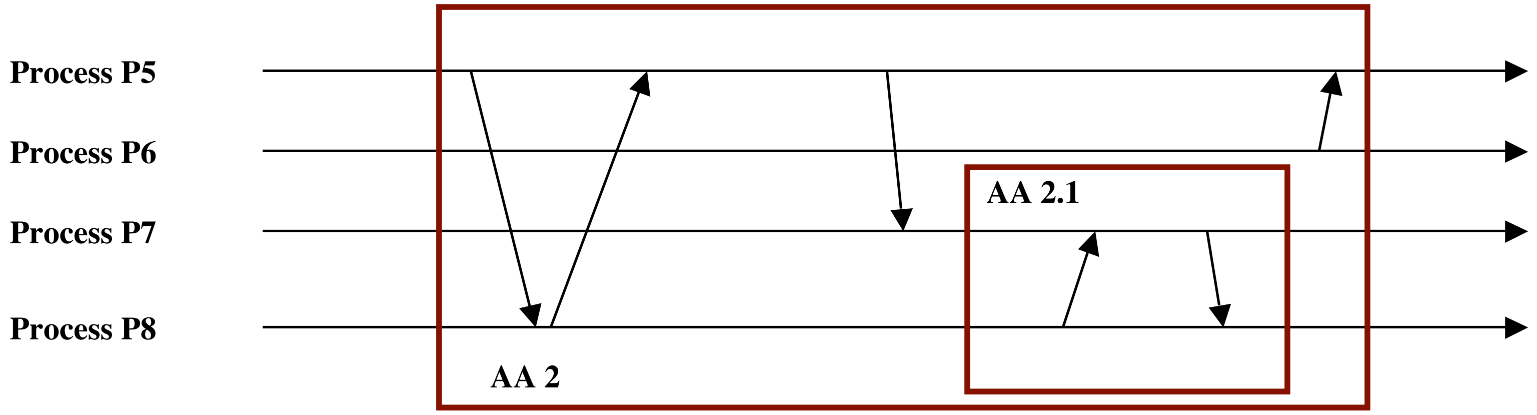
A Coordinated Atomic (CA) action can be regarded as an atomic action that also provides controlled access to shared external resources. Equally, it can be regarded as a nested multi-threaded transaction with disciplined exception handling.

scheme	Conversations	Atomic Actions	ACID Transactions	CA actions
recovery	BER	FER	BER	FER and BER
concurrency	cooperation	cooperation	competition	cooperation and competition

UNIVERSITY OF NEWCASTLE

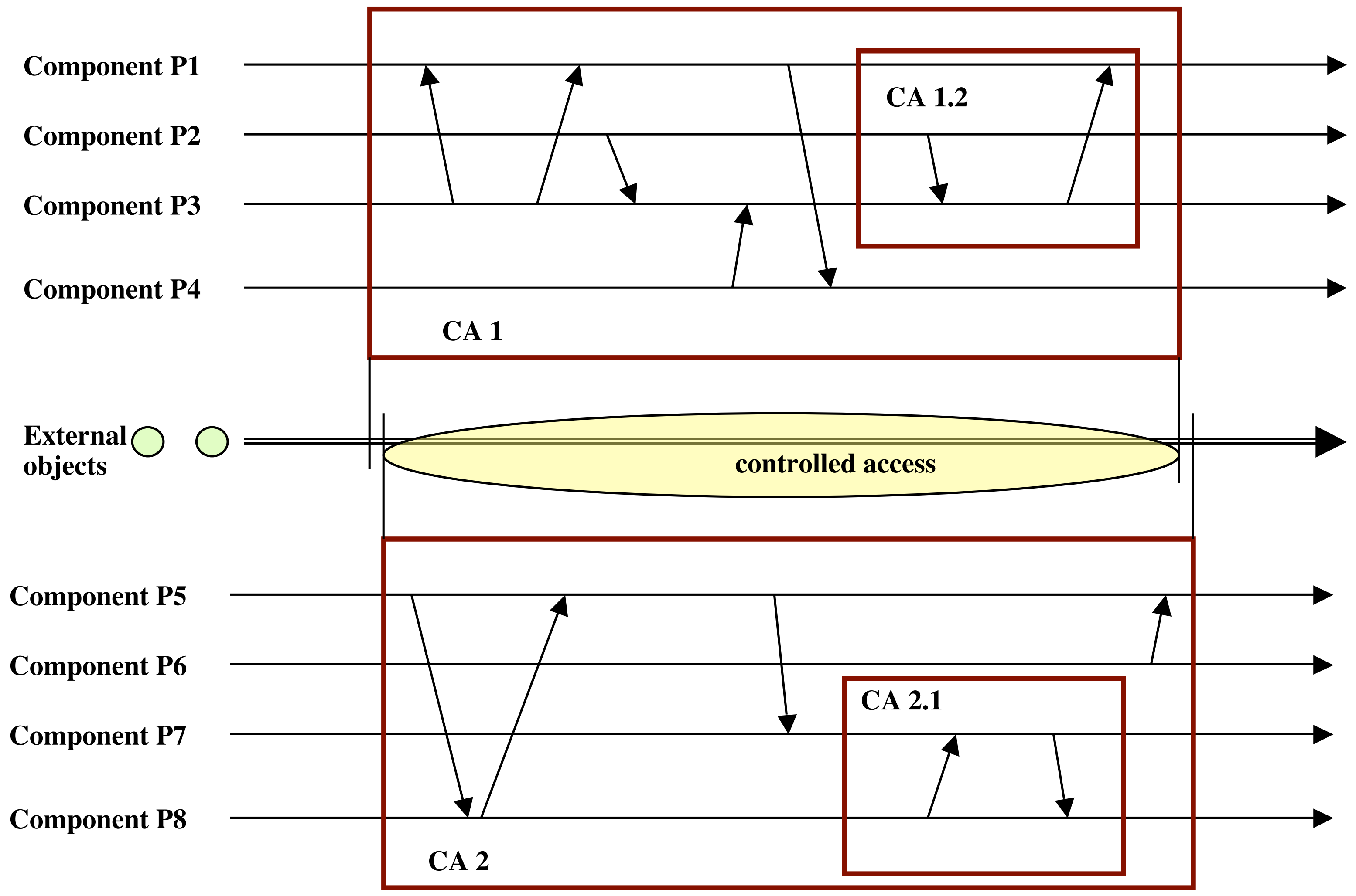
3. CA Actions: ACID Transactions and Atomic Actions





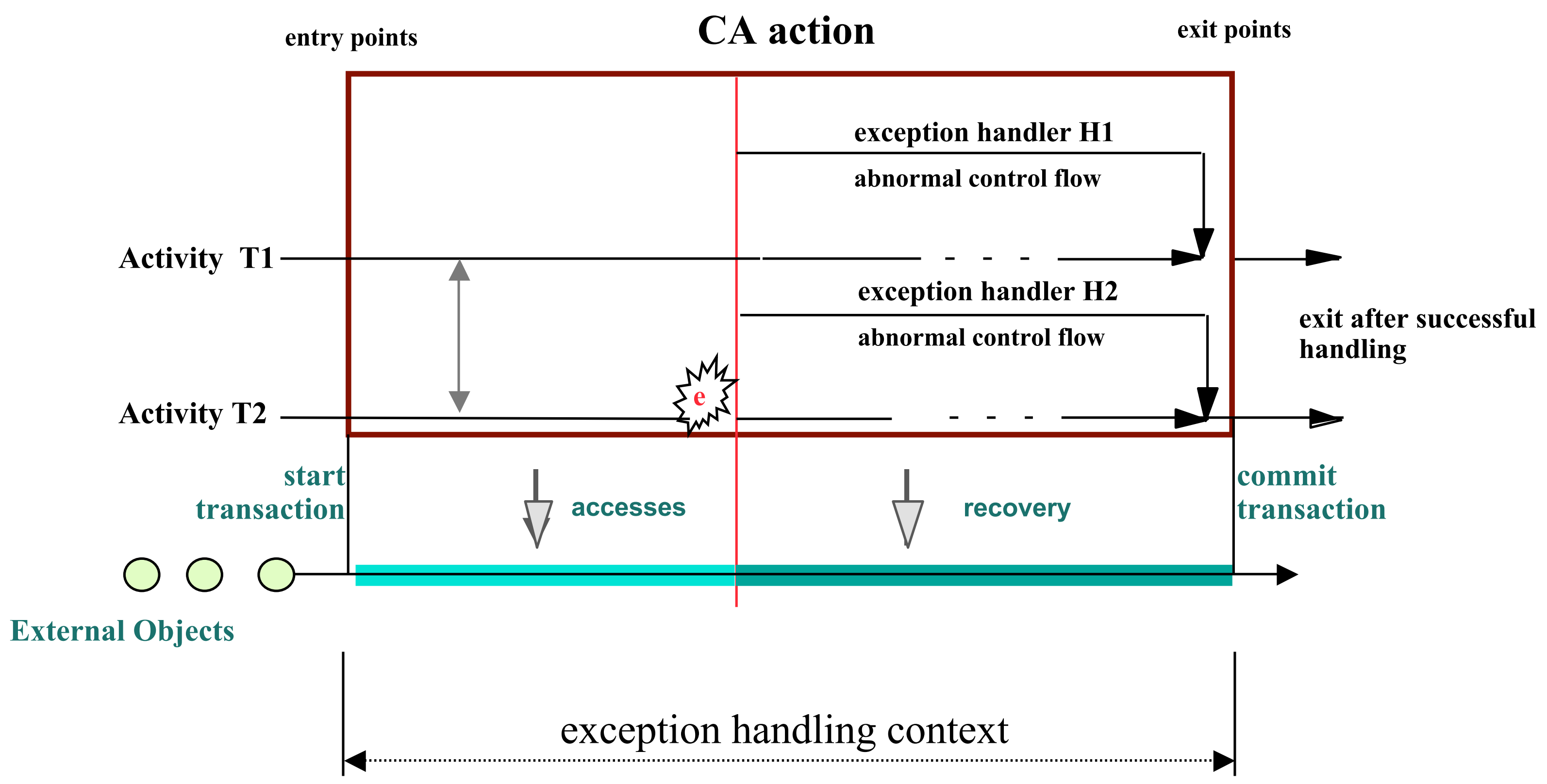
UNIVERSITY OF NEWCASTLE

3. CA Actions: ACID Transactions and Atomic Actions

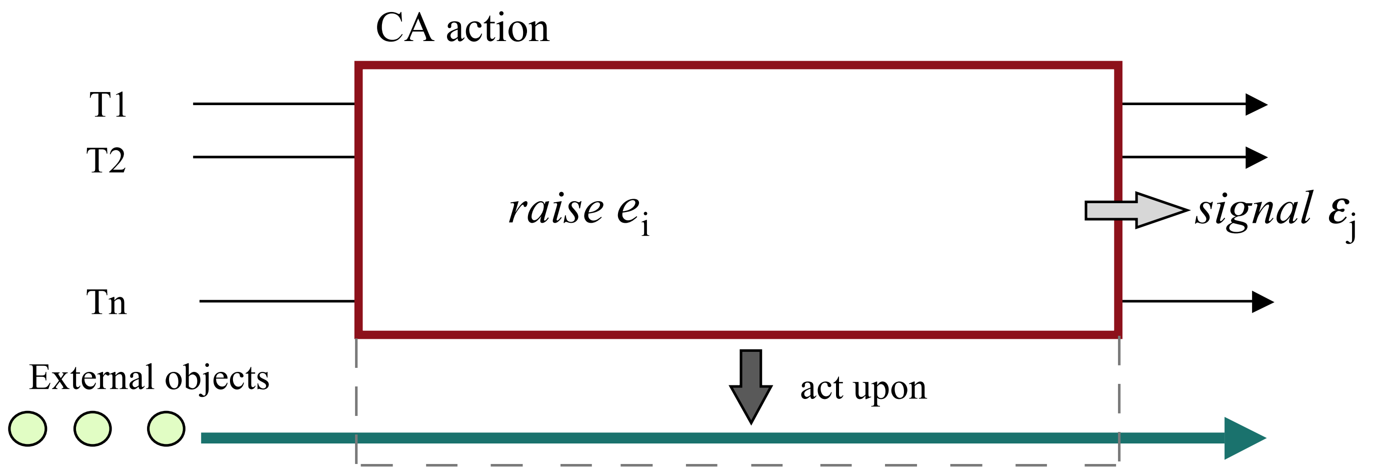


UNIVERSITY OF NEWCASTLE

3. CA Actions: Participants, Context and Internal Exceptions



3. CA Actions: Action Nesting and External Exceptions



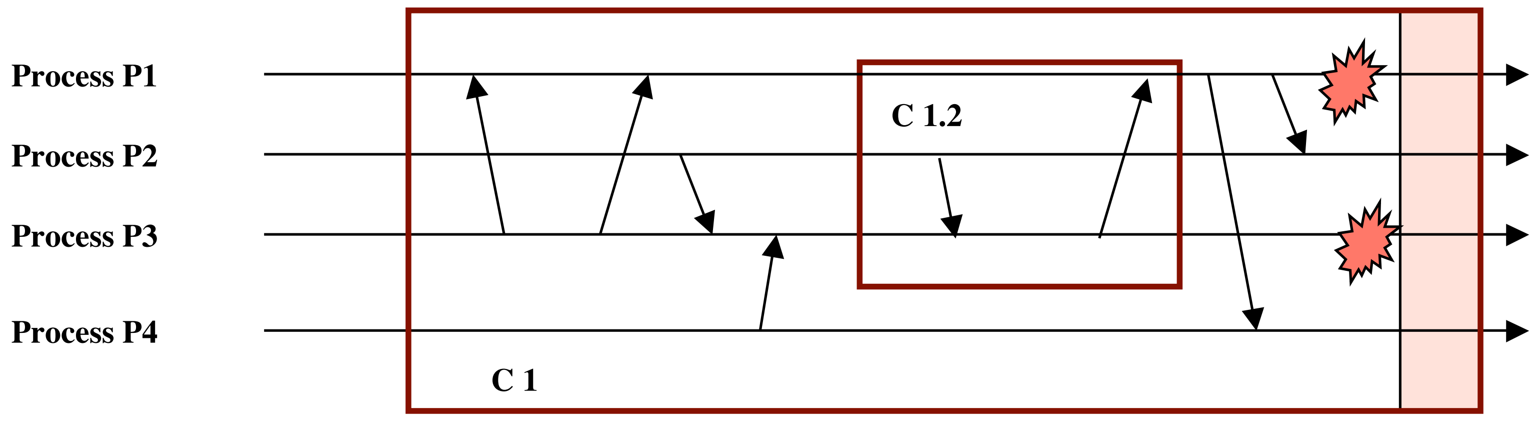
$e = \{e_1, e_2, e_3, \ldots\}$ (*Internal*) Exceptions inside the CA action must be declared with the action "body"/implementation and handled within the action

$\varepsilon = \{\varepsilon_1, \varepsilon_2, \varepsilon_3, \ldots\}$ (*External*) Exceptions to be signalled from the action to its environment (i.e. the enclosing action) must be specified in the CA action interface

Recursive relation: $\boldsymbol{\varepsilon}_{\textit{nested}}$ is a subset of $\boldsymbol{e}_{\textit{enclosing}}$

UNIVERSITY OF NEWCASTLE

3. CA Actions: Concurrent Exceptions

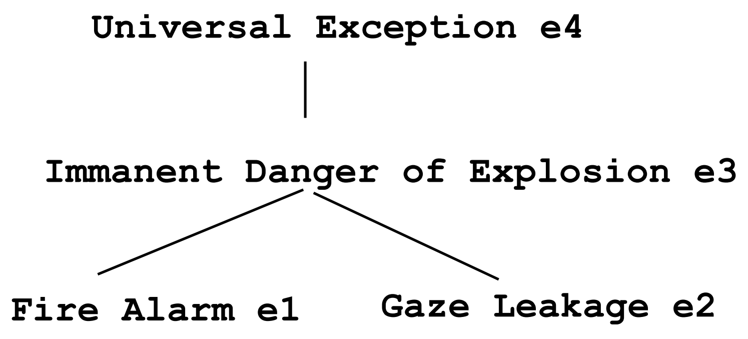


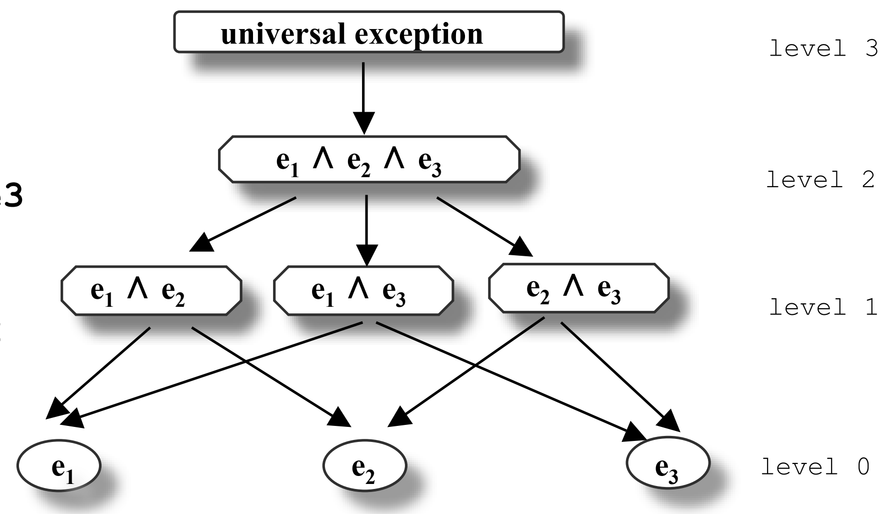
UNIVERSITY OF NEWCASTLE

3. CA Actions: Concurrent Exceptions

In a distributed system different activities may raise different exceptions and the exceptions may be raised simultaneously. Concurrent exceptions must be handled in a coordinated manner.

An exception resolution graph approach is developed in order to find the exception that "covers" all the exceptions raised concurrently. The graph imposes partial order on all internal action exceptions.





UNIVERSITY OF NEWCASTLE

3. CA Actions: Case Studies

Production cell case study

Fault tolerant production cell case study

Real time production cell case study

Internet auction system

Distributed over the Internet gamma computation

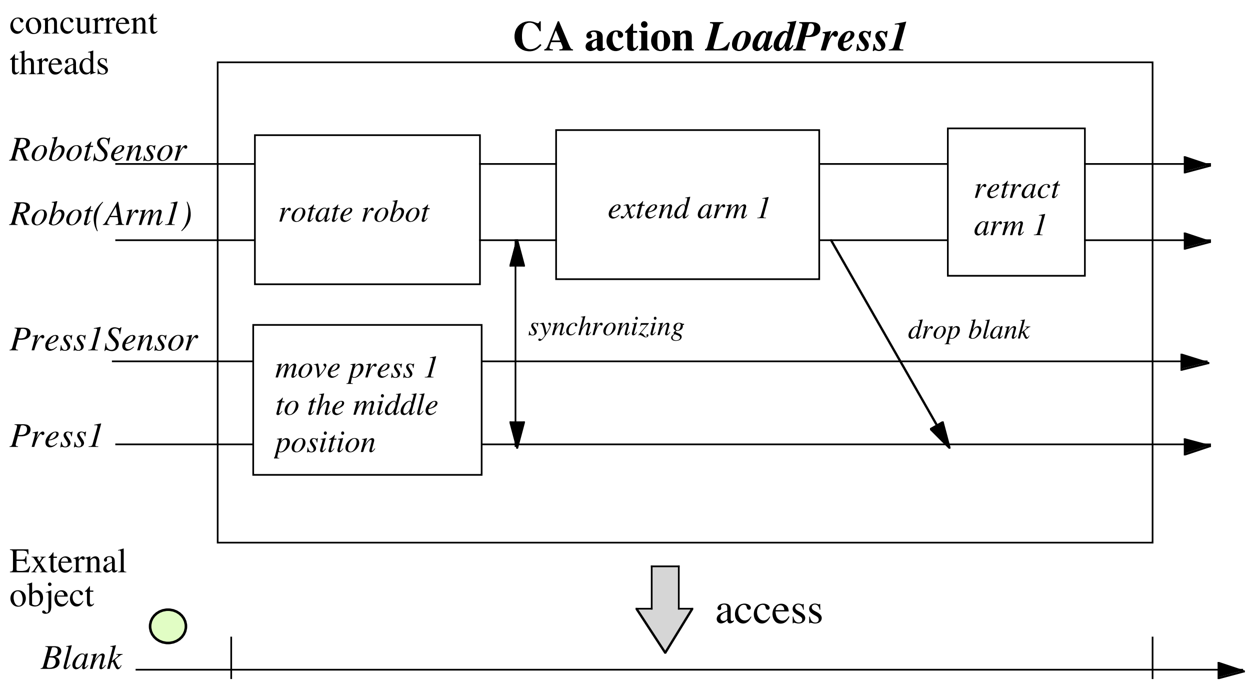
Railway control system

UNIVERSITY OF NEWCASTLE

3. CA Actions: Case Studies

Fault tolerant Production Cell case study.

The design uses 12 main CA actions; each action controls one step of blank processing and typically involves passing a blank between two devices.



3. CA Actions: Summary

CA actions allow us to design long-lived activities as they support

- exception handling: rather than always aborting and going back, it is possible to handle the problem and continue
- action nesting (recursive system structuring, choice of the right level of granularity) – lost computation can be minimised
- explicit application-specific programming of cooperation/competition with respect to shared resources – minimise periods when shared resources are locked

UNIVERSITY OF NEWCASTLE

3. CA Actions: Summary

A CA action is a generalised form of the basic atomic action structure.

Multi-Threaded Enclosure and Coordination - CA actions provide a mechanism for enclosing and coordinating interactions among activities, and ensuring consistent access to resources in the presence of complex concurrency and faults.

Fault Tolerance - If an exception is raised inside a CA action, appropriate forward and/or backward recovery measures are invoked cooperatively in order to reach some mutually consistent conclusion and, if possible, to recover.

Multiple Outcomes - CA actions combine exception handling with the nested action structure to allow multiple outcomes, e.g. a normal outcome or some possible exceptional outcomes.

CA actions (as well as ACID transactions, conversations, etc.) are nested structuring units of system design and execution.

UNIVERSITY OF NEWCASTLE

4. Trends in CA Action Research

CA actions were intentionally developed as a general concept capturing several fundamental ideas:

- recursive system structuring for fault tolerance
- dealing with heterogeneous distributed systems consisting of cooperative and competitive concurrency
- coordinated recovery inside the action and concurrent exception resolution
- normal and exceptional outcomes defined at the action interface level

The idea was not to develop a specific scheme for a specific environment, OS, platform or language.

UNIVERSITY OF NEWCASTLE

4. Trends in CA Action Research

The main trends in the recent years:

- **Software engineering.** **The main focus is on the CA action engineering. Application level fault tolerance needs to be supported at the *earlier phases* and through the whole development process as well as being *reused*.**
- **New environments, domains and paradigms**
- **New case studies:**
  - **Internet travel agency (DSoS)**
  - **Fault tolerant insulin pump control system (CORRECT)**
  - **Ambient campus (RODIN)**
- **New implementations**
- **Generalisation:**
  - **applicability of exception handling for dealing with external objects, resources and the environment (they do not have to be ACID)**
  - **concept of action participants - structuring activities involving computers (e.g. with humans or external devices)**

5. CA Action Engineering - Formal Methods

Why do we need formalisation?

- **understanding and definition of the concepts**
- **verification of system properties**
- **formal stepwise development process (cf correct fault tolerance by construction)**

UNIVERSITY OF NEWCASTLE

5. CA Action Engineering - Formal Methods

- COALA - the first model describing all main CA action abstractions and functionality in terms of CO-OPN/2: an object-oriented language based on Petri nets and partial order-sorted algebraic specifications (DeVa)
- Formalisation of CA actions using the ERT model (ERT stands for extraction, refusals and traces): CSP and a specific technique used to relate systems specified at different levels of abstraction (DeVa)
- Timed CSP based framework for representing the use of CA actions in real-time safety-critical systems; focus on modelling dynamic system structuring
- Properties: Temporal Logic of Actions - TLA. Explicit modelling of a set of action outcomes with associated post-conditions (DeVa)
- Properties: Alloy, B, ProB - proving basic, desired and application-specific properties (RODIN)
- B modelling of the CA action environment and abstractions: action participant, transactional external objects, exception resolution, action (DSoS)
- B refinement: decomposition patterns for developing mobile agent applications with actions: scopes and exception handling (RODIN)

UNIVERSITY OF NEWCASTLE

5. CA Action Engineering - Software Architecture

The Web Service Composition Actions (WSCA) architecture for dependable composition of WSs is specified at an abstract level supporting recursive system structuring and cooperative exception handling (DSoS)

Engineering dependable systems using two fault tolerance architectural styles (the idealized fault tolerant component model style and the role-based collaboration style) and a set of OO design patterns

CORRECT ongoing work.

UNIVERSITY OF NEWCASTLE

5. CA Action Engineering - UML Modelling and MDD

UML modelling:

- **UML2.0 platform-independent profile for CA actions (CORRECT)**
- **More work on UML profiling in CORRECT - ongoing**
- **Initial work at Monash**

Model-driven development:

- **DRIP Catalyst: An MDE/MDA Method for Fault-tolerant Distributed Software Families (CORRECT)**
- **Ongoing work in CORRECT**
- **Initial work at Monash**

UNIVERSITY OF NEWCASTLE

6. New Domains - WSs and SOA

No backward error recovery. Compensation is only a partial answer. Components are not controlled by the integrator.

The Web Service Composition Actions (WSCA) scheme is an extension of CA actions and their adaptation for this application area:

- Dependable composition of WSs is specified at an abstract level supporting recursive system structuring and cooperative exception handling
- The WSCA language (WSCAL) built on the W3C standards: Web Service Description Language and Web Services Conversation Language

CA Action design of Web applications (Internet Travel Agency):

- Design. Each client session is a CA action consisting of a number of nested actions performing availability checking, trip booking, trip cancellation, payment, etc.
- An experimental implementation using Java RMI, JavaServer Page (JSP) and a distributed CA action support (DRIP)

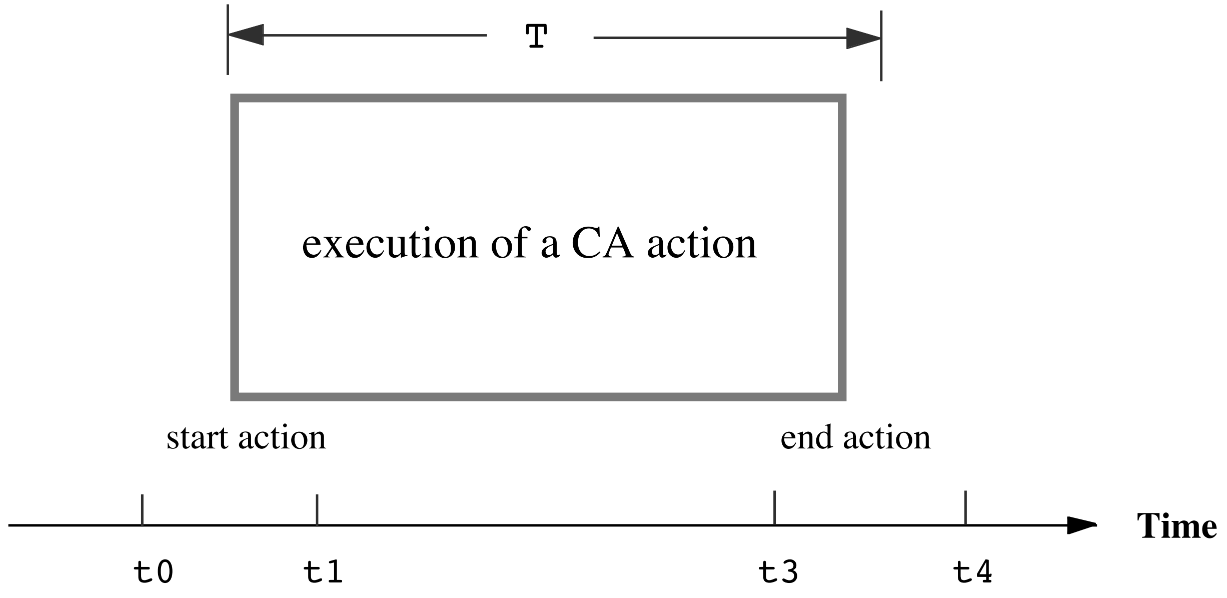
Ongoing work in CORRECT on defining UML methods and models for developing complex systems composed of WSs.

UNIVERSITY OF NEWCASTLE

6. New Domains - RT

Real-time systems and CA actions (DeVa).

Typical timing constraints on the execution of a CA action:



UNIVERSITY OF NEWCASTLE

6. New Domains - Security

Using extended action mechanisms not only for traditional fault tolerance but for tolerating malicious faults (FP5 MAFTIA)

Designing and structuring several specific security protocols using CA actions with sets of participants playing different roles (Durham University).

UNIVERSITY OF NEWCASTLE

6. New Domains - Mobile Agents

Mobile agents (asynchronous and anonymous communication, autonomous components).

Mobile coordination- (tuplespace) based systems - Newcastle University:

- Open and flexible exception handling and structuring mechanisms. Nested scopes with flexible exception handling. Several directions of exception propagation
- A set of abstractions (such as roles, scopes, agents, locations, platforms, exceptions and exception propagation)
- Stepwise formal development based on Event B refinement
- Fault tolerance formal development templates
- Agent interoperability by constructions & independent agent development
- Identifying and model checking/proving fault tolerance properties
- Formal development of middleware - the location-based CAMA middleware
- Ambient Campus case study

Rodin

Ambient conversations - OO framework, dealing with disconnections - Vrije Universiteit, Brussels

Scoping for publish/subscribe systems - Lancaster University

UNIVERSITY OF NEWCASTLE

7. Implementations

DRIP class framework in Java RMI (DeVa, 1998)

Ada 95 - OO implementations supported by patterns (DeVa, 1999)

Web Services, DRIP extension, HTTP requests from clients, JSP (DSoS, 2003)

CAA DRIP class framework in Java RMI (CORRECT, 2005)

.NET (Monash, 2006)

UNIVERSITY OF NEWCASTLE

8. Conclusions

Issues for future research:

- Identification of basic distributed middleware functionalities
- Formal refinement and CA actions
- (Eclipse?) tool environment for model engineers who use CA actions
- Move from closed to open systems:
  - ambient, pervasive and ubiquitous systems
  - more flexible and dynamic schemes
  - context-aware CA actions
- SOA, WSs and business processes (BPEL)
- Structuring critical infrastructures
- Systems involving people
- Engineering CA actions by aspects

UNIVERSITY OF NEWCASTLE

8. Conclusions

CA actions are alive and kicking.

Active area of research thanks to many people's efforts.

This research informs many developments in the area.

It is important to maintain its conceptual level and at the same time to work on concrete instances.

UNIVERSITY OF NEWCASTLE

9. Projects and Groups

Design for Validation - DeVa (EC, 1996-1999)

DeVa
Design for Validation
Esprit Long Term Research Project No. 20072

Dependable Systems of Systems - DSoS (EC, 2000-2003)

Rigorous Open Development Environment for Complex Systems - RODIN (EC, 2004-2007)

Rigorous Stepwise Development of Complex Fault Tolerant Distributed Systems: from Architectural Description to Java Implementation - CORRECT (Luxembourg National Project, 2004-2007)

9. Projects and Groups

Active groups

- **Newcastle University (UK)** - FM, SA, e-Science, WSs, virtual organisations, agents, new domains, BPEL
- **University of Luxemburg (Luxemburg) Nicolas Guelfi** - FM, SA, new domains, WSs, UML, model-based development
- **UniCamp (Brazil) Cecilia Rubira** - SA, FM properties, aspects
- **University of L'Aquila (Italy) Patrizio Pelliccione and Henry Muccini** - SA, model-based development
- **University of Geneva (Switzerland) Didier Buchs** - testing, FM, WSs
- **Aabo Akademi (Finland) Elena Troubitsyna** - agents, formal stepwise development
- **INRIA (France) Valerie Issarny** - WSs, WSCA
- **University of Wellington (New Zealand) Ian Welch** - BPEL and WSs
- **Monash University and EDS (Australia) Susan Entwisle** - UML, .NET, model-based development
- **PURCS (Brazil) Avelino Zorzo** - new applications, FM
- **Lancaster University (UK) Alessandro Garcia** - agents, aspects
- **Kent University (UK) Rogerio De Limos** - SA
- **South Bank University (UK) - Nimal Nissanke** - FM
- **BAE SYSTEMS (USA) Bob Shaifer** - systems of systems

UNIVERSITY OF NEWCASTLE

Thank you!

UNIVERSITY OF NEWCASTLE