Coordinated Atomic Actions and System Fault Tolerance

Alexander Romanovsky

School of Computing Science
University of Newcastle upon Tyne
alexander.romanovsky@newcastle.ac.uk

UNIVERSITY OF NEWCASTLE

1. Overview

2. Aims of the Talk

This talk is about past, ongoing and future work on a fault tolerance (error recovery) scheme.

Error recovery is often more complex than the normal activity. In many systems more than 50-70% of the resources are dedicated to detecting and dealing with abnormal situations. Dealing with abnormalities becomes every day issue.

The aims of the talk are:

• to introduce the concept of Coordinated Atomic (CA) actions that we developed in Newcastle in late 90th

• to present our current work on application of CA actions for building complex Web applications.

3. CA Actions: 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 expected exception conditions, and backward error recovery provides a general strategy which will cope with faults a designer did not (or chose not to) anticipate.

3. CA Actions: Exception Handling

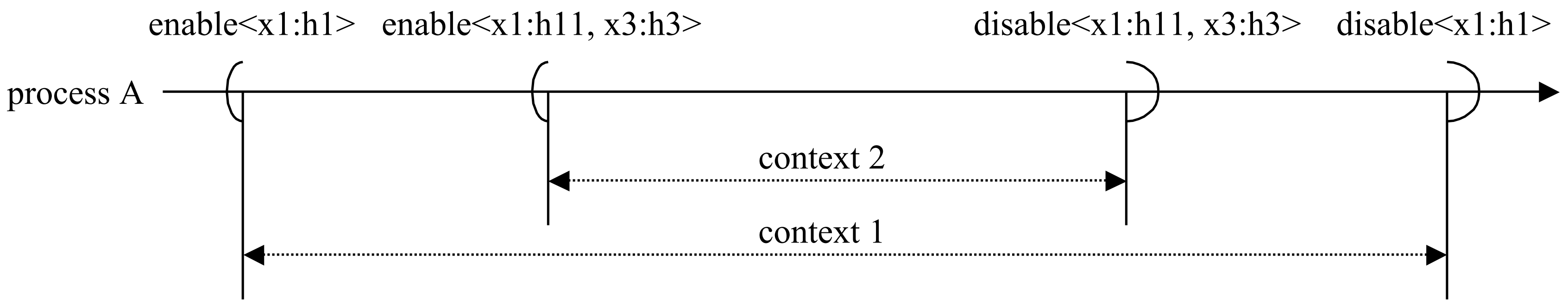
Exception handling is the most general means of providing forward error recovery.

The flow of control of a computation within a component should change as the result of a raised exception.

Such an *exceptional* flow of control is distinguished from the *normal* flow of control.

Within a program, exceptional flow of control is associated with code fragments that called exception handlers.

Exceptions, a software component, and exception handlers are linked together by a handling context. Context nesting. *System structuring!*



UNIVERSITY OF NEWCASTLE

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, e.g. several exceptions may be raised concurrently in multiple concurrent activities.

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

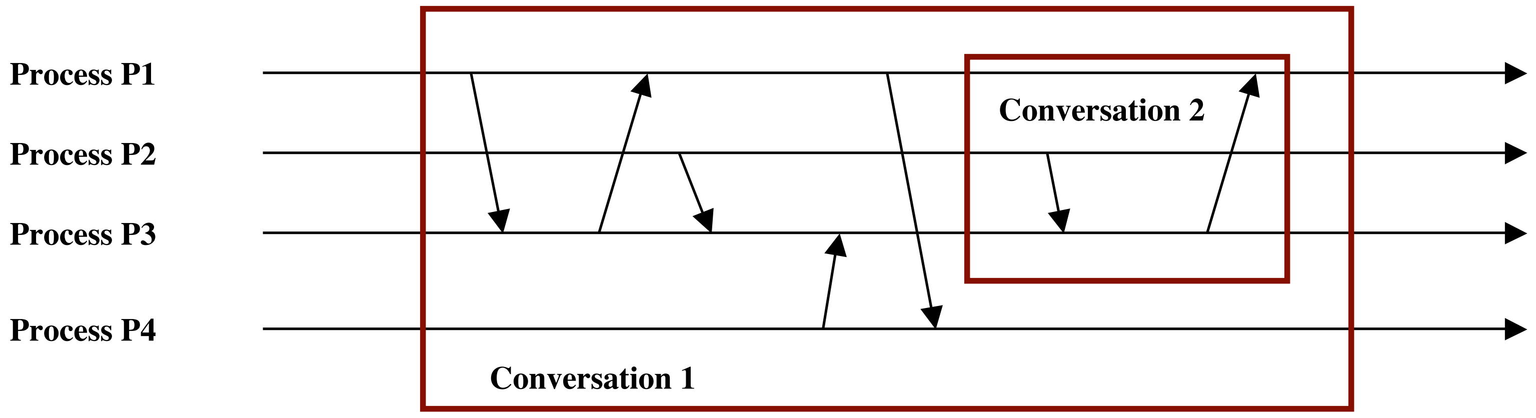
Damage confinement and assessment become more difficult in systems involving complex interactions among concurrent activities.

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 (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 + Atomic Actions

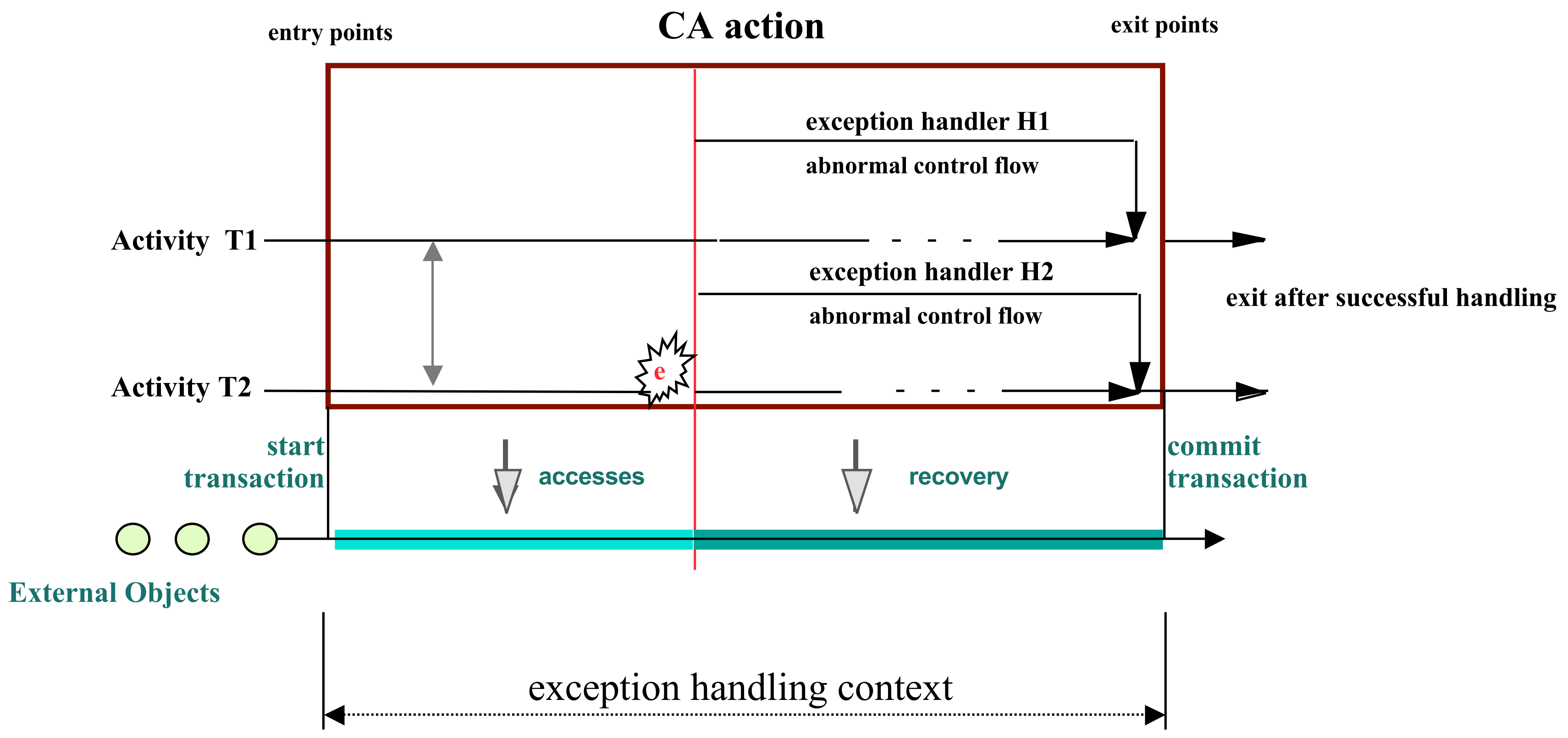
Atomic actions (R. Campbell and B. Randell) allow for exception handling in system consisting of cooperating activities - when an exception occurs, every process in the action has to switch to an appropriate handler, so that cooperative forward error recovery is performed.

A Coordinated Atomic action (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: Internal Exceptions



UNIVERSITY OF NEWCASTLE

3. CA Actions

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

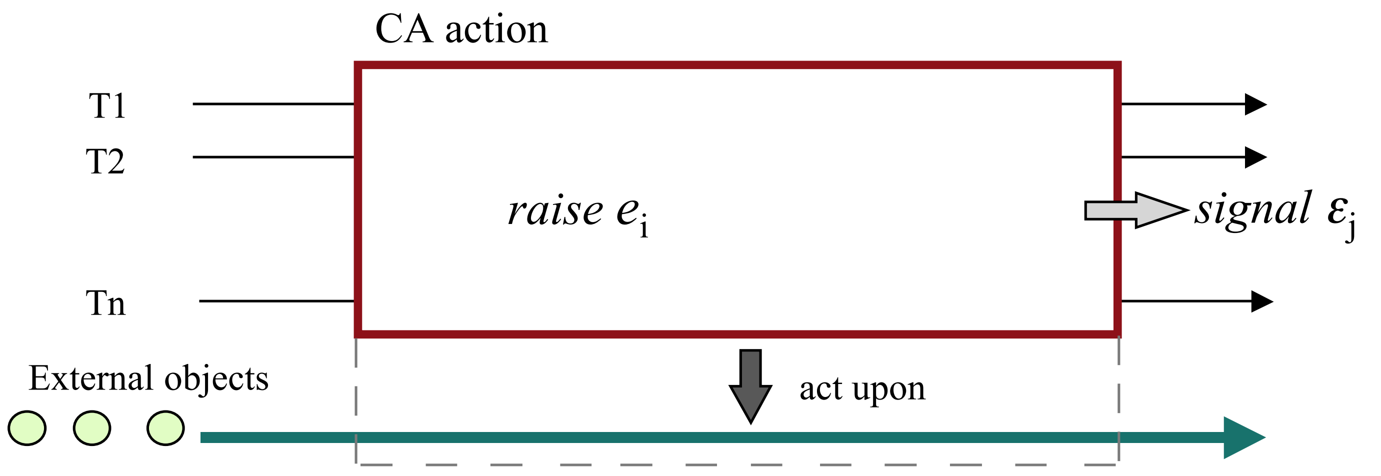
Multi-Threaded Enclosure and Coordination - CA actions provide a mechanism for enclosing and coordinating interactions among activities, and ensuring consistent access to objects (resources) in the presence of complex concurrency and potential faults.

Fault Tolerance - If an exception is raised inside a CA action, appropriate forward and/or backward recovery measures will be 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.

3. CA Actions: Action Nesting



$e = \{e_1, e_2, e_3, \ldots\}$ Exceptions inside the CA action must be declared with the action definition and handled within the action

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

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

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.

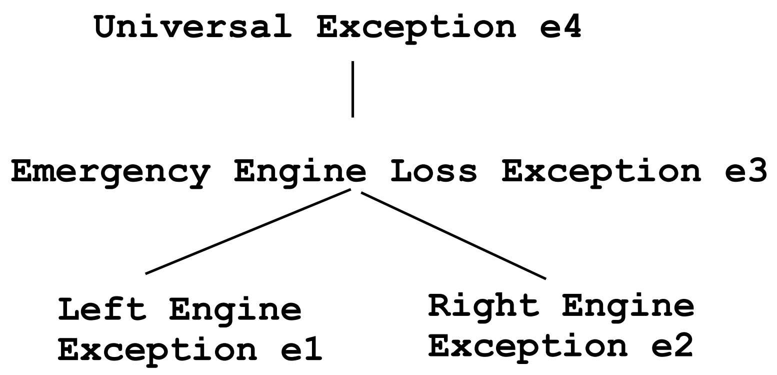
Concurrent exceptions must be handled in a coordinated manner.

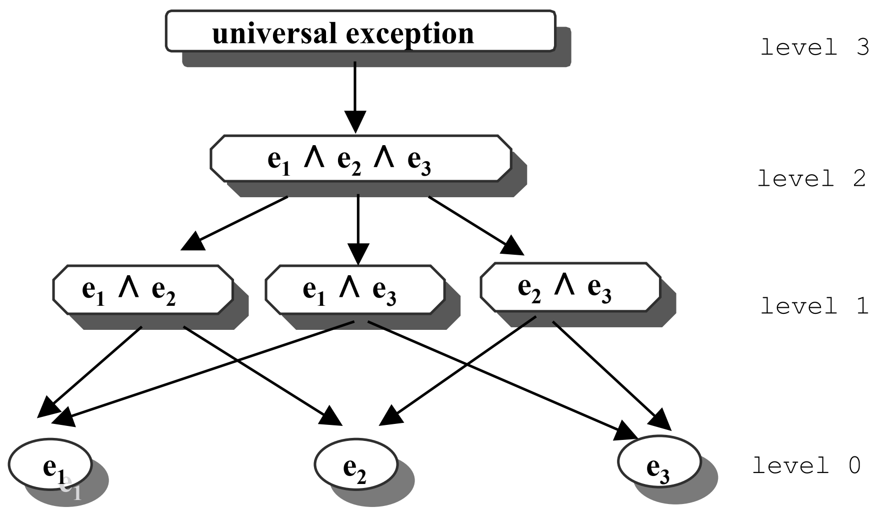
Example: If there are two exceptions `fire_alarm` and `gaze_leakage` raised concurrently in two distributed cooperating activities, their separate handling, or handling them in any order, or ignoring either of them can cause serious harm.

There is a variety of reasons why several exceptions may be raised concurrently. For example, it is often difficult to interrupt the normal operations of the other nodes immediately after an exception has been raised or the several exceptions can be symptoms of the same problems.

3. CA Actions: Concurrent Exception Resolution

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: Long-Lived Activities

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

- exception handling: you do not have to always abort and go back, you rather try 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 resource are locked

3. CA Actions: A Case Study

The FZI (Forschungszentrum Informatik, Germany) have specified and provided a simulator for the Fault-Tolerant Production Cell.

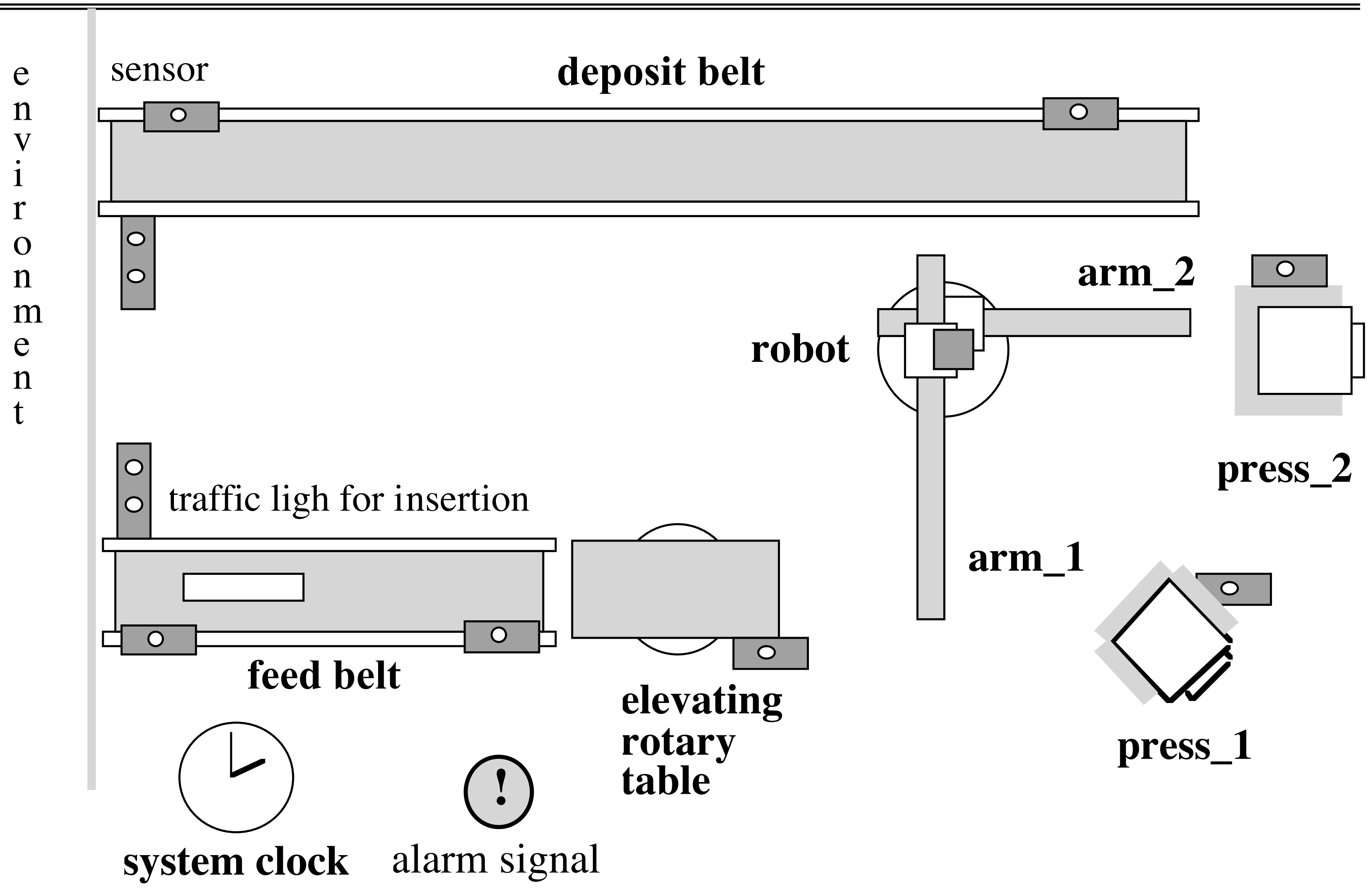
It represents a manufacturing process involving six devices: two conveyor belts (a feed belt and a deposit belt), an elevating rotary table, two presses and a rotary robot that has two orthogonal extendible arms.

The task of the cell is to get metal blanks from its "environment" via the feed belt, transform them into forged plates by using one of the presses, and then return them to the environment via the deposit belt.

The challenge posed by FZI is to design a control system that maintains specified safety and liveness properties even in the presence of a large number and variety of device and sensor failures, and which continues to operate even if one of the presses is non-operational.

Our *aim* was to show how concurrent exception handling and CA actions aid both the design and validation of this control system.

3. CA Actions: A Case Study

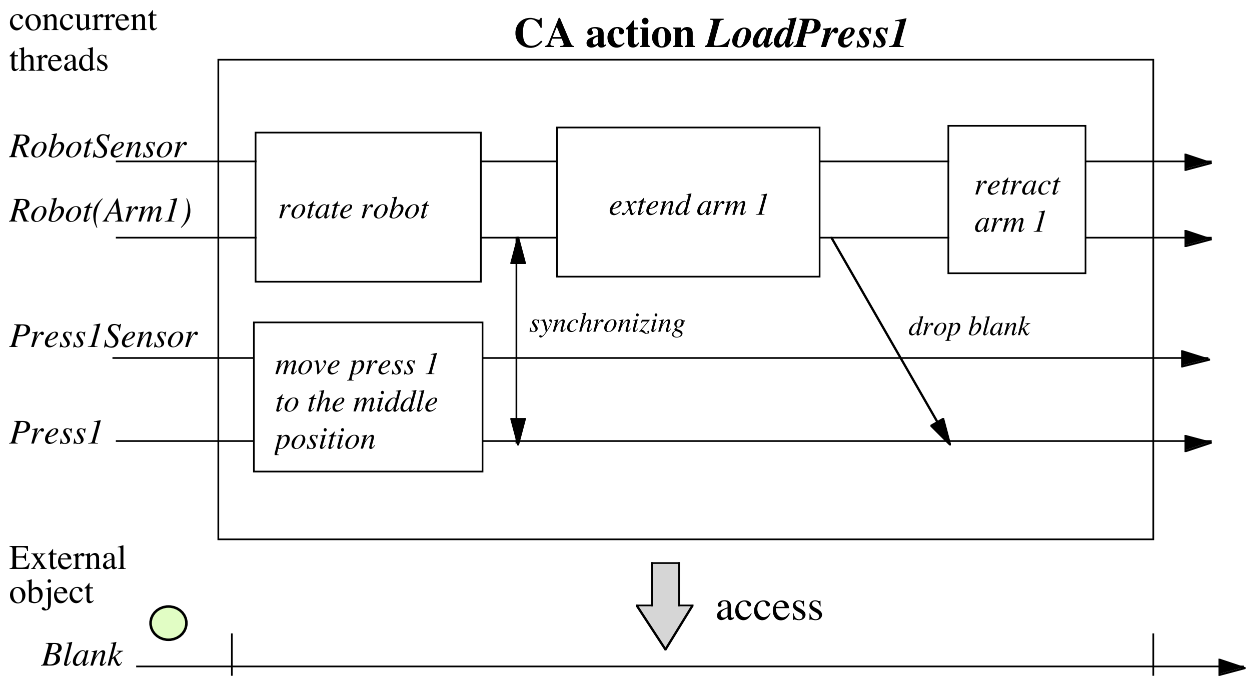


UNIVERSITY OF NEWCASTLE

3. CA Actions: A Case Study

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

A control program was implemented that uses a Java implementation of a distributed CA action support (this scheme makes use of the nested multi-threaded transaction facilities provided by the Arjuna transaction support system).



3. CA Actions: A Case Study

Pre-conditions	*post-conditions*
robot off	robot off
Blank on arm 1	no blank on arm 1
Both arms retracted	both arms retracted
robot at one of the defined angles	robot angle: arm 1 towards press 1
press 1 off	press 1 off
no blank in press 1	blank in press 1
press 1 in bottom position	press 1 in middle position

Exception to signal	*exceptional post-conditions*
	robot off
	blank on arm 1
Press 1 failure	both arms retracted
	robot angle: arm 1 towards press 2
	press 1 off
	no blank in press 1

Exception to signal	*exceptional post-conditions*
	robot off
(rotary sensor or motor failure) & Press 1 failure	blank on arm 1
	both arms retracted
	press 1 off
	no blank in press 1

UNIVERSITY OF NEWCASTLE

3. CA Actions: A Case Study

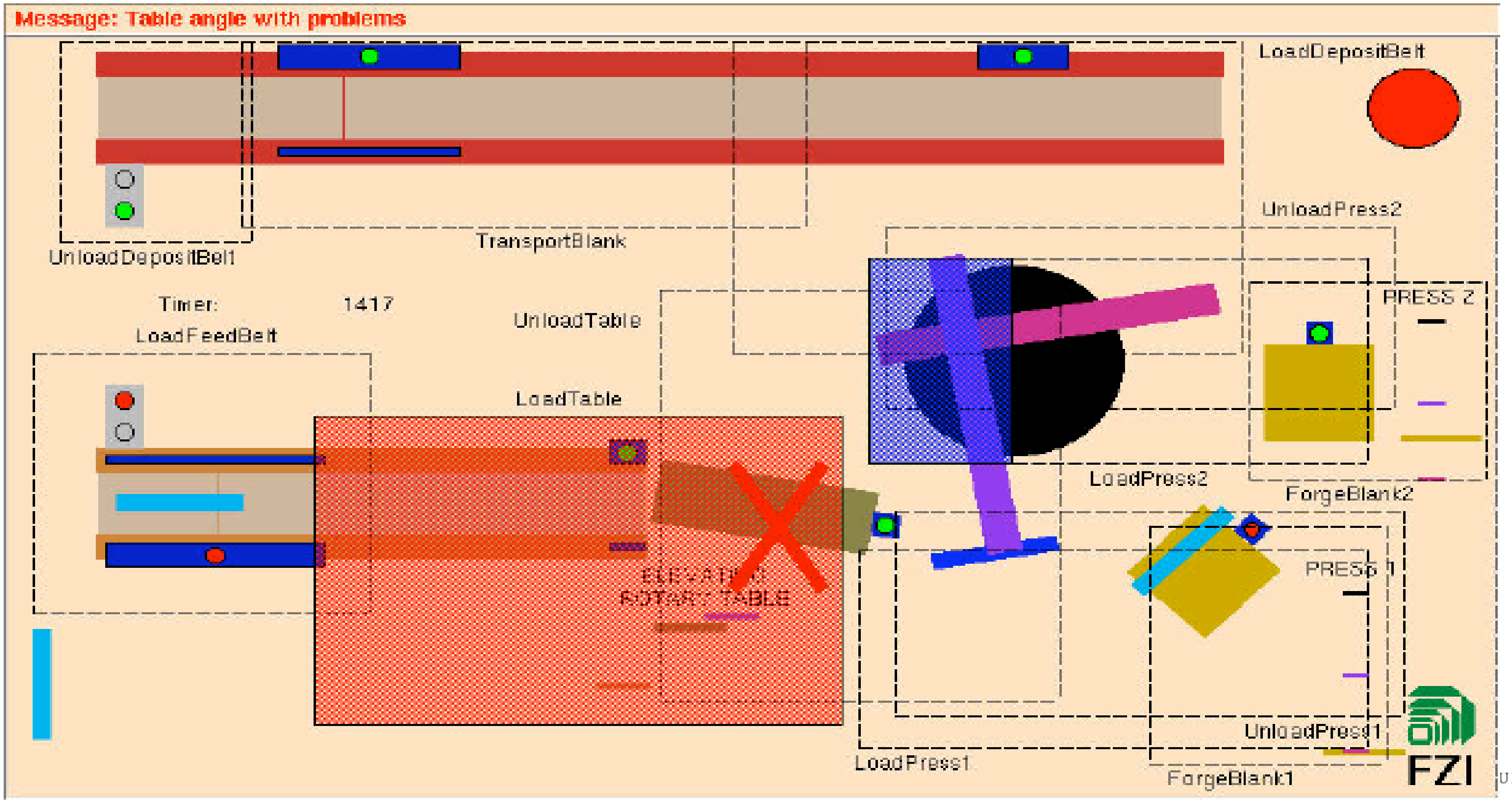
The program controls the FZI simulator. Overlaid on the screen are rectangular outlines showing the scopes of these CA actions.

As the simulator is operated each of these outlines is gradually shaded in each time the corresponding CA action is executed - the colour of this shading is changed when the CA action is involved in exception handling, in response to simulated faults.

A fault injector was implemented.

All injected device or sensor failures were caught successfully and handled immediately by our control program. A previously unknown bug in the FZI simulator was detected by a CA action and recovered automatically by the action using the retry operation.

3. CA Actions: A Case Study


Message: Table angle with problems

UNIVERSITY OF NEWCASTLE

3. CA Actions: Ongoing Research

Several more case studies have been developed:

- *a real-time Production Cell*
- *a distributed auction system*
- *a railway control system*
- *a distributed internet Gamma computation*

CA actions were intentionally developed as a general concept.

There are concrete schemes for concurrent OO systems, process-oriented systems, message-passing distributed systems, component-based systems.

These are "close" environments and architectures ...

UNIVERSITY OF NEWCASTLE

4. Integration of Complex Web Applications

The focus of this ongoing work is on developing techniques for building dependable Web applications. The complex Web applications are being and will be built by *integration of existing Web services*. This is typical of the emerging *service-oriented architecture* paradigm.

Existing and future Web applications with high dependability requirements: banking (bank portals), auctions, internet shopping, hotel/car/flight/train reservation and booking, e-business, e-science (including the Grid computing), business account management, …

4. Integration of Complex Web Applications

Complex systems of systems. Open systems. Web services to be integrated:

• are ready-made (COTS) components

• are autonomous systems without general control

• may not provide sufficient quality of service ("dirty" boxes - have bugs, do not fit, have poor specification and documentation, etc.)

• are black boxes (no source code, no spec) with known interfaces

• belong to different organizations

• are heterogeneous: they have different standards, fault assumptions, follow different conventions

• may be in operation when being integrated

• should provide individual services when integrated and when the integrated systems fails

• may change their behaviour on the fly.

4. Integration of Complex Web Applications

In addition to that:

• integrated systems are to be used by general public lacking computer-related skills

• the Internet is a poor communication medium: low quality, not predictable.

Dependability is a serious concern. These systems are inherently complex and are prone to many faults of many types. Besides, multiple abnormal situations are likely to happen concurrently.

Conventional hardware fault tolerance techniques (replication, ordered delivery, group communication, TMR techniques and retry – cf OMG Fault Tolerant CORBA service) can offer only partial solutions.

4. Integration of Complex Web Applications

We need fault tolerance mechanisms to deal with

- users' mistakes
- components mistakes
- component systems not delivering the service requested
- environmental faults
- component mismatches
- application developers' mistakes
- and with all types of errors propagated by the underlying levels (OS, middleware, hardware) when they fail to deliver the required services.

This is software fault tolerance at the application level (i.e., the level of integrated Web applications).

UNIVERSITY OF NEWCASTLE

4. Integration of Complex Web Applications

Forward error recovery because rollback/abort is not the right approach to deal with the faults of all these type. Compensation is a particular case of exception handling (BPEL focus on compensation is misleading).

Besides, canonical ACID transactions are not applicable for the Web:

• the OASIS BTP (IONA, Sun, BEA Systems, Choreology, etc.),

• WS-Transactions&WS-Cooperation (submitted to W3C by IBM, MS and BEA).

Existing schemes do not offer structured solutions, do not address cooperative/competitive systems, do not rely on cooperative exception handling.

CA actions are a good choice but there is a need for applying the general concept to dealing with external non-ACID resources/components.

5. CA Action Design of Web Applications

General requirements for CA actions in the Web context. They should allow for:

- dealing with component systems that are outside of our control
- relaxing entry/exit synchronization (people, documents, organizations, goods, etc.)
- new participants to be forked/joined
- other component systems to be invited/involved into an action when necessary

But they should

- keep atomicity, exceptions and error propagation under control
- provide co-operative exception handling

We rely on C.T. Davies' concept of *spheres of control* while extending CA actions.

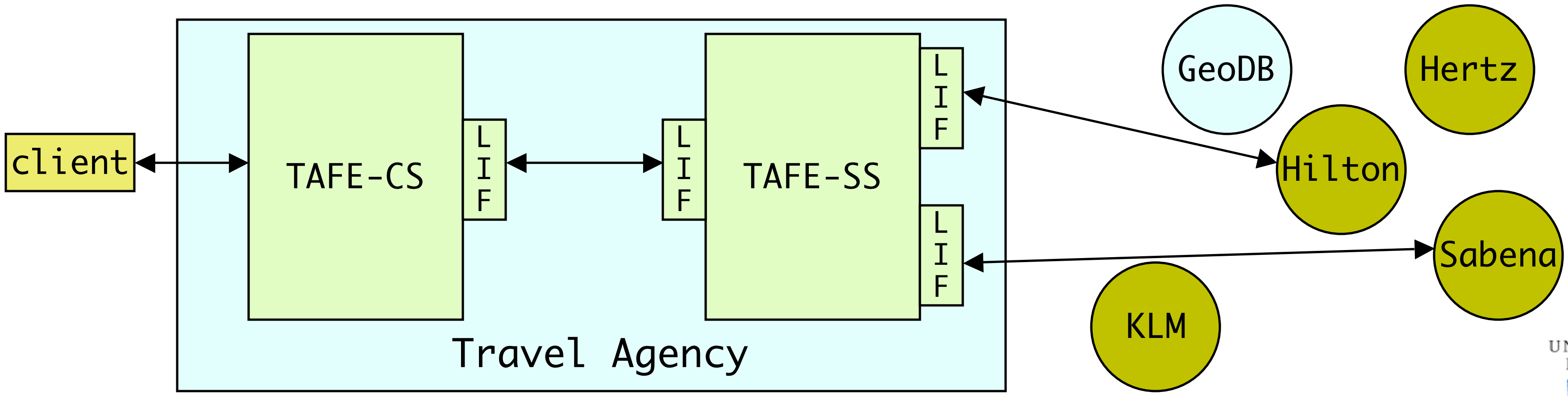
We should talk about *controlled atomicity*.

5. CA Action Design of Web Applications: Travel Agency

We use the canonical Travel Agency (TA) case study as a running example to demonstrate our ideas. TA was designed as a pair:

- the TA front end - client side (TAFE-CS): web front-end, exception handling, communication
- the TA front end - server side (TAFE-SS): access to existing services, trip composition, exception handling, component system monitoring

Linking Interface (LIF) is a reduction of the component system (WS) specification that includes its functional, temporal and dependability descriptions, which are required for integration.



UNIVERSITY OF NEWCASTLE

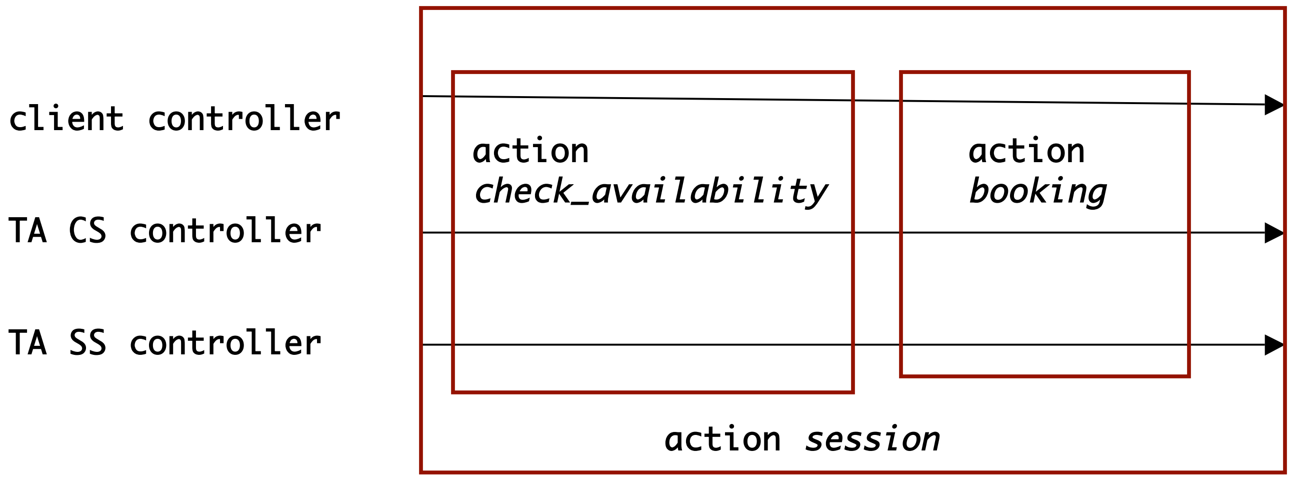
5. CA Action Design of Web Applications: Structuring

Application-level software fault tolerance in TA is achieved by structuring TA using nested CA actions and by employing disciplined exception handling within this framework.

Cooperative exception handling at the action level can involve individual WSs, people including the client, TA support, component system support (if possible).

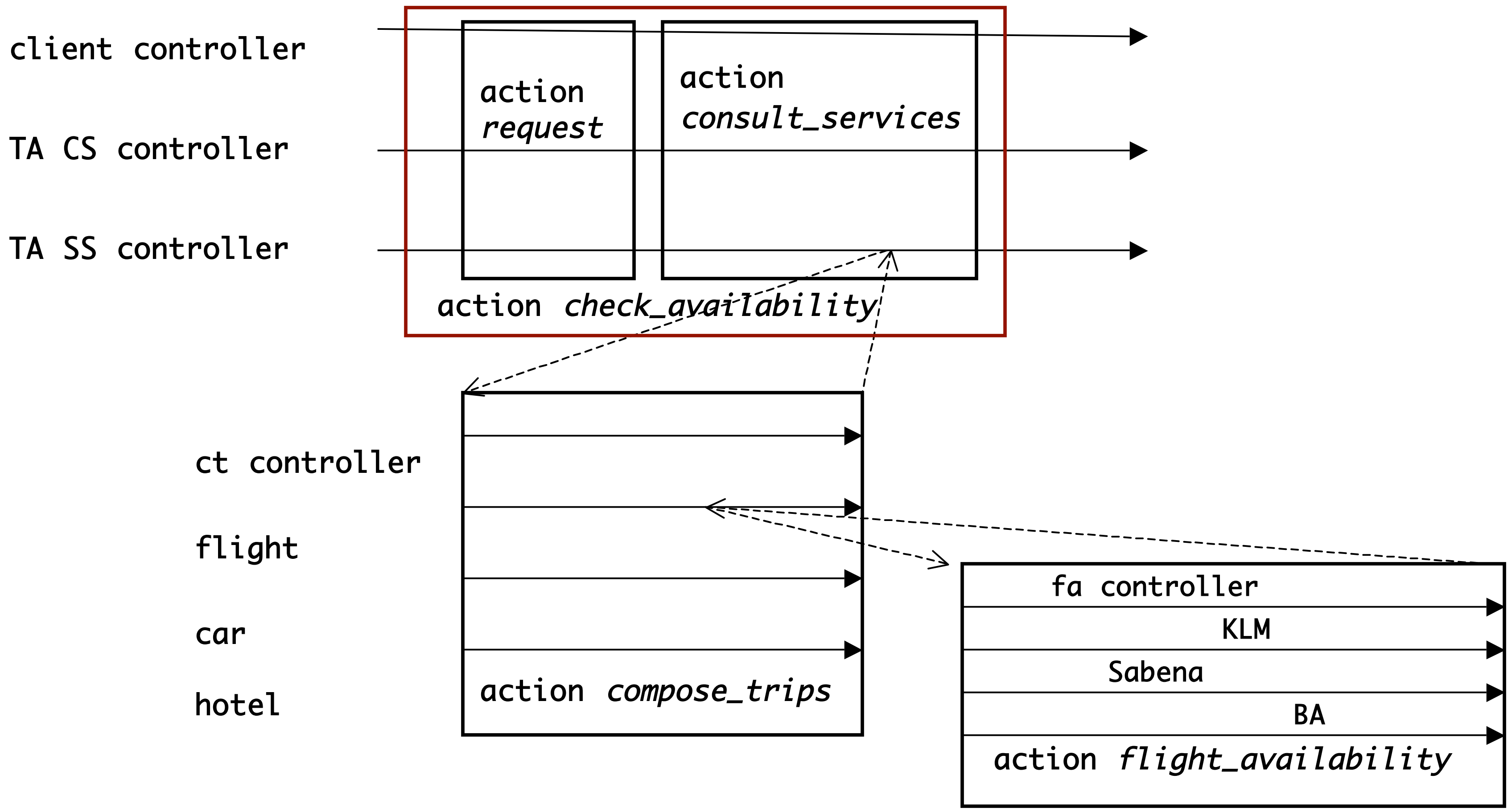
5. CA Action Design of Web Applications: Structuring

Each client session is a CA action consisting of a number of nested actions performing: availability checking, trip booking, trip cancellation, payment, etc.



UNIVERSITY OF NEWCASTLE

5. CA Action Design of Web Applications: Structuring

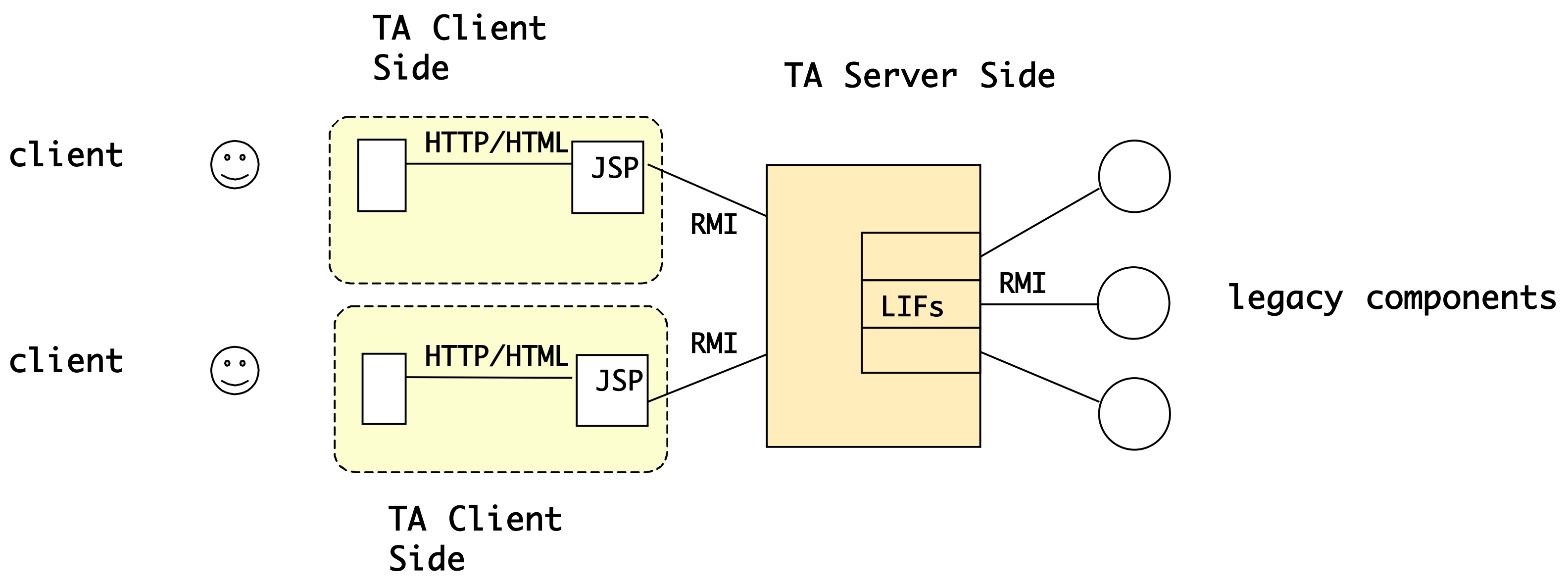


UNIVERSITY OF NEWCASTLE

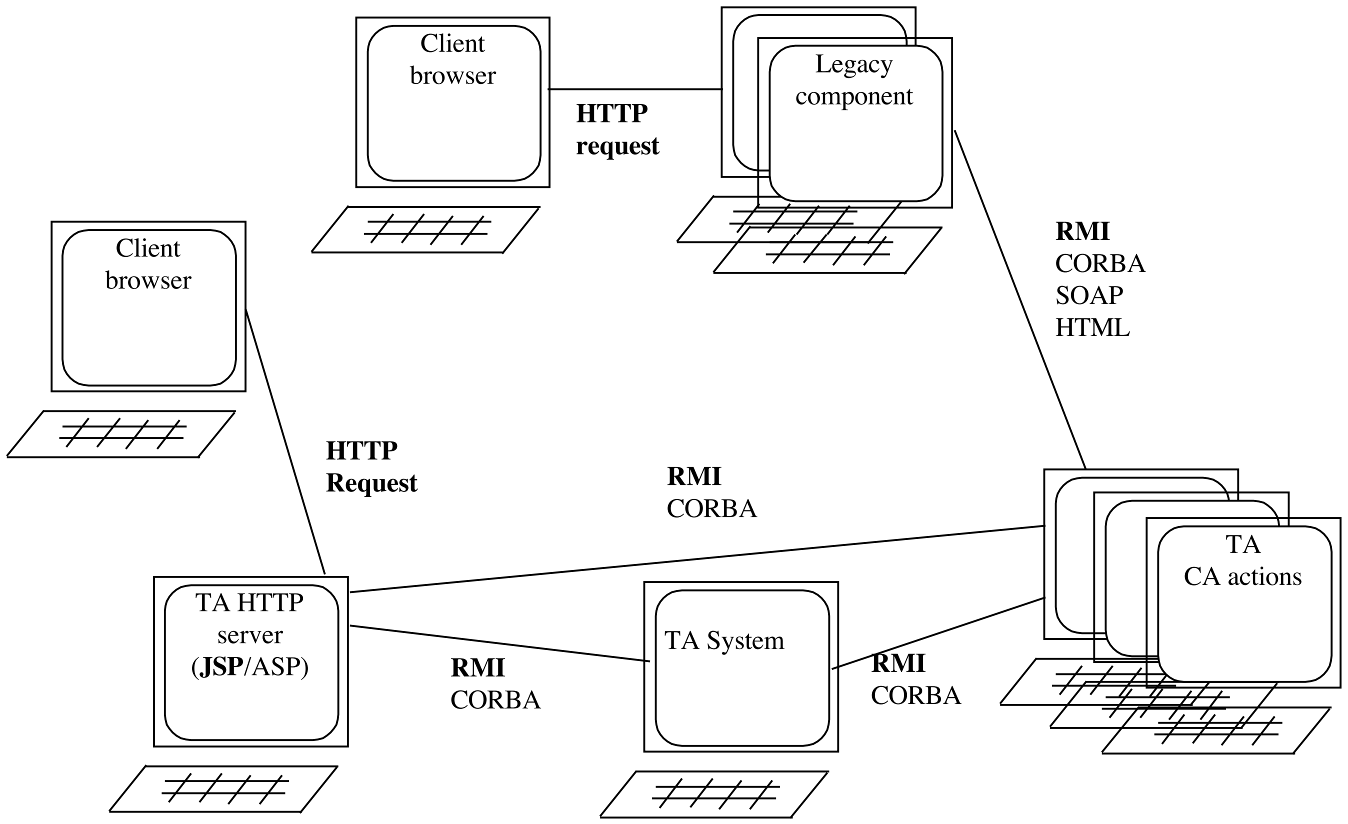
5. CA Action Design of Web Applications: Implementation

An experimental implementation of the Internet Travel Agency using Java RMI and JavaServer Page (JSP), and a distributed (RMI) CA action support developed in Newcastle.

General implementation structure:



5. CA Action Design of Web Applications: Implementation



UNIVERSITY OF NEWCASTLE

5. CA Action Design of Web Applications

Interfacing subsystems (LIFs) provide a number of functionalities, including support for:

- turning a component system into a CA action participant taking part in all action-specific activities such as cooperative exception handling (including resolution of concurrent exceptions at the action level), action entry and exit synchronisation (when necessary), etc.
- structuring the whole TA recursively using CA actions with component systems taking part in these actions
- performing systematic and disciplined local error detection and exception handling at the level of component systems and dealing with mismatches

Experience:

- structuring integrated WSs using CA actions
- systematic dealing with external non-ACID resources (weakening the atomicity).

5. CA Action Design of Web Applications: New Characteristics

CA actions allow for disciplined exception handling and system structuring for fault tolerance. New characteristics:

• control of the WSs accessed during action execution

• cooperation of WSs (normal and abnormal behaviour)

• dedicated processes representing participation of the individual WSs in the actions.

• flexible participation (using participant forking-joining)

• flexible structuring (action nesting and action composition)

UNIVERSITY OF NEWCASTLE

5. CA Action Design of Web Applications: WSCA

Ongoing joint work between INRIA Rocquencourt (V. Issarny's group) and Newcastle U.

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

An XML-based language is under development. The WSCA language (WSCAL) builds on coming W3C standards: Web Service Description Language (WSDL) and Web Services Conversation Language (WSCL).

It allows dependable composition of WSs to be specified at an abstract level supporting recursive system structuring and cooperative exception handling.

5. CA Action Design of Web Applications: Future Work

Our future work in the WS area focuses:

- **on developing a middleware WSCAL support to allow the skeletons of the composed applications to be automatically generated**
- **and on introducing extended CA actions into existing business-logic description and composition languages (BPEL4WS).**

UNIVERSITY OF NEWCASTLE

6. Future Work and Conclusions

Fault tolerance in open dynamic loosely-coupled systems:

• Mapping the CA action concept into the context of agent systems. In particular, introducing nested structuring and cooperative exception handling using LGI

• Development of the CA action-based schemes for mobile (e.g. tuple-based) environments

Software engineering issues:

• Formalisation, including action and participant refinement and decomposition

• CA actions in software architecture/ADLs

CA actions for fault tolerance in service-oriented architectures.

UNIVERSITY OF NEWCASTLE

6. Future Work and Conclusions

It is unfortunate and counterproductive to focus all efforts on making systems faultless.

Exception handling is the most general means for tolerating faults of the widest possible range.

Fault tolerance features to (e.g. exception handling mechanisms) should match the development paradigm, the specific characteristics of the application domain, the computational model.

(Recursive) system structuring and providing system fault tolerance should go hand in hand in developing complex systems.

Thanks to Brian Randell, Avelino Zorzo, Valerie Issarny, Jie Xu, Cecilia Rubira, Panos Perriorelis, Ian Welch, Robert Stroud, Galip-Ferda Tartanoglu, Cliff Jones, Joey Coleman and Nicole Levy.

Two EC Projects:

Design for Validation - DeVa (1996-1999)

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

Dependable Systems of Systems - DSoS (2000-2003)

The European Commission
Community Research
ist
information society technologies