Software Fault Tolerance

Cooperative Concurrency

Jörg Kienzle
Software Engineering Laboratory
School of Computer Science
McGill University

McGill

Overview

- Conversations
  - Synchronous vs. Asynchonous Entry
- Atomic Actions
  - Cooperative Exception Handling
  - Preemptive and Non-preemptive Execution
- Ada Implementation of Atomic Actions

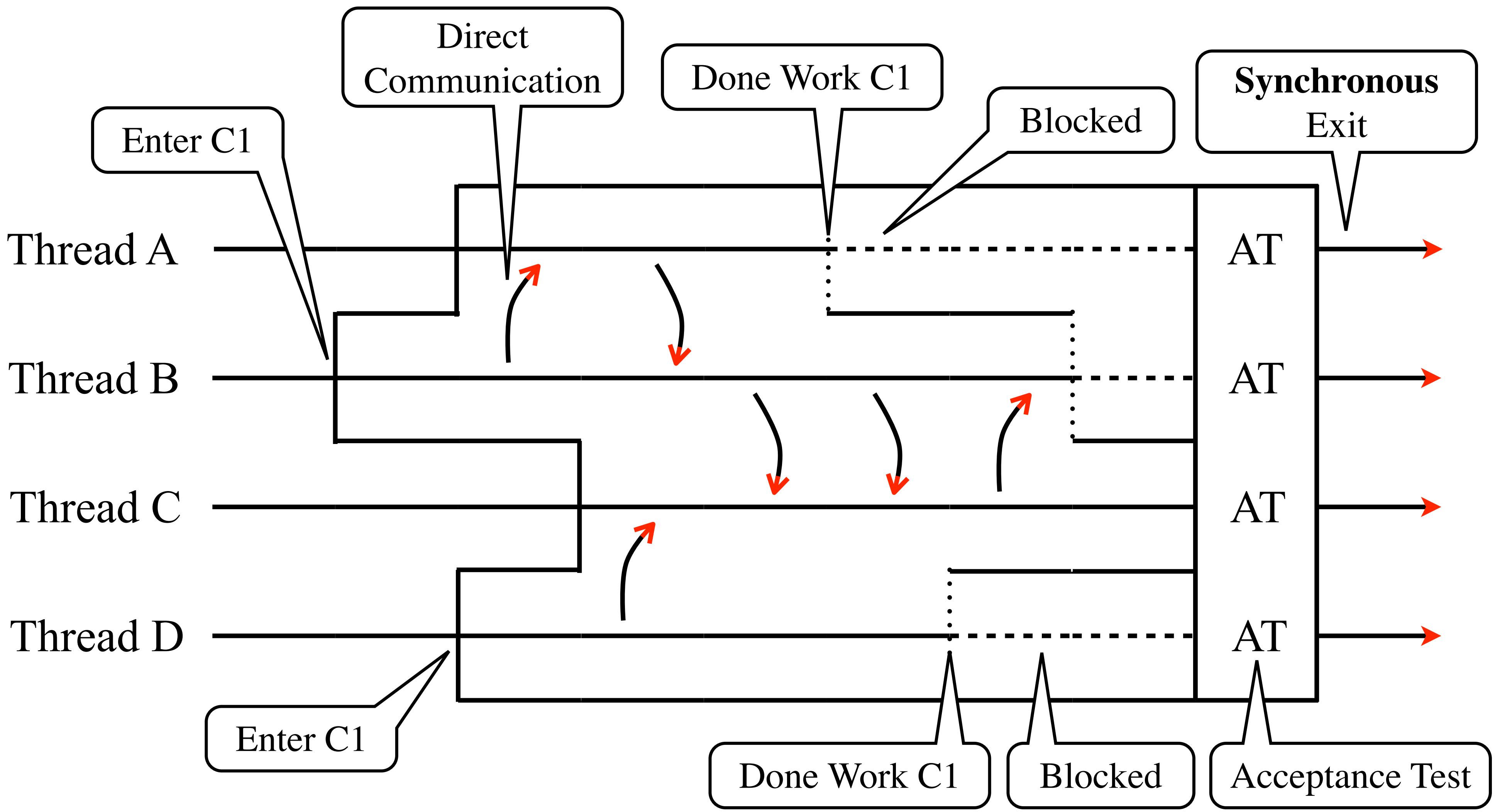
Cooperating Concurrent Systems

- Processes (or threads) running in the system have been designed together
- Are aware of each other
  - Communicate explicitly with other processes
  - Share resources
  - Cooperate to achieve a joint goal
  - Must also cooperate in case of exceptional situations

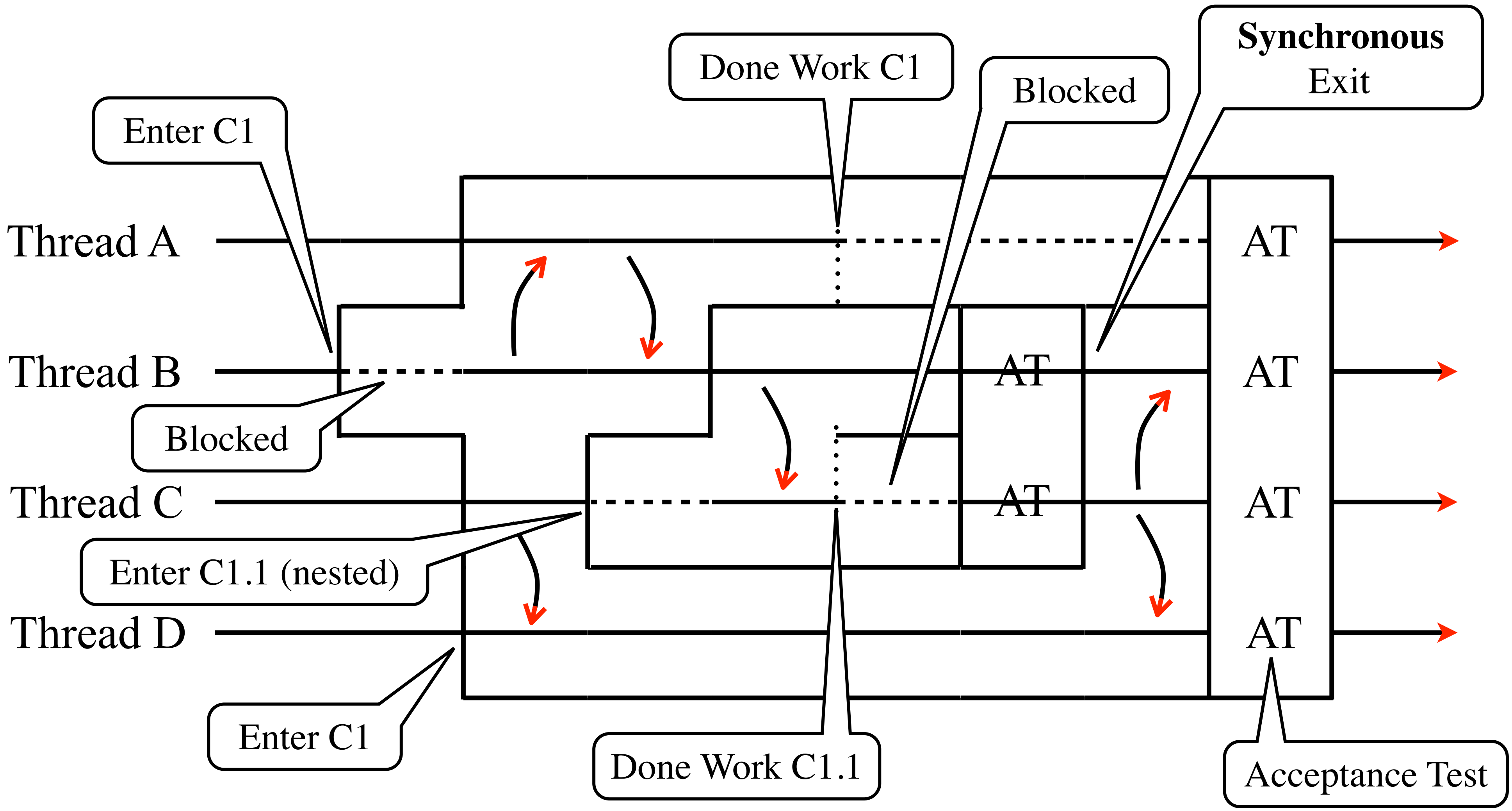
Conversations

- Introduced in 1975 [Ran75]
- "Concurrent recovery block"
- Fixed number of processes
- Upon entrance, a checkpoint is established in each of them
- Inside the conversation, the processes freely communicate
- No communication to the outside
  (side boundaries, no information smuggling!)
- When all processes come to an end, their acceptance tests are checked
  - If all are successful, the processes exit synchronously
  - Otherwise, the checkpoints are restored, and potential alternates are executed

Conversation Execution (Asynch. Entry)



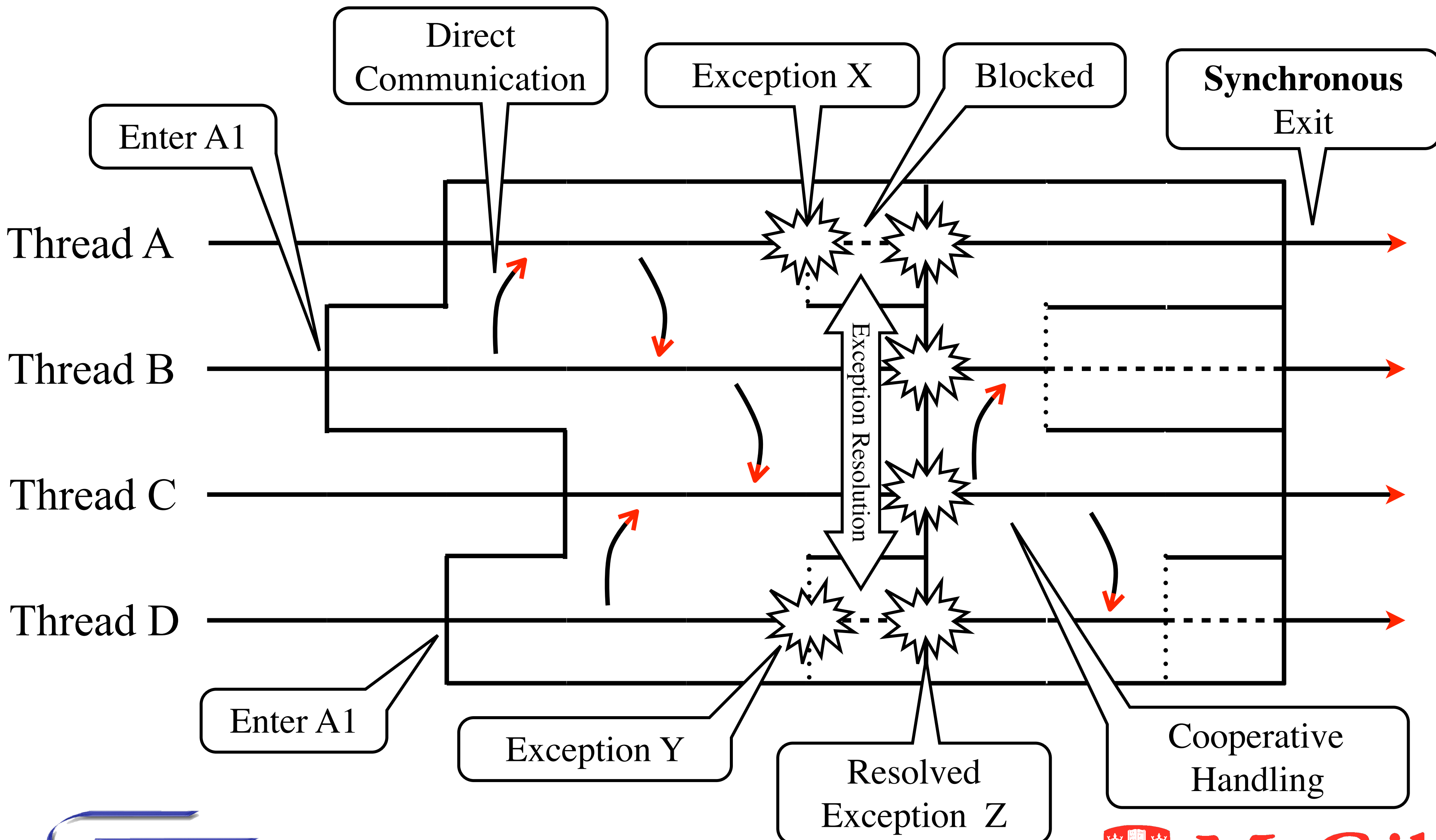
Conversation Execution (Synch. Entry)



Atomic Actions [CR86, LA90]

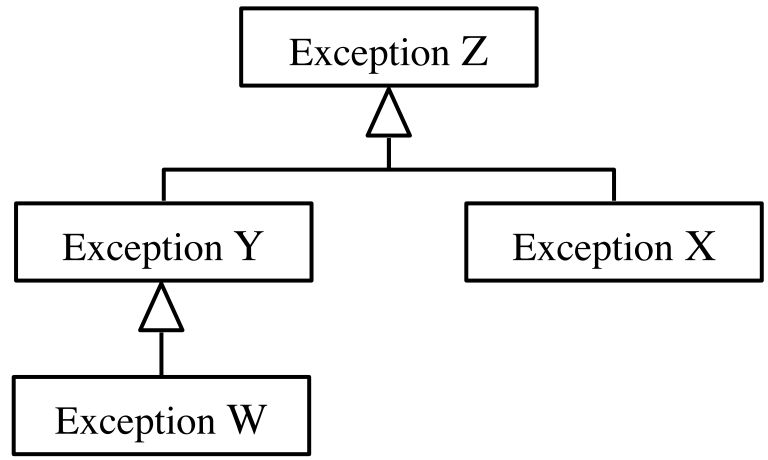
- Fixed number of processes
- Support for forward error recovery using exception handling
  - Internal exceptions
    - Handled by all participants
  - External exceptions
  - Recursive (external exceptions of a nested action are internal exceptions of the containing one)
  - Cooperative handling
    - Reasoning: Error can spread to all participants
    - Concurrent exception resolution
- Can include support for backward error recovery just like conversations with checkpointing and acceptance test

Atomic Action Execution



Atomic Action Issues

- How to perform exception resolution?
  - Resolution Tree or Resolution Graph



- How to inform participants of an exception that occurred in other participants?
  - Pre-emptive vs. non-preemptive models

Non-Preemptive

- Each participant performs its work, and at the end synchronizes with the others to inform them of success or of encountered exceptions
- Advantages
  - Participants are in a consistent state, ready for recovery
  - Nested actions have completed
- Disadvantages
  - Wasted time
  - Can’t handle infinite loops
- Optimizations
  - Abort participant when library takes control

Pre-emptive

- Other participants are interrupted / notified as soon as an error is encountered
- Advantages
  - No wasted time
  - Can handle infinite loops
- Disadvantages
  - Needs special language or OS support
  - Often results in high run-time overhead in fail-free mode
  - Consistency problems
  - Hard to prove correctness
  - Nested action abortion is problematic

Atomic Action Specification

```
with Ada.Exceptions; use Ada.Exceptions;

type Exceptions is array (Positive range <>) of Exception_ID;

generic
  with procedure Resolve(E: Exceptions) return Exception_ID;
package Atomic_Action_Support is
  type Action_Type (Participants: Positive) is tagged limited private;
  type Vote_Type is (Commit, Abort);
  generic
    with procedure Work;
    with function Exception_Handler(E: Exception_Id) return Vote_Type;
  procedure Action_Component (A: access Action_Type'Class);
  Atomic_Action_Failure: exception;
private
  -- continued on atomic action implementation slide
```

User-Defined Atomic Action (1)

```
with Atomic_Action_Support;

procedure My_Resolution(E: Exceptions) is ...;
package My_Support is new Atomic_Action_Support();
use My_Support;

package My_Atomic_Action is
  type My_Action_Type is tagged limited private;
  procedure Participant_1 (A: access My_Action_Type);
  procedure Participant_2 (A: access My_Action_Type);
  My_Action_Failure: exception;
private
  type My_Action_Type is tagged record
    Action : Action_Type (2);
  end;
end My_Atomic_Action;
```

User-Defined Atomic Action (2)

```
package body My_Atomic_Action is
  procedure Participant_1 (A: access My_Action_Type) is
    procedure My_Work is
    begin
        -- perform work ...;
    end My_Work;
    function My_Error_Handler (E: Exception_Id) return Vote_Type is
    begin
      -- handle error ...
      return Commit; -- or abort, if exception could not be handled
    end My_Error_Handler;
    procedure Work is new Action_Component(My_Work,My_Error_Handler);
  begin
    Work(A.Action'Access);
  exception
    when Atomic_Action_Failure => raise My_Action_Failure;
  end Participant_1;
  -- same for participant 2
end My_Atomic_Action;
```

Atomic Action Use

```
with My_Atomic_Action; use My_Atomic_Action;

A1 : My_Action_Type;

task Client_1;
task body Client_1 is
begin
  Participant_1(A1'Access);
exception
  when My_Action_Failure =>
    -- handle error
end Client_1;

task Client_2;
task body Client_2 is
begin
  Participant_2(A1'Access);
end Client_2;
```

Atomic Action Implementation (1)

```
  -- continuation of atomic action specification slide
private
  protected type Action_Controller(Participants: Positive) is
    entry Wait_Abort;
    procedure Work_Successful;
    entry Get_Resolved_Exception(E: out Exception_Id);
    procedure Signal_Abort (E: Exception_Id);
  private
    Stopped: Positive = 0;
    Exception_Encountered: Boolean := False;
    Exceptions: array (1 .. Participants) of Exception_ID
      := (others => Null_ID);
    Resolved_Exception: Exception_ID = Null_ID;
  end Action_Controller;
  type Action_Type (Participants: Positive) is tagged limited record
      Controller: Action_Controller (Participants);
  end record;
end Atomic_Action_Support;
```

Atomic Action Implementation (2)

```
protected body Action_Controller(Participants: Positive) is
  entry Wait_Abort when Exception_Encountered is
  begin
    Stopped := Stopped + 1;
    if Stopped = Participants then All_Stopped := True;
  end Wait_Abort;
  procedure Work_Successful is
  begin
    Stopped := Stopped + 1;
    if Stopped = Participants then All_Stopped := True;
  end Work_Successful;
  procedure Signal_Abort (E: Exception_Id) is
  begin
    Stopped := Stopped + 1;
    if Stopped = Participants then All_Stopped := True;
    Exceptions(Stopped) := E;
    Exception_Encountered := True;
  end Signal_Abort;
  -- continued on next slide
```

Atomic Action Implementation (2)

```
    -- continued from previous slide
    entry Get_Resolved_Exception(E: out Exception_Id)
      when All_Stopped is
    begin
      if Stopped = Participants then
        Resolved_Exception := Resolve(Exceptions);
      end if;
      E := Resolved_Exception;
      Stopped := Stopped - 1;
      if Stopped = 0 then
        All_Stopped := False;
        Exception_Encountered := False;
        Resolved_Exception := Null_ID;
        Exceptions := (others => Null_ID);
      end if;
    end Get_Resolved_Exception;
  end Action_Controller;
end Atomic_Action_Support;
```

Atomic Action Implementation (3)

```
procedure Action_Component (A: access Action_T'Class) is
   X: Exception_Id; Decision: Vote_Type;
begin
   select
      A.Controller.Wait_Abort;
      A.Controller.Get_Resolved_Exception(X);
      Raise_Exception(X);
  then abort
      begin
         Work;
         A.Controller.Work_Successful;
      exception
         when E: others =>
            A.Controller.Signal_Abort(Exception_Identity(E));
      end;
      A.Controller.Get_Resolved_Exception(X);
      if X /= Null_ID then Raise_Exception(X);
   end select;
exception
   when E: others =>
      Decision := Exception_Handler(Exception_Identity(E));
      if Decision = Aborted then raise Atomic_Action_Failure;
      end if;
end Action_Component;
```

References

- [Ran75]
  Randell, B.: “System Structure for Software Fault Tolerance”, IEEE Transactions on Software Engineering 1(2), 1975, pp. 220 – 232.
- [CR86]
  Campbell, R. H.; Randell, B.: “Error Recovery in Asynchronous Systems”, IEEE Transactions on Software Engineering SE-12(8), August 1986, pp. 811 – 826.
- [LA90]
  Lee, P. A.; Anderson, T.: “Fault Tolerance - Principles and Practice”, in Dependable Computing and Fault-Tolerant Systems, Springer Verlag, 2nd ed., 1990.
- [RMW96]
  Romanovsky, A.; Michell, S. E.; Wellings, A.: Implementing Atomic Actions in Ada 95”, Technical Report, University of Newcastle Upon Tyne.

McGill