Formalizing Dependability Mechanisms in B: *From Specification to Development Support*

G. Ferda Tartanoglu

ARLES Research Project, INRIA Rocquencourt

In collaboration with:

V. Issarny (ARLES, INRIA Rocquencourt)

N. Levy (PRISM, U. Versailles Saint-Quentin-en-Yvelines)

A. Romanovsky (U. Newcastle upon Tyne)

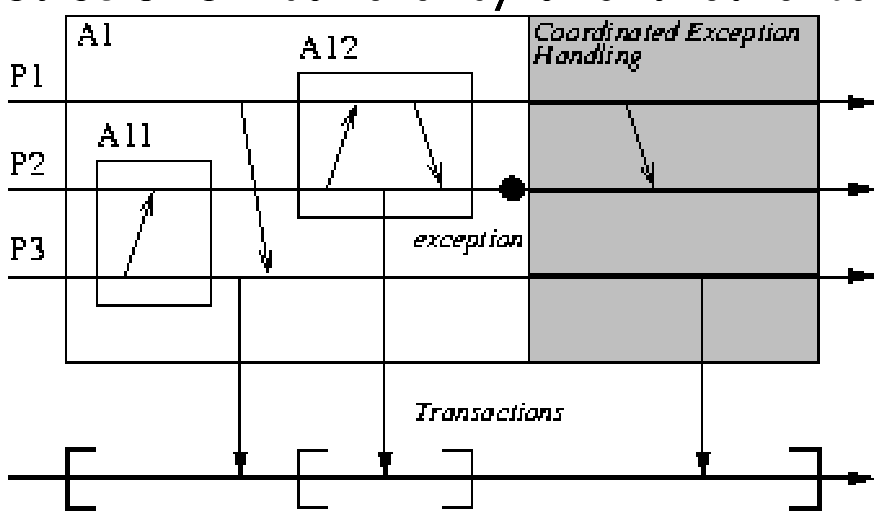
Introduction

- We formalize the notion of Coordinated Atomic Actions using the B method
  - Validate dependability mechanisms
    - Transactional access to external objects
    - Coordinated Exception Handling
    - Atomicity
  - Provide an XML-based declarative language for building dependable systems
    - The B formal specification is refined to obtain an implementation of the associated runtime support

Architecting Dependable Systems with Coordinated Atomic Actions

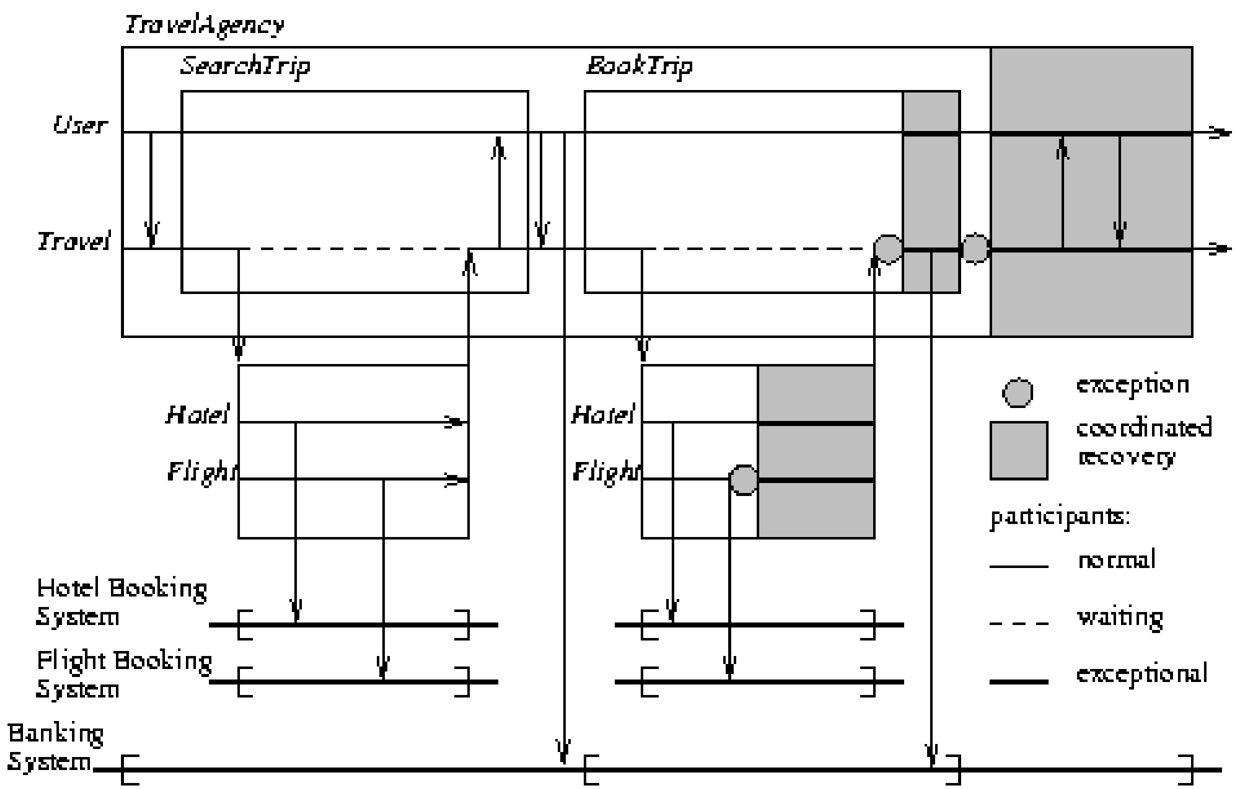
Coordinated Atomic Actions

- Coordinated Atomic Actions (J. Xu, B. Randell, A. Romanovsky et al., 1995)
  - Structuring mechanism for developing dependable concurrent systems
  - **Atomic actions** : for controlling cooperative concurrency
    - Coordinated error recovery using **exception handling**
  - **Transactions** : coherency of shared external resources



CA Actions Composition

- Allows the design of distributed systems built out of several CA actions [Tartanoglu et al., ICSE-WADS 2002]



Specifying CA Actions in B

- Offer a general framework that can be instantiated to describe the implementation of a specific system that is developed using CA actions
  - Dependability properties associated with CA actions will be enforced for any system based on them

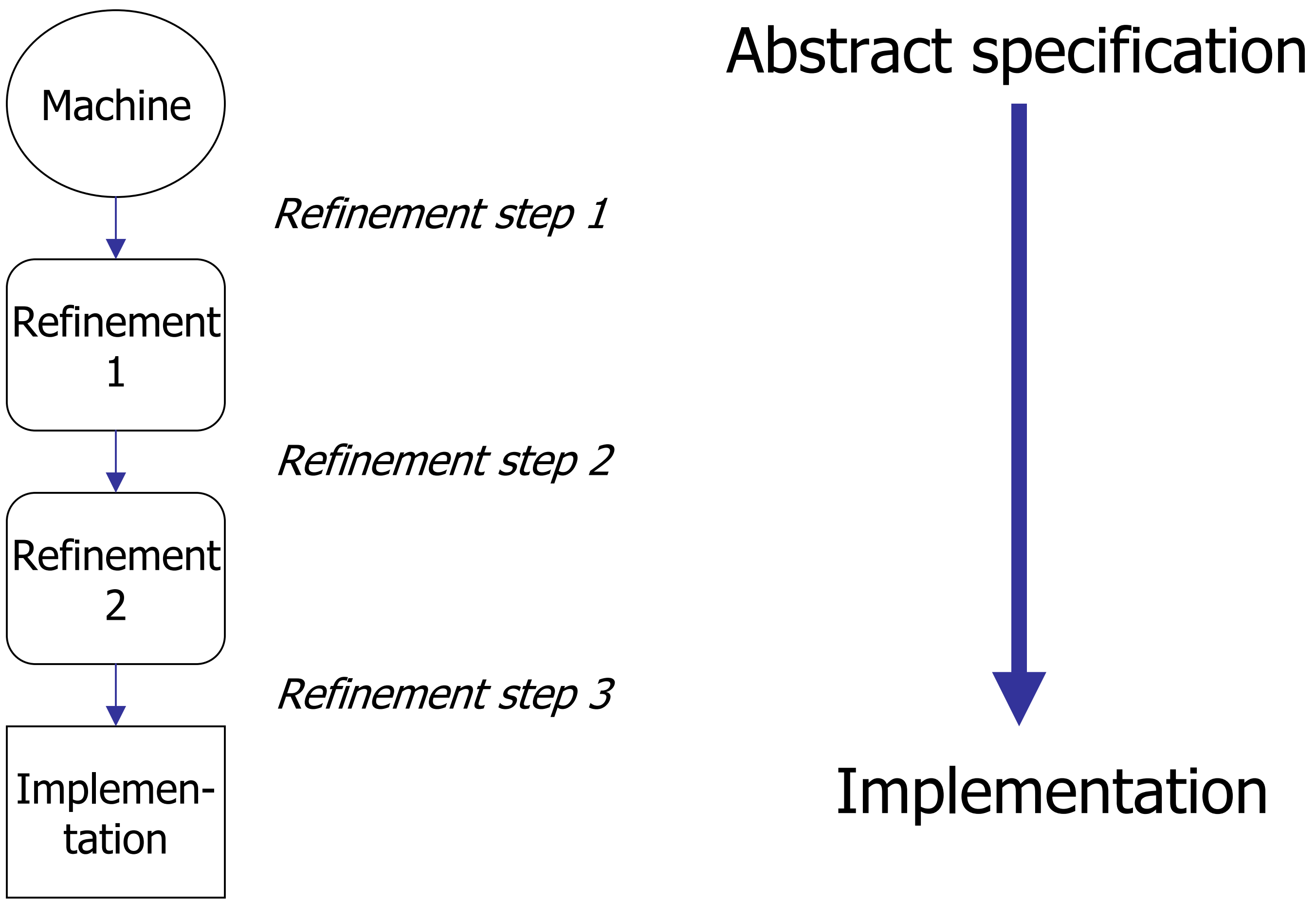
The B Method

- A **model-based** (state-based) method built on **set theory** and **predicate logic** and extended by **generalized substitutions**

- Specifications are represented by **abstract machines**

- A machine encapsulates **operations** and **states**
    - Set of variables

Refinement in B



Proofs

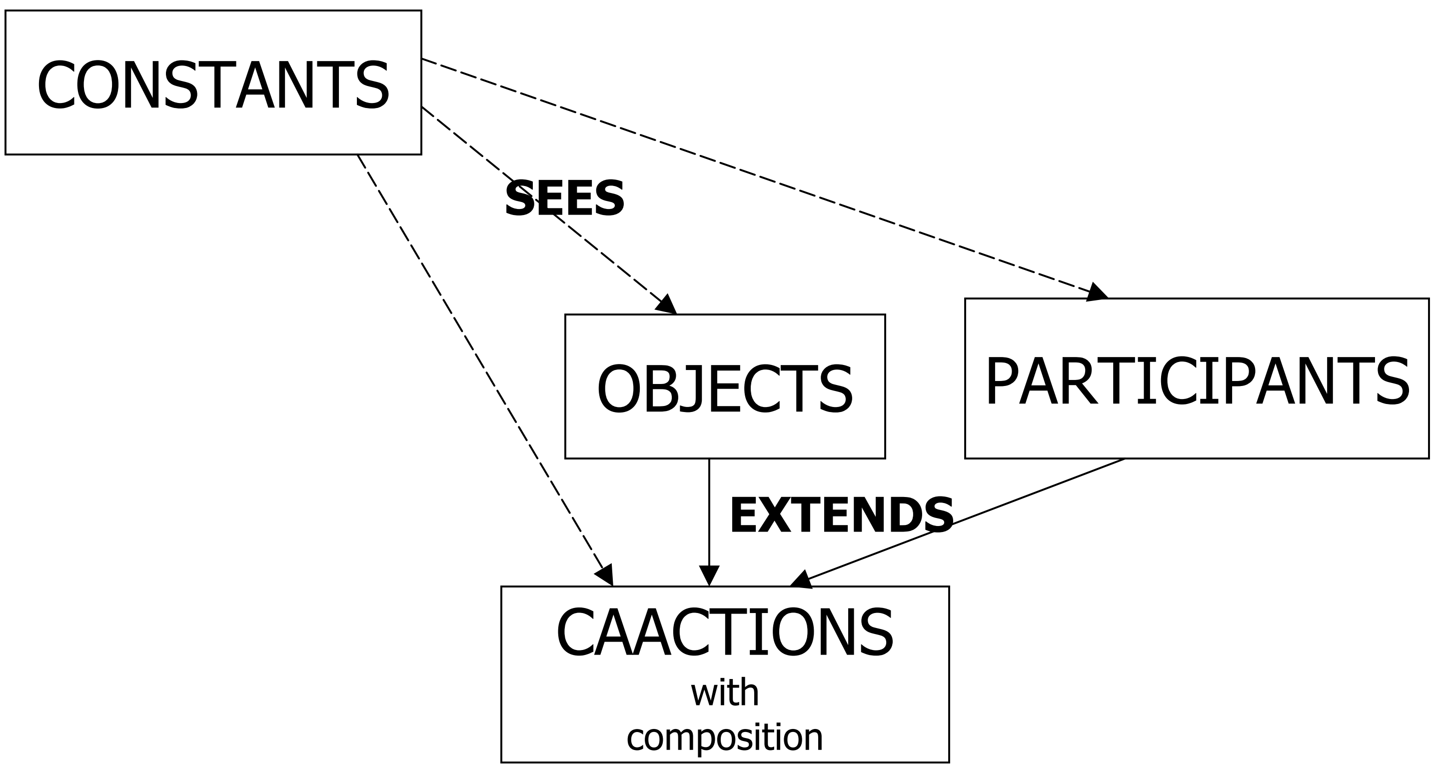
- In B, we prove that
  - All **operations** preserve the **invariants** of the machine
  - **Implementations** and **refinements** preserve the **invariant** and the **behavior** of the initial abstract machine

B Tools

- Atelier B (ClearSy, France)
- B-Toolkit (B-Core, UK)
- Both tools include
  - type checker
  - animator
  - proof obligation generator
  - theorem prover
  - code translators
  - documentation facilities

Modelling CA Actions

- Structure of the B specification



States and Operations

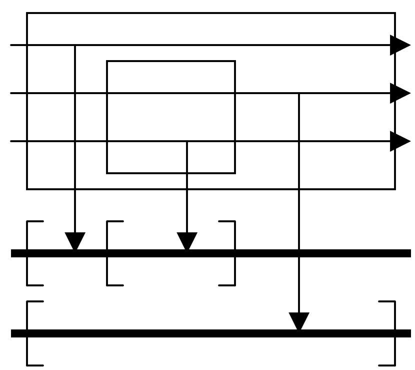
- **CAACTIONS abstract machine attributes**
  - CAACTION_STATE={caa_normal,caa_exceptional}
  - caaction_state $\in$ caaction $\rightarrow$ CAACTION_STATE
  - participant_of_caaction $\in$ caaction $\rightarrow$ **P**(participant)
  - caaction_of_participant $\in$ participant $\rightarrow$ **seq**(caaction)
  - caaction_ext_objects $\in$ caaction $\leftrightarrow$ objects
- **Pre-conditioned operations**
  - create_{main,nested,composed}_caaction
  - {send,recv}_message
  - {read,write}_object(participant,participant,message)
  - raise_exception(participant,exception)
  - propagate_exception(participant)
  - abort_{main,nested,composed} (caaction)
  - terminate_{main,nested,composed} (caaction)

Formalizing Dependability Mechanisms

- Transactions on external objects
- Atomicity of CA Actions
- Coordinated exception handling

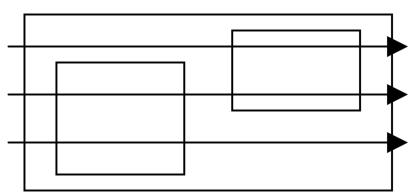
Transactions on External Objects

- Participants *setpar* of nested CA action *caa1* can only access subset *setobj* of external objects associated to containing CA action *caa2*
  - $\forall obj.(obj \in setobj \Rightarrow obj \in caaction\_ext\_objects[\{caa2\}])$



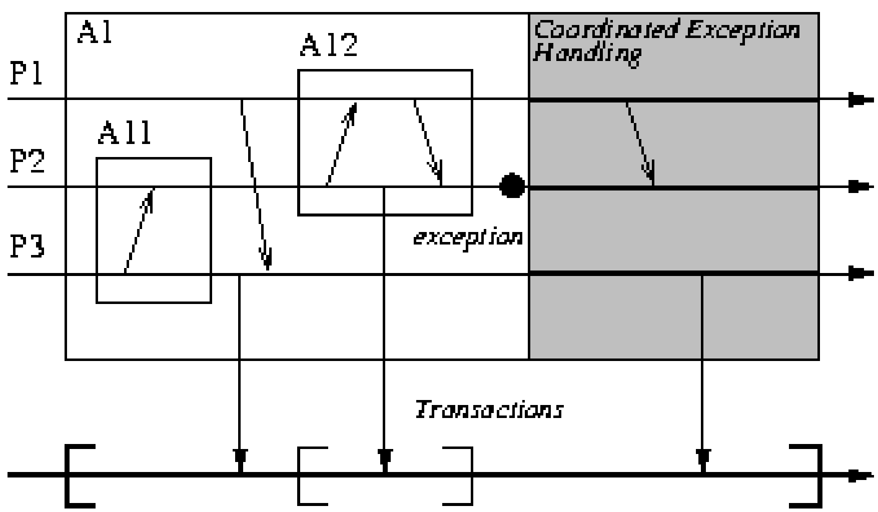
Atomicity of CA Actions

- Participants of nested CA action *caa1* are also participants of containing action *caa2*
  - $\forall$ (caa1,caa2).((caa1 $\in$ caaction Λ caa2 $\in$ caaction Λ (caa1,caa2) $\in$ is_nested) $\Rightarrow$ participants_of_caaction(caa1) $\subseteq$ participants_of_caaaction(caa2))
- A participant can only enter one sibling nested CA action at a time
  - **card**(**ran**({p,c | p $\in$ setpar Λ c $\in$ CAACTION Λ c=**last**(caaction_of_participants(p))})) = 1



Coordinated Exception Handling

- A CA action is set to an exceptional state if all of its participants are in the exceptional state
  - $\forall$ (caa). (caa $\in$ caaction Λ caaction_state(caa)=caa_exceptional
    $\Rightarrow$ $\forall$ (p).(p $\in$ participant_of_caaction(caa)
    $\Rightarrow$ (**last**(participant_state(p)) $\in$ EXCEPTIONAL_STATE)))



From the B Specification to the Development Support

Refinement

- In order to have an implementation of the CA action’s runtime support, the abstract machines are refined
  - B operations offered as a programming library
- Existing libraries are used
- To be able to prove the correctness of the implementation
  - Formal specification of the behavior of these methods
  - Prove that the refinement of the machines that use these methods are correct

Declarative Language

- XML-based declarative language for building CA action-based systems

```
<caaction name="nmtoken"? >
  <composedActions> ?
      <action name="qname" /> *
  </composedActions>
  <nestedActions> ?
      <nested name="nmtoken" /> *
  </nestedActions>
  <external> ?
      <object name="nmtoken" /> *
  </external>
```

Participant Behavior

```
<participants>
   <participant name="nmtoken"> +
        <var>
           <element name="nmtoken" type="qname"/> *
        </var>
        <behavior>
                <normal>
                        statements …
                </normal>
                <exceptional handle="qname"> *
                        statements …
                </exceptional>
        </behavior>
   </participant>
</participants>
```

Statements

<**invoke action**=“qname” **input**=“qname”? **output**=“qname”?/>

➔ *create_composed*

<**send rcpt**=“qname” **input**=“qname”/>

➔ *send_message*

<**recv from**=“qname” **output**=“qname”/>

➔ *recv_message*

<**call rcpt**=“qname” **input**=“qname” **output**=“qname” />

➔ *{read,write}_object*

<**assign element**=“qname” **value**=“xpath”/>

➔ *set_value*

<**raise exception**=“qname” **message**=“qname”? />

➔ *raise_exception*

<**nest nestedaction**=“qname”? > **behavior** … </**nest**>

➔ *create_nested*

Future Work

- Development support
  - Implementation of a compiler/code generator for the declarative language
- Extend the base CA Action model using the formal model
  - Already used to introduce CA Action composition
  - Relax atomicity properties

Web Services Composition Actions

- Relaxes the transactional requirements over external interactions
  - Relaxed atomicity
  - Compensations when available : *semantic atomicity*