LABORATOIRE D'INFORMATIQUE DE NANTES ATLANTIQUE
Behaviour Abstraction from Code

# The Kmelia Component Model

## Hierarchical Service Description and Analysis

P. André, G. Ardourel, C. Attiogbé

COLOSS-LINA

september, 3-7 2007

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ DE NANTES

# Introduction

The motivation: need of models and practical tools to assist users in formal component-based development.

- component abstract definition and composition
- properties verification: safety, interoperability, compatibility...
- from components to code

The proposal: Kmelia, COSTO

- simple, formal, abstract component model
- service-based approach
  (functionalities, contracts, behaviour, composition)
- property checking (safety, liveness)
- tool assistance for the development (open platform)

# This Talk

Overview

- The Kmelia component model
- Property Formal Verification

  The COmponent Study TOolbox (COSTO)

Illustration on an ATM Case Study

- Specification
- Analysis

Discussion

# Outline of the part

# Kmelia Abstract Component Model

Kmelia: a simple and abstract component model based on services.

- simple: component, service, assembly, composition
  reduced number of concepts
- extensible: protocols, adaptors, aspects...
  add new concepts from the kernel
- abstract: over programming models
  reasoning, refinement, code generation...
- formal: types, contracts, LTS...
  support automated processing
- service: first class elements and not only messages
  basis for component description and component assembly

# Kmelia Components

- A component is a structuring unit that encapsulates a state and services; has an interface with usage constraints.
- A component interface: provided services and required services.

**Component** C1
**Interface** <Interface descr>
**Types** <Type Defs>
**Variables** <Var list>
**Invariant**
<Predicate>
**Initialisation**
... // var. assignments
**Services**
... // described later
**end**

# Kmelia Services

- A service is an abstraction of a functionality; it has a behaviour.
  signature + contract + dynamic evolution
- A service interface: subservices and requirements.
  hierarchical structuration
- Support for component connection and interaction.
  assembly links

```
Provided aService_1 ()
  Interface     <Interface descr>
  Pre           <Predicate>
  Post          <Predicate>
  Behaviour     // eLTS
  init          aStatel
  final         aState
  { state_i —label—> state_j
    ... }
end
Required aService_2 ()
    ...
```

# Kmelia Services: Behaviour Specification

The behaviour of a service *ss* $\langle S_{ss}, L_{ss}, \delta_{ss}, \Phi_{ss}, S_{0_{ss}}, S_{F_{ss}} \rangle$.

## Extended LTS

- States, initial state, final states
- Transitions: `source--label-->target`

  `label ::= [guard] actions*`

  Actions:
  - elementary action
  - communication: *service call/response* or message *communication*.

```
Communication ::= Channel(!|?|!!|??)message(param*)

Channel ::= SELF | __CALLER | RequiredServiceName
```

- Annotations

# Kmelia Services: Hierarchical Service Description

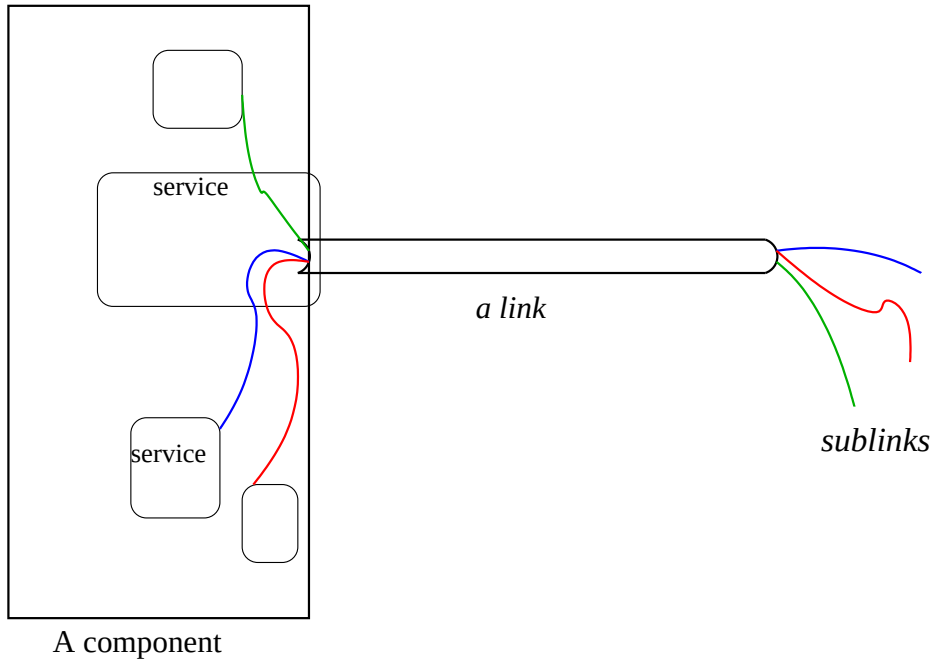
## Service Composition

- Horizontal Structuring: Interaction between Services
  - linking required and provided services
    (internally, by the caller, by a third component)
  - service call handled with communication mechanisms:
    !! for a service call `channel!!message(param*)`
    ?? for a service wait `channel??message(param*)`
- Vertical Structuring: State Annotation, Transition Annotation
  eLTS size reduction, service sharing, mastering service complexity
  - state annotation: <<...>> permits optionnal behaviours
  - transition annotation:[[...]] permits mandatory behaviours

Service Behaviour $\leftrightharpoons$ Service Interface

# Kmelia Assemblies and Compositions

- Connecting components on pairwise services.
  assembly links
- Conforms to the service interfaces.
  links and sublinks
- Encapsulation
  promotion links




Partial usage allowed only fulfilled chains are usable

# A Bank ATM Example

ATM = interface between clients and banks

Clients access via a User_Interface.

The banks are represented by one local bank (which can deliver cash) and a bank consortium (the authority center).

A Bank ATM usual functionalities: cash withdrawal, account query, cash deposit, cash transfer.

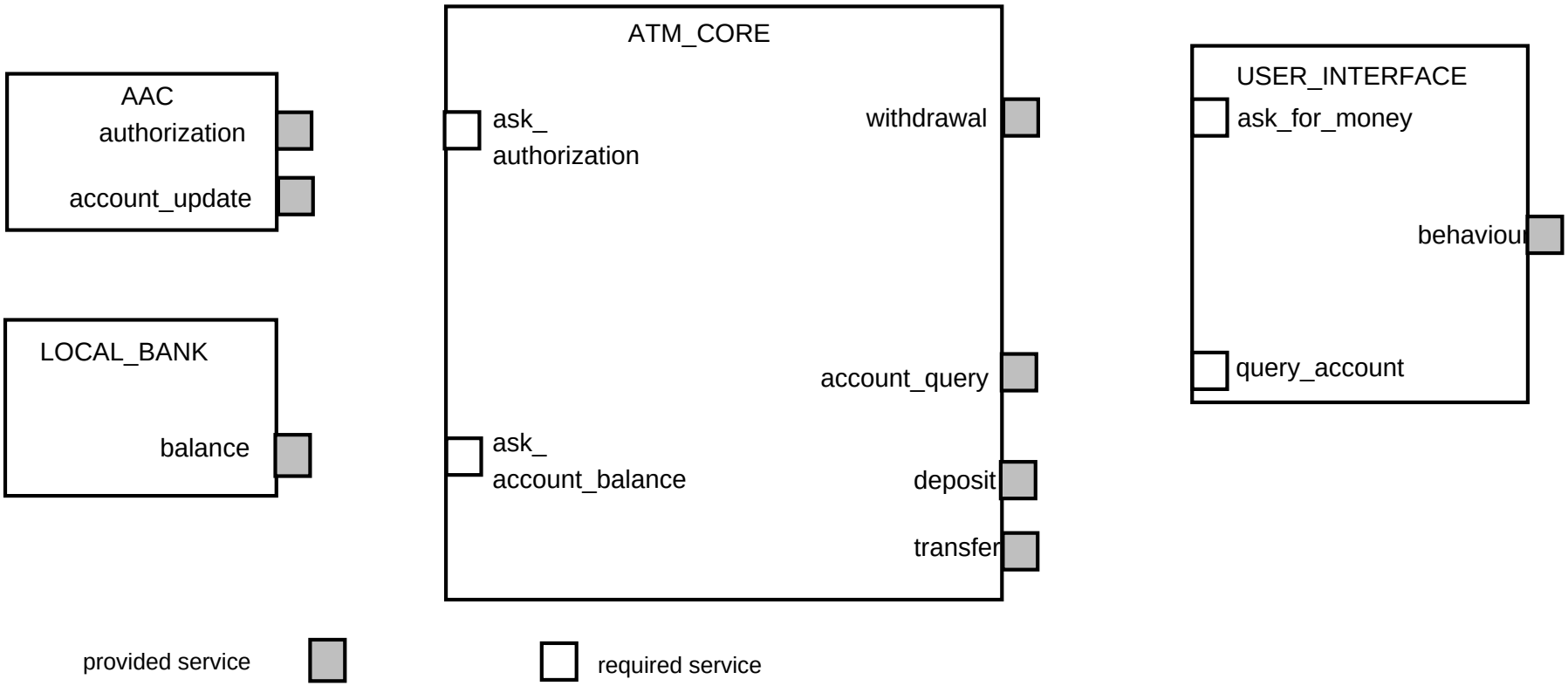
An ATM abstract specification $\rightsquigarrow$ 4 components

Application domain: concurrent systems with interactive services

# A Bank ATM Architecture

*ATM Case Study*

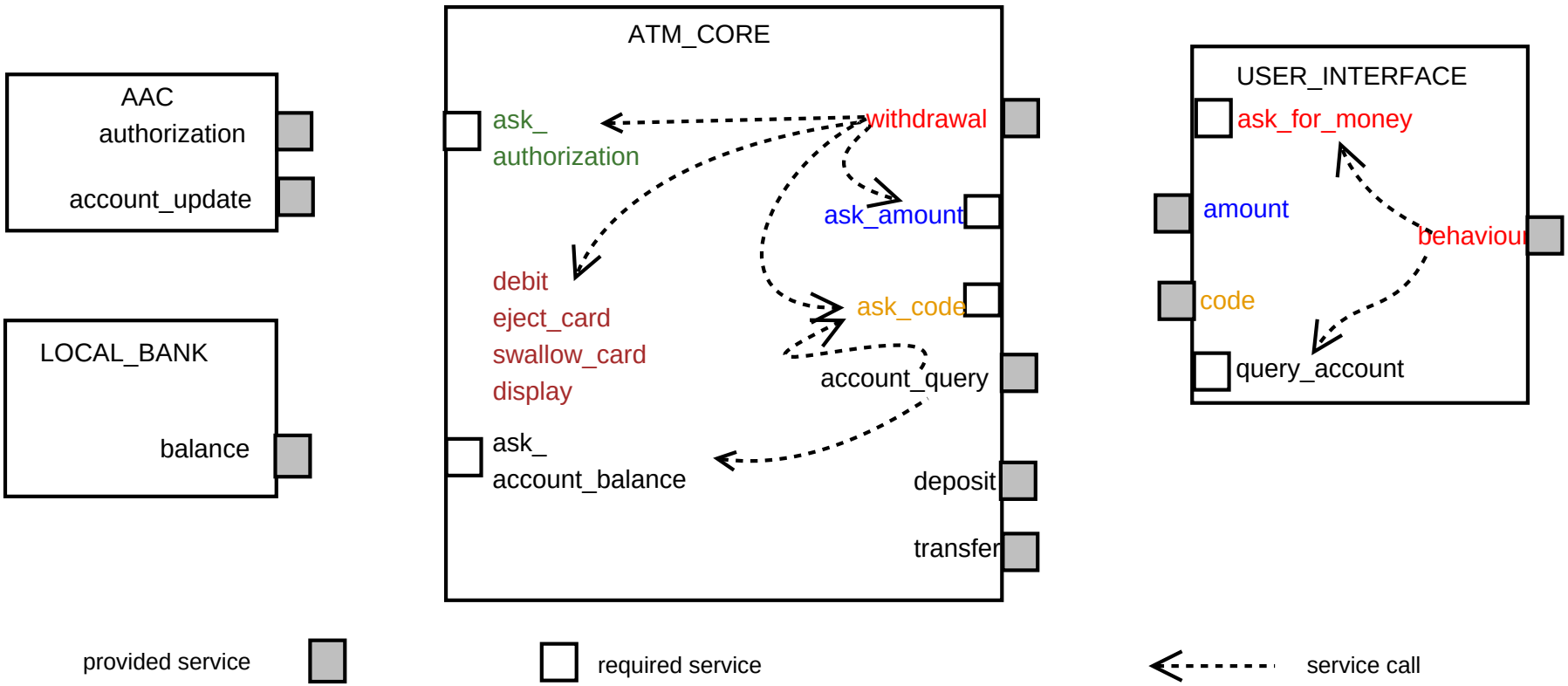
1 → *components + interfaces*



# A Bank ATM Architecture

*ATM Case Study*

2 → *service calls*

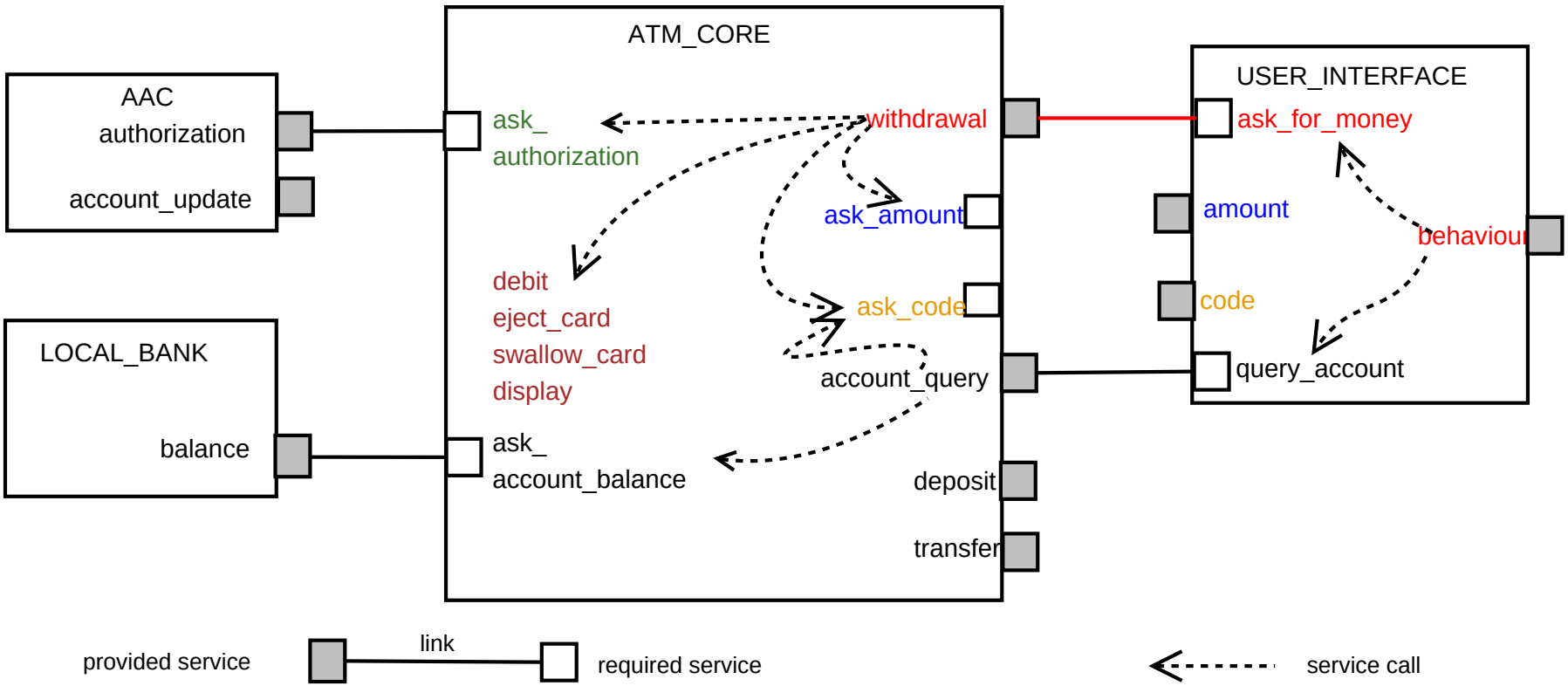


# A Bank ATM Architecture

*ATM Case Study*

3 → *links*

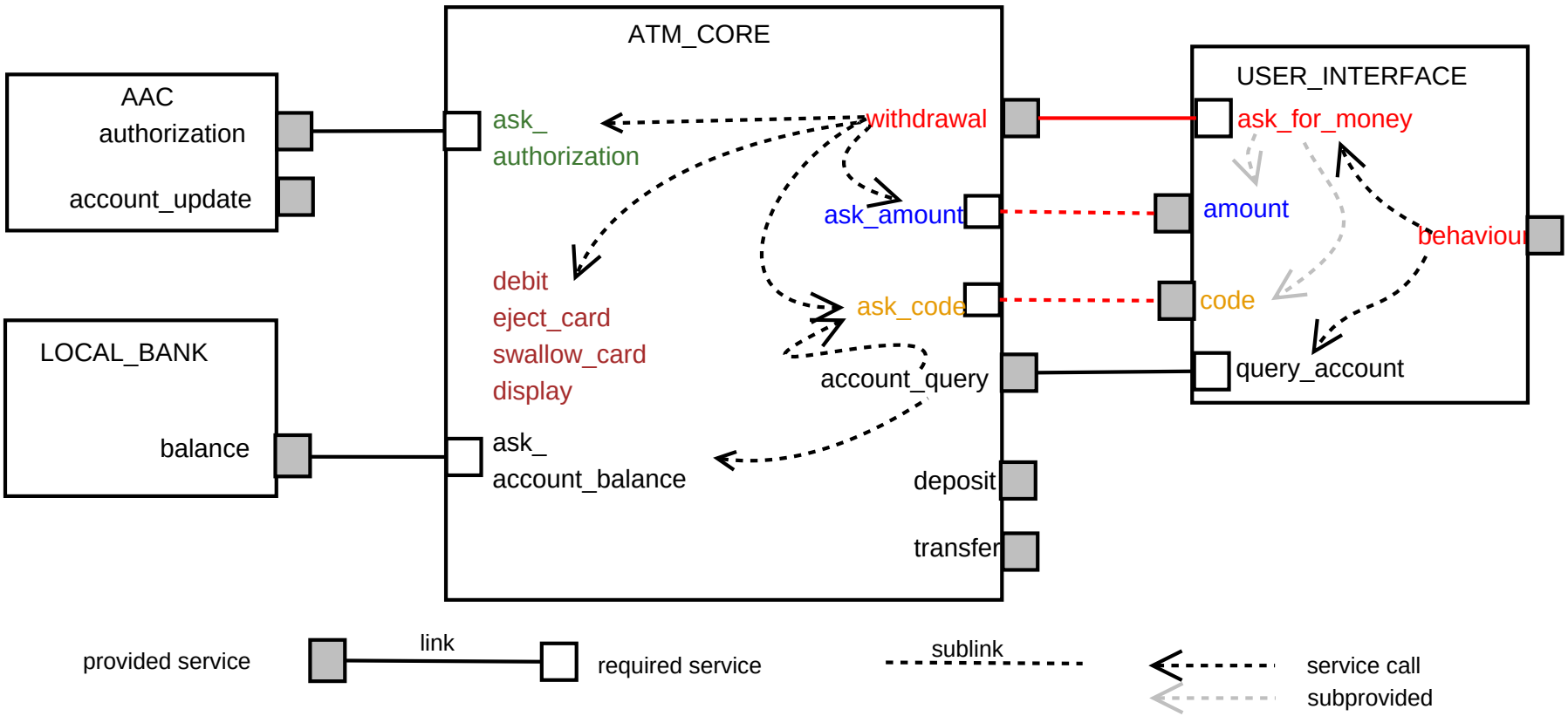
Connecting components on paired services



# A Bank ATM Architecture

*ATM Case Study* Connecting components on paired services

4 → *sublinks, subservices*

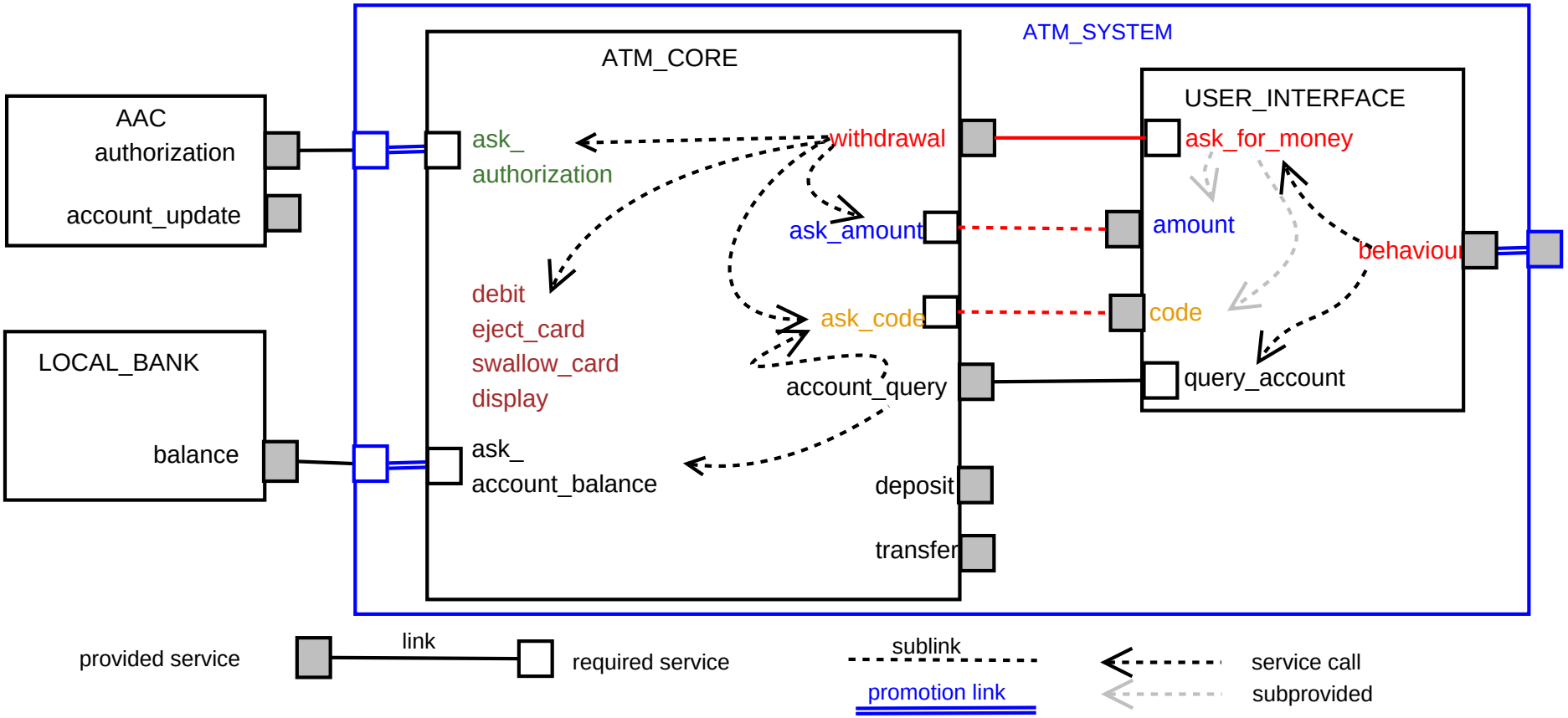


# A Bank ATM Architecture

*ATM Case Study*

5 → *encapsulation*

Connecting components on paired services



# Illustration of Component Specification

```
COMPONENT ATM_CORE
INTERFACE
  provides : {withdrawal, account_query, deposit, transfer}
  requires : {ask_authorization, ask_account_balance}

TYPES
  CashCard  :  struct {code:Integer, id:Integer, limit:Integer}
CONSTANTS
  available_cash : Integer := 100,
  swallowed_size : Integer := 100
VARIABLES
  name : String,
  swallowed_cards : Set,
  available_notes : Integer
PROPERTIES
 cash_disp: available_notes >= 0,
 card_capacity: size(swallowed_cards) <= swallowed_size
INITIALIZATION
  name := "ATM203";
  ...
```

# Illustration of Component Specification

```
COMPONENT ATM_CORE
INTERFACE
  provides : {withdrawal, account_query, deposit, transfer}
  requires : {ask_authorization, ask_account_balance}

TYPES
  CashCard  :  struct {code:Integer, id:Integer, limit:Integer}
CONSTANTS
  available_cash : Integer := 100,
  swallowed_size : Integer := 100
VARIABLES
  name : String,
  swallowed_cards : Set,
  available_notes : Integer
PROPERTIES
 cash_disp: available_notes >= 0,
 card_capacity: size(swallowed_cards) <= swallowed_size
INITIALIZATION
  name := "ATM203";
  ...
```

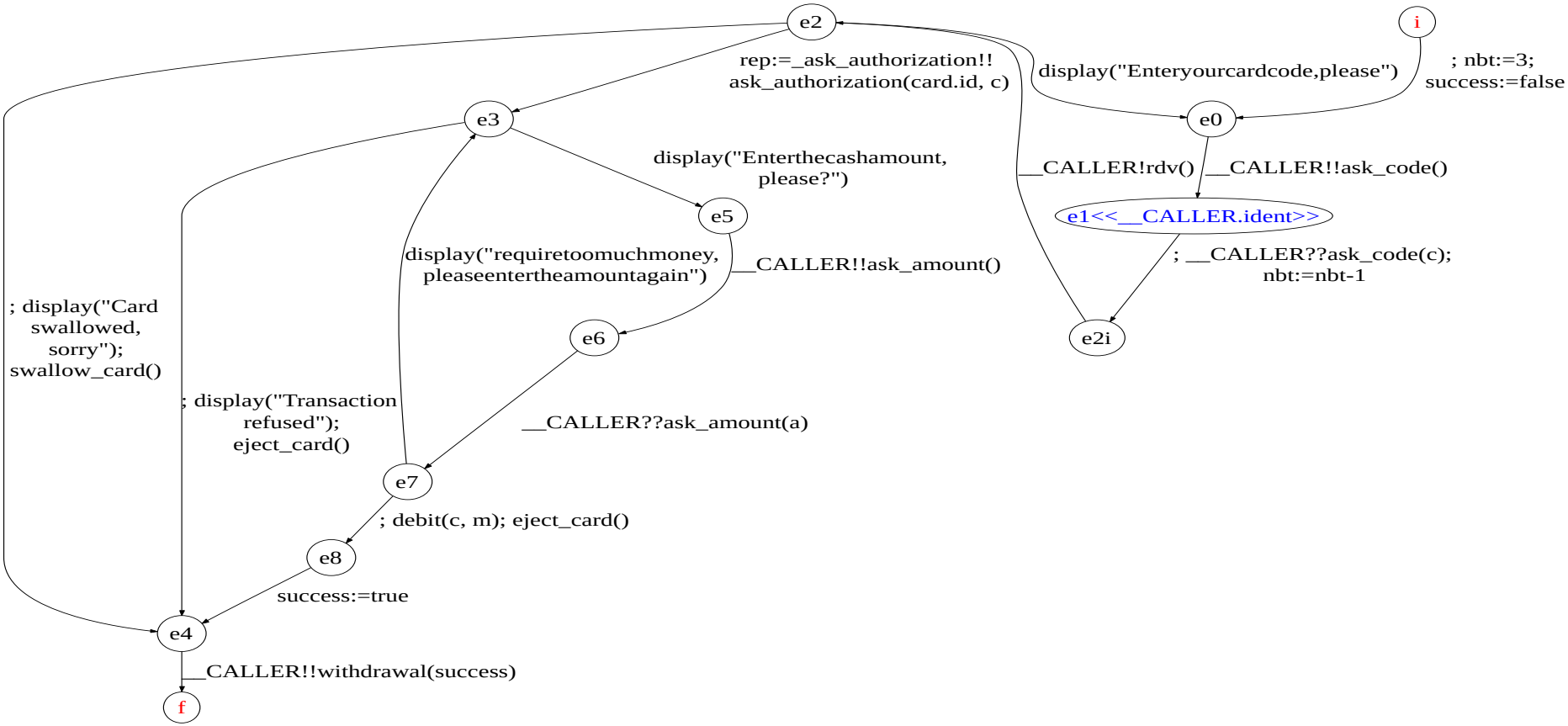
# Illustration of Service Specification

```
withdrawal (card : CashCard)
Interface
   subprovides : {ident}
   calrequires : {ask_code, ask_amount} //required from the caller
   extrequires : {ask_authorization}

Pre
  available_notes >= available_cash

Variables
  nbt : Integer,    // nbt : number of authorized trials of code entering
  c : Integer,      // c : input code given by the user
  a : Integer,      // a : input amount given by the user
  rep : Boolean,    // rep : reply from the authorization request
  success : Boolean // success : result of the withdrawal request
Behavior
  ...
Post
  available_notes <= pre(available_notes)
end
```

# Illustration of Service Specification

```
withdrawal (card : CashCard)
Interface
   subprovides : {ident}
   calrequires : {ask_code, ask_amount} //required from the caller
   extrequires : {ask_authorization}

Pre
  available_notes >= available_cash

Variables
  nbt : Integer,     // nbt : number of authorized trials of code entering
  c : Integer,       // c : input code given by the user
  a : Integer,       // a : input amount given by the user
  rep : Boolean,     // rep : reply from the authorization request
  success : Boolean // success : result of the withdrawal request
Behavior
  ...
Post
  available_notes <= pre(available_notes)
end
```

# Illustration of Service Behaviour Specification: eLTS




The behaviour of the withdrawal service

# Outline of the part

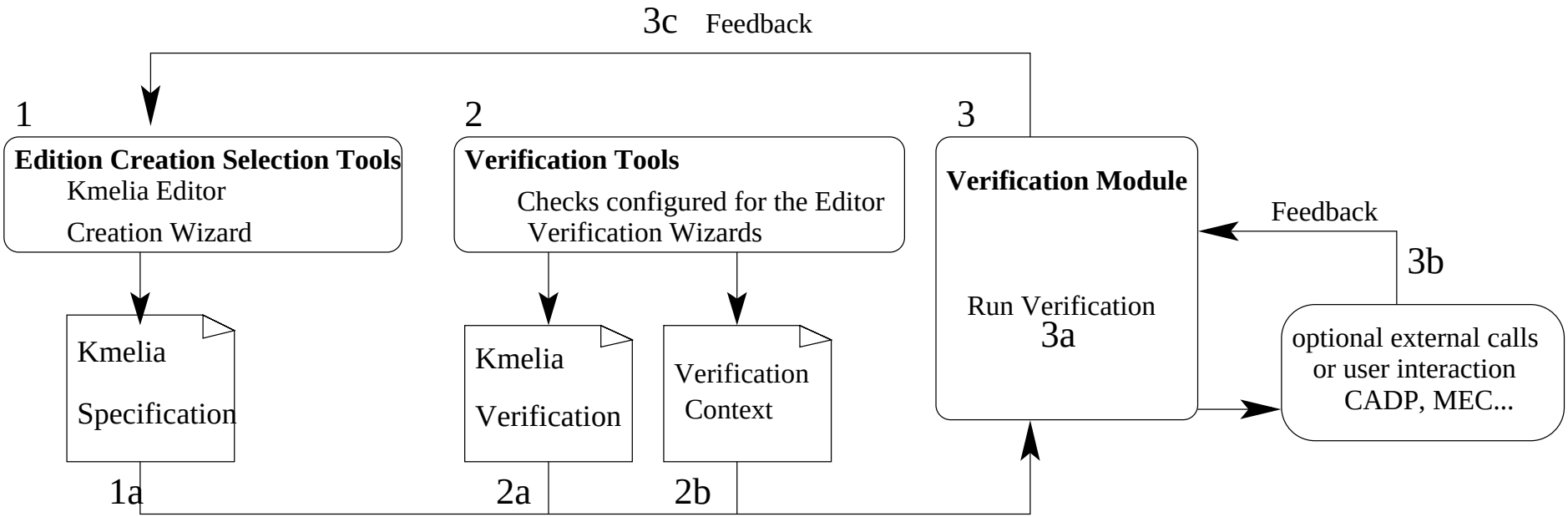
# Property Verification in COSTO

Principles

- Formal verification of properties
- Reduced User Interaction (automation)
- Customisable verification Process
- Open framework: interconnection with appropriate tools
  - Specific property analysis modules: COSTO Modules
  - General property analysis: connection with existing tools (Model-checking or theorem-proving)

Example: Behavioural Compatibility

# The COSTO Toolbox / Verification Scenario




Using Eclipse plugins to verify Kmelia components

# Behavioural Compatibility Verification in COSTO

Scope: correctness of components and assemblies compositions

- availability of components and services
- compatibility of linked interfaces
- service compatibility, four levels of control:
  - service signatures,
  - enhanced service interfaces (sub-services)),
  - contracts: pre/post conditions
  - behaviours: correct interactions (caller service / called service).
- diagnosis on mismatch

Illustration with MEC

- Choice of MEC: existing powerful tools, LTS, simple
- Systematic translation into input formalisms and feedback

# Behavioural Compatibility with MEC: coverage

Correct interaction between linked services.

- Assumption: compatibility of signatures, enhanced interfaces and assertions.
- Interaction Context: a triplet based on an assembly link

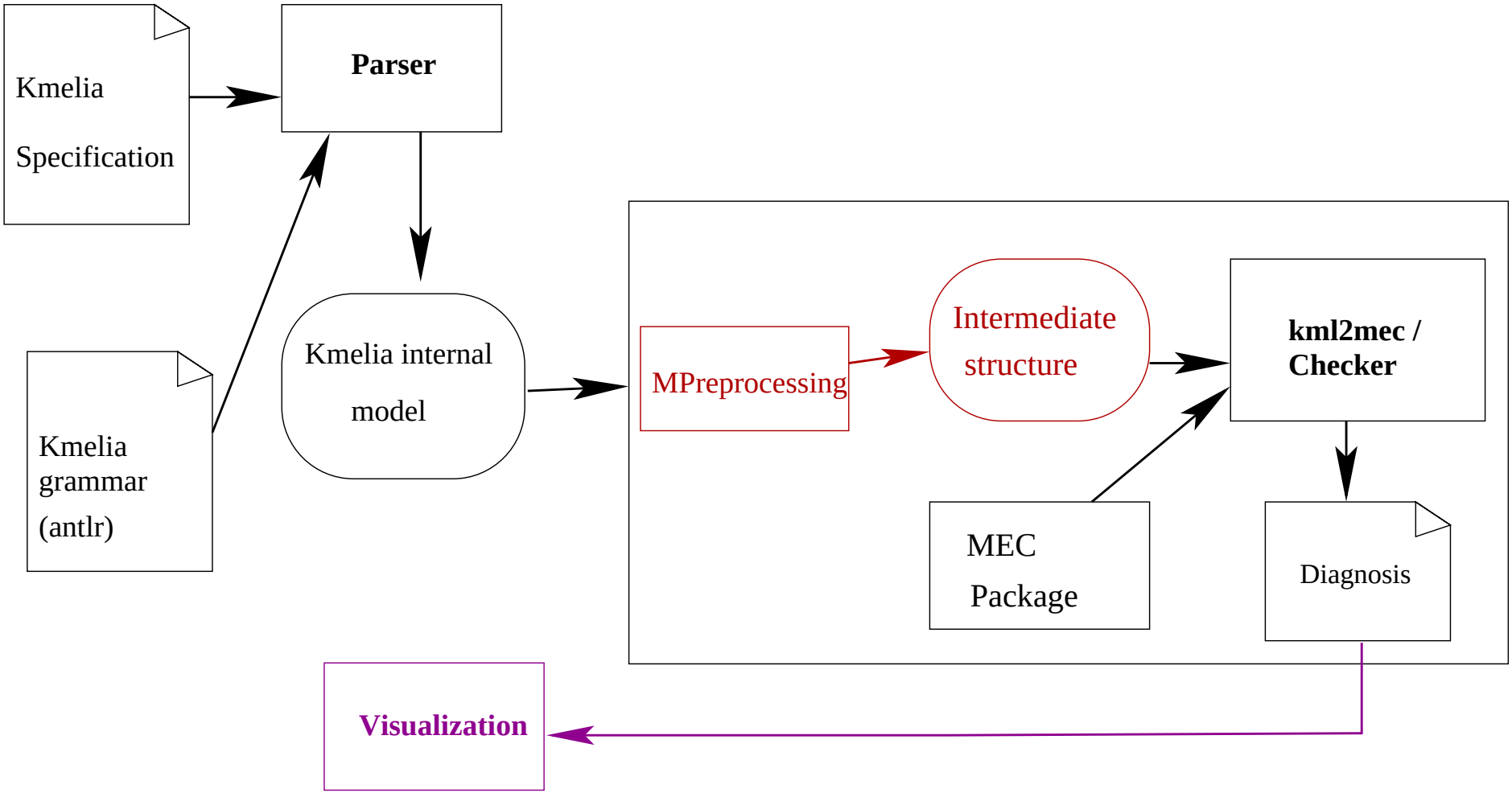
  called service / required service / caller service

  Interactions on messages and services
- Local verification $\rightsquigarrow$ global verification
  One MEC translation per context

Kmelia Behavioural compatibility $\Longrightarrow$ Deadlock freeness with MEC

1. Translation rules
2. Execution with MEC 4
3. Feedback with COSTO

# Behavioural Analysis with COSTO/MEC: Overview




Verification process

# Behavioural Analysis with COSTO/MEC

## Translation rules

- Generate the individual transition systems (from the context)
  - caller service ⇒ MEC TS
  - called service ⇒ MEC TS
  - subservices ⇒ service expansion or MEC TS
- Translate the actions
  - elementary actions ⇒ convert into MEC labels
  - communication actions
    - outside the context ⇒ elementary actions
    - inside the context ⇒ MEC synchronisation
- Generate the MEC synchronised transition system description
  - define the STS
  - compute the synchronisation vector
- Generate the MEC property definitions and computations

Execution with MEC 4

Feedback with COSTO

# Behavioural Analysis with COSTO/MEC

Translation rules

Execution with MEC 4

- mec input file
- run (interactive, linux binary, ascii interface)
- customizable mec output file

Feedback with COSTO

# Behavioural Analysis with COSTO/MEC

Translation rules

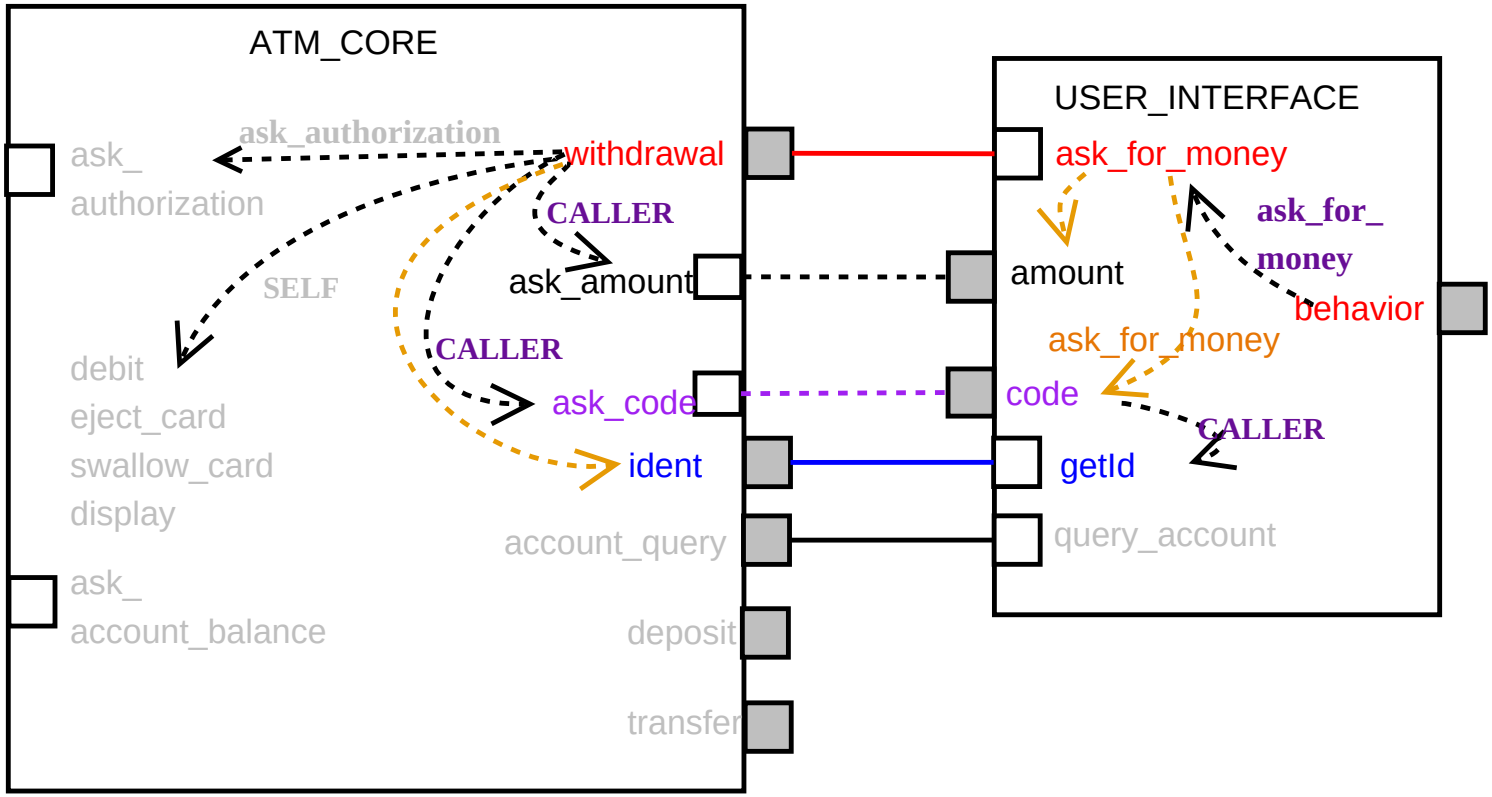
Execution with MEC 4

Feedback with COSTO

- MEC output compiler (ANTLR)
- Dot generator
- Syntax highlighting

# Experimental Results: ATM Case Study

## Deadlock detection



# Experimental Results: MEC input file

```
synchronization_system Verif_ATM_CORE_withdrawal <width = 3 ;
   list = ( ATM_CORE_withdrawal, USER_INTERFACE_behavior, USER_INTERFACE_amount )>;

( eject_card__ . e . e );
...

( e . call_query_account_query_account . e );
( e . rcv_result_query_account_query_account . e );
...

( start__CALLER_withdrawal . call_ask_for_money_ask_for_money . e );
( emit_result__CALLER_withdrawal . rcv_result_ask_for_money_ask_for_money . e );
( emit__CALLER_rdv . rcv_ask_for_money_rdv . e );
( call__CALLER_ask_code . start_ask_for_money_code . e );
( rcv_result__CALLER_ask_code . emit_result_ask_for_money_code . e );
( start__CALLER_ident . call_ask_for_money_getId . e );
( emit_result__CALLER_ident . rcv_result_ask_for_money_getId . e )
...

\* ------- Mec expressions ------- *\
sync(Verif_ATM_CORE_withdrawal, Verif_ATM_CORE_withdrawal);
dts(Verif_ATM_CORE_withdrawal);
finalStates := final[1] /\ final[2];
deadf := (* - src(*)) - finalStates;
```

# Experimental Results: MEC input file

```
synchronization_system Verif_ATM_CORE_withdrawal <width = 3 ;
   list = ( ATM_CORE_withdrawal, USER_INTERFACE_behavior, USER_INTERFACE_amount )>;

( eject_card__ . e . e );
...

( e . call_query_account_query_account . e );
( e . rcv_result_query_account_query_account . e );
...

( start__CALLER_withdrawal . call_ask_for_money_ask_for_money . e );
( emit_result__CALLER_withdrawal . rcv_result_ask_for_money_ask_for_money . e );
( emit__CALLER_rdv . rcv_ask_for_money_rdv . e );
( call__CALLER_ask_code . start_ask_for_money_code . e );
( rcv_result__CALLER_ask_code . emit_result_ask_for_money_code . e );
( start__CALLER_ident . call_ask_for_money_getId . e );
( emit_result__CALLER_ident . rcv_result_ask_for_money_getId . e )
...

\* ------- Mec expressions ------- *\
sync(Verif_ATM_CORE_withdrawal, Verif_ATM_CORE_withdrawal);
dts(Verif_ATM_CORE_withdrawal);
finalStates := final[1] /\ final[2];
deadf := (* - src(*)) - finalStates;
```

# Experimental Results: MEC input file

```
synchronization_system Verif_ATM_CORE_withdrawal <width = 3 ;
   list = ( ATM_CORE_withdrawal, USER_INTERFACE_behavior, USER_INTERFACE_amount )>;

( eject_card__ . e . e );
...

( e . call_query_account_query_account . e );
( e . rcv_result_query_account_query_account . e );
...

( start__CALLER_withdrawal . call_ask_for_money_ask_for_money . e );
( emit_result__CALLER_withdrawal . rcv_result_ask_for_money_ask_for_money . e );
( emit__CALLER_rdv . rcv_ask_for_money_rdv . e );
( call__CALLER_ask_code . start_ask_for_money_code . e );
( rcv_result__CALLER_ask_code . emit_result_ask_for_money_code . e );
( start__CALLER_ident . call_ask_for_money_getId . e );
( emit_result__CALLER_ident . rcv_result_ask_for_money_getId . e )
...

\* ------- Mec expressions ------- *\
sync(Verif_ATM_CORE_withdrawal, Verif_ATM_CORE_withdrawal);
dts(Verif_ATM_CORE_withdrawal);
finalStates := final[1] /\ final[2];
deadf := (* - src(*)) - finalStates;
```

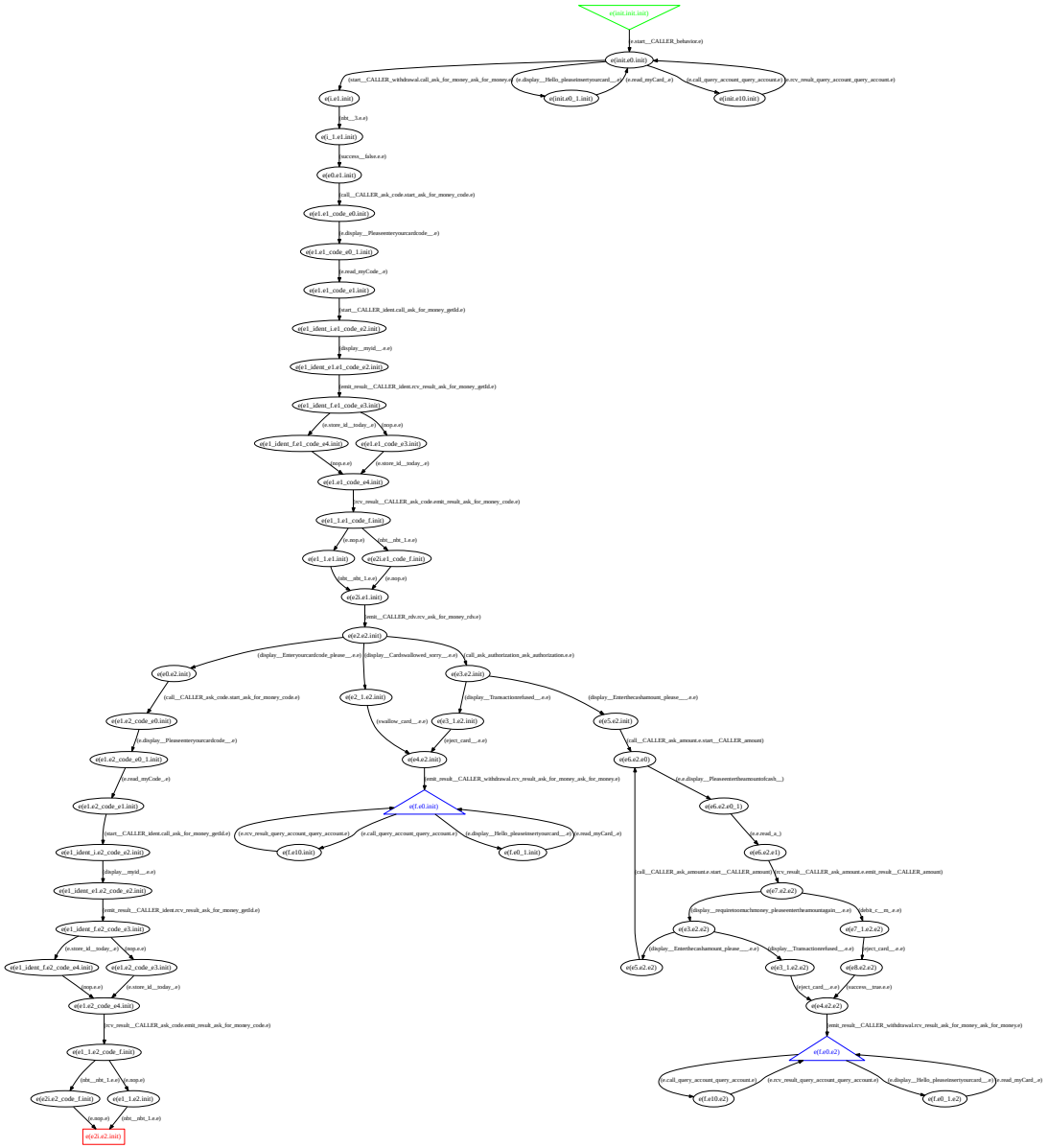
# Experimental Results: MEC input file

```
synchronization_system Verif_ATM_CORE_withdrawal <width = 3 ;
   list = ( ATM_CORE_withdrawal, USER_INTERFACE_behavior, USER_INTERFACE_amount )>;

( eject_card__ . e . e );
...

( e . call_query_account_query_account . e );
( e . rcv_result_query_account_query_account . e );
...

( start__CALLER_withdrawal . call_ask_for_money_ask_for_money . e );
( emit_result__CALLER_withdrawal . rcv_result_ask_for_money_ask_for_money . e );
( emit__CALLER_rdv . rcv_ask_for_money_rdv . e );
( call__CALLER_ask_code . start_ask_for_money_code . e );
( rcv_result__CALLER_ask_code . emit_result_ask_for_money_code . e );
( start__CALLER_ident . call_ask_for_money_getId . e );
( emit_result__CALLER_ident . rcv_result_ask_for_money_getId . e )
...

\* ------- Mec expressions ------- *\
sync(Verif_ATM_CORE_withdrawal, Verif_ATM_CORE_withdrawal);
dts(Verif_ATM_CORE_withdrawal);
finalStates := final[1] /\ final[2];
deadf := (* - src(*)) - finalStates;
```

# Experimental Results: MEC input file

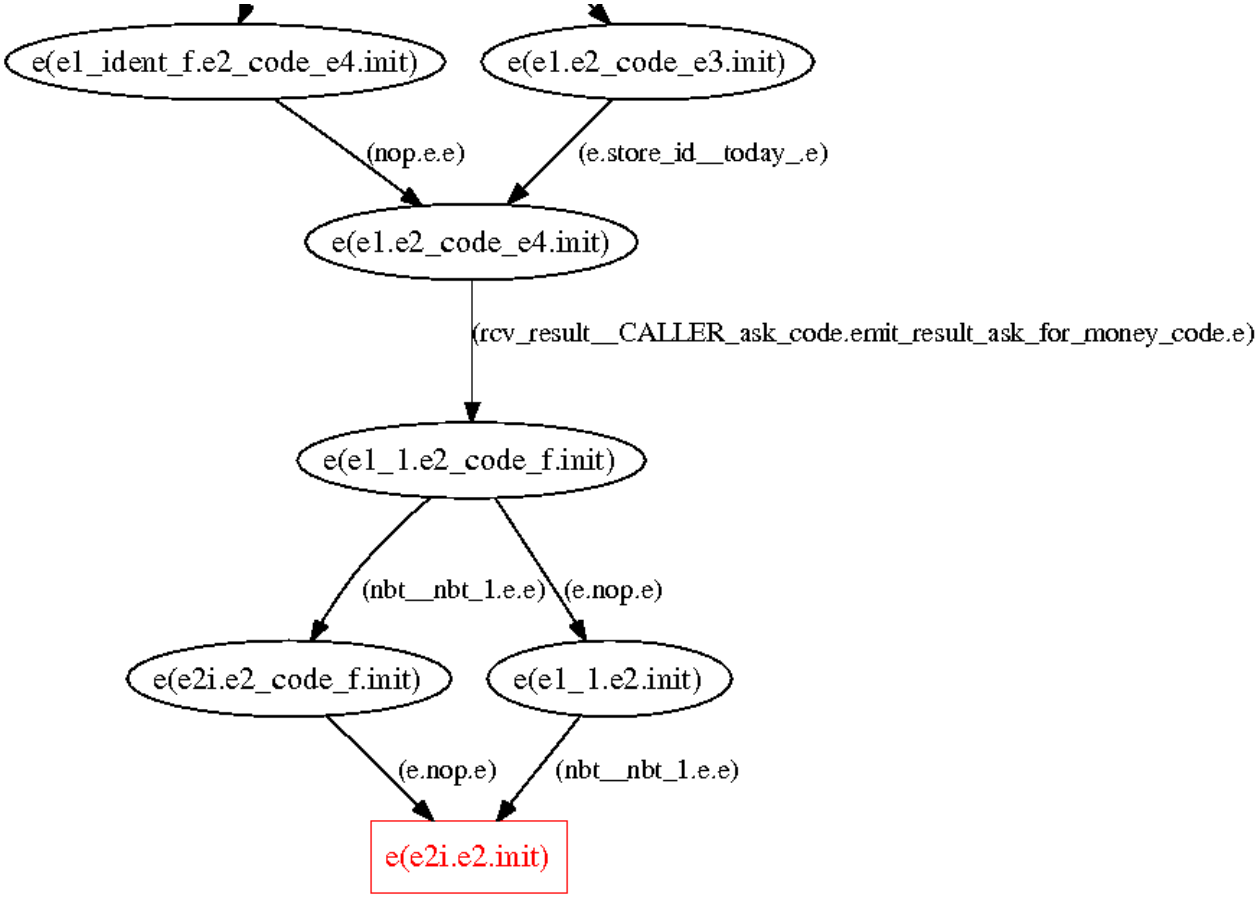
```
synchronization_system Verif_ATM_CORE_withdrawal <width = 3 ;
   list = ( ATM_CORE_withdrawal, USER_INTERFACE_behavior, USER_INTERFACE_amount )>;

( eject_card__ . e . e );
...

( e . call_query_account_query_account . e );
( e . rcv_result_query_account_query_account . e );
...

( start__CALLER_withdrawal . call_ask_for_money_ask_for_money . e );
( emit_result__CALLER_withdrawal . rcv_result_ask_for_money_ask_for_money . e );
( emit__CALLER_rdv . rcv_ask_for_money_rdv . e );
( call__CALLER_ask_code . start_ask_for_money_code . e );
( rcv_result__CALLER_ask_code . emit_result_ask_for_money_code . e );
( start__CALLER_ident . call_ask_for_money_getId . e );
( emit_result__CALLER_ident . rcv_result_ask_for_money_getId . e )
...

\* ------- Mec expressions ------- *\
sync(Verif_ATM_CORE_withdrawal, Verif_ATM_CORE_withdrawal);
dts(Verif_ATM_CORE_withdrawal);
finalStates := final[1] /\ final[2];
deadf := (* - src(*)) - finalStates;
```

# Experimental Results: Feedback



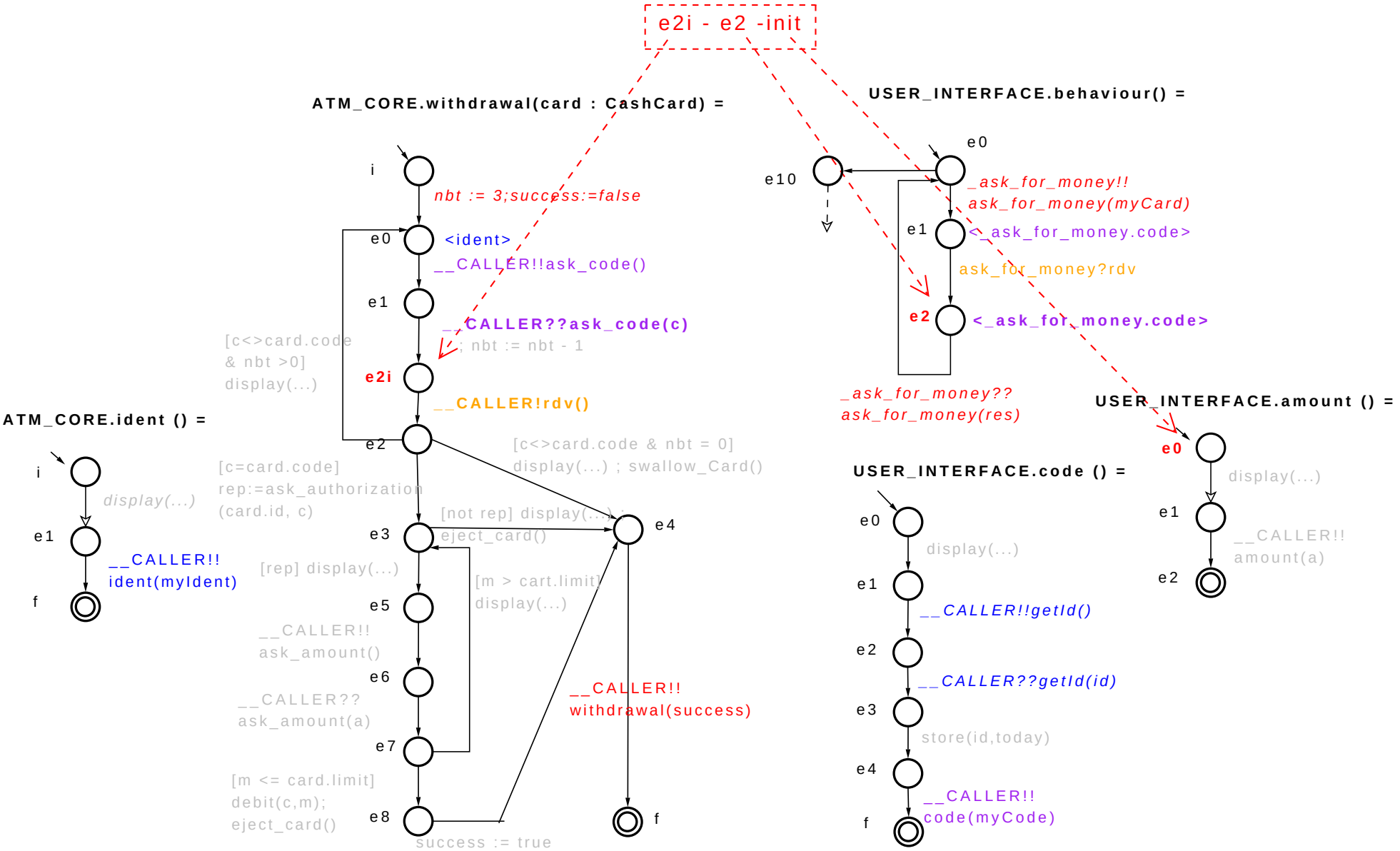
# Experimental Results: Feedback

## Zoom on the deadlock




*Occur only when a second card code trial proceeds.*

# Experimental Results: Interpretation



# Experimental Results: Synthesis

- Non trivial deadlock found
- Limited interaction via the plugin (MEC is unix only)
- Visual feedback for the deadlock property
- User interpretation and correction
- Error detection is not correctness (guards, parameters)

# Outline of the Talk

# Summary and Perspectives

## Summary

- Kmelia abstract component model with complex services
- Illustration with an ATM system
- Verification of service behavioural compatibility with MEC
- COSTO toolbox

## Ongoing Work

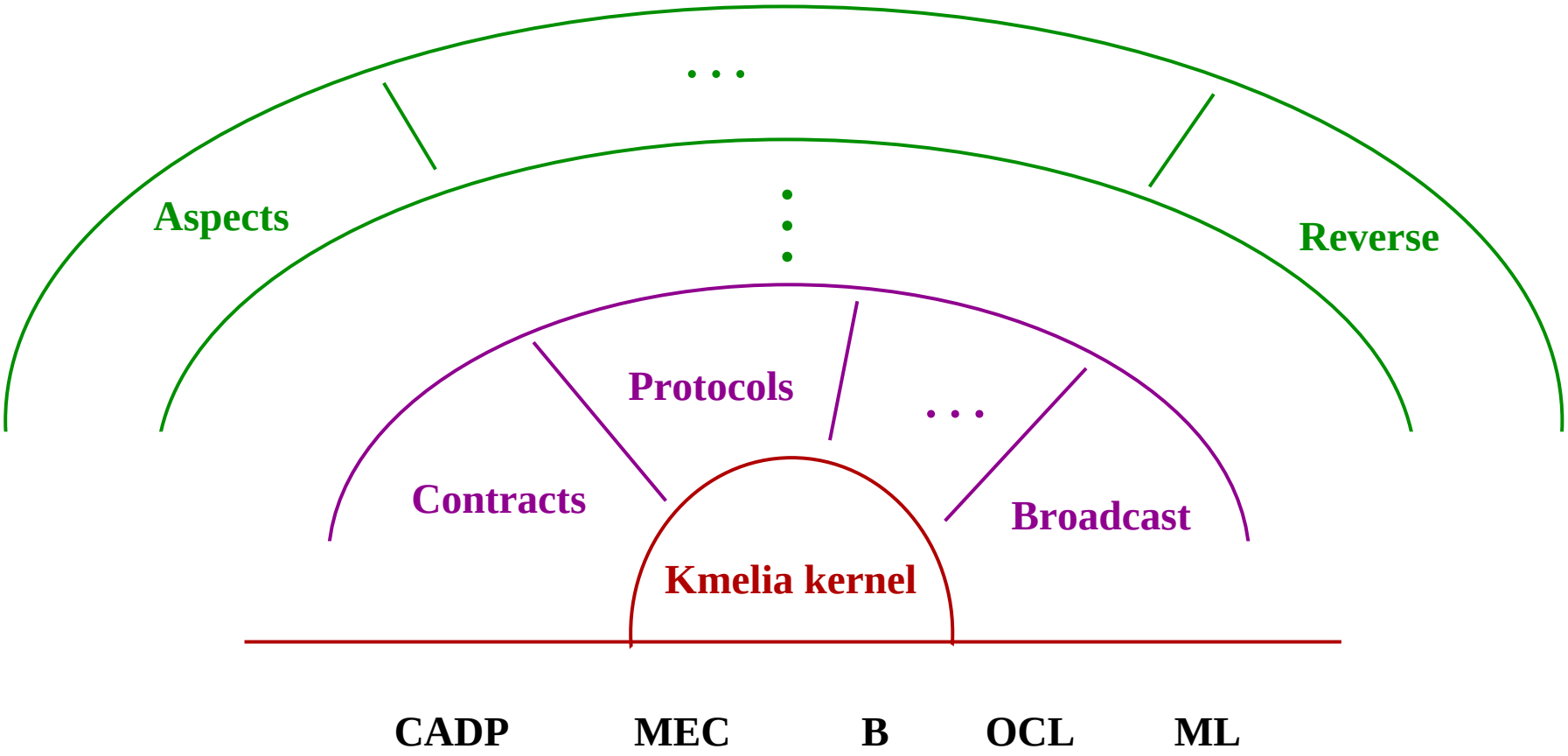
- Extending the Kmelia data and assertion language
- Extending Kmelia to multiple component instances
- Extending the COSTO Toolbox to deal with consistency using theorem proving
- Connection with other tools (Fractal, SOFA, etc)

# End

Thanks for your Attention !
Merci de votre Attention!

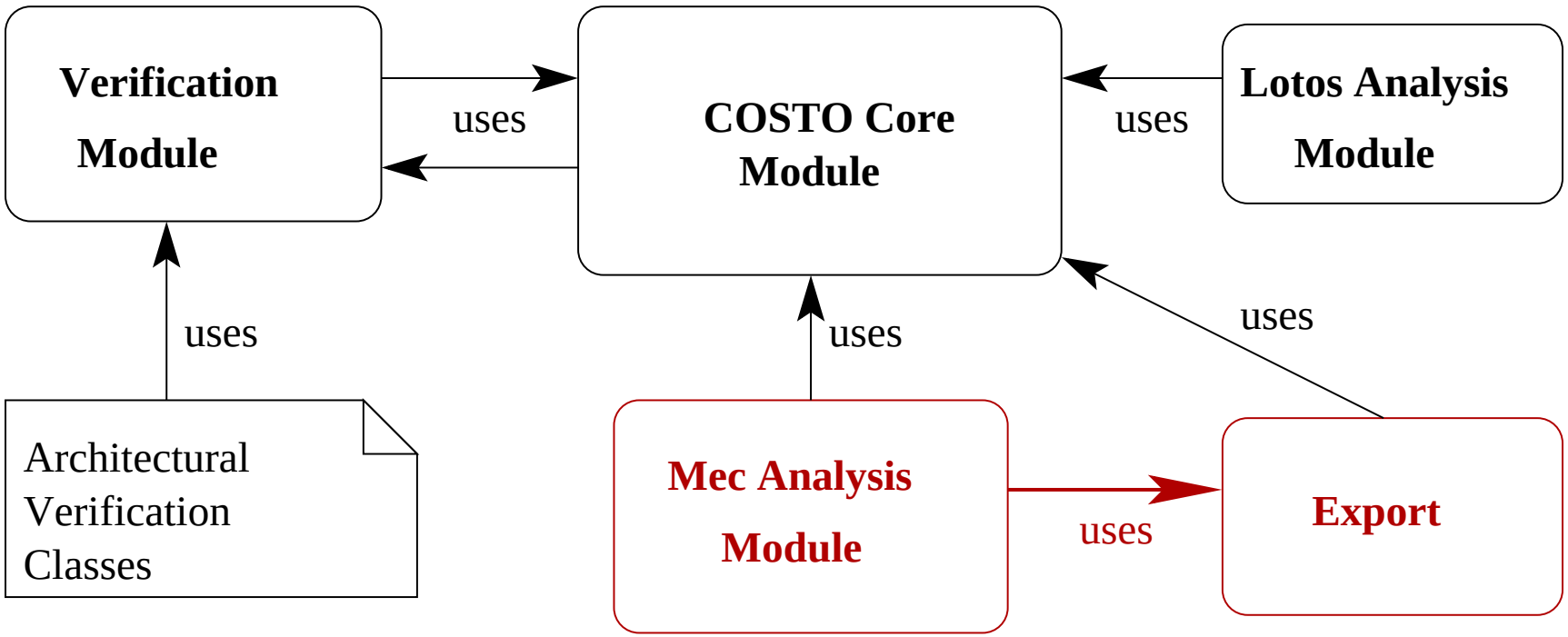
http://www.lina.sciences.univ-nantes.fr/coloss/

# The Kmelia Project

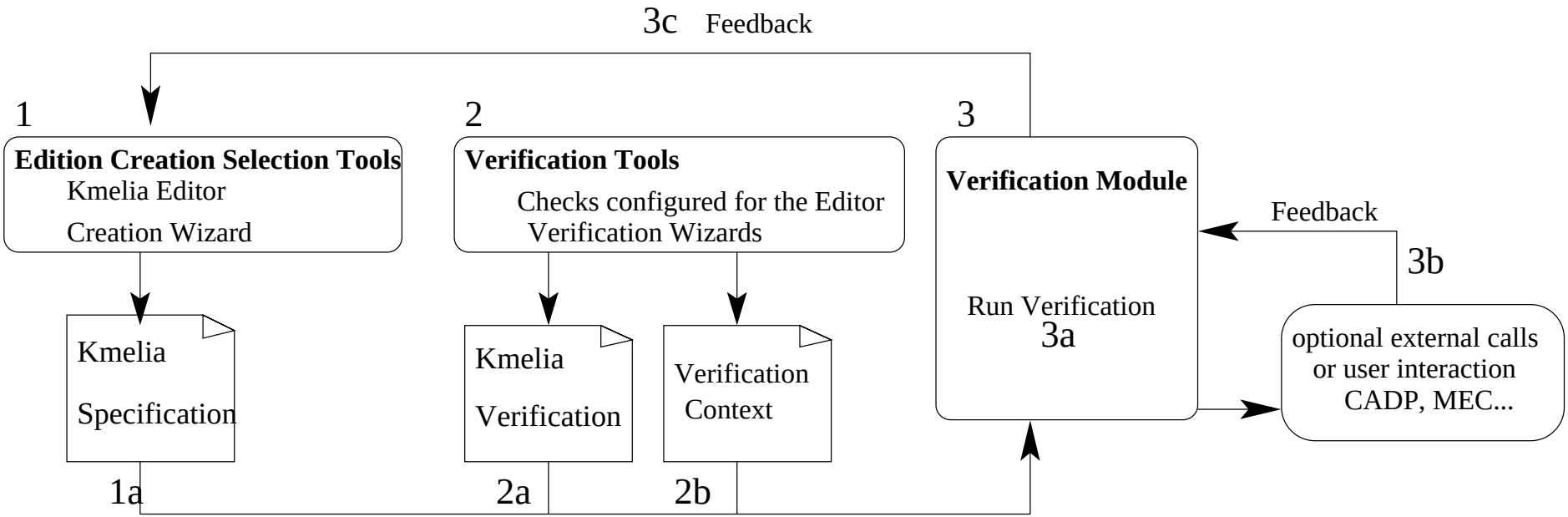



http://www.lina.sciences.univ-nantes.fr/coloss/

# The COSTO Toolbox / Current Architectural Overview



# The COSTO Toolbox / Eclipse plugins




Using Eclipse plugins to verify Kmelia components

# The COSTO Toolbox / Implementation

The COSTO modules are being developed with JAVA.
The modules have been integrated into the Eclipse IDE as plug-ins.
The `costolib.base` plug-in: CORE, Mec Analysis, LOTOS Analysis, Verification, Export

The `costolib.ui` plug-in contains:

- A text editor for Kmelia specifications;
- A tree-based view that outlines a Kmelia specification;
- Wizards for creating Kmelia components and assemblies;
- Menu actions for exporting a Kmelia specification to the various supported formats (such as LaTeX, dot, ps);
- Wizards for creating verification contexts and starting the verifications with MEC or LOTOS.