Checking Component Composability

Christian ATTIOGBÉ, P. ANDRÉ, G. ARDOUREL
University of Nantes, CNRS
LINA FRE CNRS 2729, COLOSS Team

Outline of the Talk

1. Introduction
2. The Kmelia Component Model
3. Service and Component Composability
4. Behavioural Verification of Service Composability
5. The COSTO Toolbox
6. Conclusion and Perspectives

1. Introduction

The motivation for this work: sound basis

- to develop correct software within CBSE (components, composition)
- to propose techniques for property verification.

The goal:

- to provide developers with component models and guidance,
- to build practical toolbox.

The article:

- to check the composability of components in assemblies.

1. Introduction - cont'd.

To Check Component Composability

1. formal descriptions for
   - components: state, interfaces, rules
   - services: static and dynamic features
   - composition: components linked by their services

1. Introduction - cont'd.

To Check Component Composability

1. formal descriptions for
   - components: state, interfaces, rules
   - services: static and dynamic features
   - composition: components linked by their services
2. a formal definition for Composability
   correctness of component assemblies according to the service specifications.
   as a layered property to support progressive check and Interoperability (e.g IDL, BIDL...)

1. Introduction - cont'd.

To Check Component Composability

1. formal descriptions for
   - components: state, interfaces, rules
   - services: static and dynamic features
   - composition: components linked by their services
2. a formal definition for Composability as a layered property to support progressive check and Interoperability (e.g IDL, BIDL...)
3. verification techniques and tools

Outline of the Talk

1. Introduction
2. The Kmelia Component Model
3. Service and Component Composability
4. Behavioural Verification of Service Composability
5. The COSTO Toolbox
6. Conclusion and Perspectives

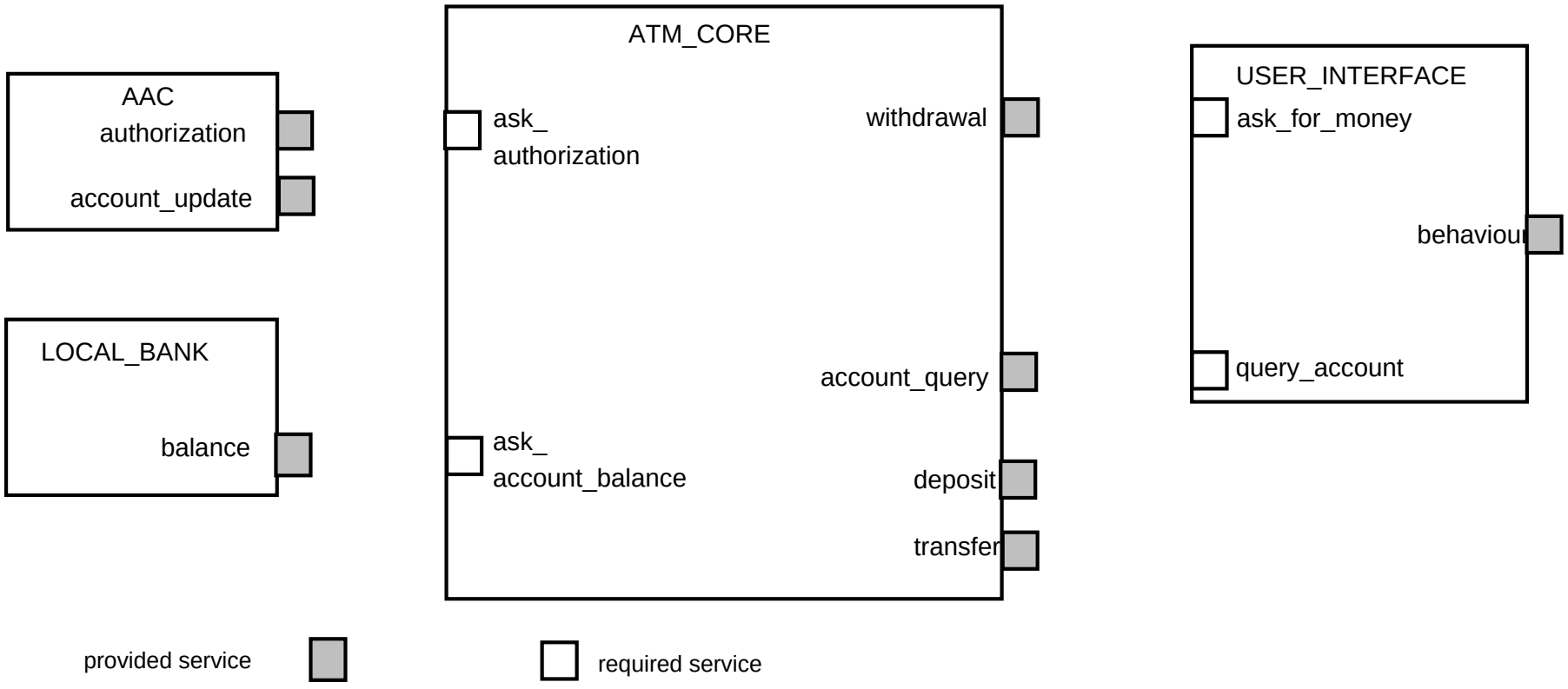
2. *The* Kmelia *Component Model*

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

- A component is a structuring unit that encapsulates a state and services in an interface with usage constraints.
- A component interface: interactions on provided services and required services.
- A service encodes a functionality; it has a behaviour.
- A service interface: subservices and requirements.

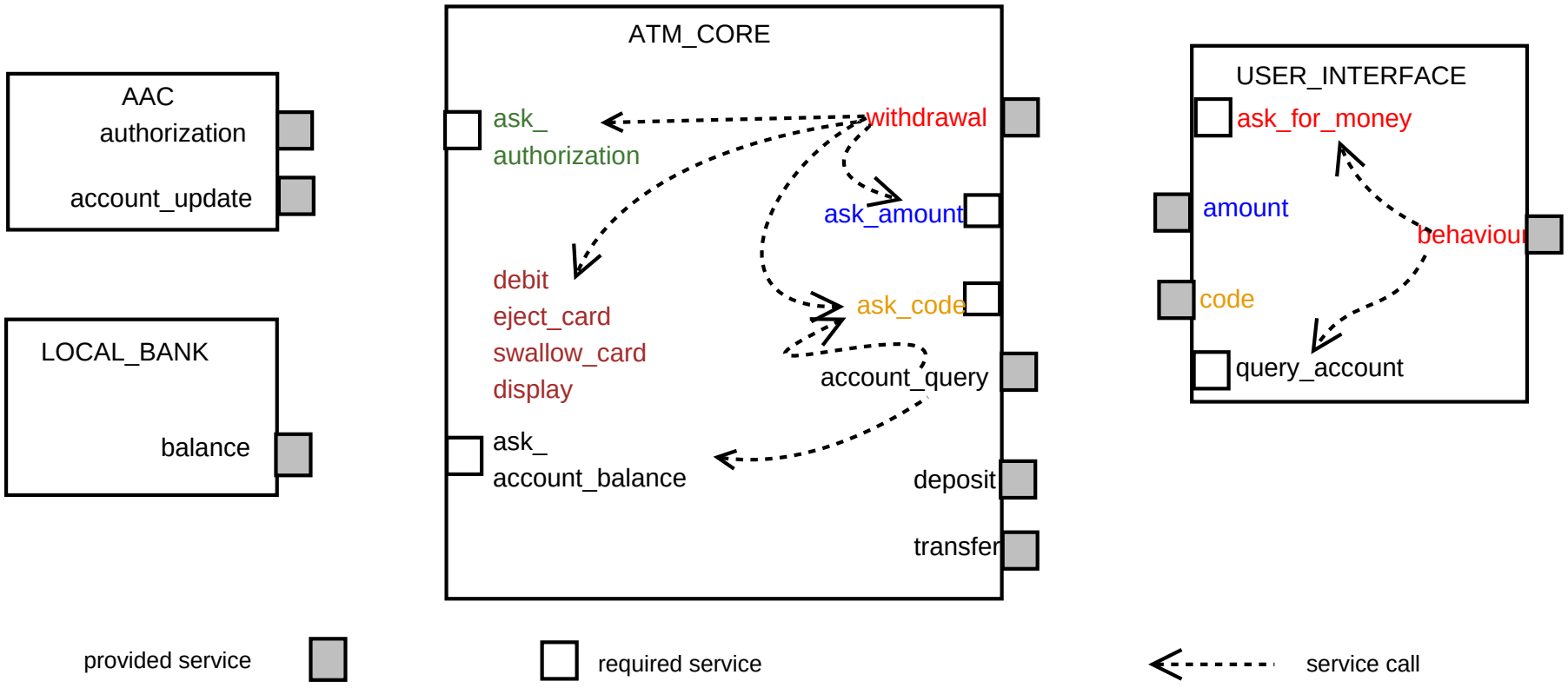
2. *The* Kmelia *Component Model*

A Kmelia assembly for the ATM - *components, interfaces*



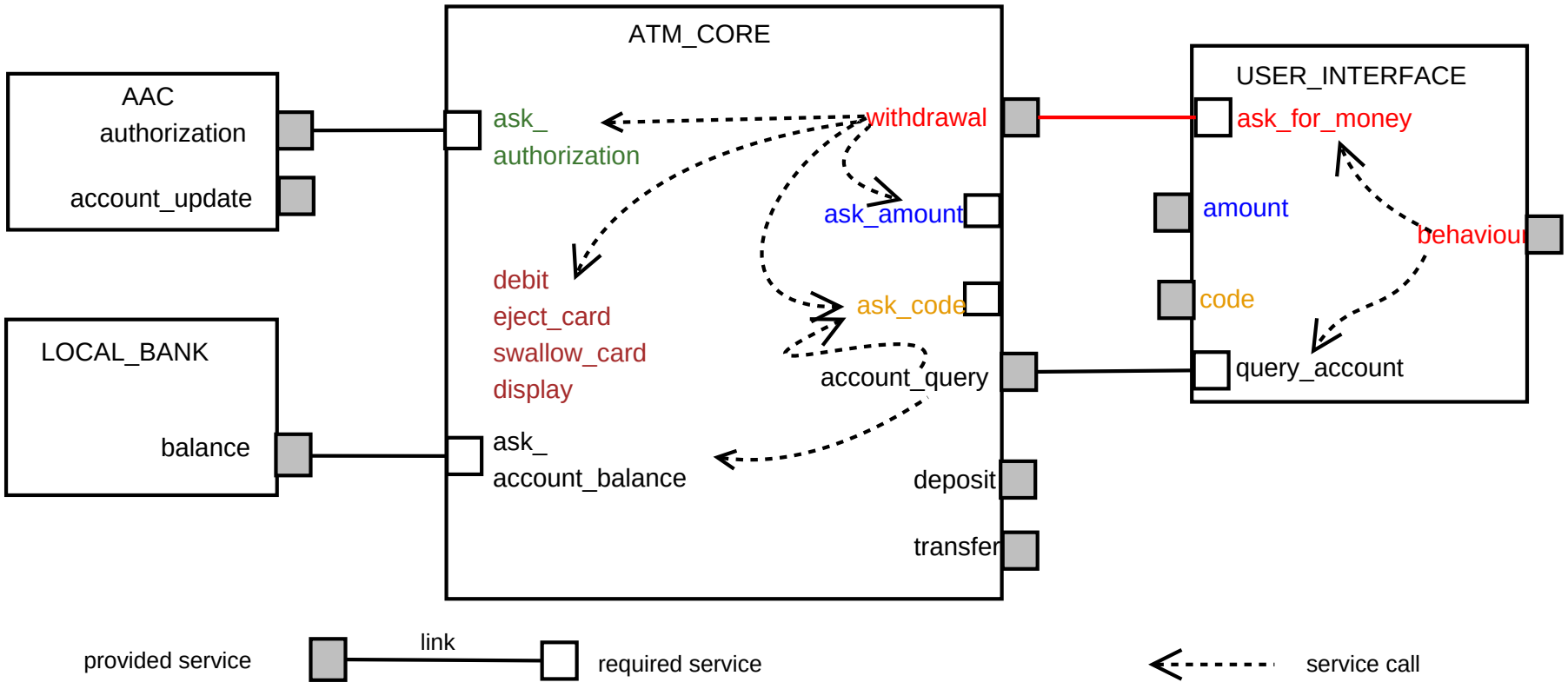
2. *The* Kmelia *Component Model*

A Kmelia assembly for the ATM - *service calls*



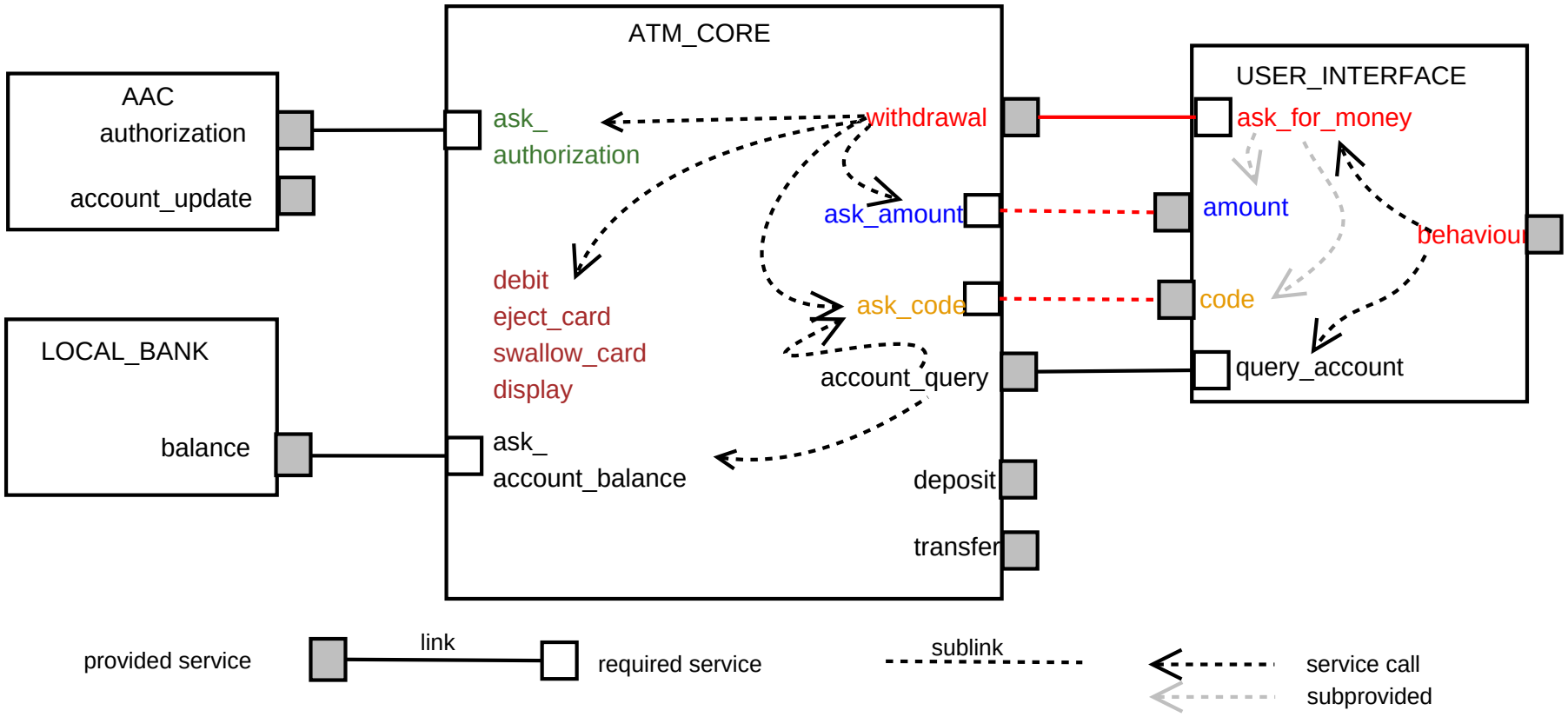
2. *The* Kmelia *Component Model*

A Kmelia assembly for the ATM - *links*



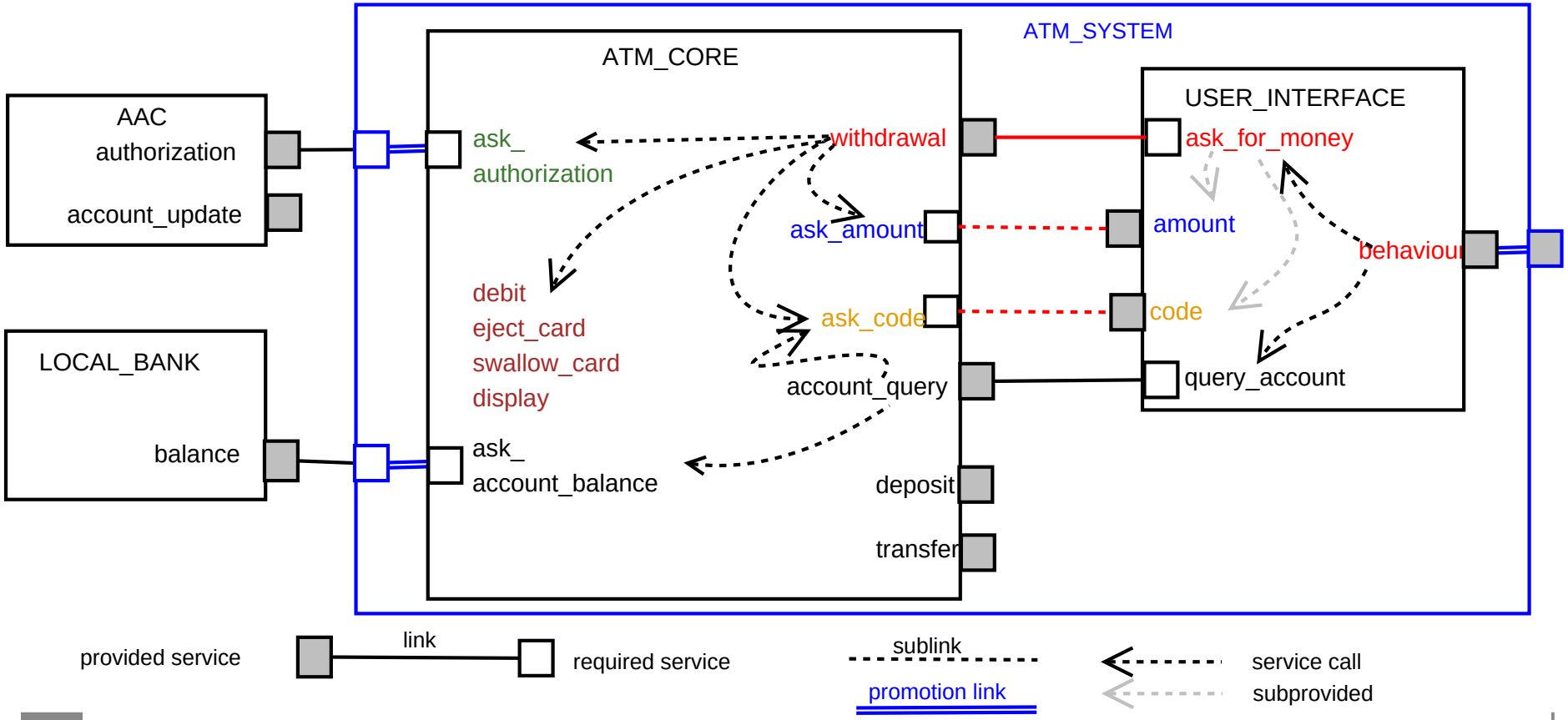
2. *The* Kmelia *Component Model*

A Kmelia assembly for the ATM - *sublinks, subservices*



2. *The* Kmelia *Component Model*

A Kmelia assembly for the ATM - *composition*



2. *The* Kmelia *Component Model*

A composition links components via their services.

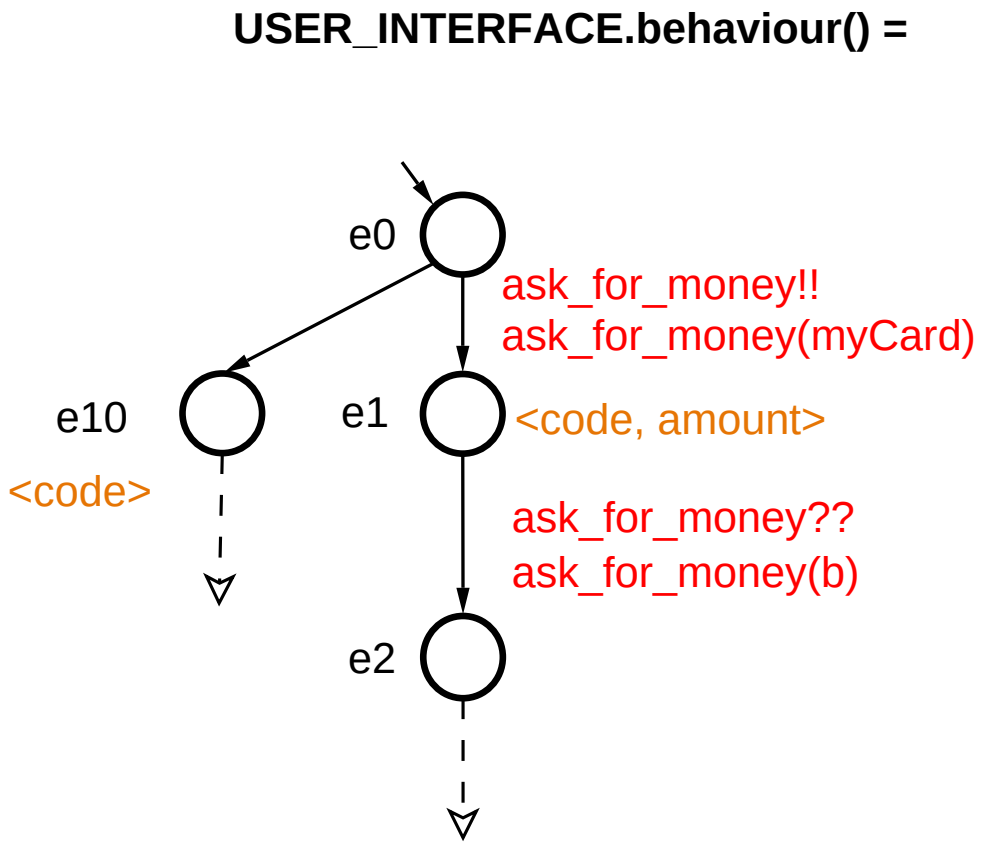
- Horizontal:
  - assembly links structure service interactions
  - assembly sublinks support the structuring of larger services
- Vertical: promotion links denote the structuring of larger components

The service concept is central to Kmelia:

- Support for component connection and interaction
- First class elements and not only messages
- Service composition

2. Service description

- Signature
- Local variables
- Assertions
- Interface: required services, provided services
- Behaviour: extended Labelled Transition System



annotation: possible service calls

2. Service description: eLTS

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

  `label ::= [guard] actions*`

  An action is:
  - An elementary action
  - A communication:
    a *service call/response* or
    a message *communication*.

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

    `Channel ::= SELF | CALLER | RequiredServiceName`
- Extensions (branching states and transitions)

Outline of the Talk

1. Introduction
2. The Kmelia Component Model
3. Service and Component Composability
4. Behavioural Verification of Service Composability
5. The COSTO Toolbox
6. Conclusion and Perspectives

3. Service/Component Composability

The scope is the correctness of components and their compositions:

- availability of components and services,
- compatibility of linked interfaces,
- correct interaction between services,
- diagnosis on mismatching.

Flexibility:

- partial use of components,
- incomplete description of services.

3. Composability (Service level)

A provided service $sp_{C_i} = \langle\langle \sigma_p, P_p, Q_p, V_{sp}, S_{sp}\rangle, \mathcal{B}_{sp}\rangle$ *of a component* C_i *and a required service* $sr_{C_j} = \langle\langle \sigma_r, P_r, Q_r, V_{sr}, S_{sr}\rangle, \mathcal{B}_{sr}\rangle$ *of a component* C_j *are s-composable (noted s-composable*(sp_{C_i}, sr_{C_j})*) when* sr_{C_j} *is required in the behaviour* $\mathcal{B}_s$ *of a service* s *of* C_j *if:*

1. *the interfaces of* sp_{C_i} *and* sr_{C_j} *are compatible; that is,*

3. Composability (Service level)

A provided service $sp_{C_i} = \langle\langle \sigma_p, P_p, Q_p, V_{sp}, S_{sp}\rangle, \mathcal{B}_{sp}\rangle$ of a component C_i and a required service $sr_{C_j} = \langle\langle \sigma_r, P_r, Q_r, V_{sr}, S_{sr}\rangle, \mathcal{B}_{sr}\rangle$ of a component C_j are s-composable (noted s-composable(sp_{C_i}, sr_{C_j})) when sr_{C_j} is required in the behaviour $\mathcal{B}_s$ of a service s of C_j if:

1. *the interfaces of sp_{C_i} and sr_{C_j} are compatible; that is,*
   (a) *their signatures are matching (no type conflict: σ_p and σ_r are identical),*

3. Composability (Service level)

A provided service $sp_{C_i} = \langle\langle \sigma_p, P_p, Q_p, V_{sp}, S_{sp}\rangle, \mathcal{B}_{sp}\rangle$ of a component C_i and a required service $sr_{C_j} = \langle\langle \sigma_r, P_r, Q_r, V_{sr}, S_{sr}\rangle, \mathcal{B}_{sr}\rangle$ of a component C_j are s-composable (noted s-composable(sp_{C_i}, sr_{C_j})) when sr_{C_j} is required in the behaviour $\mathcal{B}_s$ of a service s of C_j if:

1. *the interfaces of sp_{C_i} and sr_{C_j} are compatible; that is,*
   (a) *their signatures are matching (no type conflict: σ_p and σ_r are identical),*
   (b) *the assertions (pre/post) are consistent ($post(sp_{C_i}) \sim post(sr_{C_j})$) and*

3. Composability (Service level)

A provided service $sp_{C_i} = \langle\langle \sigma_p, P_p, Q_p, V_{sp}, S_{sp}\rangle, \mathcal{B}_{sp}\rangle$ *of a component* C_i *and a required service* $sr_{C_j} = \langle\langle \sigma_r, P_r, Q_r, V_{sr}, S_{sr}\rangle, \mathcal{B}_{sr}\rangle$ *of a component* C_j *are s-composable (noted s-composable*(sp_{C_i}, sr_{C_j})*) when* sr_{C_j} *is required in the behaviour* $\mathcal{B}_s$ *of a service* s *of* C_j *if:*

1. *the interfaces of* sp_{C_i} *and* sr_{C_j} *are compatible; that is,*
   (a) *their signatures are matching (no type conflict:* σ_p *and* σ_r *are identical),*
   (b) *the assertions (pre/post) are consistent (*$post(sp_{C_i}) \sim post(sr_{C_j})$*) and*
   (c) *their mutually dependent services* S_{sp}, S_{sr} *are not conflicting: the inner required-provided relationship is preserved: that means they involve a well-formed assembly.*

3. Composability (Service level)

A provided service $sp_{C_i} = \langle\langle \sigma_p, P_p, Q_p, V_{sp}, S_{sp}\rangle, \mathcal{B}_{sp}\rangle$ *of a component* C_i *and a required service* $sr_{C_j} = \langle\langle \sigma_r, P_r, Q_r, V_{sr}, S_{sr}\rangle, \mathcal{B}_{sr}\rangle$ *of a component* C_j *are s-composable (noted s-composable*(sp_{C_i}, sr_{C_j})*) when* sr_{C_j} *is required in the behaviour* $\mathcal{B}_s$ *of a service* s *of* C_j *if:*

1. *the interfaces of* sp_{C_i} *and* sr_{C_j} *are compatible; that is,*
   (a) *their signatures are matching (no type conflict:* σ_p *and* σ_r *are identical),*
   (b) *the assertions (pre/post) are consistent (*$post(sp_{C_i}) \sim post(sr_{C_j})$*) and*
   (c) *their mutually dependent services* S_{sp}, S_{sr} *are not conflicting: the inner required-provided relationship is preserved: that means they involve a well-formed assembly.*
2. *the behaviour* $\mathcal{B}_{sp}$ *of* sp_{C_i} *and* $\mathcal{B}_s$ *are compatible:* $compatible(\mathcal{B}_{sp}, \mathcal{B}_s)$*; that is, their eLTSs are matching; either they evolve independently or they perform complementary communication actions until a termination without a deadlock.*

3. Composability (Service level)

A provided service $sp_{C_i} = \langle\langle \sigma_p, P_p, Q_p, V_{sp}, S_{sp}\rangle, \mathcal{B}_{sp}\rangle$ of a component C_i and a required service $sr_{C_j} = \langle\langle \sigma_r, P_r, Q_r, V_{sr}, S_{sr}\rangle, \mathcal{B}_{sr}\rangle$ of a component C_j are s-composable (noted s-composable(sp_{C_i}, sr_{C_j})) when sr_{C_j} is required in the behaviour $\mathcal{B}_s$ of a service s of C_j if:

1. *the interfaces of sp_{C_i} and sr_{C_j} are compatible; that is,*
   (a) *their signatures are matching (no type conflict: σ_p and σ_r are identical),*
   (b) *the assertions (pre/post) are consistent ($post(sp_{C_i}) \sim post(sr_{C_j})$) and*
   (c) *their mutually dependent services S_{sp}, S_{sr} are not conflicting: the inner required-provided relationship is preserved: that means they involve a well-formed assembly.*
2. *the behaviour $\mathcal{B}_{sp}$ of sp_{C_i} and $\mathcal{B}_s$ are compatible: $compatible(\mathcal{B}_{sp}, \mathcal{B}_s)$; that is, their eLTSs are matching; either they evolve independently or they perform complementary communication actions until a termination without a deadlock.*

4 levels of s-composability

$\rightsquigarrow$ coarse/fi ne grain, interoperability

3. Composability (Component level)

Two components C_i and C_j are c-composable according to a set of service pairs ss, if all the pairs (s_i, s_j) of ss are s-composable:

$$\textit{c-composable}(C_i, C_j, ss) \Leftrightarrow \forall (s_i, s_j) \in ss \bullet \textit{s-composable}(s_i, s_j)$$

3. Composability (Component level)

Two components C_i and C_j are c-composable according to a set of service pairs ss, if all the pairs (s_i, s_j) of ss are s-composable:

c-composable$(C_i, C_j, ss) \Leftrightarrow \forall (s_i, s_j) \in ss \bullet$ *s-composable*(s_i, s_j)

To check an assembly

- carry out pairwise verifications (link oriented),
- check the completeness of used services,
- support composition and promotion.

Outline of the Talk

1. Introduction
2. The Kmelia Component Model
3. Service and Component Composability
4. Behavioural Verification of Service Composability
5. The COSTO Toolbox
6. Conclusion and Perspectives

4. Behavioural Compatibility

A verification context =

- a caller (provided) service
- a required service
- a called (provided) service
- dependent subservices

A (contextual) behavioural compatibility results in the simultaneous state-based examination of two (or more) flattened services

During the flattening, the channels may be renamed according to the verification context.

4. Behavioural Compatibility - cont'd.

The current output transitions are checked $\Rightarrow$

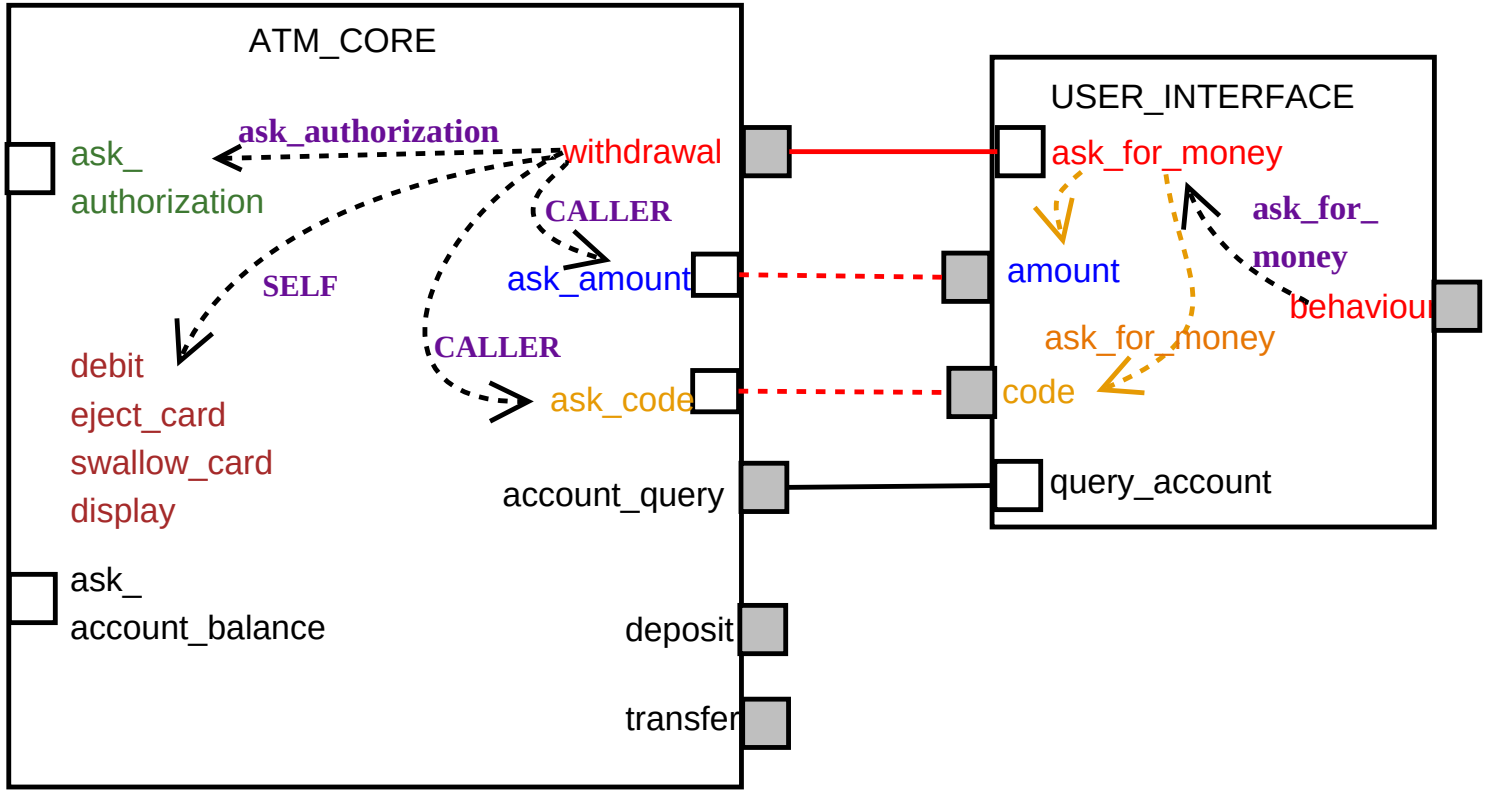
- Independent actions
- Matching actions (with identical channel):

send(!)	receive(?)
call service(!!)	start service(??),
emit service result(!!)	get service result(??)

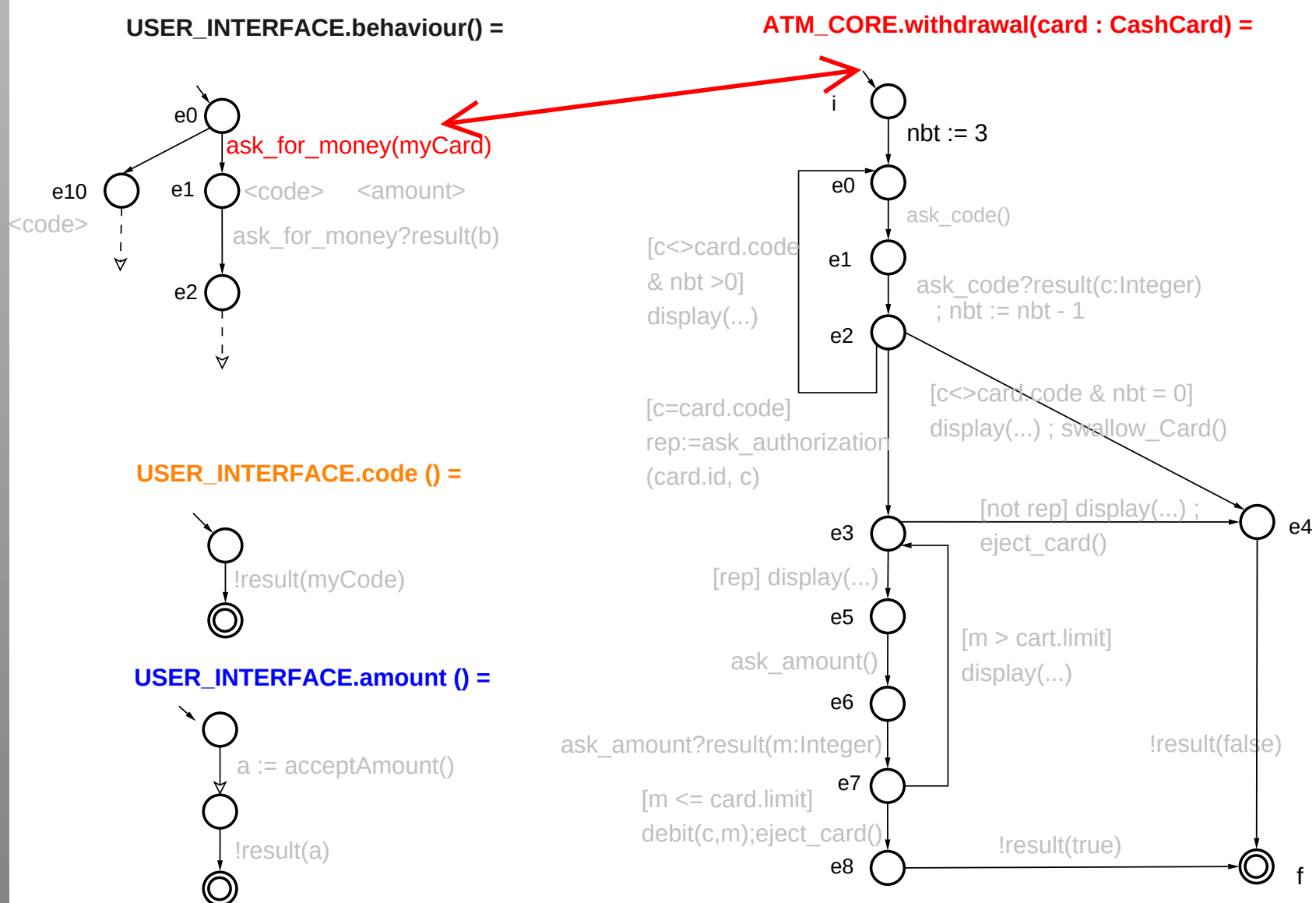
until final states without blocking(deadlock)

4. Behavioural Compatibility (ATM)

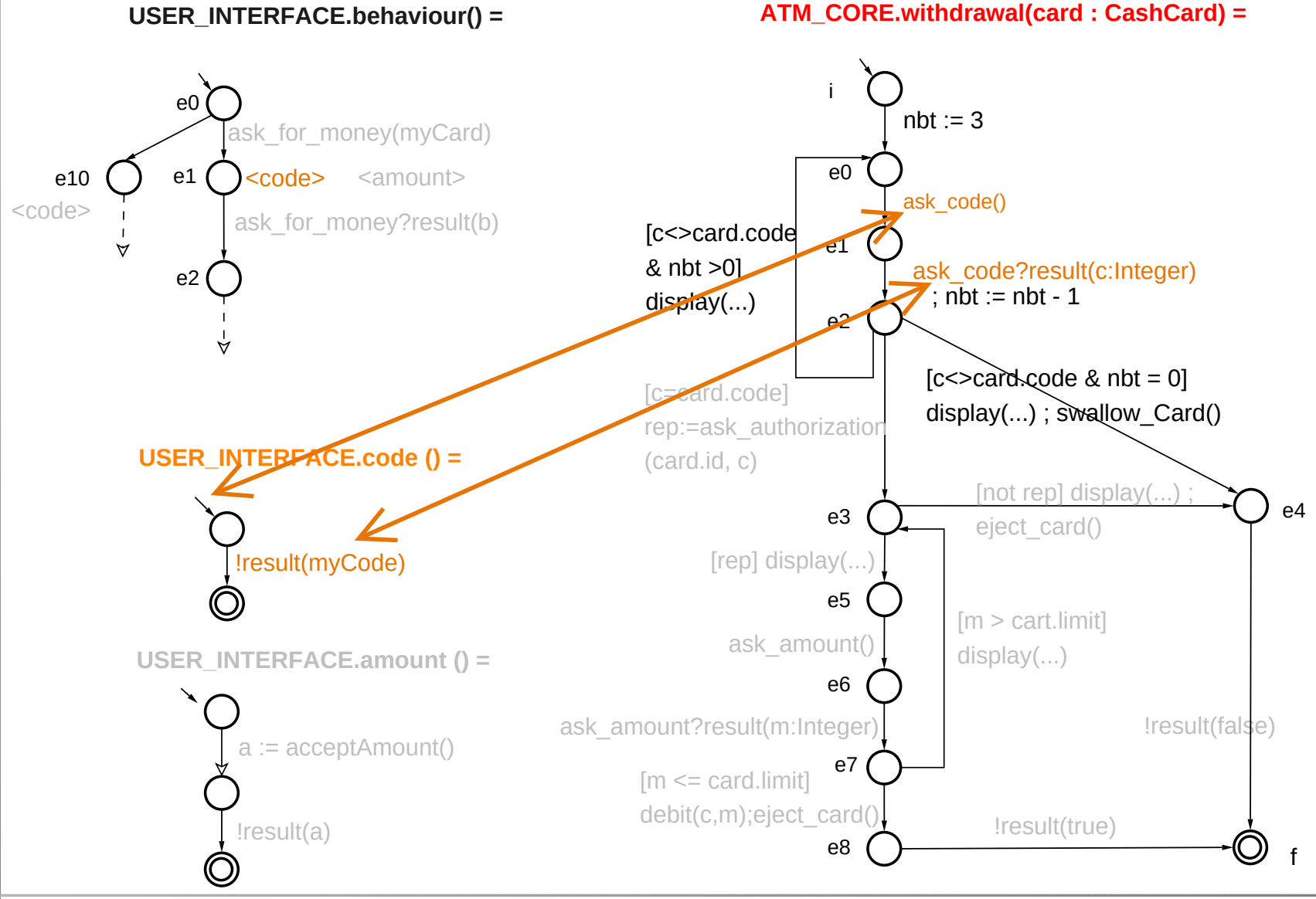
A verification context =
a caller service / a required service / a called service / dependent subservices



4. Behavioural Compatibility (ATM)

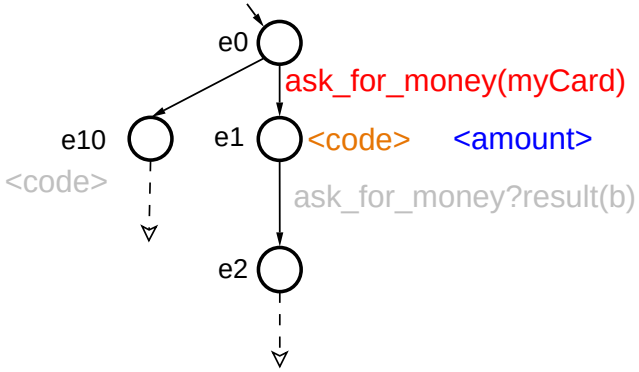


4. Behavioural Compatibility (ATM)

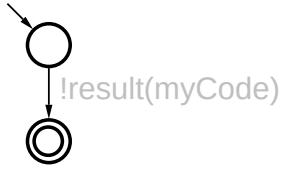


4. Behavioural Compatibility (ATM)

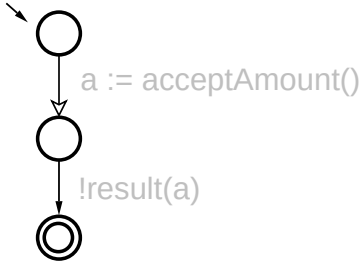
USER_INTERFACE.behaviour() =



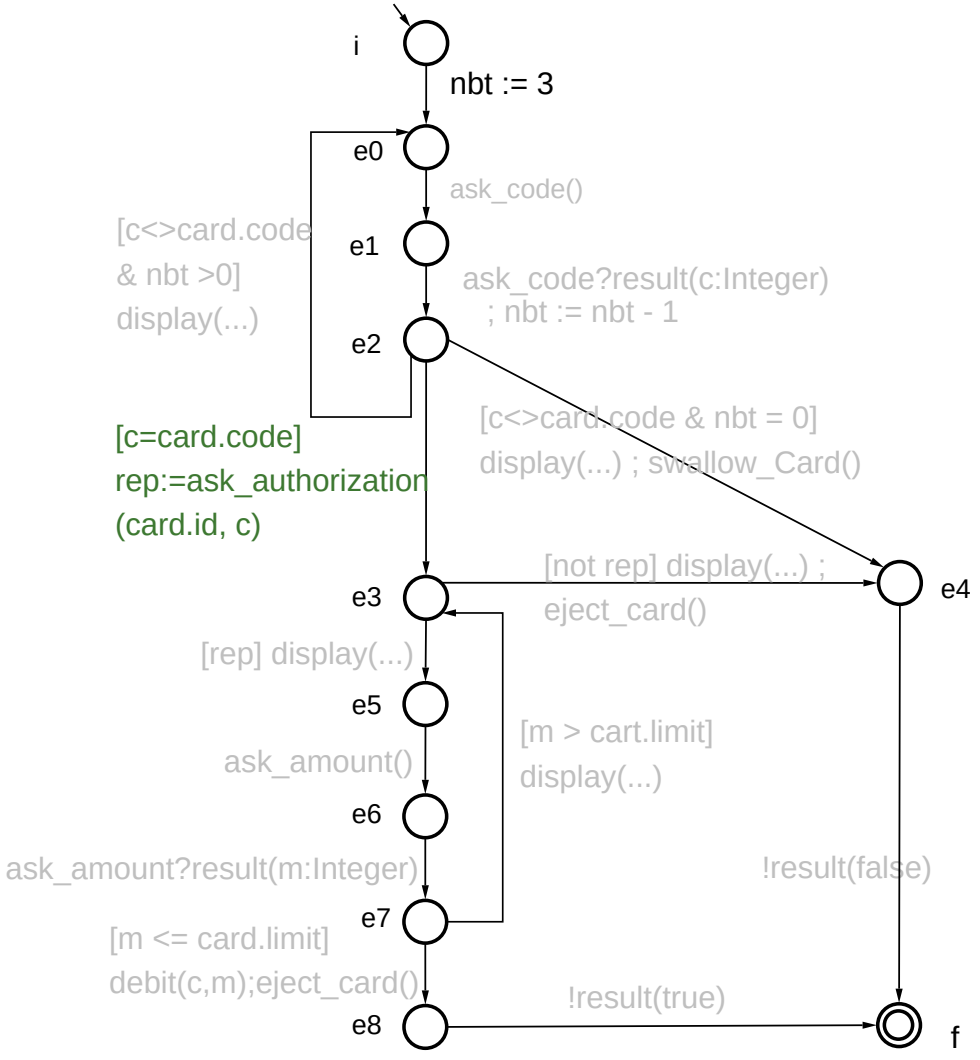
USER_INTERFACE.code () =



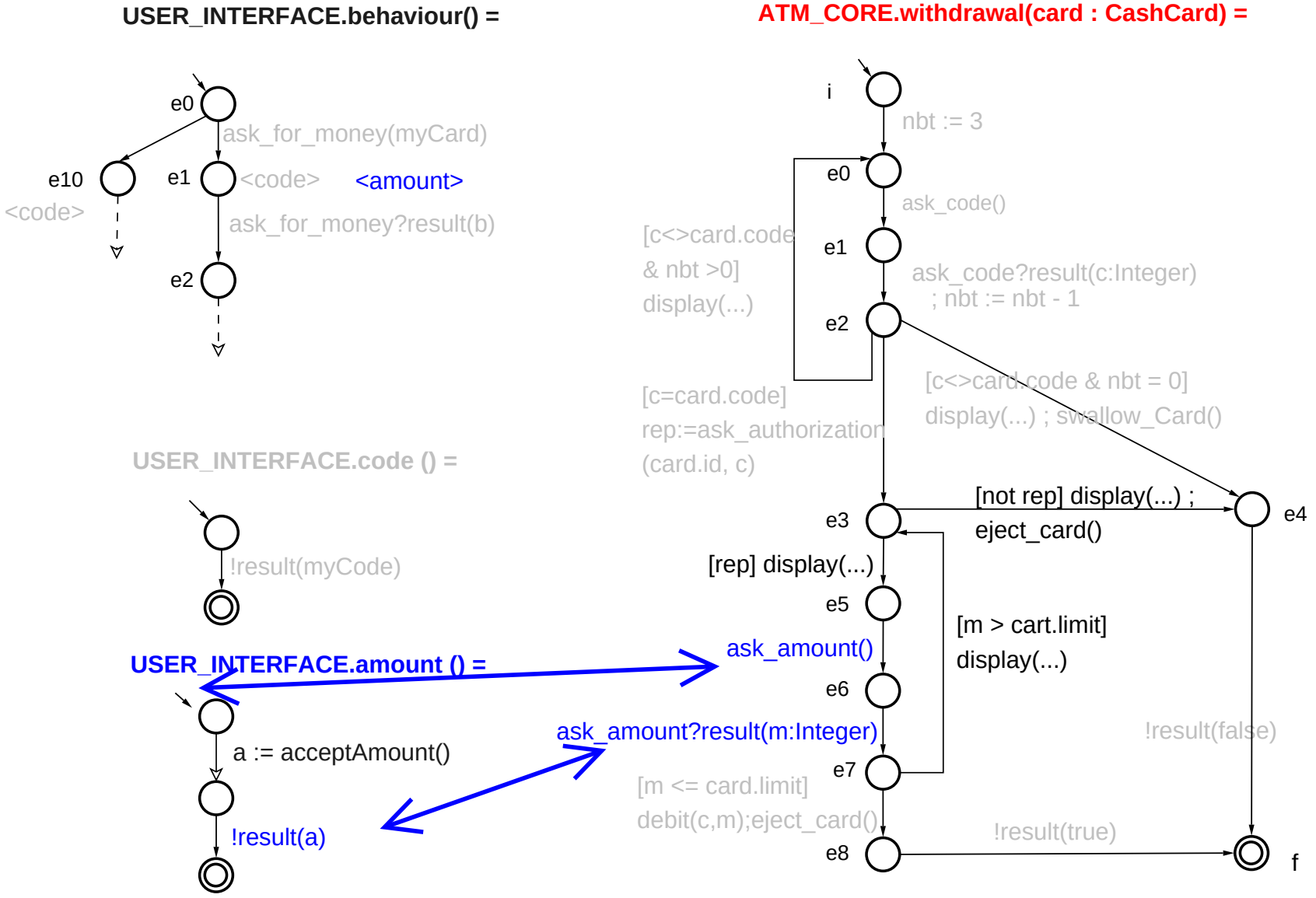
USER_INTERFACE.amount () =



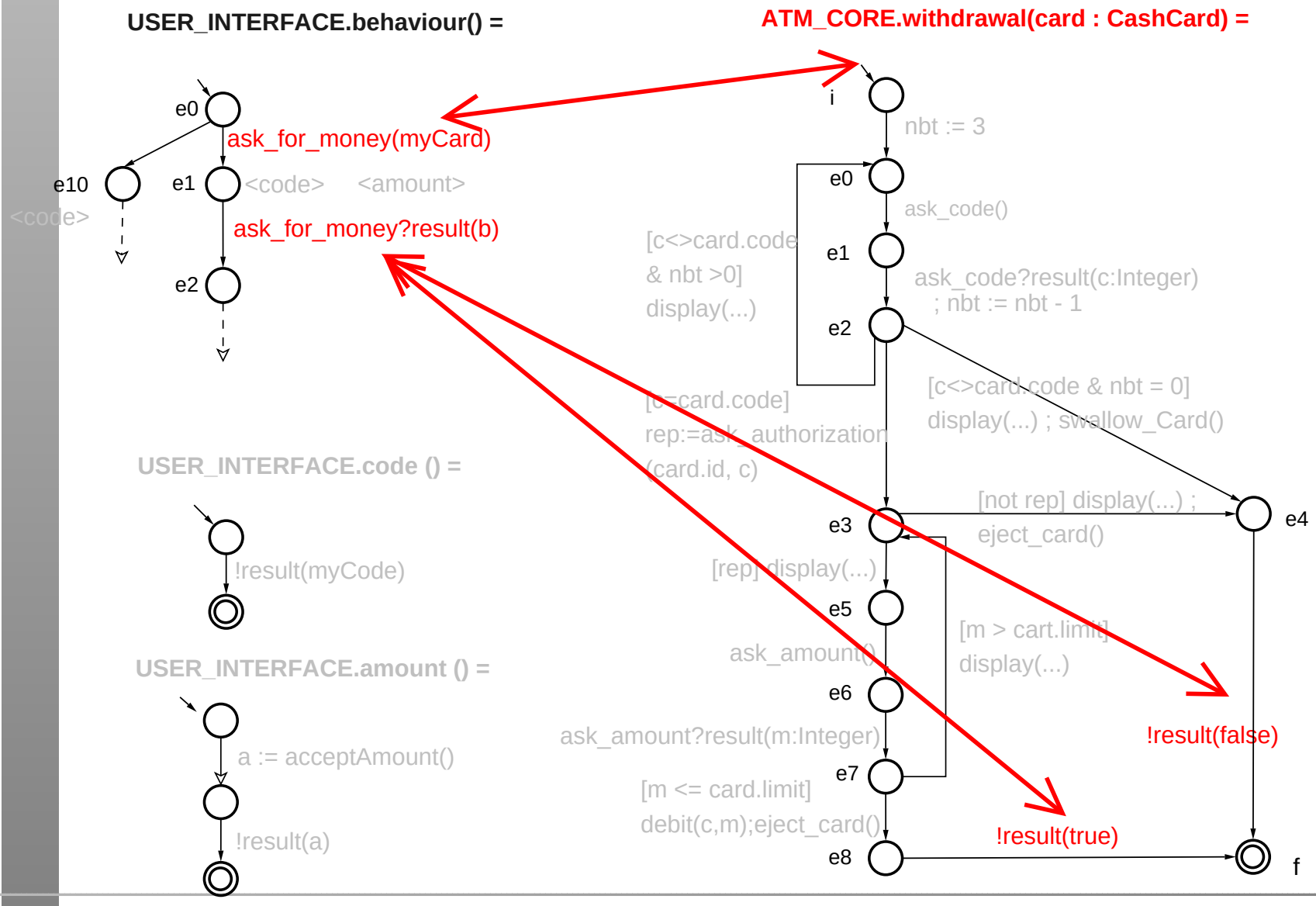
ATM_CORE.withdrawal(card : CashCard) =



4. Behavioural Compatibility (ATM)



4. Behavioural Compatibility (ATM)



Possible failures...

4. Experimentation with Lotos/CADP

The behavioural compatibility coincides with the Lotos |[L]| composition operator

- Translation of Kmelia services (s1, s2) into Lotos processes (s1Process, s2Process).
- Synchronisation of the Lotos processes

```
s1Process [...]
  |[channels]|
s2Process [...]
```

Lotos/CADP tools

diagnosis+feedback

Outline of the Talk

1. Introduction
2. The Kmelia Component Model
3. Service and Component Composability
4. Behavioural Verification of Service Composability
5. The COSTO Toolbox
6. Conclusion and Perspectives

5. The COSTO Toolbox

- Kmelia compiler (Antlr, Java)
- Architectural correctness checker
- Translators into Lotos (and Mec)
- Behavioural compatibility checker
- Graphical visualisation with dot
- GUI under work

7. Conclusion and Perspectives

Summary

- Kmelia component model: Services, components, assemblies
- Composability
- Experimentations with COSTO + existing tools.

Perspectives

- Correctness: functional properties
- Protocols, N-ary links
- Refinement
- Improving the COSTO tool