An Algebraic Semantics for Contract-Based Software Components

Rolf Hennicker

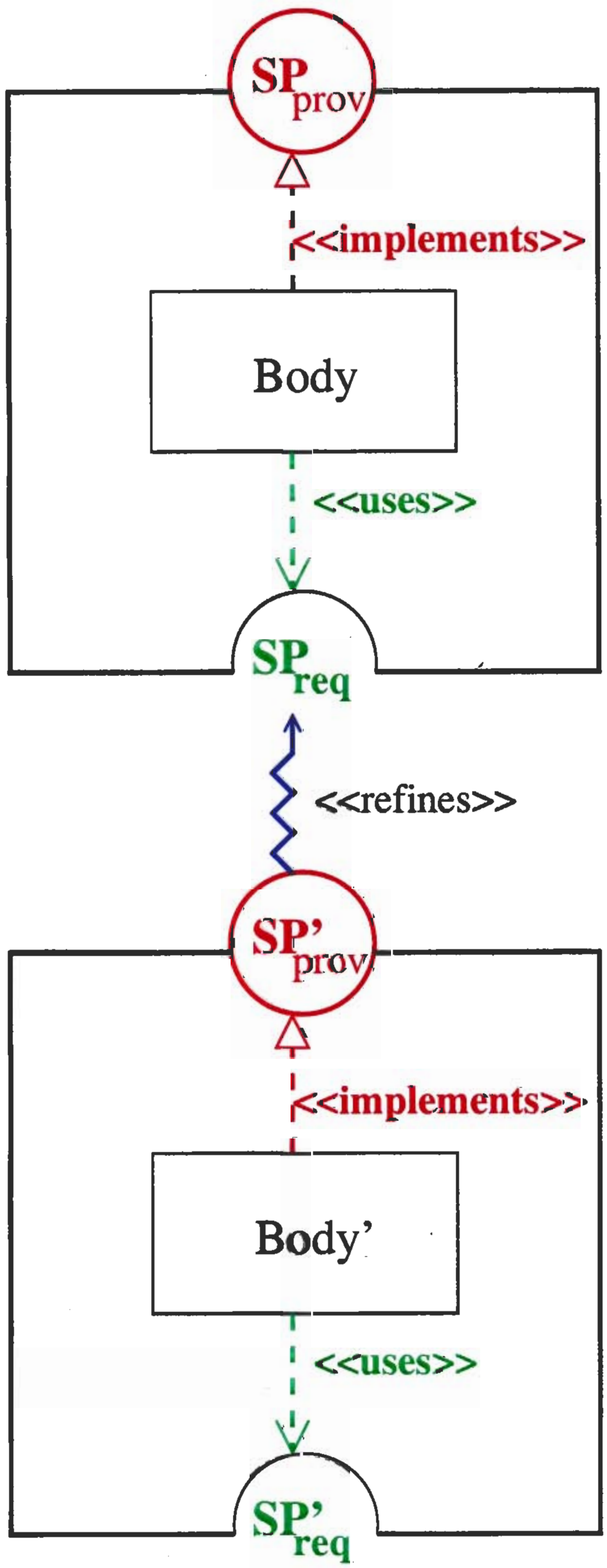
Ludwig-Maximilians-Universität München

Michel Bidoit

Centre de recherche INRIA Saclay

Scope: Assertion-based (sequential) SW components

- user
- implementor



provided interface specification

implementation of the provided operations

required interface specification

Literature:

ACT TWO
CASL architec. specs
CommUnity
+ Cos
OO: Eiffel
JML
Spec#
ADLs: Wright
Darwin

...

Here:

Model-theoretic approach

- signatures, sentences
- models, satisfaction relation

$$M \models \varphi$$

- loose semantics for
  * implementation relations

    $\text{Body} \dashrightarrow \mathit{SP}_{prov}$
  * refinement relations

    $\mathit{SP}'_{prov} \rightsquigarrow \mathit{SP}_{req}$

Abstract framework to be applicable to

- concrete specification formalisms
- concrete implementation languages

Concrete enough to support

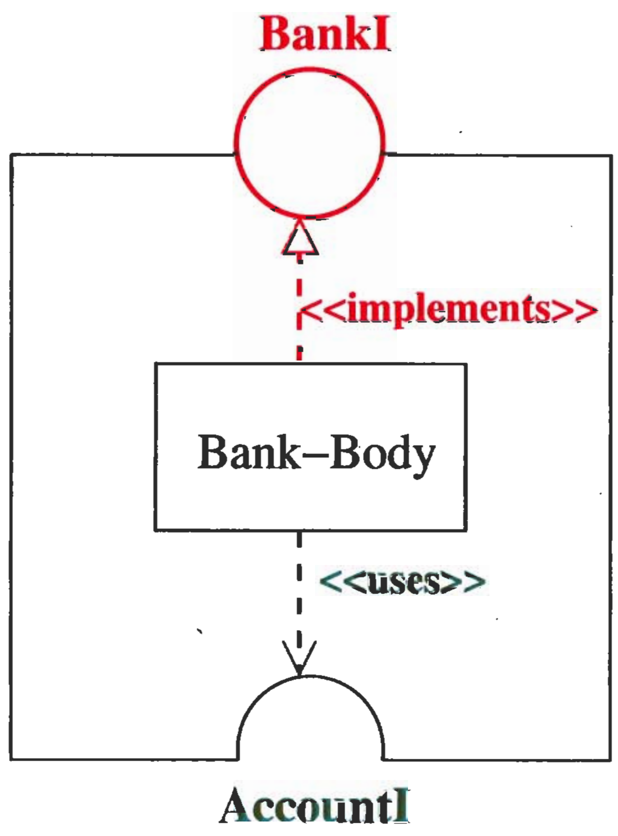
- definition of states (abstract + concrete)
- dynamic evolution of component instances and configurations
- strong encapsulation of states
- contract principle

Background: observ. alg. specs, architec. specs
OO-specs

Plan of the Talk

1. Interface specifications
2. Component bodies and how they use required interface specs
3. Component bodies and how they implement provided interface specs
4. Semantics of components
5. Component composition

Example: The *Bank* Component



1. Interface Specifications $SP_I = (\Sigma_I, Ax)$

interface spec AccountI =

```
primitive types int, ...
observer getBal: -> int;
operations
  credit(int i: i >= 0);
  withdraw(int i: i >= 0 and getBal >= i);
inv  getBal >= 0;
effects
  credit : getBal = getBal@pre + i;
  withdraw : getBal = getBal@pre - i;
```

observer signature $\Sigma_{Obs} = (\Sigma_{prim}, Obs)$

domain constraint

state predicate

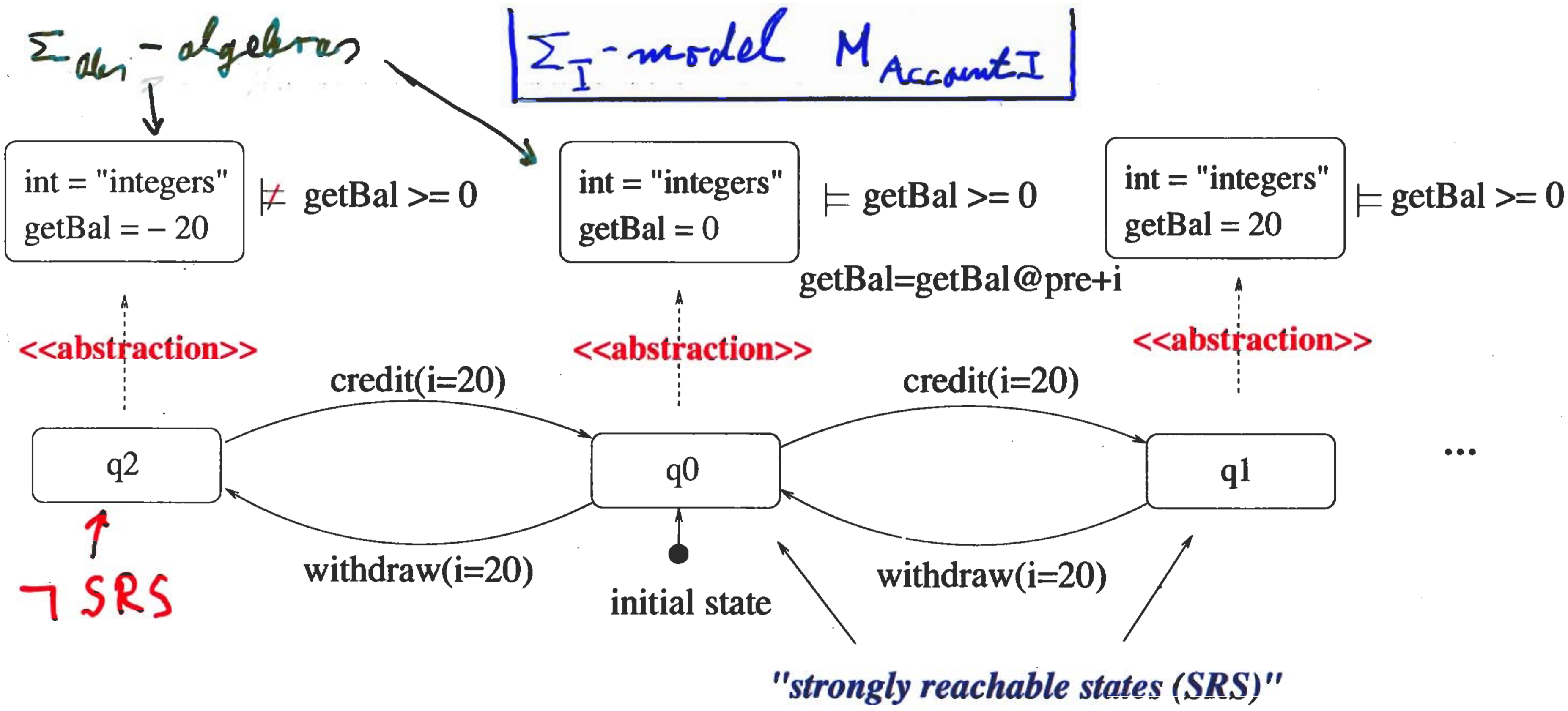
transition predicates

interface sig. $\Sigma_I = (\Sigma_{Obs}, Op)$

axioms

Semantics: "class of all correct realizations" (Hoare)

$[\![SP_I]\!]$ = class of all Σ_I-models satisfying Ax



$M_{AccountI} \models_I$ *inv* getBal >= 0 i.e. "getBal >= 0 *holds in all (abstracted) SRS* "

$M_{AccountI} \models_I$ credit: getBal = getBal@pre + i

i.e. "getBal=getBal@pre+i *holds for all transistions on SRS* "

How Σ_I-models reflect the "contract principle"

<u>Responsibility</u> <u>of</u> <u>the</u> <u>user</u>:

Any system run goes only through strongly reachable states!

<u>Responsibility</u> <u>of</u> <u>the</u> <u>implementor</u>:

Any system run through strongly reachable states respects the given axioms!

2. Component bodies and how they use required interface specifications

required type

body Bank-Body =

```
let Account : AccountI;
private Map<nat,Account> accounts;

public int balAcc(nat no)
 {Account acc = accounts.get(no);
  return acc.getBal();}

public void transfer(nat from,to; int i)
 {Account source = accounts.get(from); source.withdraw(i);
  Account target = accounts.get(to); target.credit(i);}
```

attribute sig. Σ_{Att}

body sig. $\Sigma_{Body} = (\Sigma_{Att}, Op)$

Op - Impl

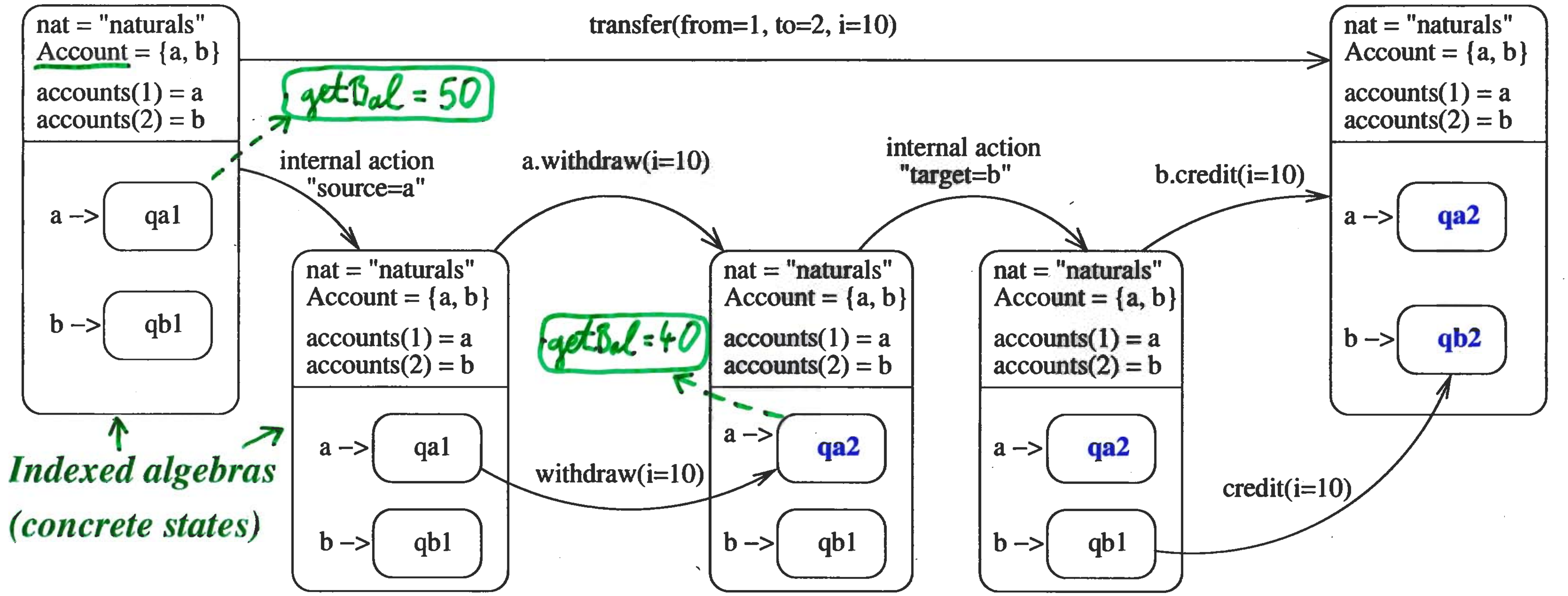
We assume

$[\![Op\text{-}Impl]\!] : [\![SP_{req}]\!] \to$ class of Σ_{Body}-models

$\in$

$M_{req} \longmapsto \Sigma_{Body}$-model M <u>over</u> M_{req}

Σ_{Body} - model M_{Bank} over $M_{AccountI}$



!! *Requirements* (strong encapsulation)

- Internal actions do not modify the state of instances of a required type!
- Calling a required operation on an instance o can only modify the state of o!

Σ_{Body} - model : Transition system with indexed algebras as states satisfying the above requirements

Theorem

Let $M_{req} \in [\![SP_{req}]\!]$, let M be a Σ_{Body}-model <u>over</u> M_{req}.

If M is a "correct user" of M_{req} then:

1. $M_{req} \models_I inv\ \varphi$ $\overset{\Rightarrow\checkmark}{\underset{\Leftarrow ?}{\text{iff}}}$ $M \models inv\ \forall x{:}rt\ \ x.\varphi$

   $\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ $\llcorner$ req. type

   e.g.

   $M_{AccountI} \models_I inv\ getBal \geq 0$ iff $M_{Bank} \models$
   $\forall x{:}Account\ \ x.getBal \geq 0$

2. $M_{req} \models_I op_{req} : \pi$ iff $M \models op_{req} : \forall x{:}rt\ \ x.\pi$

   e.g.

   $M_{AccountI} \models_I withdraw : getBal = getBal@pre - i$ iff

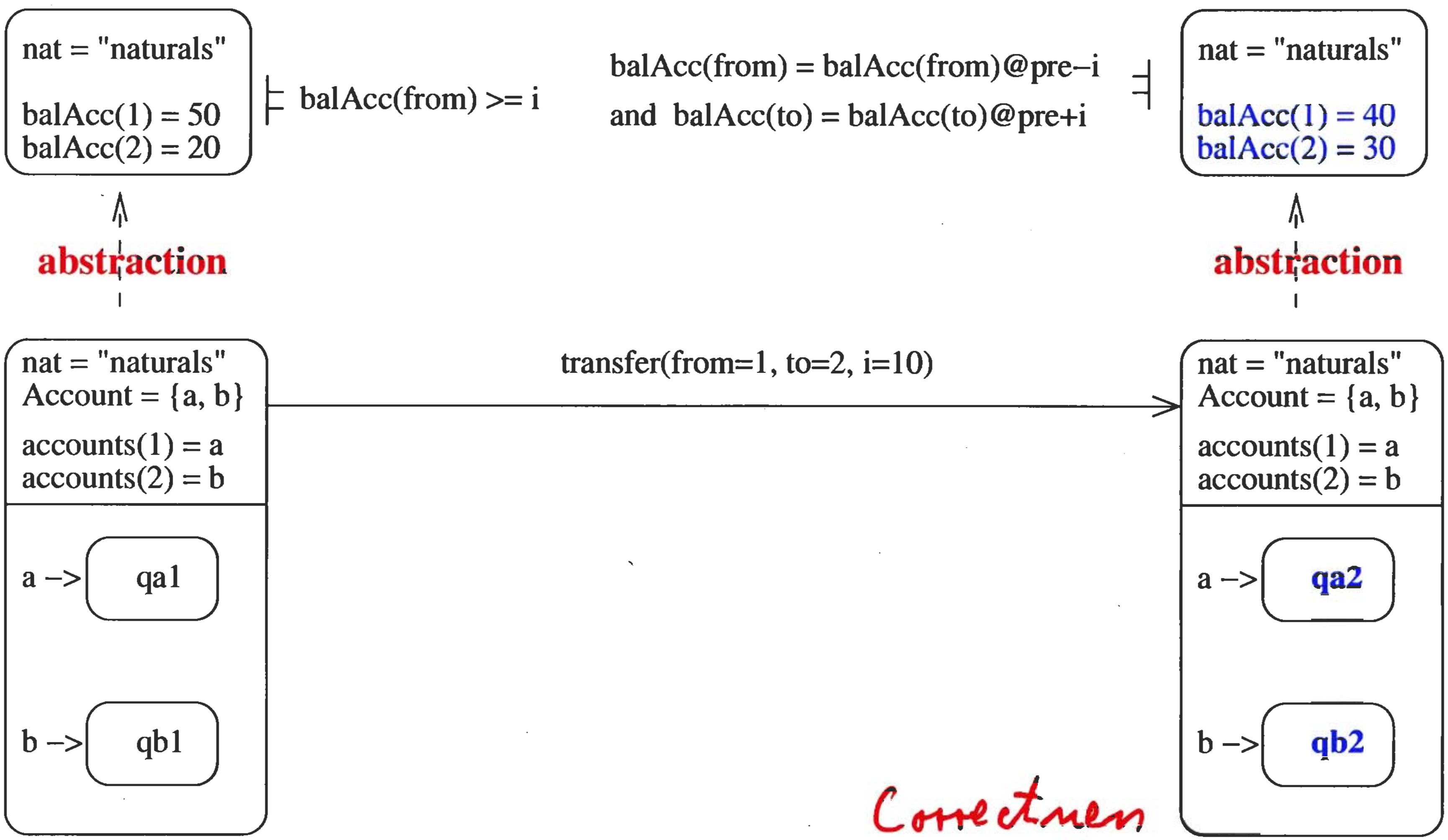
   $M_{Bank} \models withdraw : \forall x{:}Account\ \ x.getBal = x.getBal - i$

3. Component bodies and how they implement provided interface specifications

provided interface spec BankI =

```
primitive types int, nat, ...
observers
  balAcc: nat -> int;
                                      domain constraint
operations
  transfer(nat from, to; int i: i >= 0 and balAcc(from) >= i);
  ...
inv  balAcc(no) >= 0;
effect
  transfer : balAcc(from) = balAcc(from)@pre - i and
             balAcc(to) = balAcc(to)@pre + i;
```

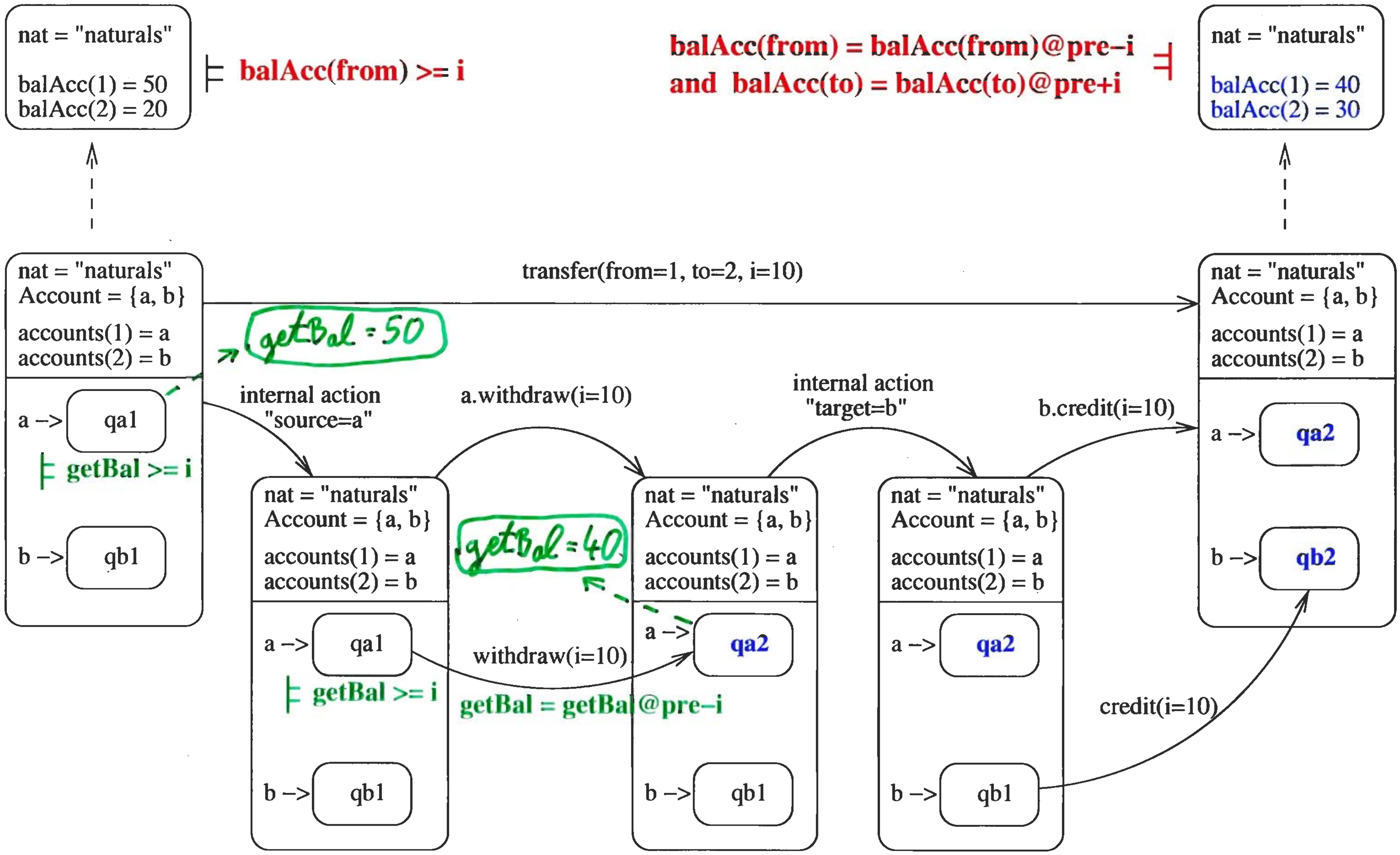
How to construct interface models out of body models



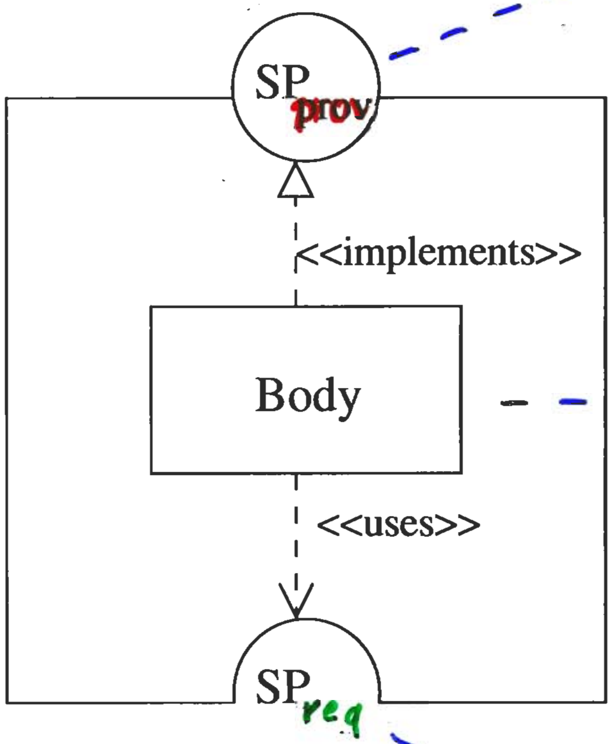
Correctness

$Restrict_{Sig(BankI)}(⟦Bank\text{-}Body⟧(M_{AccountI})) \in ⟦BankI⟧$!

Modular Verification



4. Components



$(\Sigma^{I}_{prov}, Ax^{I}_{prov})$

$\quad (\Sigma^{obs}_{prov}, Op_{prov}, dom_{prov})$

$\quad\quad (\Sigma_{prim}, Obs_{prov})$

$(\Sigma_{Body}, José : [\![SP_{req}]\!] \longrightarrow Mod(\Sigma_{Body}))$

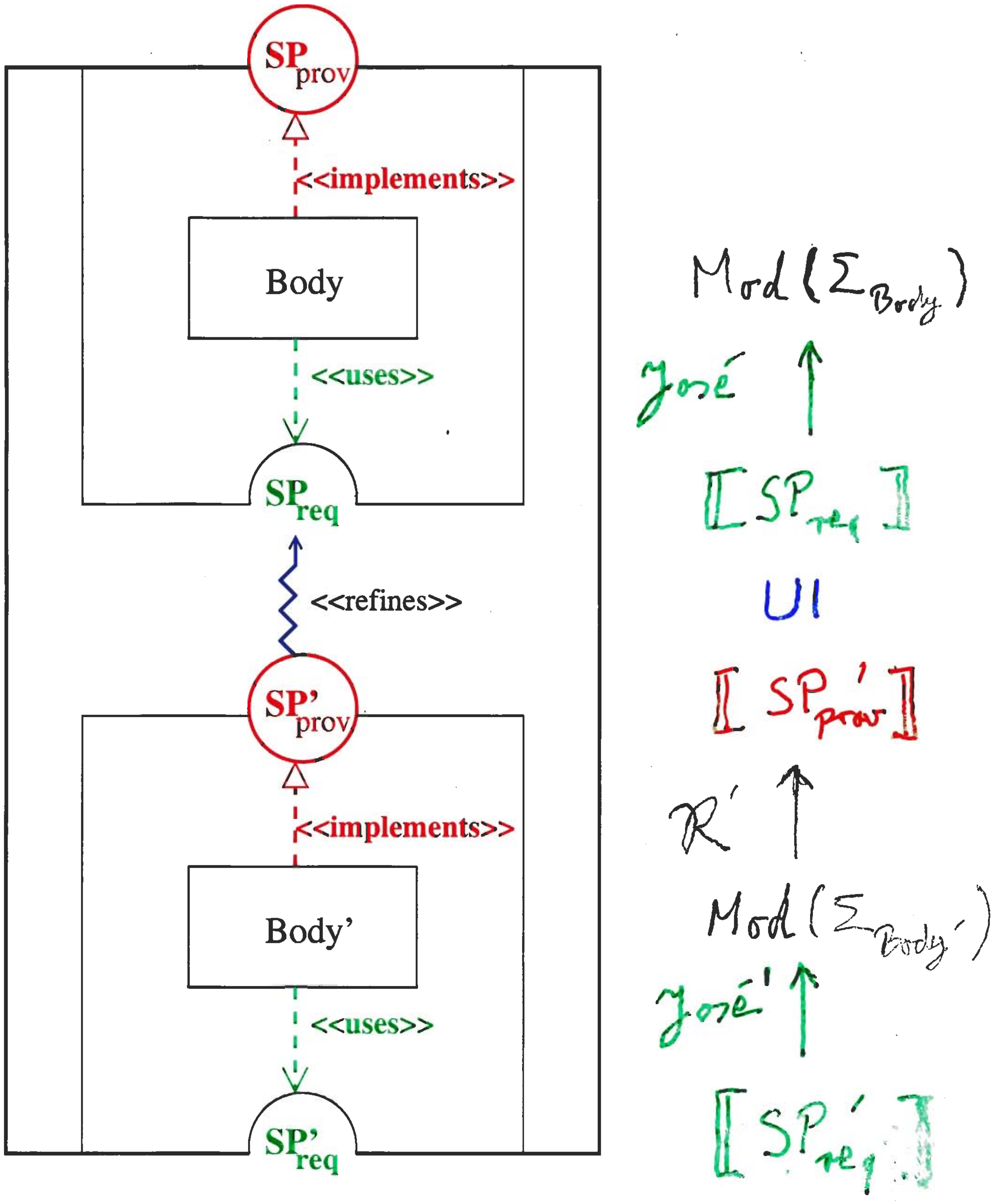
$\quad (\Sigma_{Att}, Op)$

$\quad\quad (st : SP_{req}, \Sigma_{prim} + st, Att)$

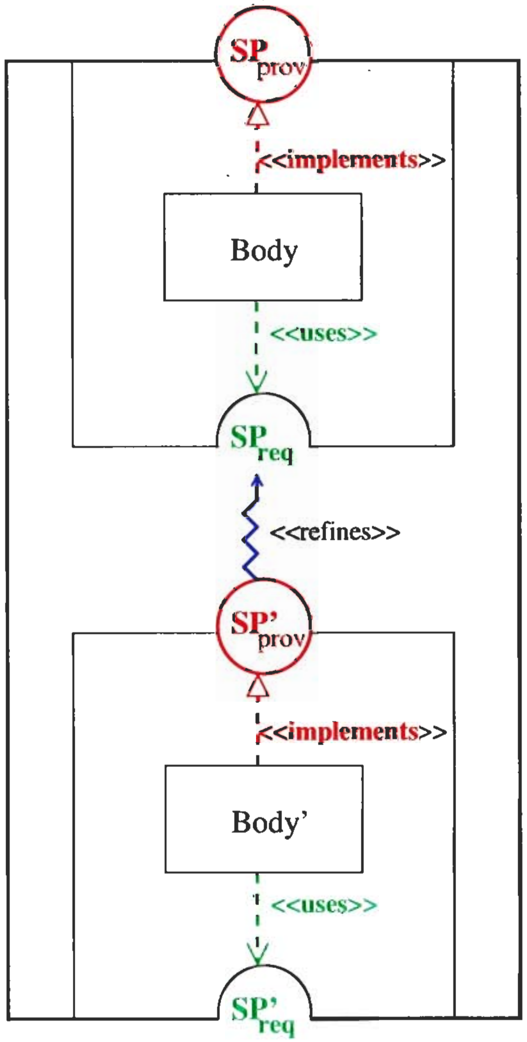
$(\Sigma^{I}_{req}, Ax^{I}_{req})$

Correctness:

$$\mathcal{R}_{\Sigma^{I}_{prov}}(José([\![SP_{req}]\!])) \subseteq [\![SP_{prov}]\!]$$


SP prov
<<implements>>
Body
<<uses>>
SP req
<<refines>>
SP' prov
<<implements>>
Body'
<<uses>>
SP' req
Mod(Σ Body)
José
⟦SP req⟧
∪
⟦SP' prov⟧
R'
Mod(Σ Body')
José'
⟦SP' req⟧

5. Component composition



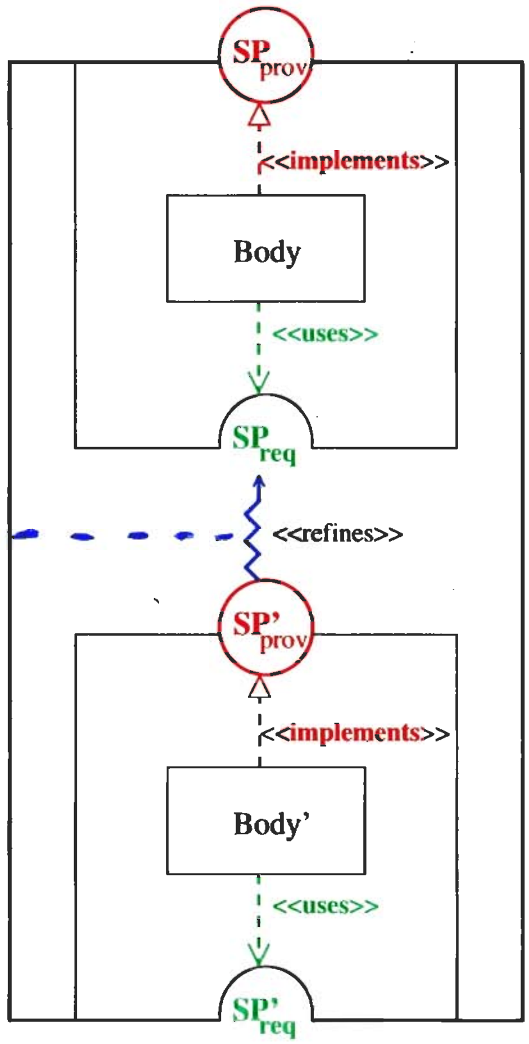
$$\begin{tikzcd} Mod(\Sigma_{Body}) \\ [\![SP_{req}]\!] \arrow[u, "\text{José}"] \\ [\![SP'_{prov}]\!] \arrow[u, phantom, "\cup\!\mid"] \\ Mod(\Sigma_{Body'}) \arrow[u, "\mathcal{R}'"] \\ [\![SP'_{req}]\!] \arrow[u, "\text{José}'"] \end{tikzcd}$$

5. Component composition

$[\![SP_{req}]\!]$

$\cup|$

$[\![SP'_{prov}]\!]$



$[\![Body]\!](\quad) \in \Sigma_{Body}\text{-models}$

$Restr'(\quad)$

$[\![Body']\!](\quad)$

$M'_{req} \in [\![SP'_{req}]\!]$

Fact: Correctness of subcomponents propagates to CC

Conclusion

- Rigorous model-theoretic foundation of (sequential) component-based systems supporting
  - contract-based system development
  - strong encapsulation of states
  - dynamic evolution based on "indexed algebras"
- Future work
  - addition of behaviour protocols
  - extension to concurrent, distributed components