

Formal Modelling of Service-Oriented Systems

José Fiadeiro

Laura Bocchi, João Abreu



**University of
Leicester**

Antónia Lopes



**UNIVERSIDADE
DE LISBOA**

aims of this tutorial

aims of this tutorial

- to provide you with an overview of a formal approach to service-oriented modelling that we have been developing in the SENSORIA project
 - a 'prototype' modelling language – SRML
 - (part of) its semantics
 - methodological aspects of an engineering approach to service-oriented systems

aims of this tutorial

- to provide you with an overview of a formal approach to service-oriented modelling that we have been developing in the SENSORIA project
 - a 'prototype' modelling language – SRML
 - (part of) its semantics
 - methodological aspects of an engineering approach to service-oriented systems
- a companion paper is available from:
 - www.cs.le.ac.uk/people/jfiadeiro

plan of this tutorial

plan of this tutorial

- **Setting the scene**
 - the context – SENSORIA
 - what we mean by 'service'
 - what we mean by 'modelling'

plan of this tutorial

■ Setting the scene

- the context – SENSORIA
- what we mean by 'service'
- what we mean by 'modelling'

■ Engineering service-oriented systems

- why (we think that) it is not the same as for component-based systems
- social complexity
- service consumers (activities) vs service providers
- static vs dynamic aspects

plan of this tutorial

plan of this tutorial

■ SRML

- Use Cases for SOC
- A language and model of interactions for SOC
- Orchestration
- 'Provides' and 'Requires' interfaces
- Connectors and interaction protocols
- Internal configuration policies
- External configuration policies – SLA's

plan of this tutorial

■ SRML

- Use Cases for SOC
- A language and model of interactions for SOC
- Orchestration
- 'Provides' and 'Requires' interfaces
- Connectors and interaction protocols
- Internal configuration policies
- External configuration policies – SLA's

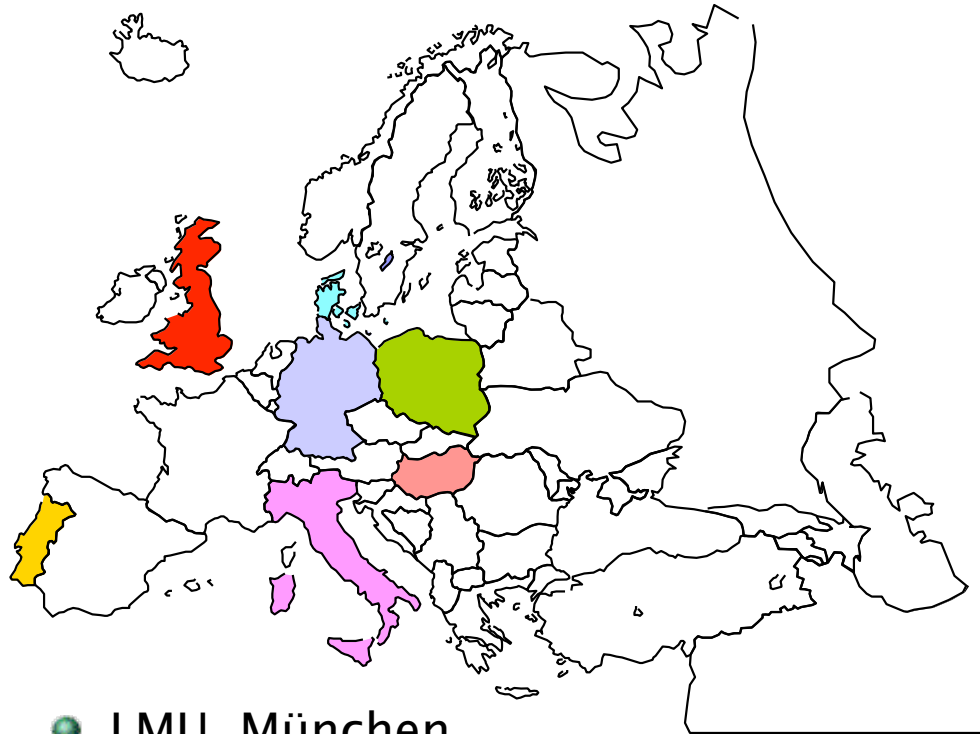
■ Semantics of service discovery and binding

the context

the context

SENSORIA

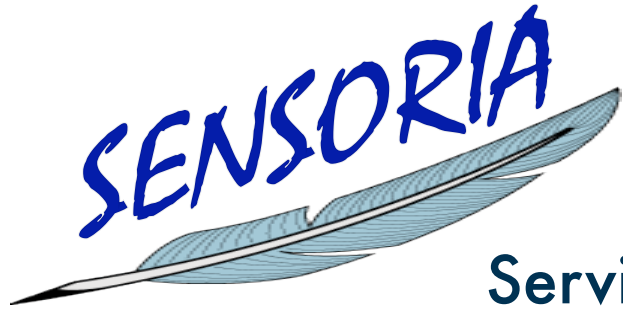




- LMU München
- FAST GmbH
- S&N AG
- TU Denmark at Lyngby
- Warsaw University
- Budapest University of Technology and Economics

- Università di Pisa
- Università di Firenze
- Università di Bologna
- ISTI Pisa
- Telecom Italia Lab
- Università di Trento
- University of Leicester
- University of Edinburgh
- Imperial College London
- University College London
- Universidade de Lisboa
- ATX Software SA

more precisely...

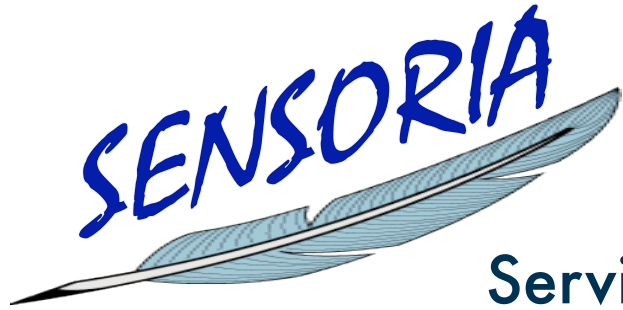


An IST-FET Integrated Project Sept05–Aug09

Software Engineering for Service-Oriented Overlay Computers

The aim of SENSORIA is to develop a novel comprehensive approach to the engineering of software systems for service-oriented overlay computers where foundational theories, techniques and methods are fully integrated in a pragmatic software engineering approach.

even more precisely...



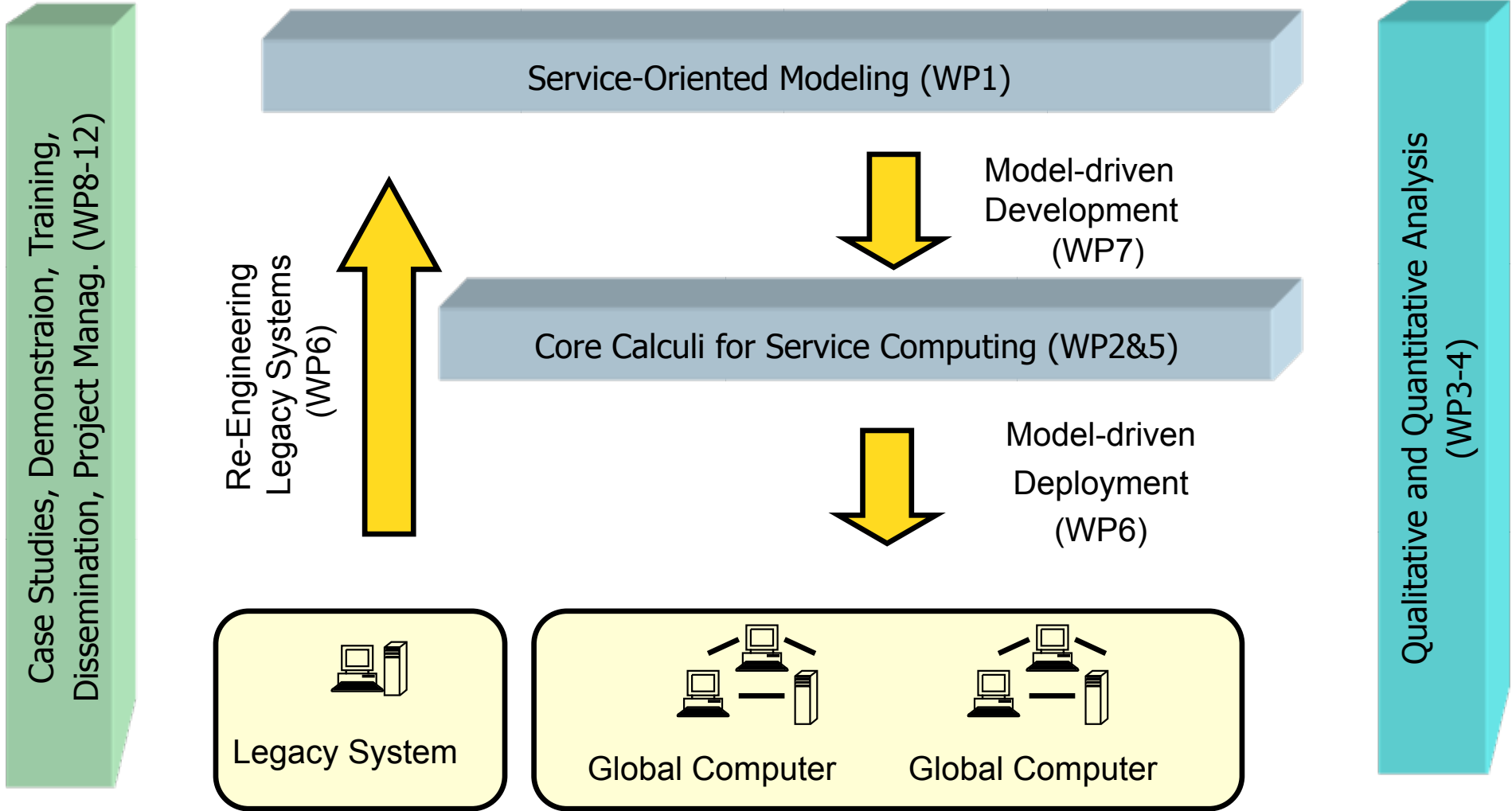
An IST-FET Integrated Project **Sept05–Aug09**

Software Engineering for Service-Oriented Overlay Computers

- WP1** Provide support for service-oriented modelling at high levels of abstraction, i.e. independently of the hosting middleware and hardware platforms, and the languages in which services are programmed.

even more precisely...





Services?

Services?

A personal experience...

Services?



You're closer than you think to the perfect mortgage


2 year **fixed** rate mortgage
3.69% **5.8% APR**
[click here](#)

Lowest



FIRST TIME BUYERS  **JUST 3% DEPOSIT**

4.29%
2 Year Fixed

 **FFWD**
your mortgage

With a Woolwich Openplan Offset Mortgage you could pay off your mortgage sooner.

[Click Here](#)



Escape to a better rate

[tracker online](#)

too many products

too many products

■ how to choose a mortgage?

- how to choose the *right* mortgage?

- how to choose the mortgage that is right for *me*?

too many products

- how to choose a mortgage?
 - how to choose the *right* mortgage?
 - how to choose the mortgage that is right for *me*?
- it was more than a mortgage that I needed...
 - which bank would I use for the monthly payments?
 - what about life insurance?
 - and protection against job loss?
 - and perhaps a saving scheme?

from products to *services*

from products to *services*



HERTFORDSHIRE MORTGAGE SERVICES

Are you looking for a mortgage package suitable for your own personal needs?

Welcome to the Hertfordshire Mortgage Services website. As independent mortgage advisors, we aim to bring you the ideal mortgage for your own needs. We are computer linked to all the UK's lenders, so we are able to match you to your perfect mortgage.

We aim to provide the very best service

Whether you are a first time buyer, looking to remortgage, or thinking of becoming a landlord we can match the right package to your individual needs.

If you are self employed, have been declined by a lender or have county court judgments we can still help you find the right mortgage for you click [here](#) to find out how.

from products to *services*



**HERTFORDSHIRE
MORTGAGE SERVICES**

Are you looking for a mortgage package suitable for your own personal needs?

Welcome to the Hertfordshire Mortgage Services website. As independent mortgage advisors, we aim to bring you the ideal mortgage for your own needs. We are computer linked to all the UK's lenders, so we are able to match you to your perfect mortgage.

We aim to provide the very best service

Whether you are a first time buyer, looking to remortgage, or thinking of becoming a landlord we can match the right package to your individual needs.

If you are self employed, have been declined by a lender or have county court judgments we can still help you find the right mortgage for you click [here](#) to find out how.

The
**Mortgage
Code**



from products to *services*

- Abstracts away the **identity** of the component(s) out of which the service is provided;
- Provides an explicit representation of the **role** under which the service was procured, and which led to the choice of specific components;
- The **choice** of the configuration of components that provides the required service is performed by **experts** in a more **restricted** domain;
- Service providers have to abide to rules that ensure certain levels of **quality**

judgments we can still help you find the right mortgage for you click [here](#) to

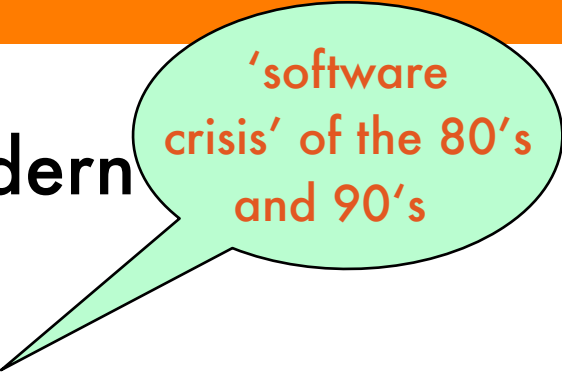
social complexity

social complexity

- The major source of complexity in modern software-intensive systems is 'social':

social complexity

- The major source of complexity in modern software-intensive systems is 'social':
 - Systems are not necessarily 'big chunks of software'...



'software crisis' of the 80's and 90's

social complexity

- The major source of complexity in modern software-intensive systems is 'social':
 - Systems are not necessarily 'big chunks of software'...
 - ... but they may exhibit complex and dynamic/evolving **interactions** among possibly huge numbers of parties

'software crisis' of the 80's and 90's

social complexity

- The major source of complexity in modern software-intensive systems is 'social':
 - Systems are not necessarily 'big chunks of software'...
 - ... but they may exhibit complex and dynamic/evolving **interactions** among possibly huge numbers of parties
 - The major concern is in having representations of the 'business' roles that parties play within a system...
 - ... and in having the means for **procuring** and **interconnecting** the parties required to execute a given business process, only **when** they are required (not so much in **developing** the parties themselves)

'software crisis' of the 80's and 90's

services vs components

- In **CBD**, software components are “taken out of a box” and **plugged** into a system (possibly with the addition of some “glue” code) to provide a “service” (see Broy et al, TOSEM February 2007)
- In **SOC**, each time a service is invoked, a different provider may be chosen to negotiate terms and conditions, and then the service is finally **bound** (see Elfatatry, CACM August 2007)

A bank will use components for calculating interests, charging commissions, etc, that it will use in different products (savings, loans, ...)

The same bank is likely to rely on external courier services that are procured according to the delivery address, speed, cost, ...

services vs components

- In **CBD**, software components are “taken out of a box” and **plugged** into a system (possibly with the addition of some “glue” code) to provide a “service” (see Broy et al, TOSEM February 2007)
- In **SOC**, each time a service is invoked, a different provider may be chosen to negotiate terms and conditions, and then the service is finally **bound** (see Elfatatry, CACM August 2007)

CBD assumes **early binding**: the “architecture” is defined at design time.
(physiological complexity)

SOC adopts **late binding**: binding is deferred to run time, enabling the choice of provision each time and change in the quality of the requirements.
(social complexity)

engineering SOC

engineering SOA

■ Stakeholders

● service providers

- do not develop 'bespoke' software to user's requirements
- need to offer services that correspond to 'market' demands

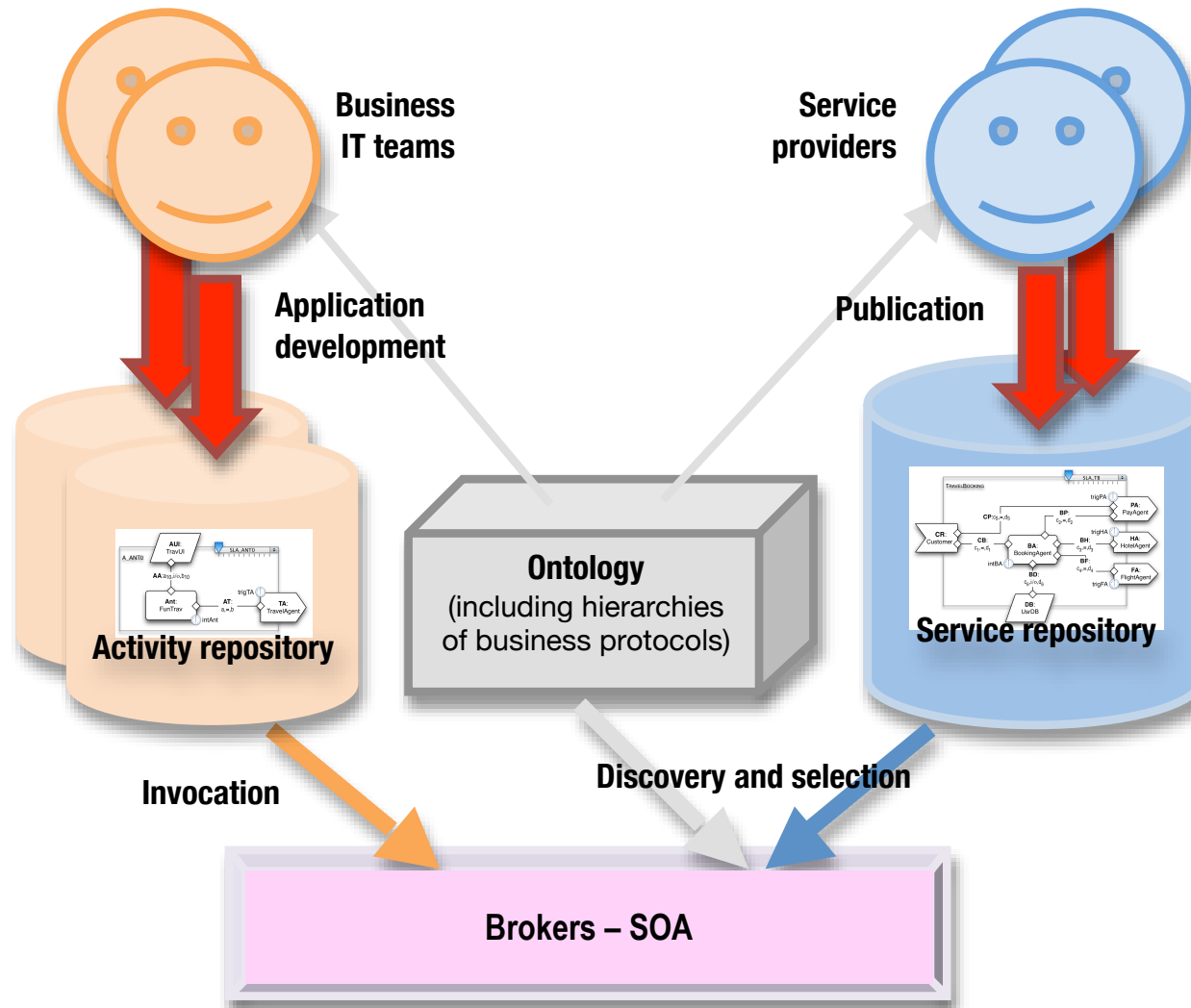
● service consumers

- are applications, not people
- are decoupled from the providers
- bind to services at run time, not design time

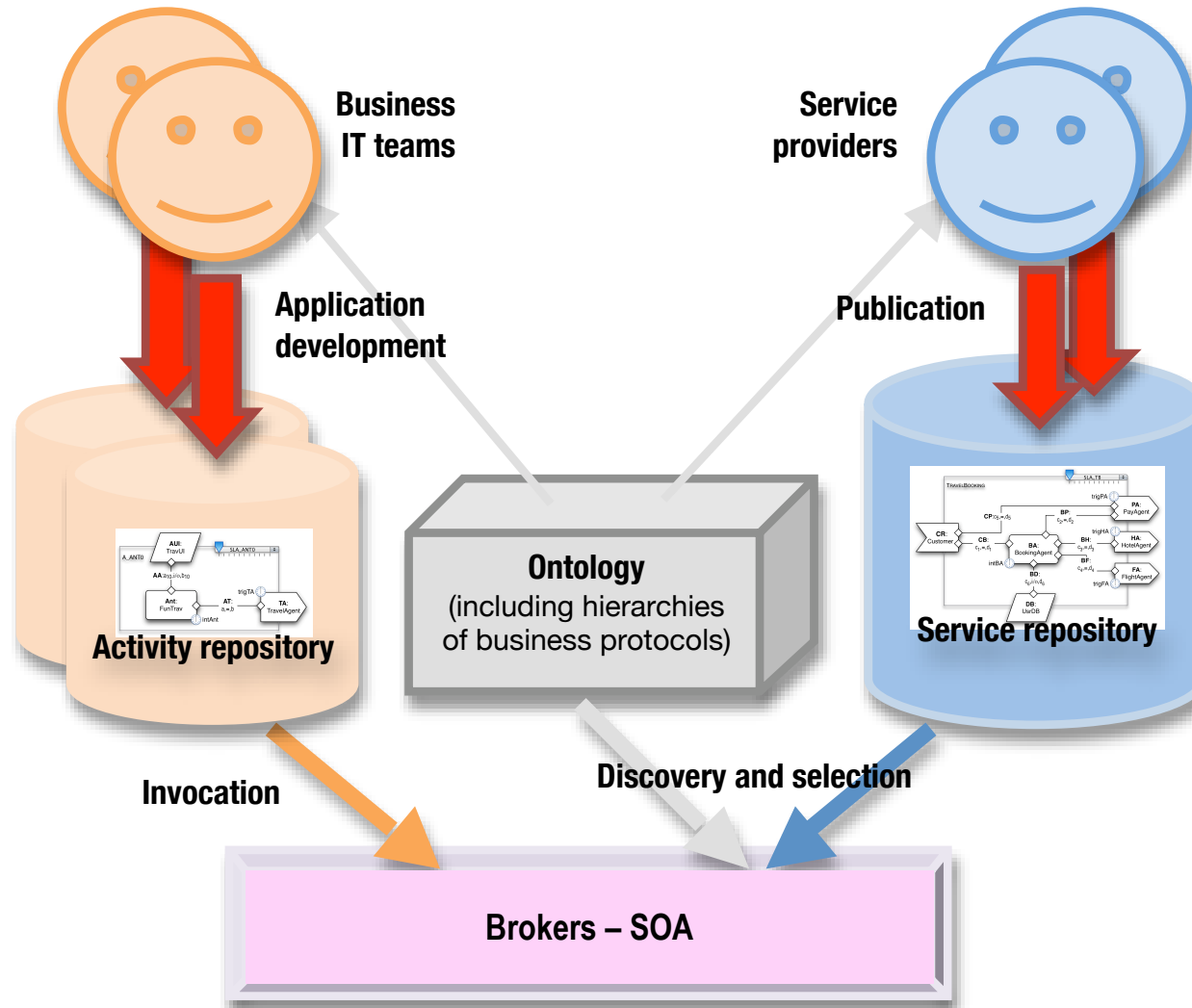
● service brokers

- manage registries
- binds consumer and provider
- offered as middleware in SOAs

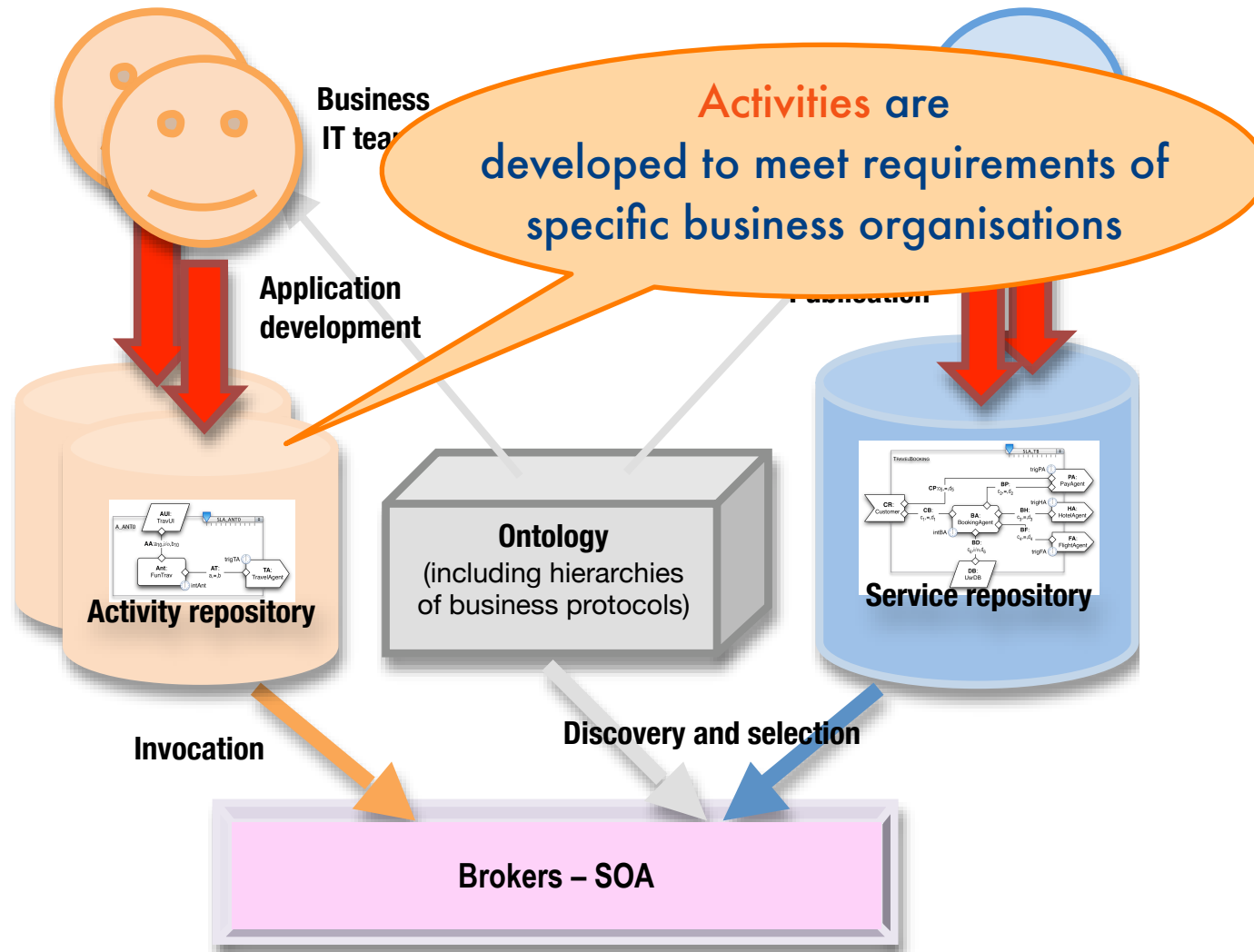
engineering architecture



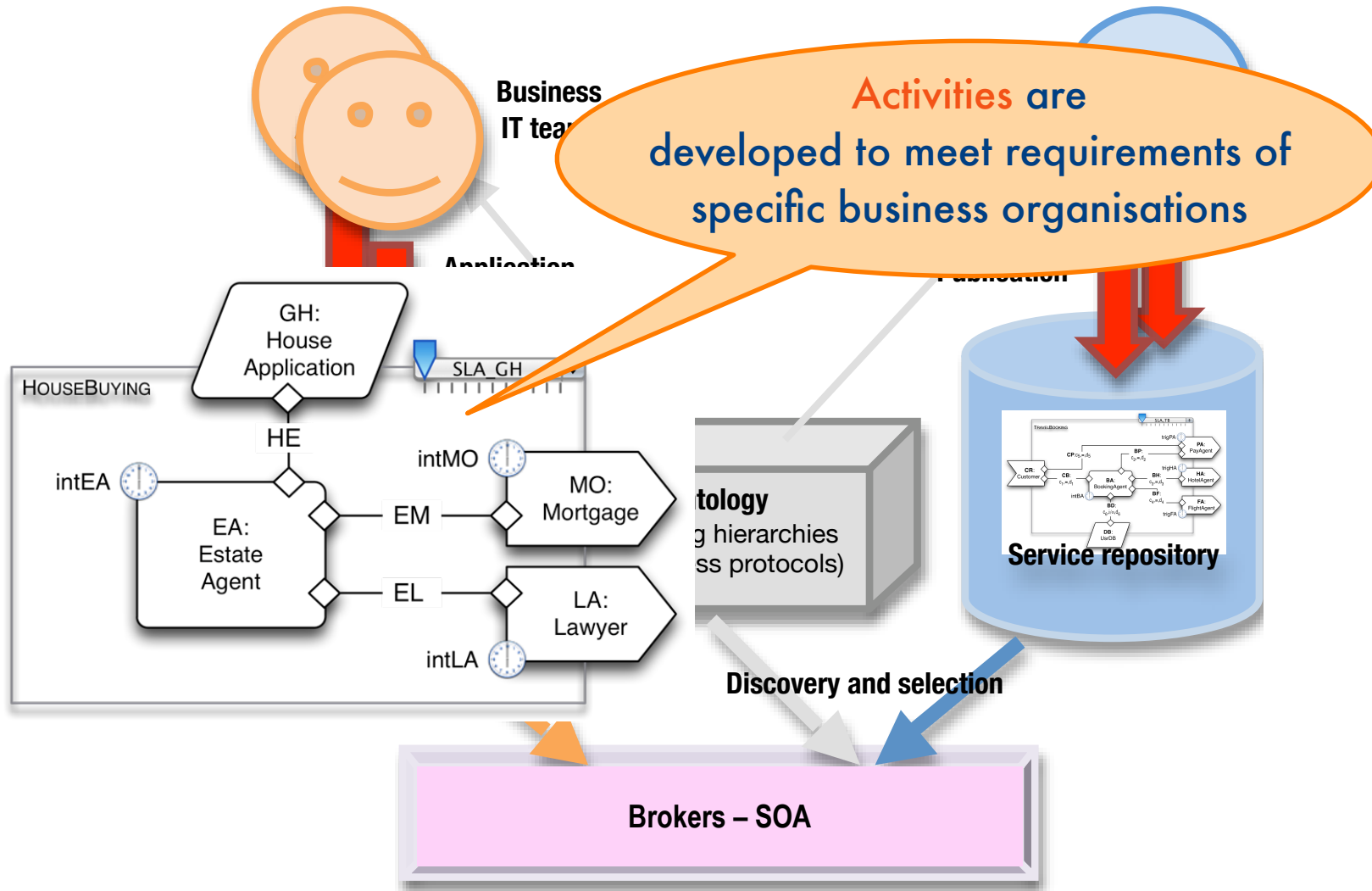
modelling



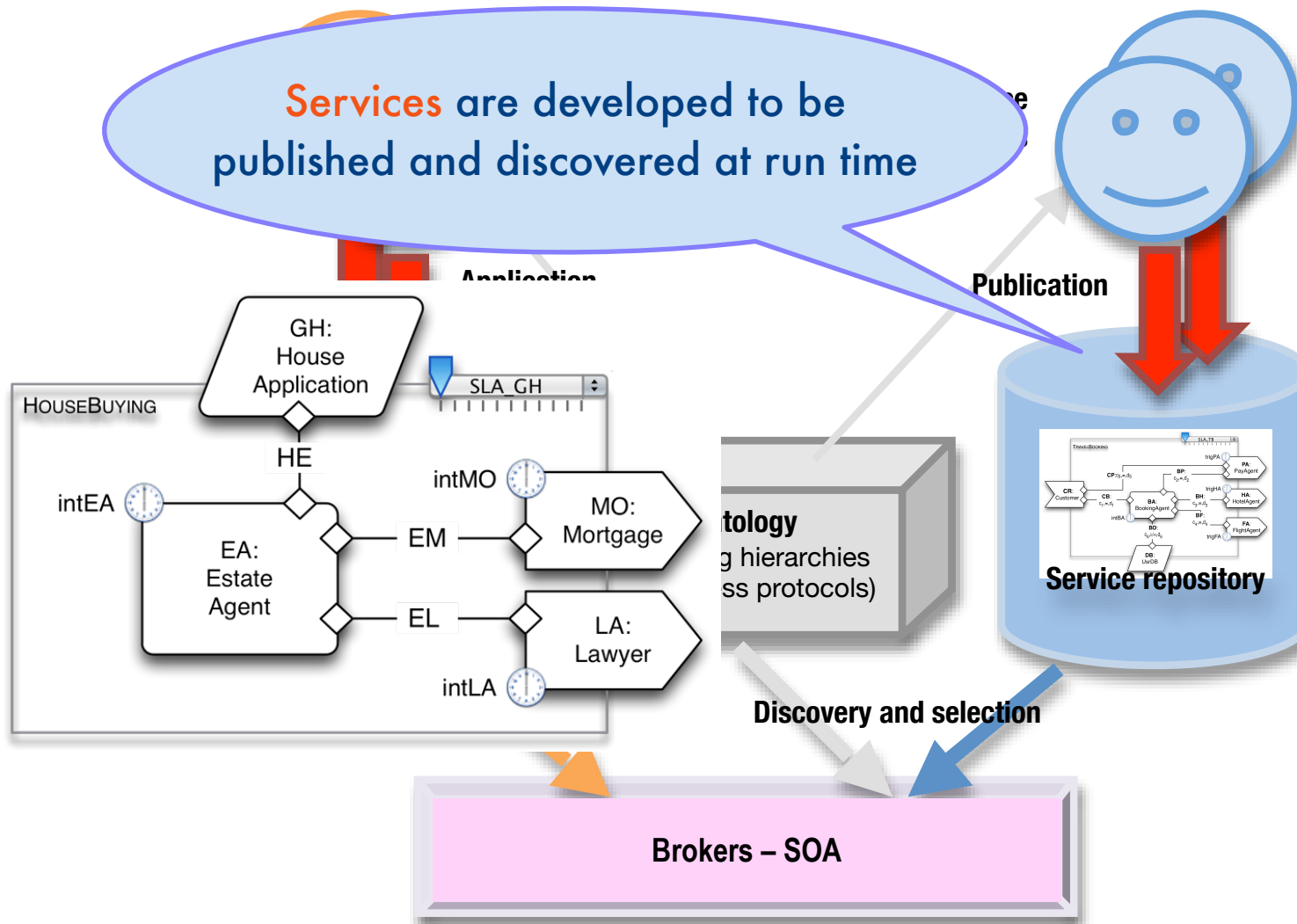
modelling



modelling

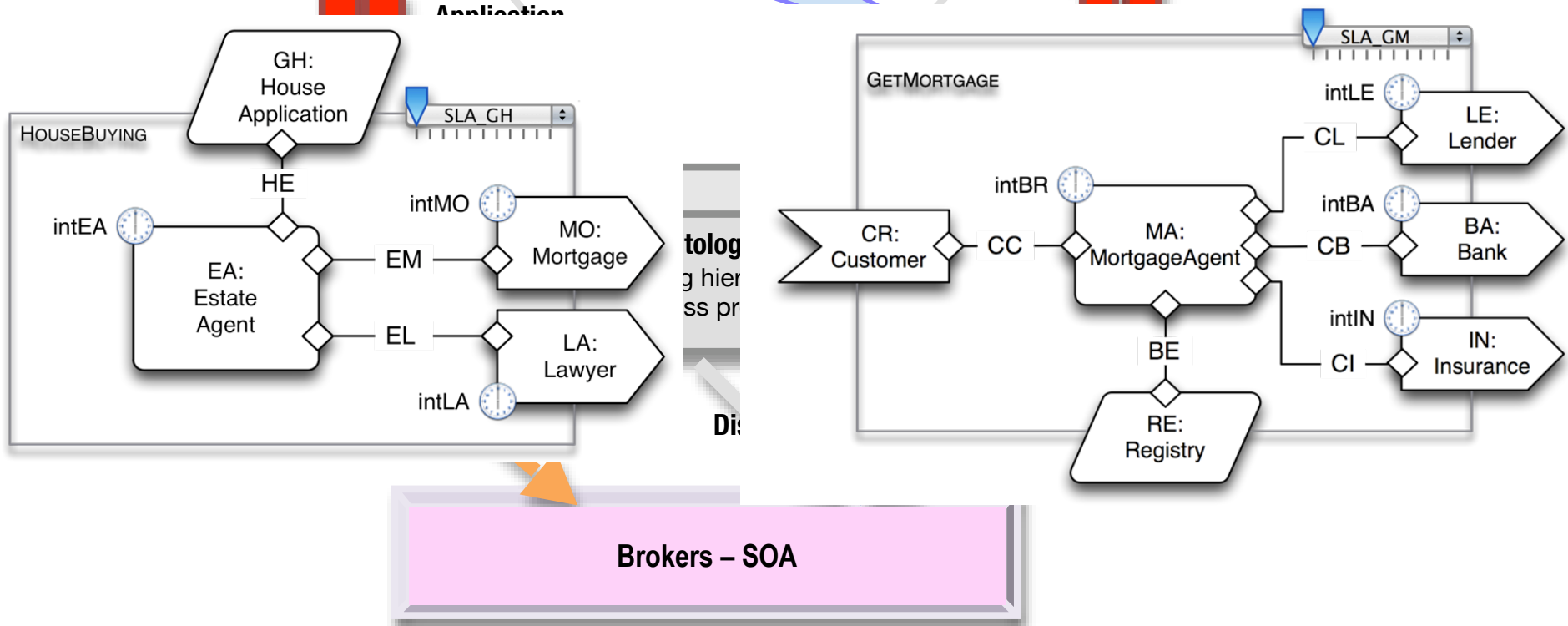


modelling

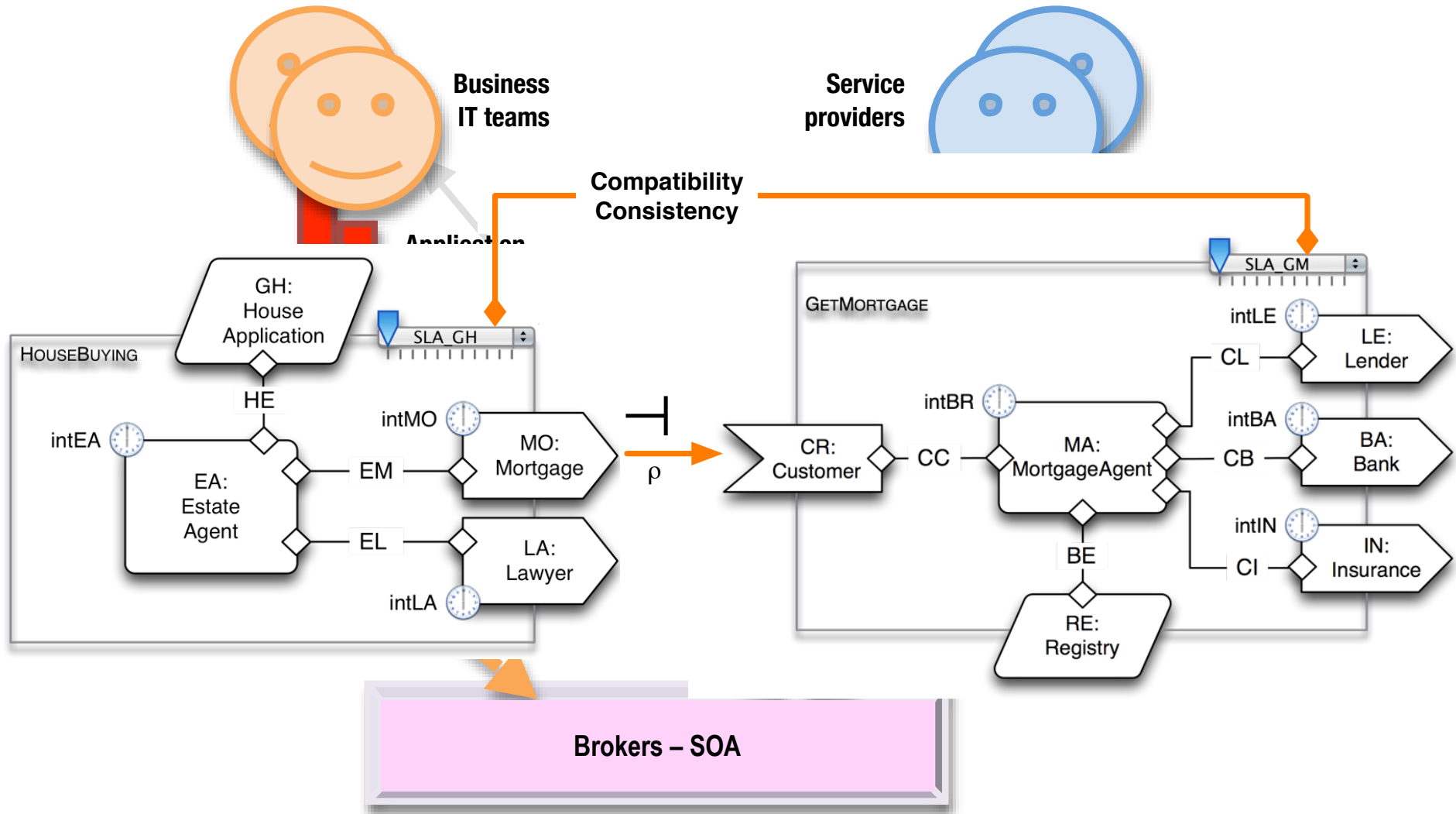


modelling

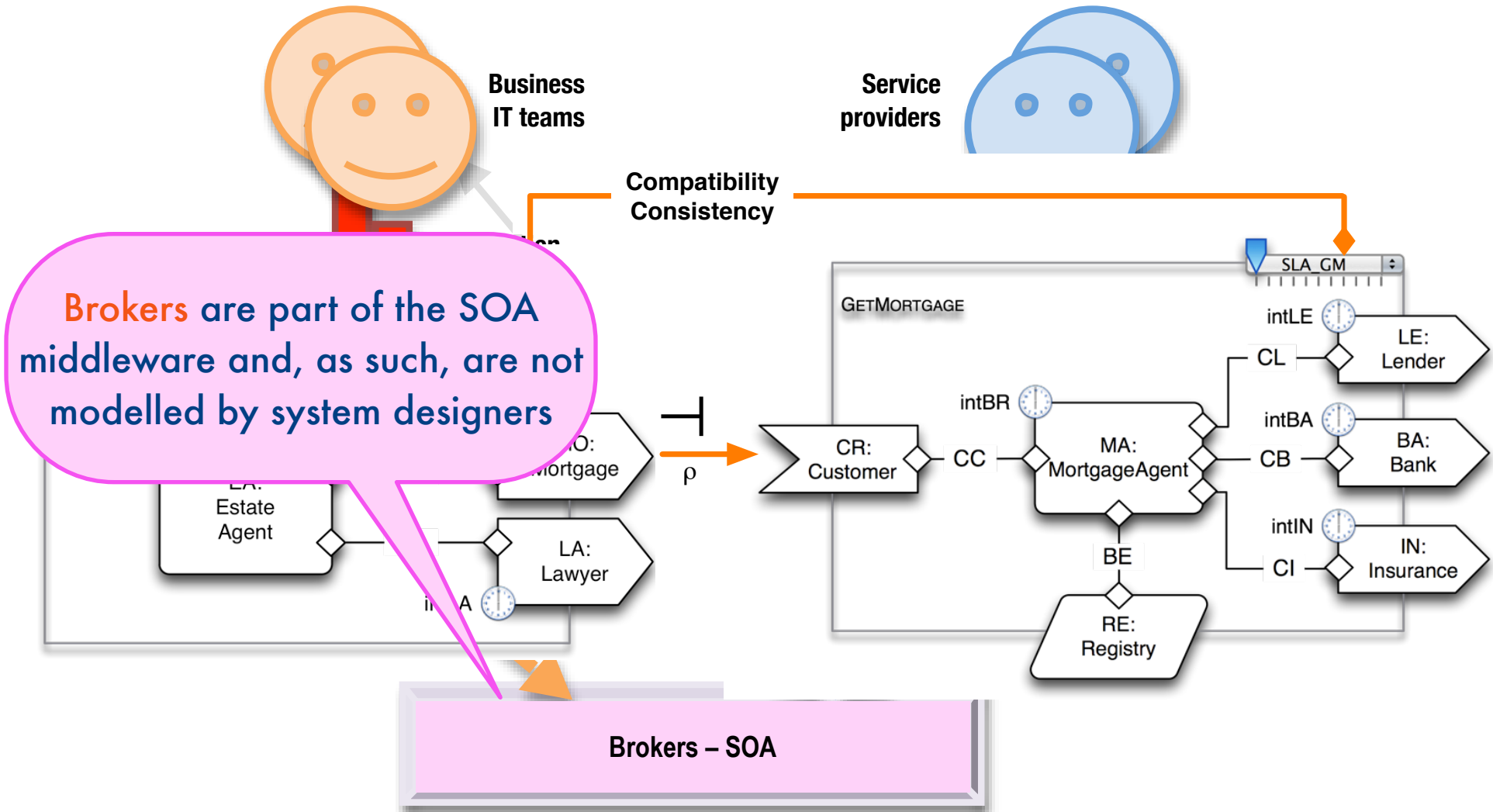
Services are developed to be published and discovered at run time



modelling



modelling



modelling

modelling

- different from programming
 - closer to the business domain (addresses the business logic and reflects business roles)
 - not necessarily executable
 - validation and verification independent of the implementation

modelling

■ different from programming

- closer to the business domain (addresses the business logic and reflects business roles)
- not necessarily executable
- validation and verification independent of the implementation

■ level of abstraction

- builds over the facilities offered by a SOA – brokers, session handling and message correlation mechanisms, ...

modelling

modelling

■ Static aspects:

- How can we account for the behaviour of services provided by collections of interconnected parties? – orchestration, conversation protocols (pledges, compensations, ...)

■ Dynamic aspects:

- How can we account for the run-time aspects of service-oriented systems that result from the SOA middleware mechanisms of service discovery, instantiation and binding?

Sensoria Reference Modelling Language

Sensoria Reference Modelling Language

■ Inspired by SCA:

- set of standards proposed by BEA, IBM, IONA, Oracle, Interface 2.1, SAP, Siebel, Sysbase
- *Service Component Architecture (SCA) is a specification that [...] aims to simplify the **creation and integration of business applications** built using a Service Oriented Architecture (SOA).*
- *[...] relatively coarse-grained business components are exposed as services, with well-defined interfaces and contracts. Interfaces are expressed using **technology-agnostic business terms and concepts**.*
- *SCA builds on emerging best practices of removing or **abstracting middleware programming model dependencies from business logic**.*
- *SCA allows developers to focus on writing **business logic**.*

Sensoria Reference Modelling Language

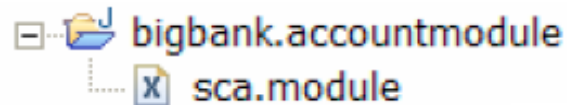
Sensoria Reference Modelling Language

- However, when it comes to 'semantics':

Sensoria Reference Modelling Language

■ However, when it comes to 'semantics':

- *"In this step you learn how to create an SCA module. A module is represented by a folder in the file system with an sca.module file at the folder root."*



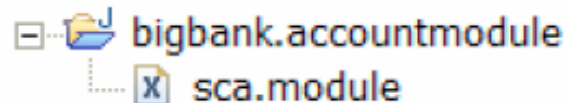
```
<?xml version="1.0" encoding="ASCII"?>
<module xmlns="http://www.osoa.org/xmlns/sca/0.9"
        name="bigbank.accountmodule" >

</module>
```

Sensoria Reference Modelling Language

■ However, when it comes to 'semantics':

- "In this step you learn how to create an SCA module. A module is represented by a folder in the file system with an *sca.module* file at the folder root."

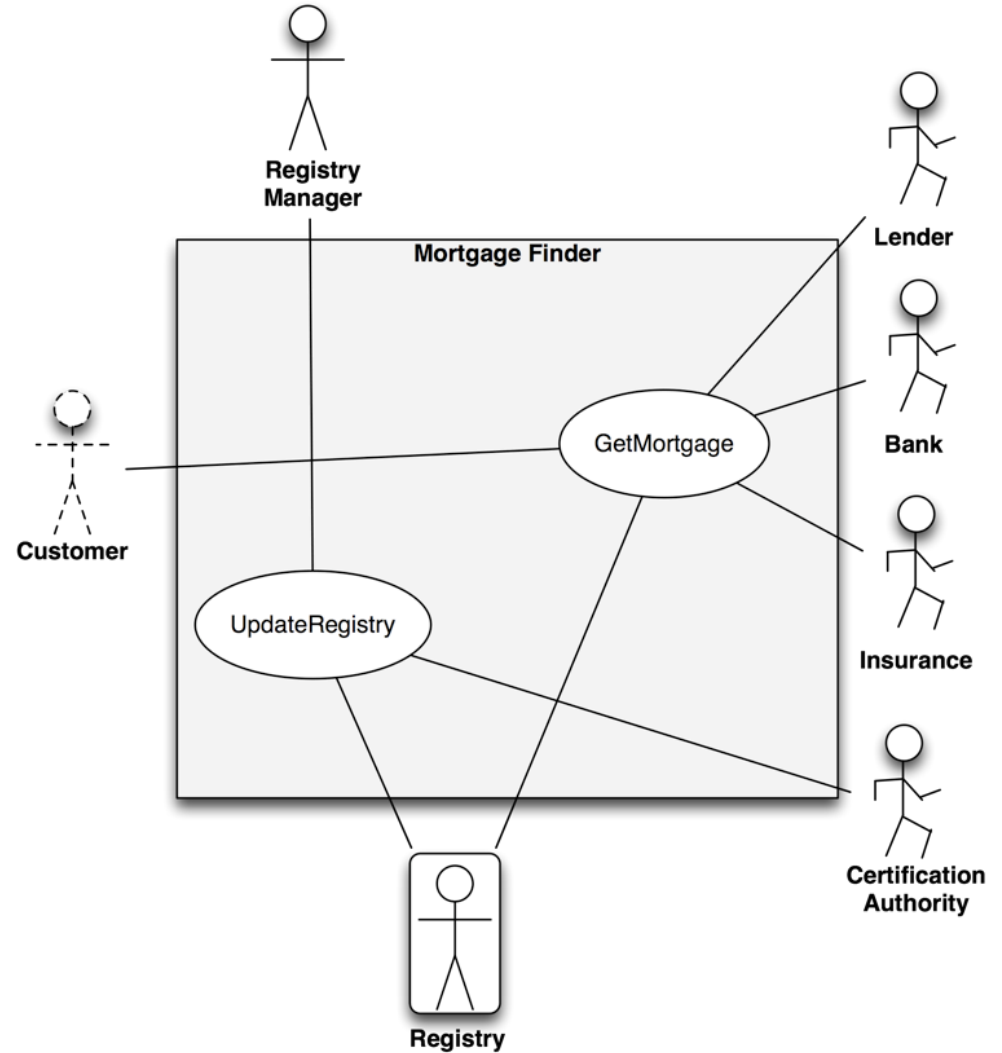


```
<?xml version="1.0" encoding="ASCII"?>
<module xmlns="http://www.osoa.org/xmlns/sca/0.9"
        name="bigbank.accountmodule" >

</module>
```

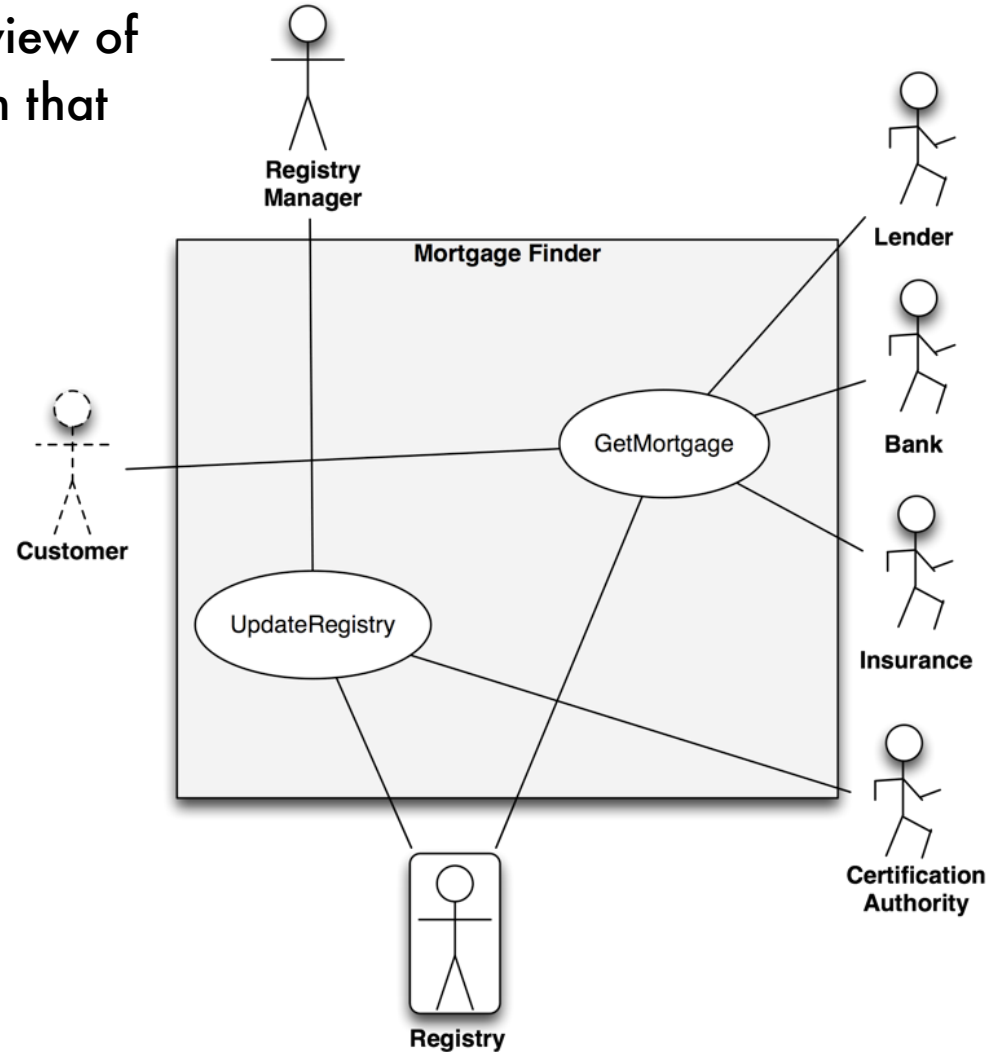
- Although it adopts an SCA-like structure for composite services, SRML is a *modelling* language with a formal semantics that offers descriptions of business logic based on conversational interactions.

use cases for SOC



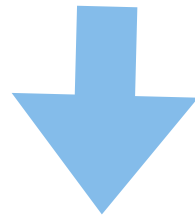
use cases for SOC

- Use Case diagrams give an overview of usage requirements for the system that has to be built

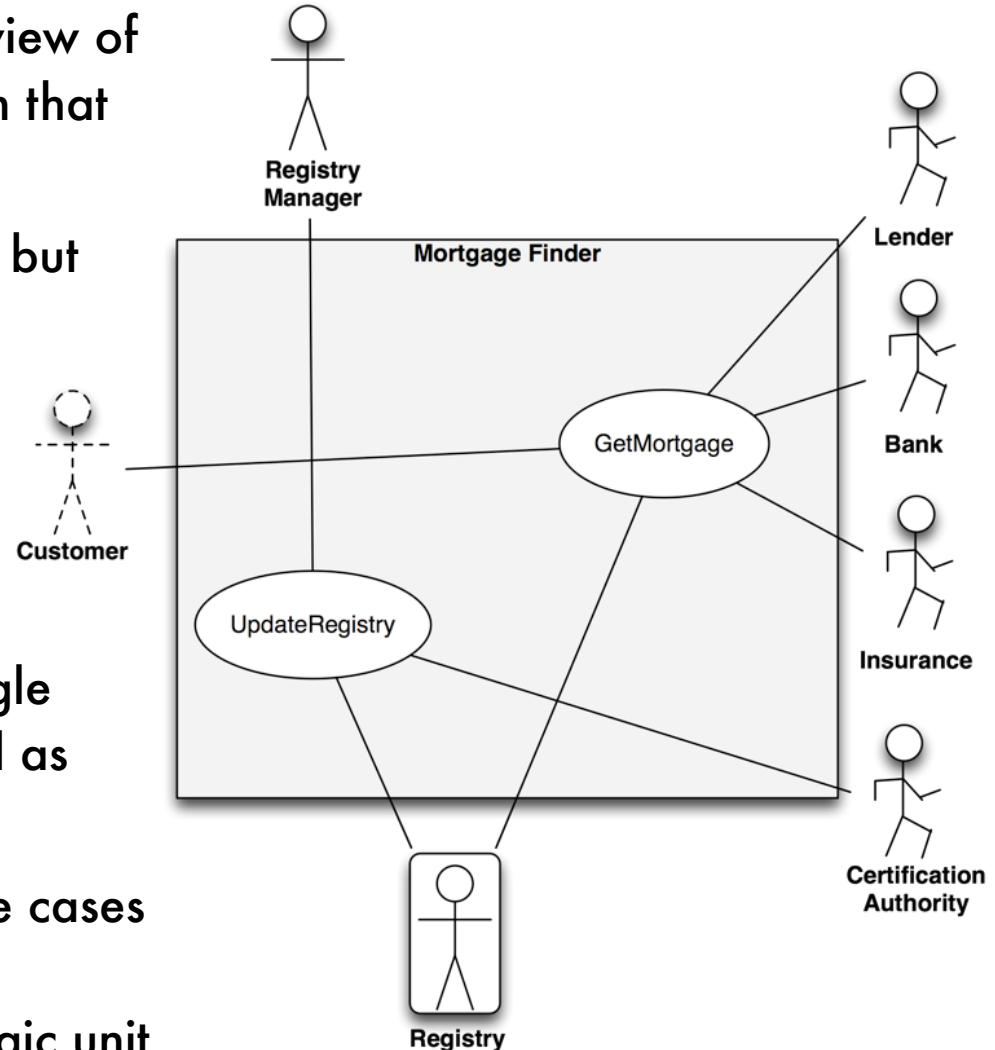


use cases for SOC

- Use Case diagrams give an overview of usage requirements for the system that has to be built
- In SOC we do not build 'systems' but services and activities

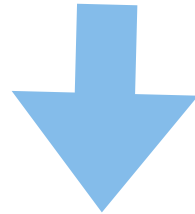


- Each service/activity satisfies a single usage requirement and is modelled as one use case
- The scope includes a number of use cases which are developed by the same company and constitute a single logic unit

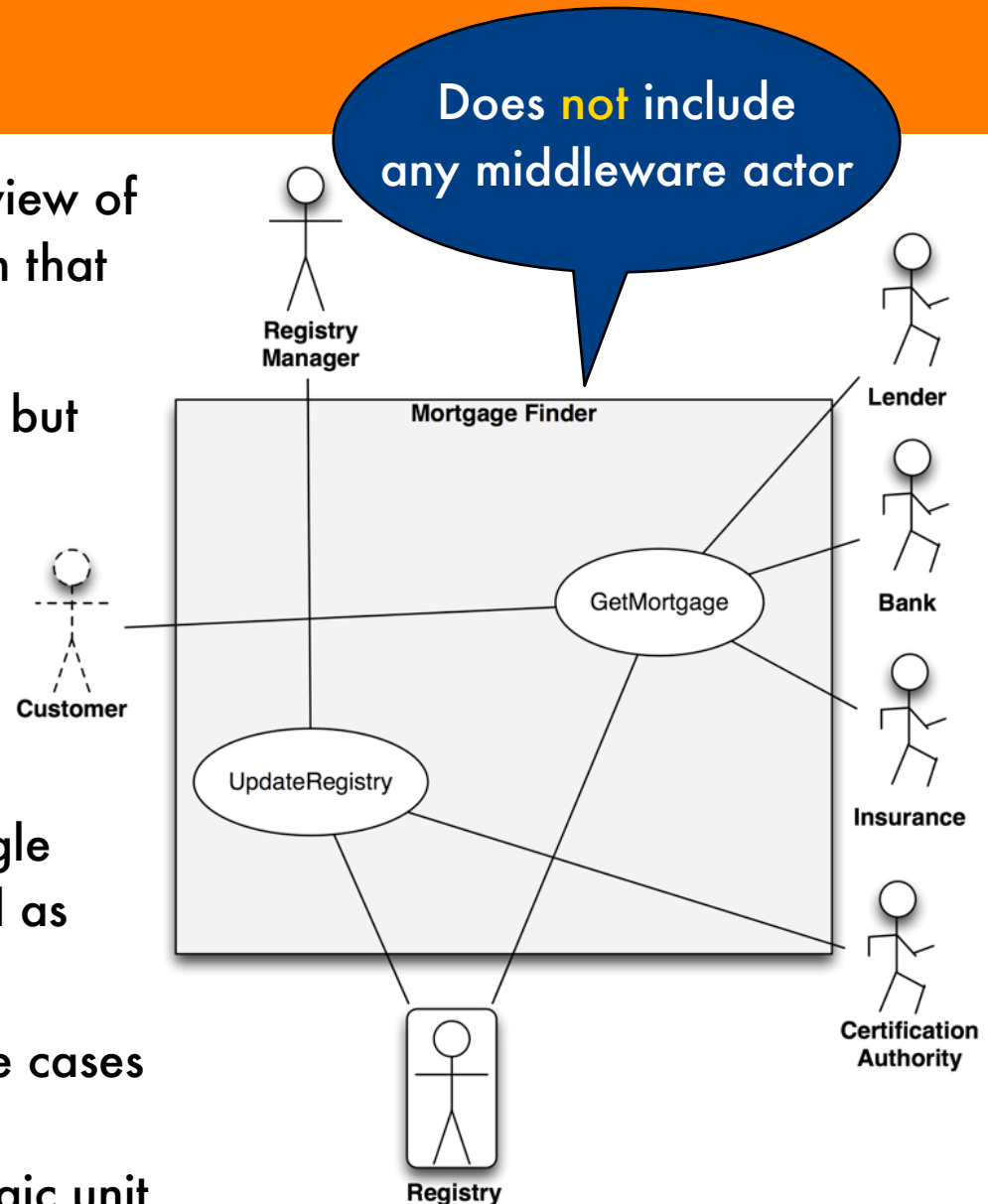


use cases for SOC

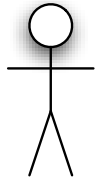
- Use Case diagrams give an overview of usage requirements for the system that has to be built
- In SOC we do not build 'systems' but services and activities



- Each service/activity satisfies a single usage requirement and is modelled as one use case
- The scope includes a number of use cases which are developed by the same company and constitute a single logic unit



actors for SOC



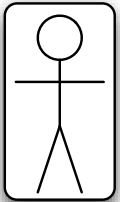
user-actor



requester-actor



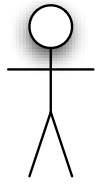
service-actor



resource-actor

actors for SOC

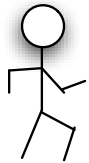
- **Primary Actors** represent entities that initiate the use case and whose goals are fulfilled through the successful completion of the use case



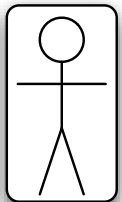
user-actor



requester-actor



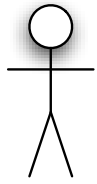
service-actor



resource-actor

actors for SOC

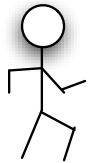
- **Primary Actors** represent entities that initiate the use case and whose goals are fulfilled through the successful completion of the use case
 - **User-actors** instantiate activities (people, machines, ...)



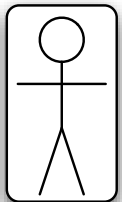
user-actor



requester-actor



service-actor

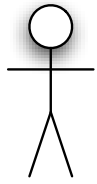


resource-actor

actors for SOC

- **Primary Actors** represent entities that initiate the use case and whose goals are fulfilled through the successful completion of the use case

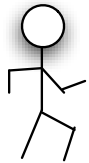
- **User-actors** instantiate activities (people, machines, ...)
- **Requester-actors** are service consumers that trigger the discovery/instantiation of services



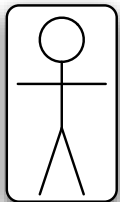
user-actor



requester-actor

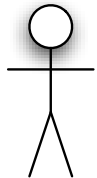


service-actor



resource-actor

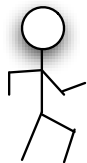
actors for SOC



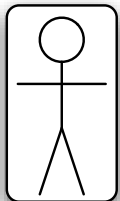
user-actor



requester-actor



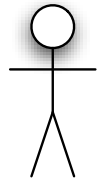
service-actor



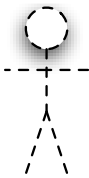
resource-actor

- **Primary Actors** represent entities that initiate the use case and whose goals are fulfilled through the successful completion of the use case
 - **User-actors** instantiate activities (people, machines, ...)
 - **Requester-actors** are service consumers that trigger the discovery/instantiation of services
- **Supporting Actors** represent external entities that need to be relied upon in order to achieve the underlying business goal

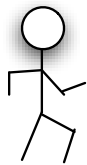
actors for SOC



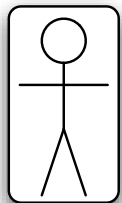
user-actor



requester-actor



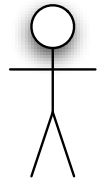
service-actor



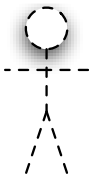
resource-actor

- **Primary Actors** represent entities that initiate the use case and whose goals are fulfilled through the successful completion of the use case
 - **User-actors** instantiate activities (people, machines, ...)
 - **Requester-actors** are service consumers that trigger the discovery/instantiation of services
- **Supporting Actors** represent external entities that need to be relied upon in order to achieve the underlying business goal
 - **Service-actors** represent functionalities to be procured on the fly (typically, the provider varies from instance to instance)

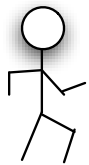
actors for SOC



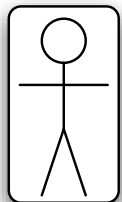
user-actor



requester-actor



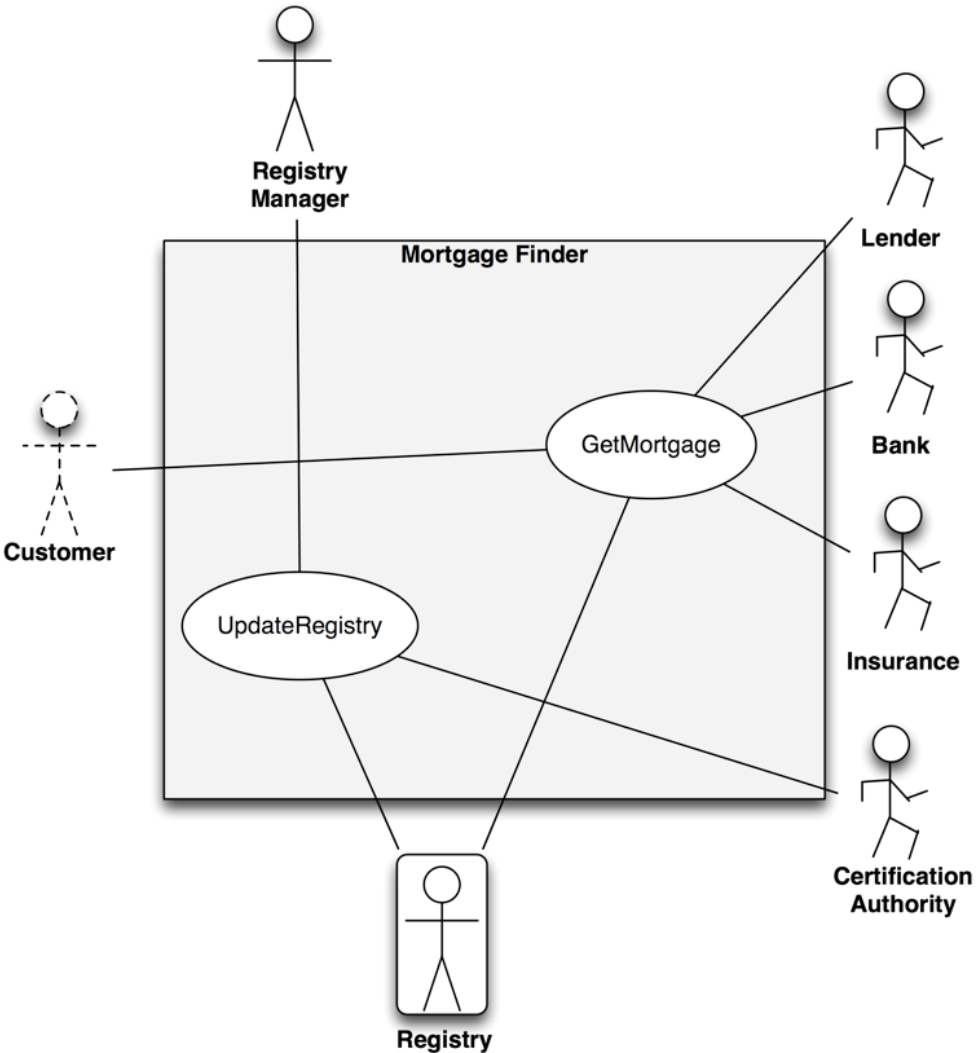
service-actor



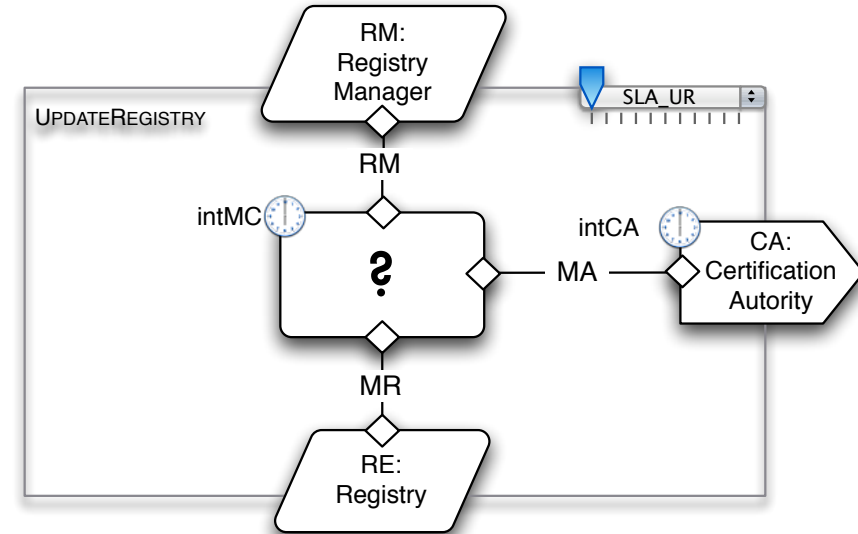
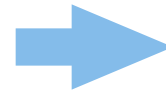
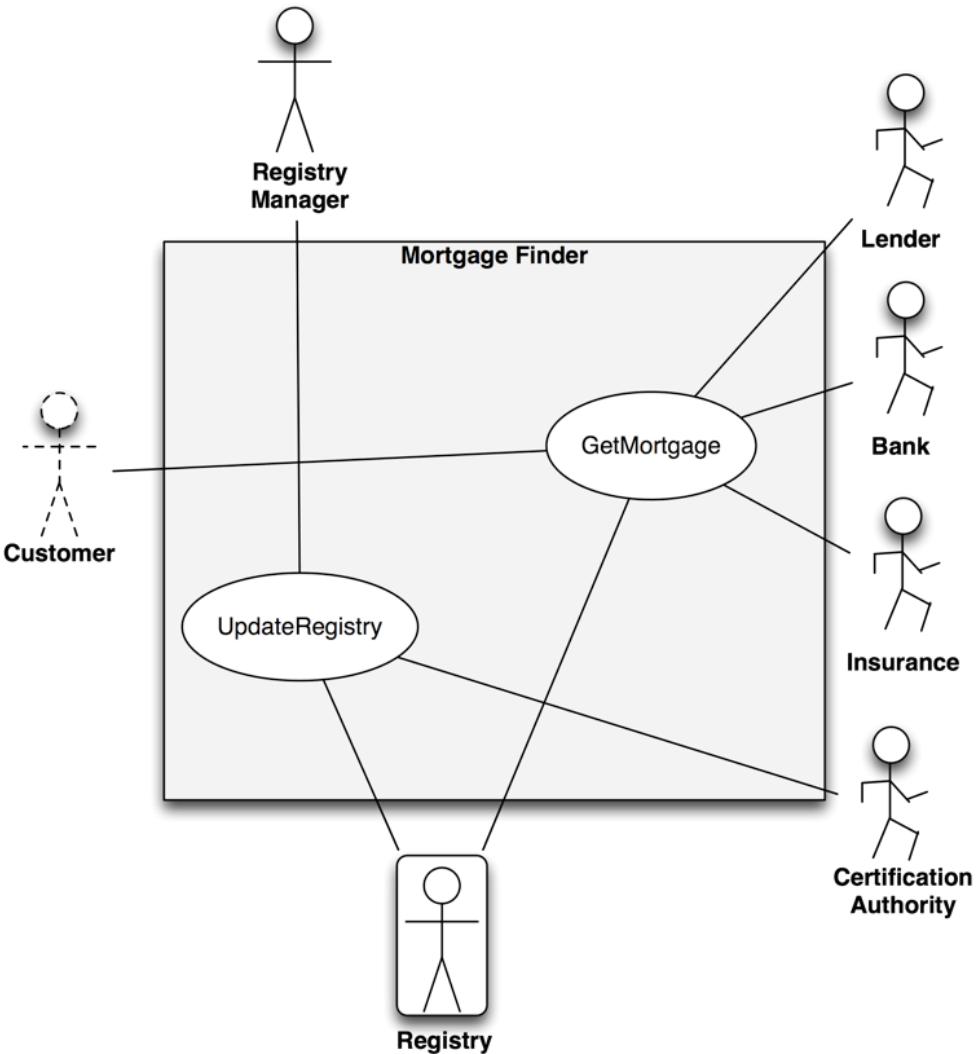
resource-actor

- **Primary Actors** represent entities that initiate the use case and whose goals are fulfilled through the successful completion of the use case
 - **User-actors** instantiate activities (people, machines, ...)
 - **Requester-actors** are service consumers that trigger the discovery/instantiation of services
- **Supporting Actors** represent external entities that need to be relied upon in order to achieve the underlying business goal
 - **Service-actors** represent functionalities to be procured on the fly (typically, the provider varies from instance to instance)
 - **Resource-actors** are statically bound and persistent (they are the same for all instances)

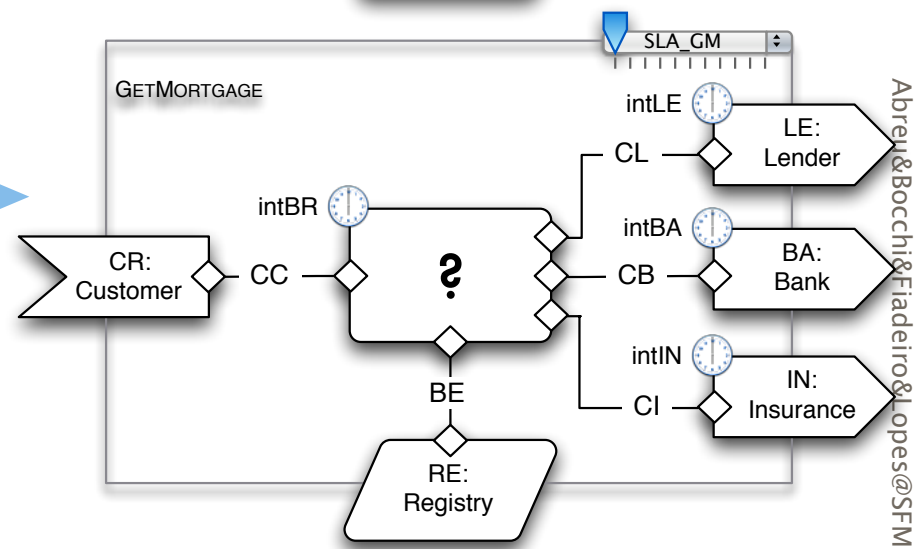
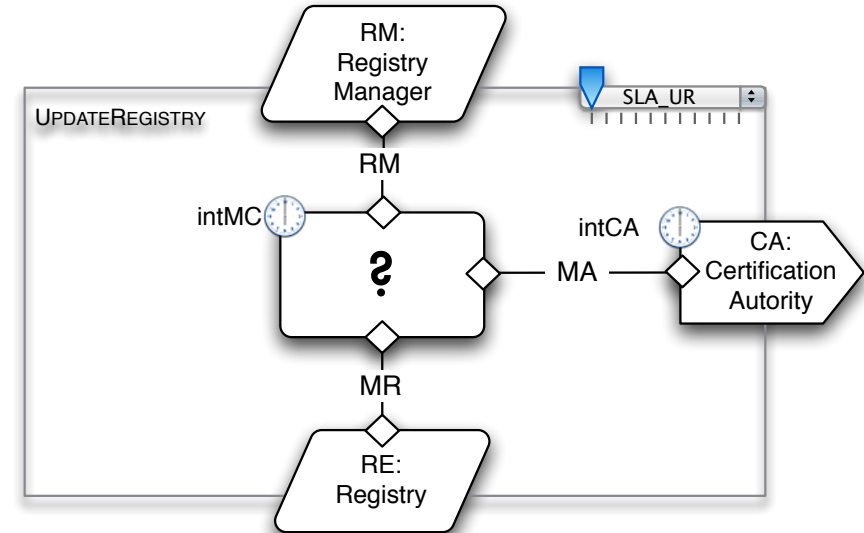
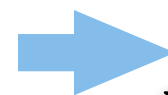
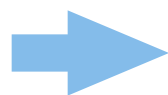
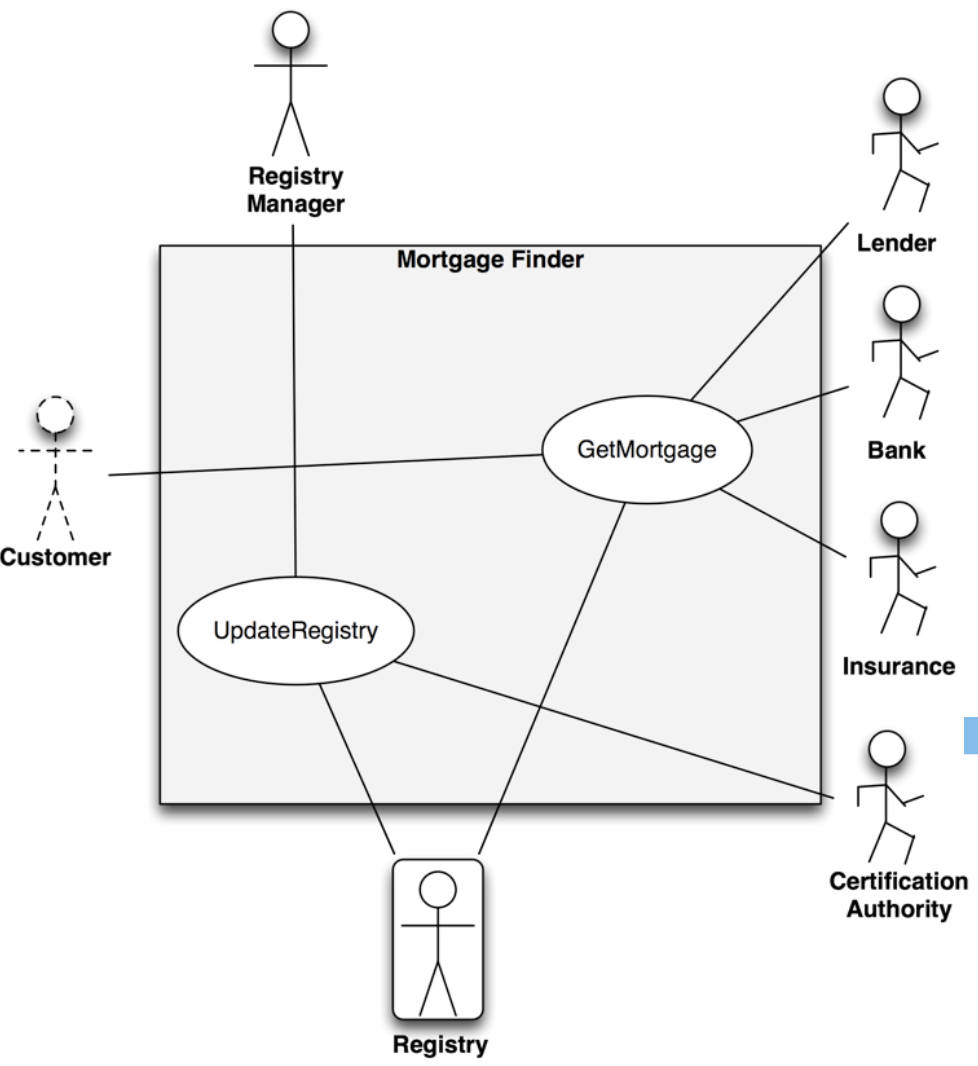
from use case diagrams to SRML



from use case diagrams to SRML

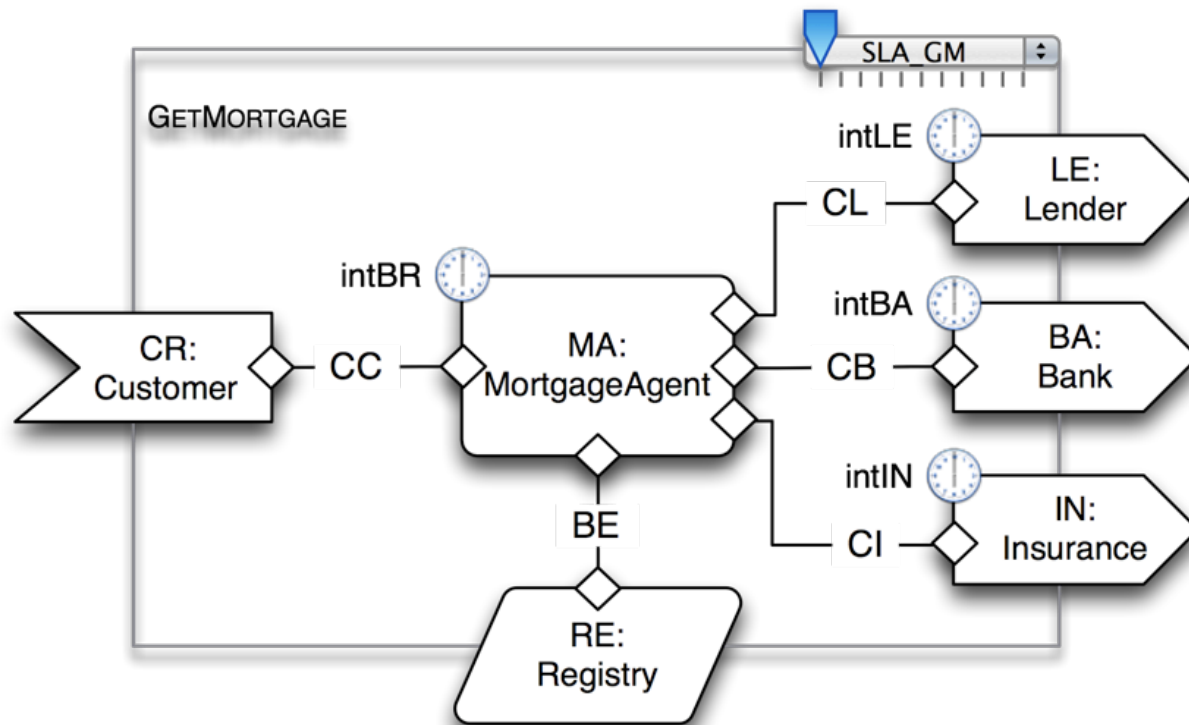


from use case diagrams to SRML

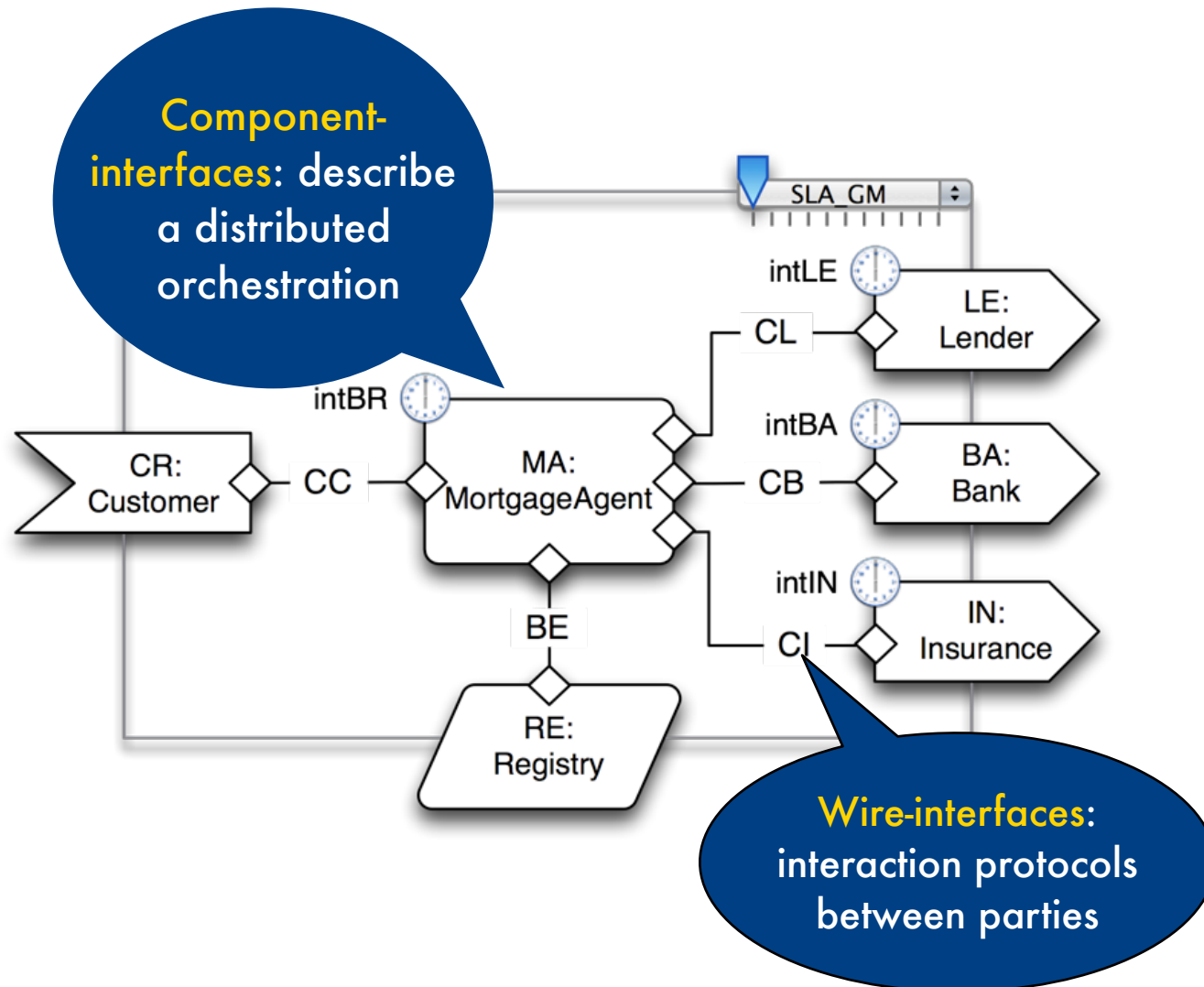


SRML service modules

Service modules model (possibly composite) services that can be published. Their discovery is triggered by a requester-actor.



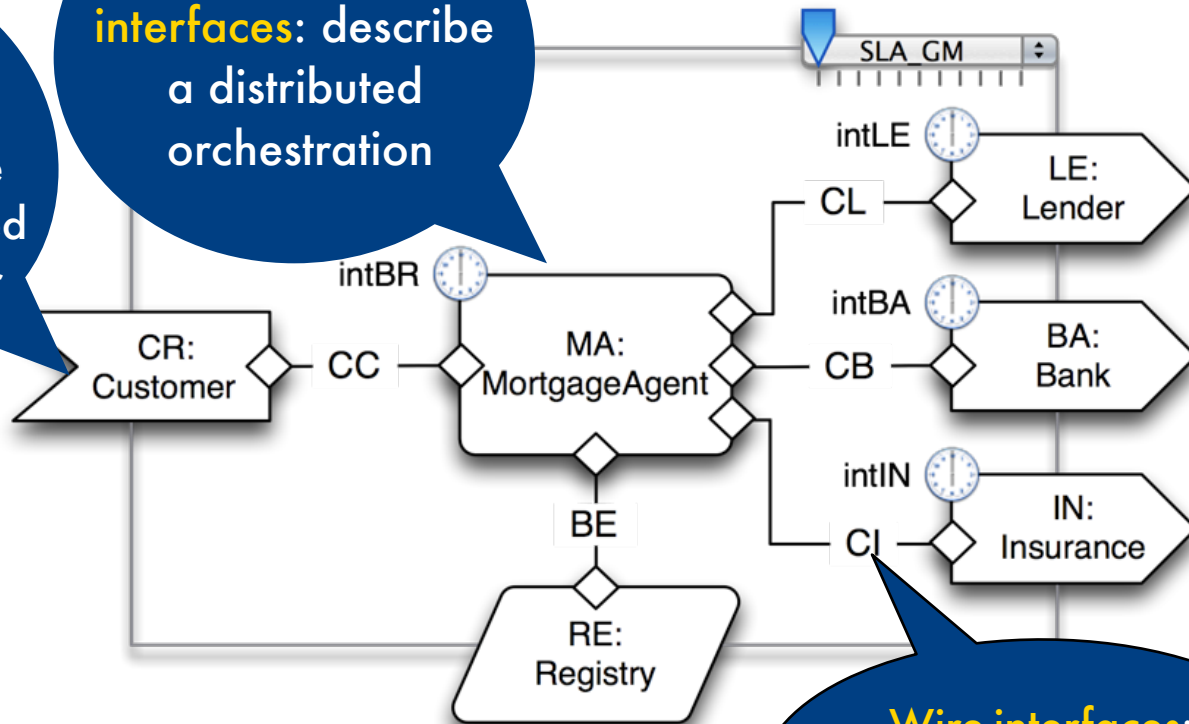
SRML service modules



SRML service modules

Provides-interface: a description of the properties provided to the requester

Component-interfaces: describe a distributed orchestration



Wire-interfaces: interaction protocols between parties

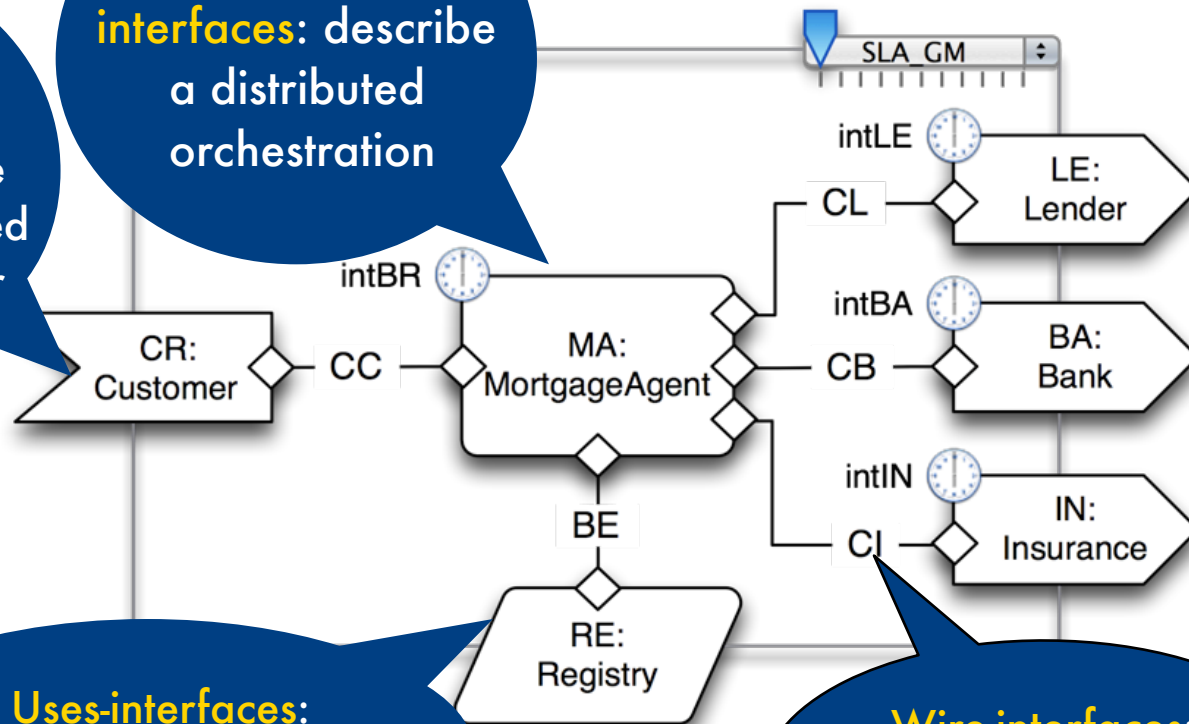
SRML service modules

Provides-interface: a description of the properties provided to the requester

Component-interfaces: describe a distributed orchestration

Uses-interfaces: statically bound to persistent resources

Wire-interfaces: interaction protocols between parties



SRML service modules

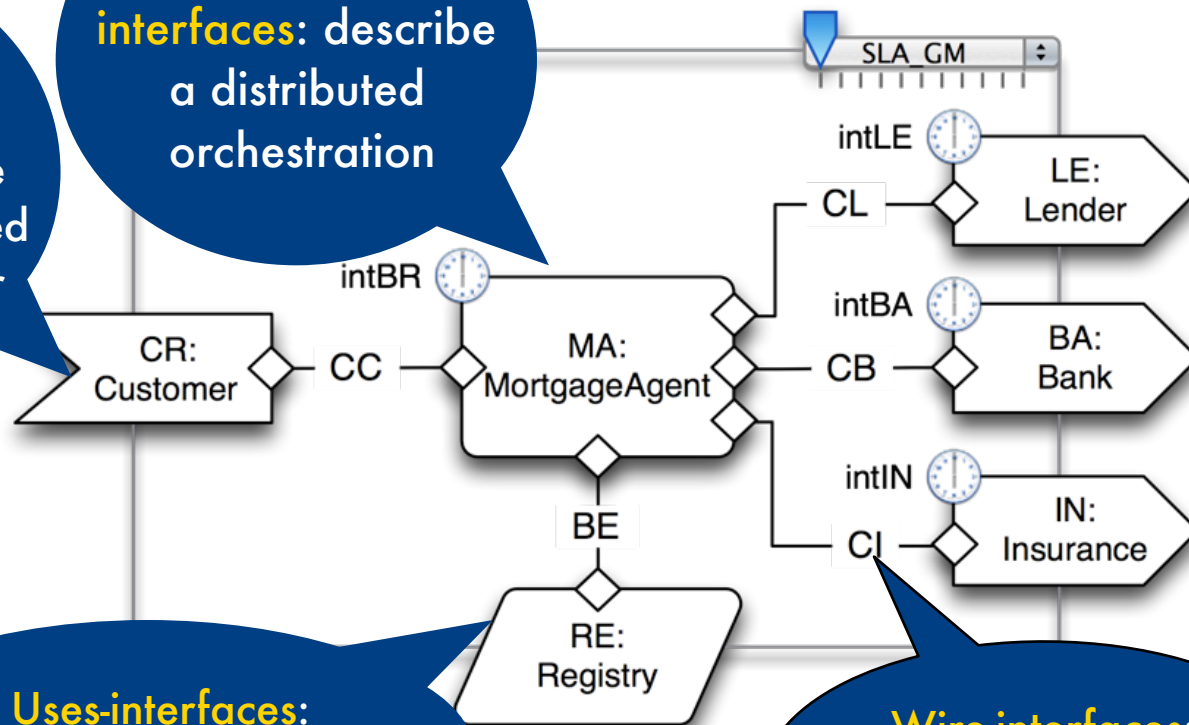
Provides-interface: a description of the properties provided to the requester

Component-interfaces: describe a distributed orchestration

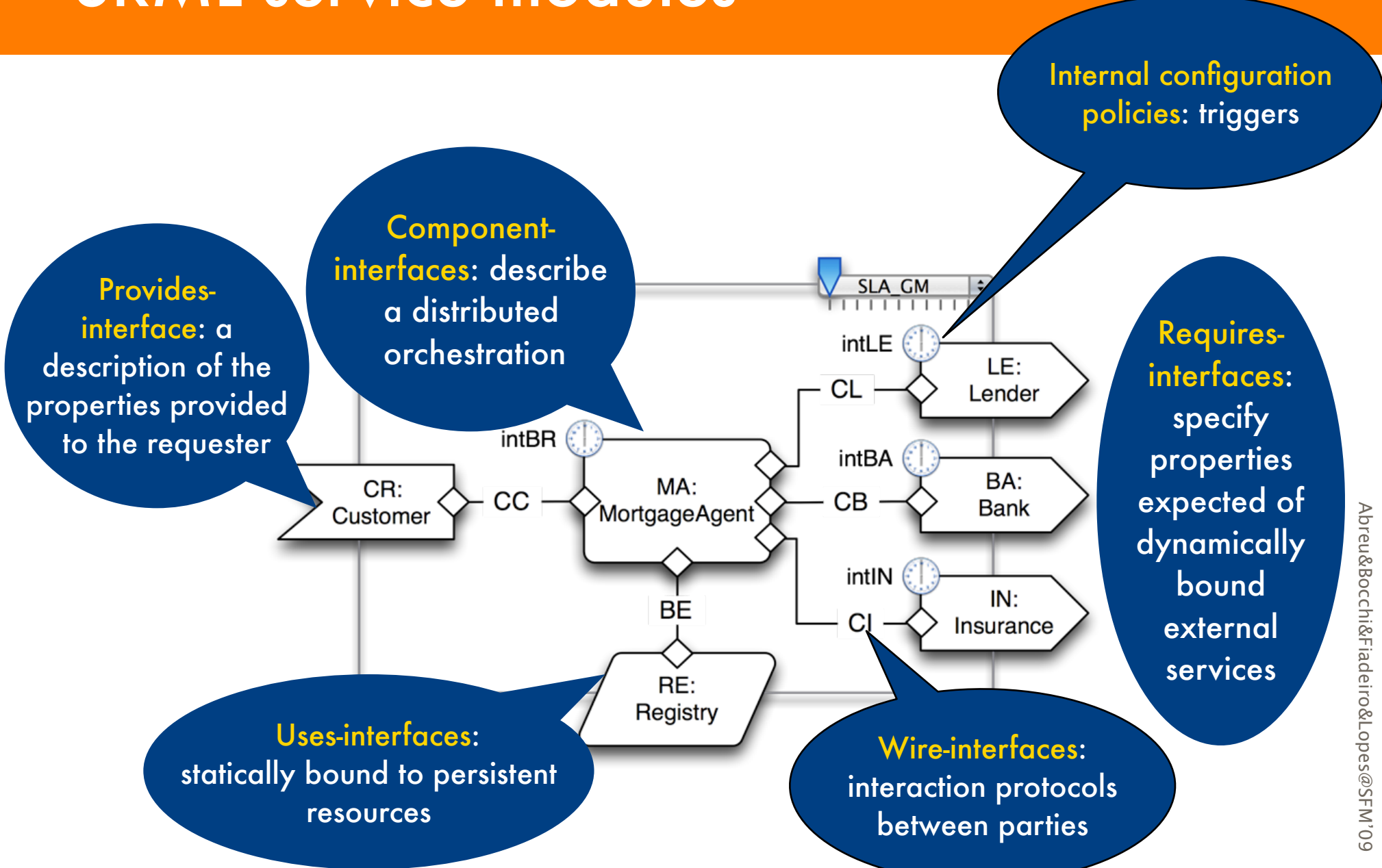
Uses-interfaces: statically bound to persistent resources

Wire-interfaces: interaction protocols between parties

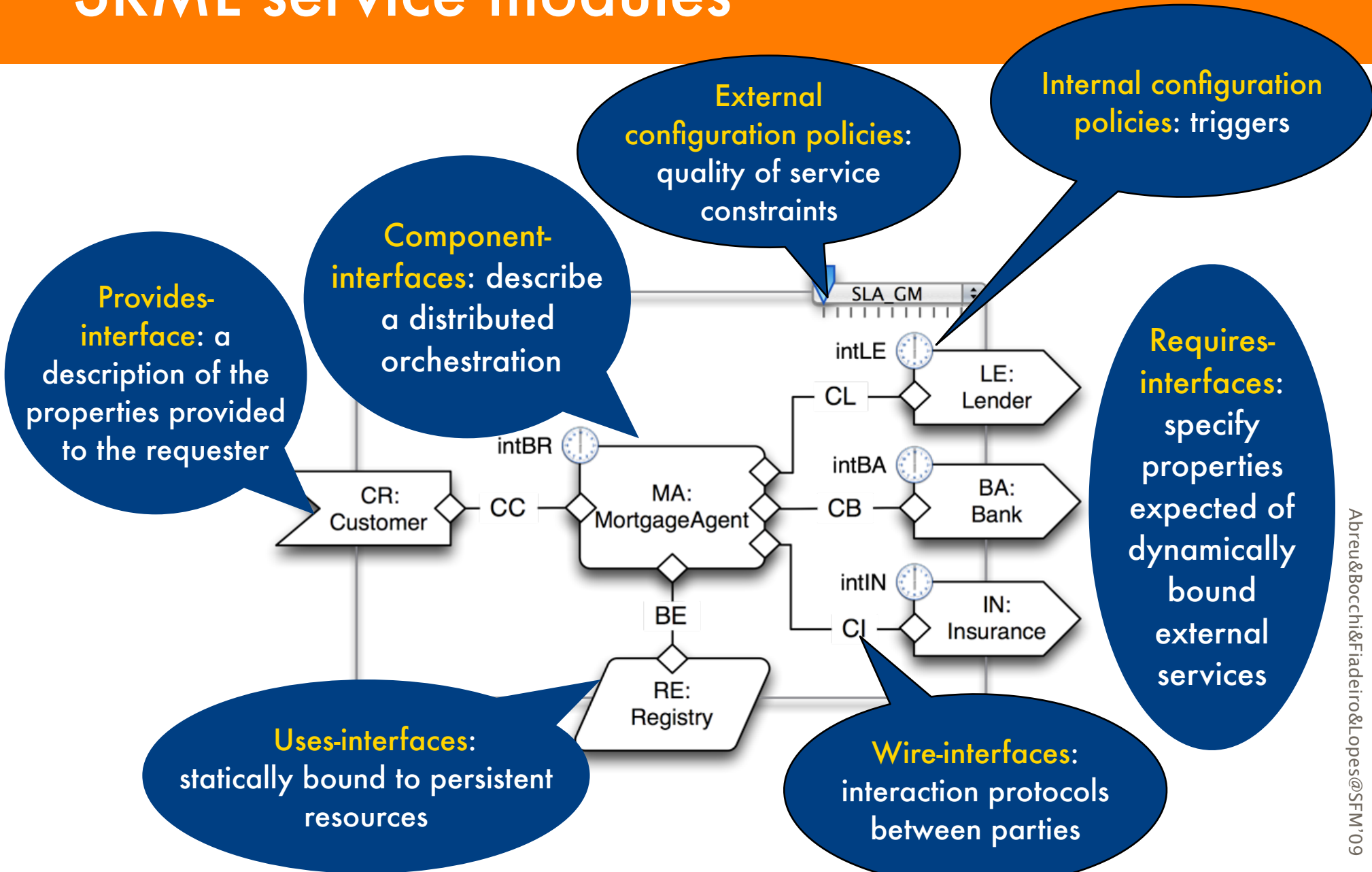
Requires-interfaces: specify properties expected of dynamically bound external services



SRML service modules



SRML service modules



interaction types

interaction types

r&s	stateful, 2-way asynchronous	The interaction is initiated by the co-party, which expects a reply. The co-party does not block while waiting for the reply.
s&r	stateful, 2-way asynchronous	The interaction is initiated by the party and expects a reply from its co-party. While waiting for the reply, the party does not block.

interaction types

r&s	stateful, 2-way asynchronous	The interaction is initiated by the co-party, which expects a reply. The co-party does not block while waiting for the reply.
s&r	stateful, 2-way asynchronous	The interaction is initiated by the party and expects a reply from its co-party. While waiting for the reply, the party does not block.
rcv	one-way asynchronous	The co-party initiates the interaction and does not expect a reply.
snd	one-way asynchronous	The party initiates the interaction and does not expect a reply.

interaction types

r&s	stateful, 2-way asynchronous	The interaction is initiated by the co-party, which expects a reply. The co-party does not block while waiting for the reply.
s&r	stateful, 2-way asynchronous	The interaction is initiated by the party and expects a reply from its co-party. While waiting for the reply, the party does not block.
rcv	one-way asynchronous	The co-party initiates the interaction and does not expect a reply.
snd	one-way asynchronous	The party initiates the interaction and does not expect a reply.
ask	synchronous	The party synchronises with the co-party to obtain data.
rpl	synchronous	The party synchronises with the co-party to transmit data

interaction types

r&s	stateful, 2-way asynchronous	The interaction is initiated by the co-party, which expects a reply. The co-party does not block while waiting for the reply.
s&r	stateful, 2-way asynchronous	The interaction is initiated by the party and expects a reply from its co-party. While waiting for the reply, the party does not block.
rcv	one-way asynchronous	The co-party initiates the interaction and does not expect a reply.
snd	one-way asynchronous	The party initiates the interaction and does not expect a reply.
ask	synchronous	The party synchronises with the co-party to obtain data.
rpl	synchronous	The party synchronises with the co-party to transmit data
tll	synchronous	The party requests the co-party to perform an operation and blocks.
prf	synchronous	The party performs an operation and frees the co-party that requested it.

events associated with an interaction a

a 🔔: the event of **initiating** a

a ✉️: the **reply**-event of a

a ✓: the **commit**-event of a

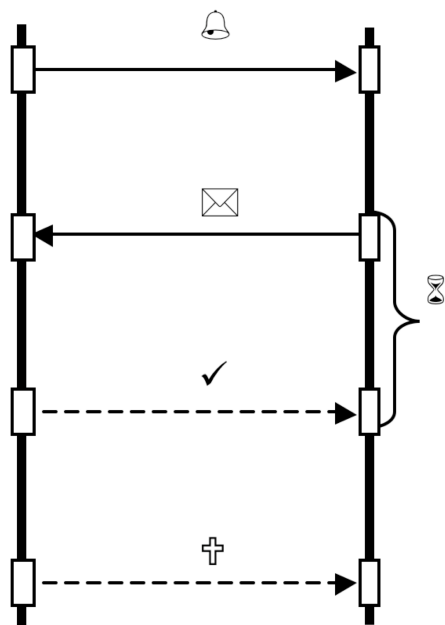
a ✖️: the **cancel**-event of a

a †: the **revoke**-event of a

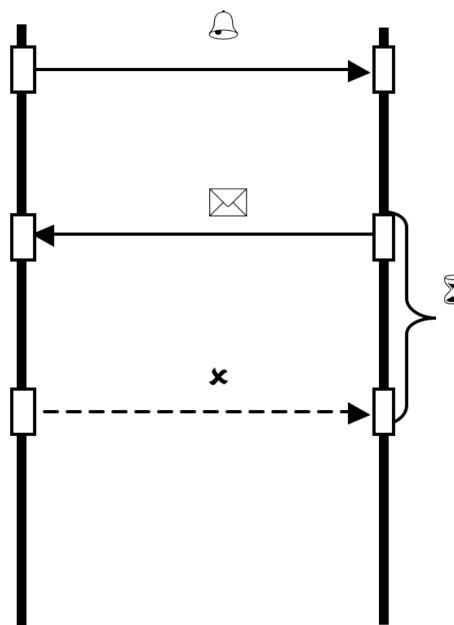
a ⌚: the **pledge** associated with a – a condition that is guaranteed to hold from the moment a positive reply-event occurs until either the commit-event, the cancel-event or the expiration time occurs.

a 🕒: the **validity interval** associated with the pledge

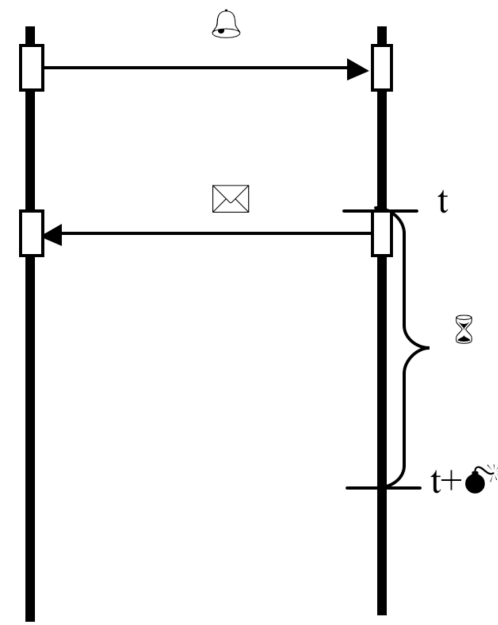
PartyS PartyR



PartyS PartyR

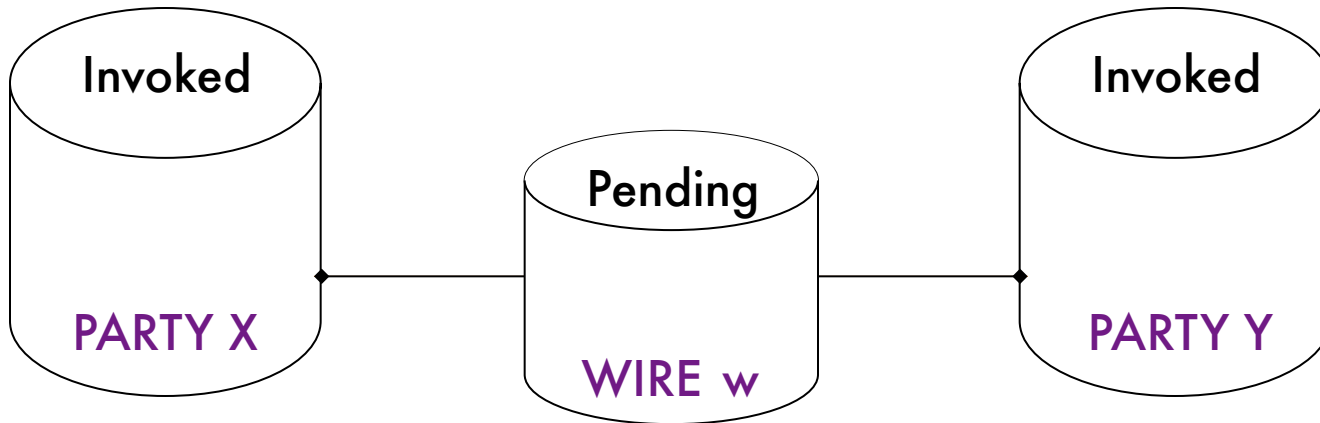


PartyS PartyR



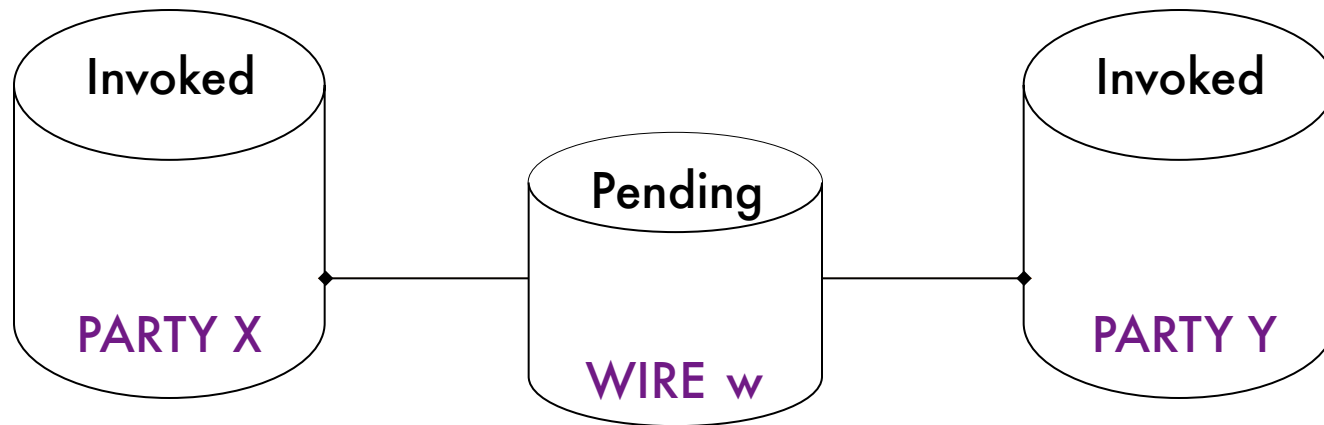
a computational model

Example: Two-way interaction a from X to Y (connected by w)



a computational model

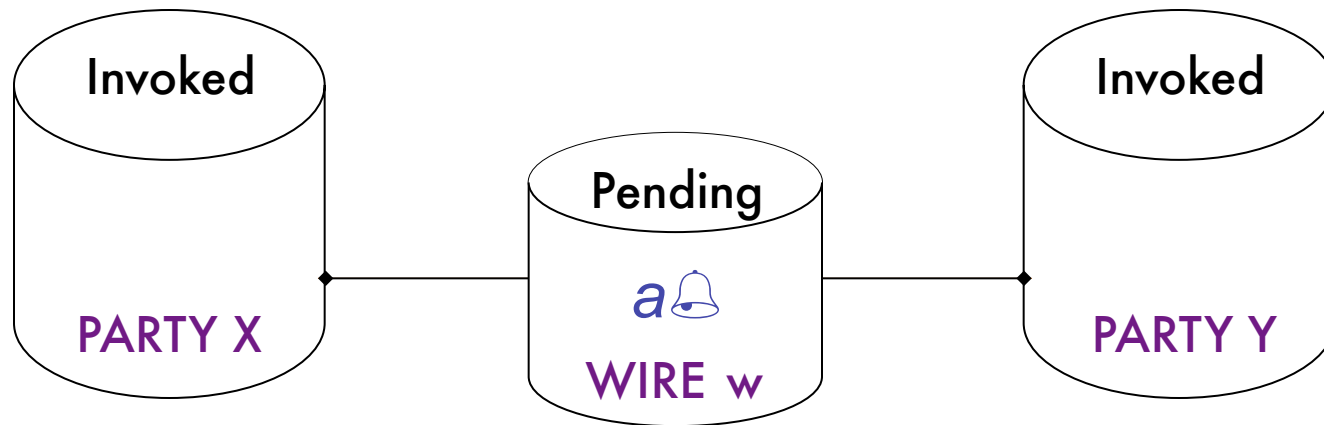
Example: Two-way interaction a from X to Y (connected by w)



- Events occur in state transitions of both parties involved in the interaction
- When a party publishes an event (*event!*), the event is transferred to the buffer of the wire that connects the party with the co-party.
- The wire delivers the event to the co-party, which stores it for processing.
- The co-party can either execute the event (*event?*) or discard it (*event?*)

a computational model

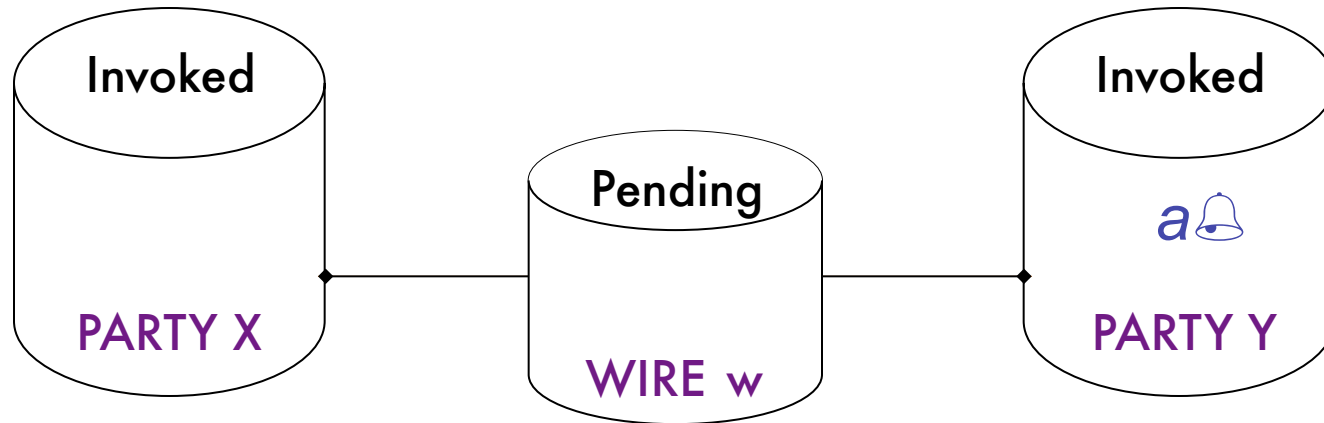
Example: Two-way interaction a from X to Y (connected by w)



- Events occur in state transitions of both parties involved in the interaction
- When a party publishes an event (*event!*), the event is transferred to the buffer of the wire that connects the party with the co-party.
- The wire delivers the event to the co-party, which stores it for processing.
- The co-party can either execute the event (*event?*) or discard it (*event?*)

a computational model

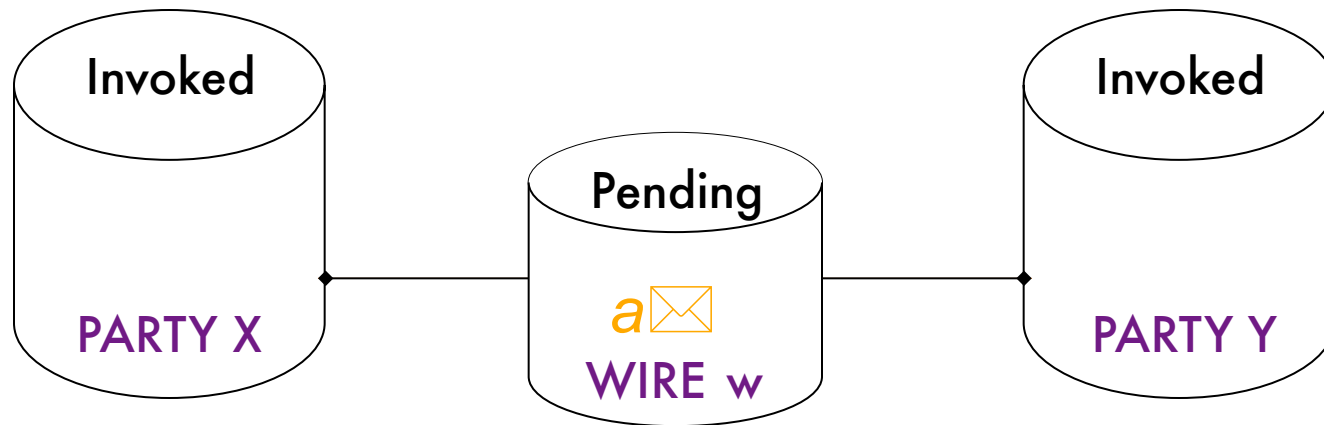
Example: Two-way interaction a from X to Y (connected by w)



- Events occur in state transitions of both parties involved in the interaction
- When a party publishes an event (*event!*), the event is transferred to the buffer of the wire that connects the party with the co-party.
- The wire delivers the event to the co-party, which stores it for processing.
- The co-party can either execute the event (*event?*) or discard it (*event?*)

a computational model

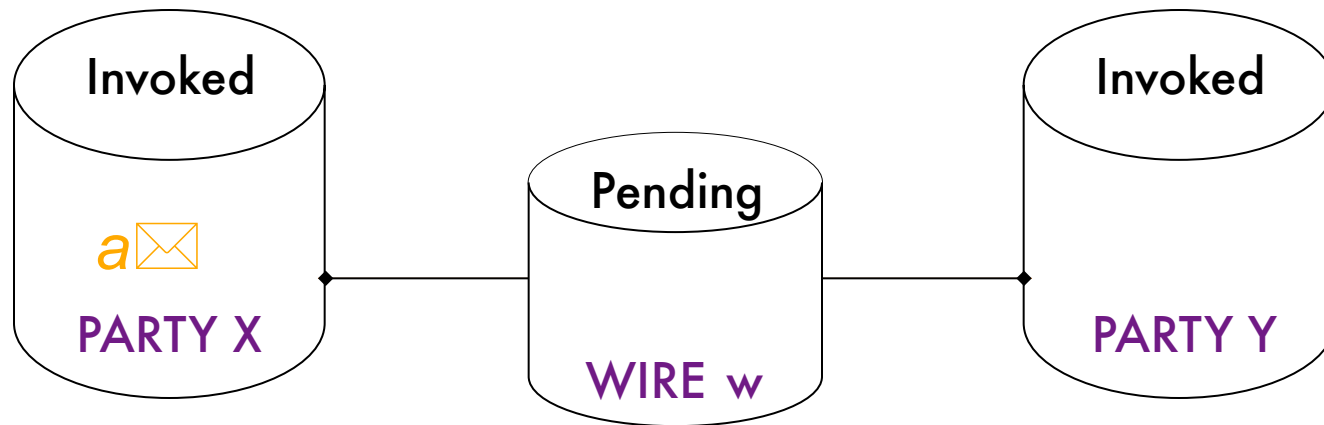
Example: Two-way interaction a from X to Y (connected by w)



- Events occur in state transitions of both parties involved in the interaction
- When a party publishes an event (*event!*), the event is transferred to the buffer of the wire that connects the party with the co-party.
- The wire delivers the event to the co-party, which stores it for processing.
- The co-party can either execute the event (*event?*) or discard it (*event?*)

a computational model

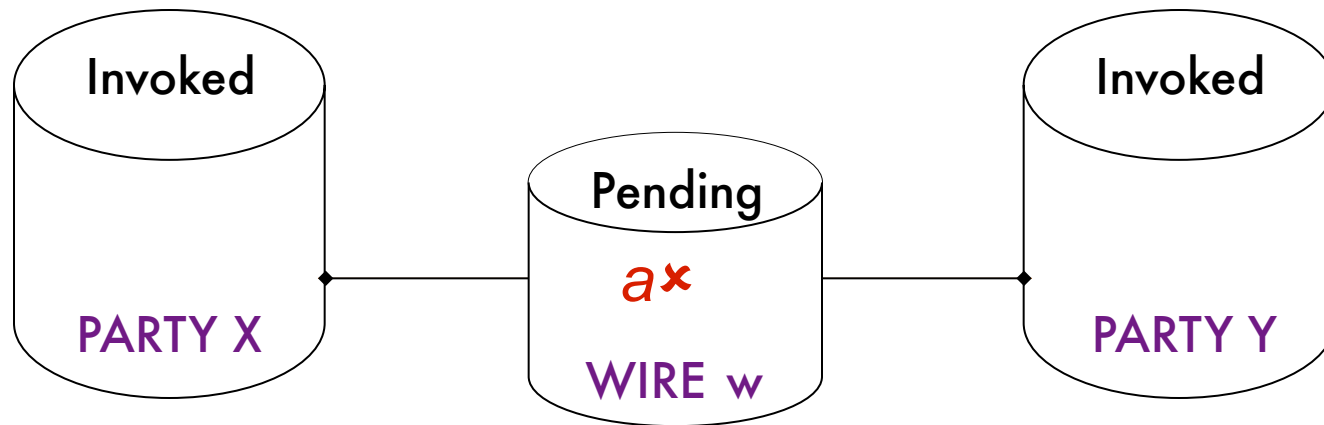
Example: Two-way interaction a from X to Y (connected by w)



- Events occur in state transitions of both parties involved in the interaction
- When a party publishes an event (*event!*), the event is transferred to the buffer of the wire that connects the party with the co-party.
- The wire delivers the event to the co-party, which stores it for processing.
- The co-party can either execute the event (*event?*) or discard it (*event?*)

a computational model

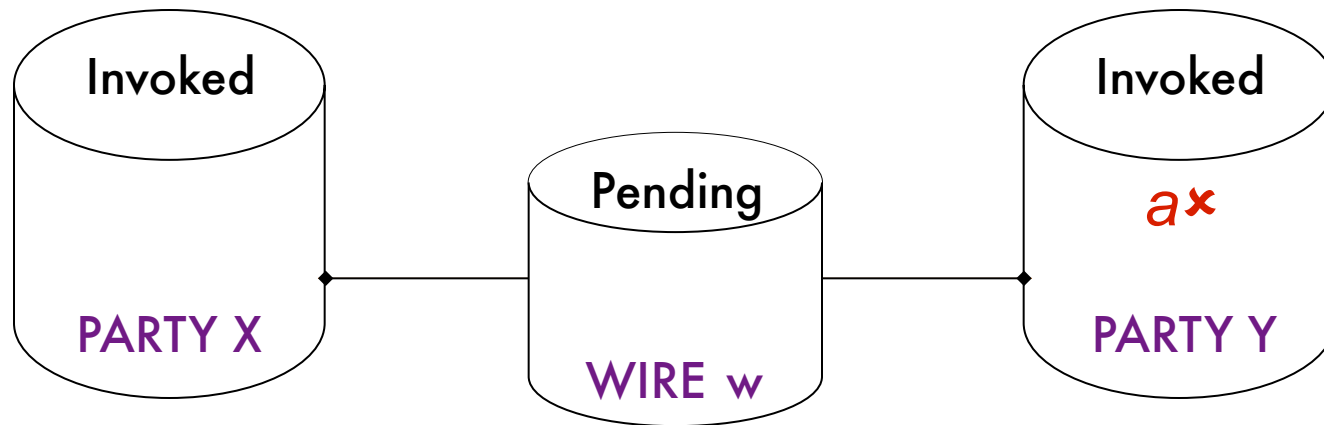
Example: Two-way interaction a from X to Y (connected by w)



- Events occur in state transitions of both parties involved in the interaction
- When a party publishes an event (*event!*), the event is transferred to the buffer of the wire that connects the party with the co-party.
- The wire delivers the event to the co-party, which stores it for processing.
- The co-party can either execute the event (*event?*) or discard it (*event?*)

a computational model

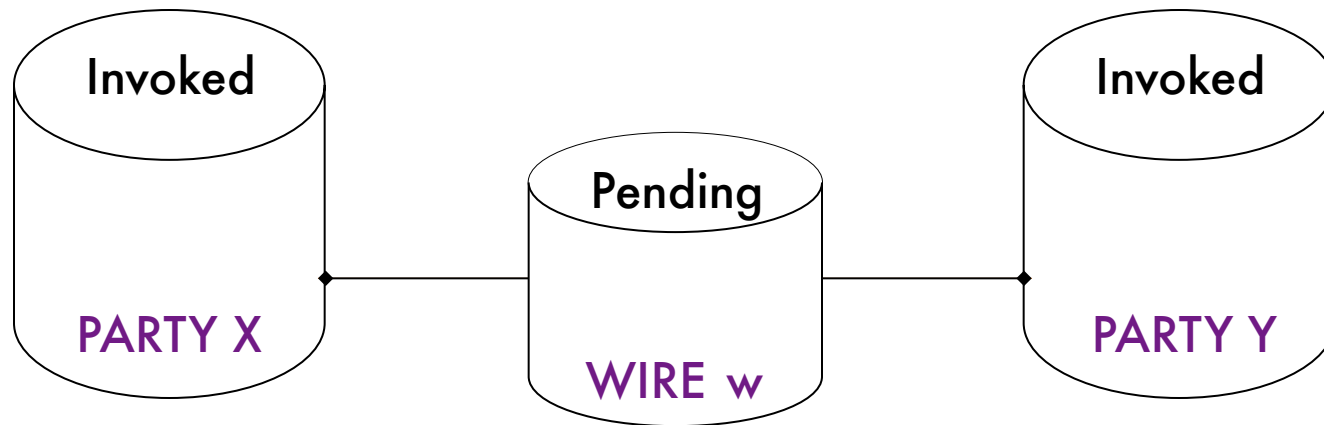
Example: Two-way interaction a from X to Y (connected by w)



- Events occur in state transitions of both parties involved in the interaction
- When a party publishes an event (*event!*), the event is transferred to the buffer of the wire that connects the party with the co-party.
- The wire delivers the event to the co-party, which stores it for processing.
- The co-party can either execute the event (*event?*) or discard it (*event?*)

a computational model

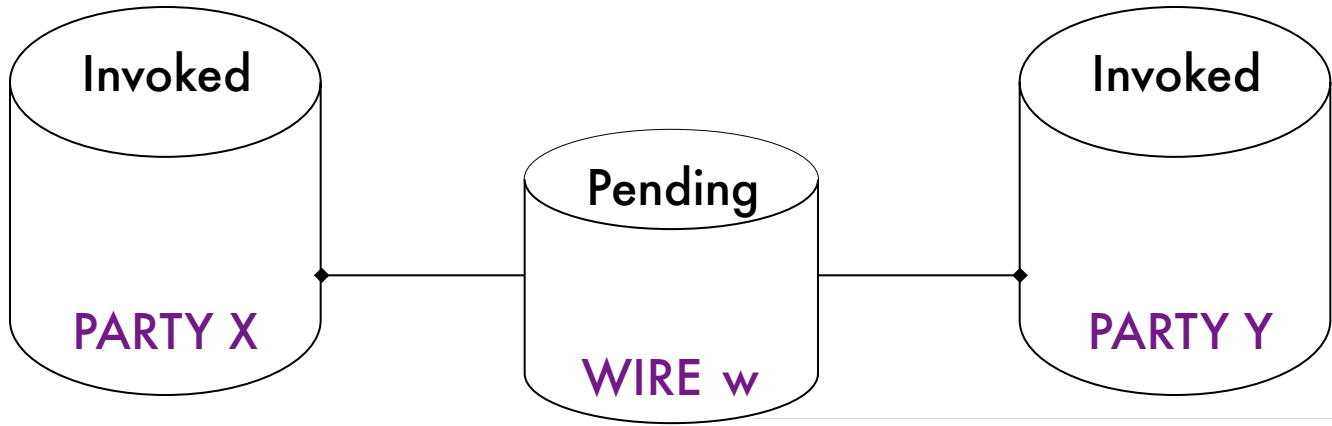
Example: Two-way interaction a from X to Y (connected by w)



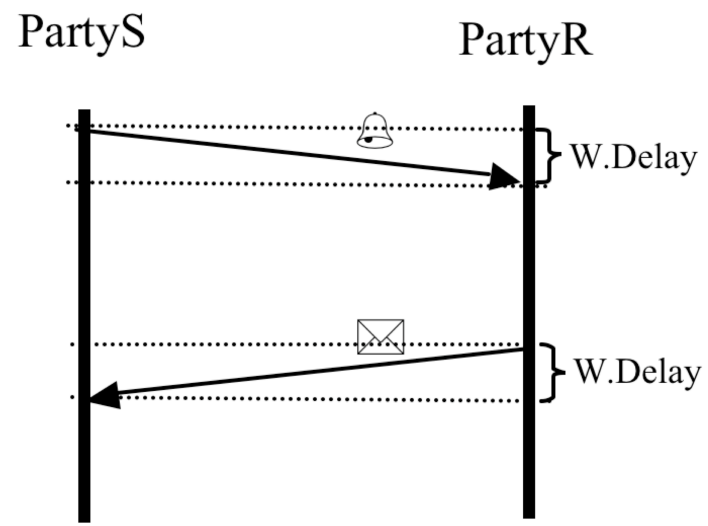
- Events occur in state transitions of both parties involved in the interaction
- When a party publishes an event (*event!*), the event is transferred to the buffer of the wire that connects the party with the co-party.
- The wire delivers the event to the co-party, which stores it for processing.
- The co-party can either execute the event (*event?*) or discard it (*event?*)

a computational model

Example: Two-way interaction a from X to Y (connected by w)

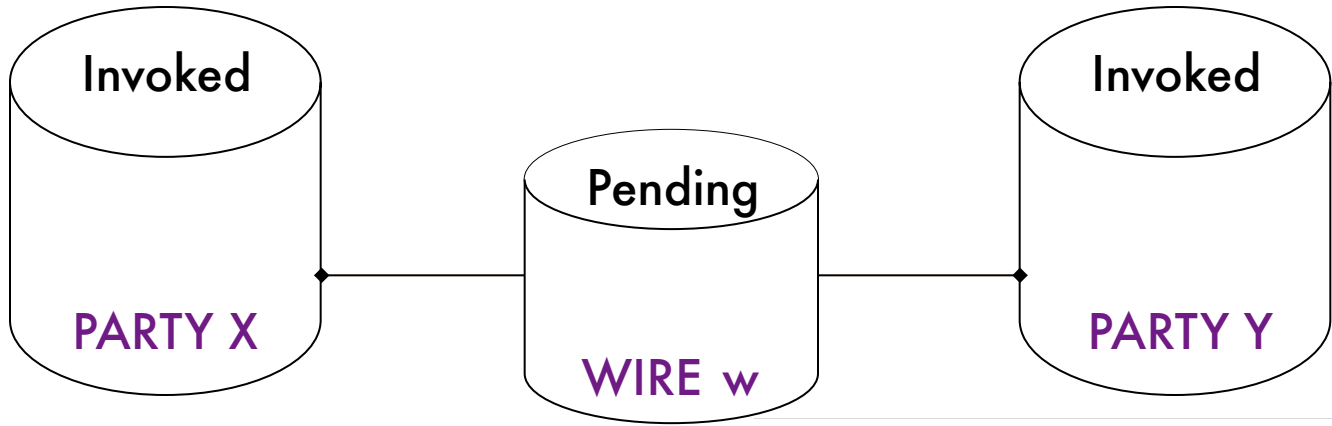


- The occurrence of *event!* and *event?* may not coincide in time



a computational model

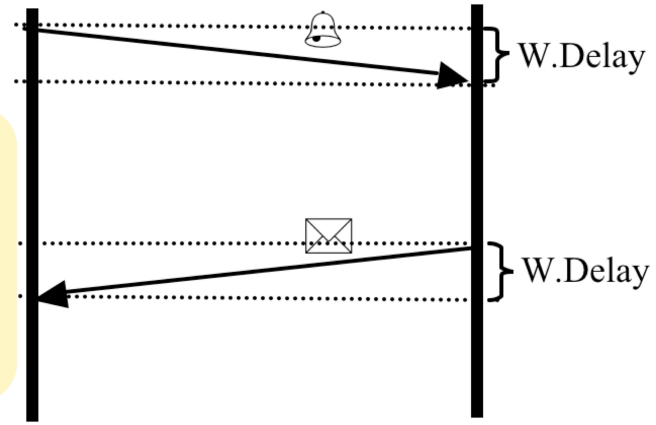
Example: Two-way interaction a from X to Y (connected by w)



PartyS

PartyR

- The occurrence of *event!* and *event?* may not coincide in time



L. Bocchi, J. Fiadeiro, S. Gilmore, J. Abreu, M. Solanki, V. Vankayala (2009) A Formal Model for Timing Aspects of Service-Oriented Systems. Submitted.

computation states

A *computation state* is a tuple $\langle PND, INV, TIME, PLG \rangle$ where:

- *PND* – the set of events that are *pending* in each wire
- *INV* – the set of events that are waiting (*invoked*) to be processed in each component
- *TIME* – the *instant* of time at which the state is observed
- *PLG* – the set of *pledges* that hold in that state

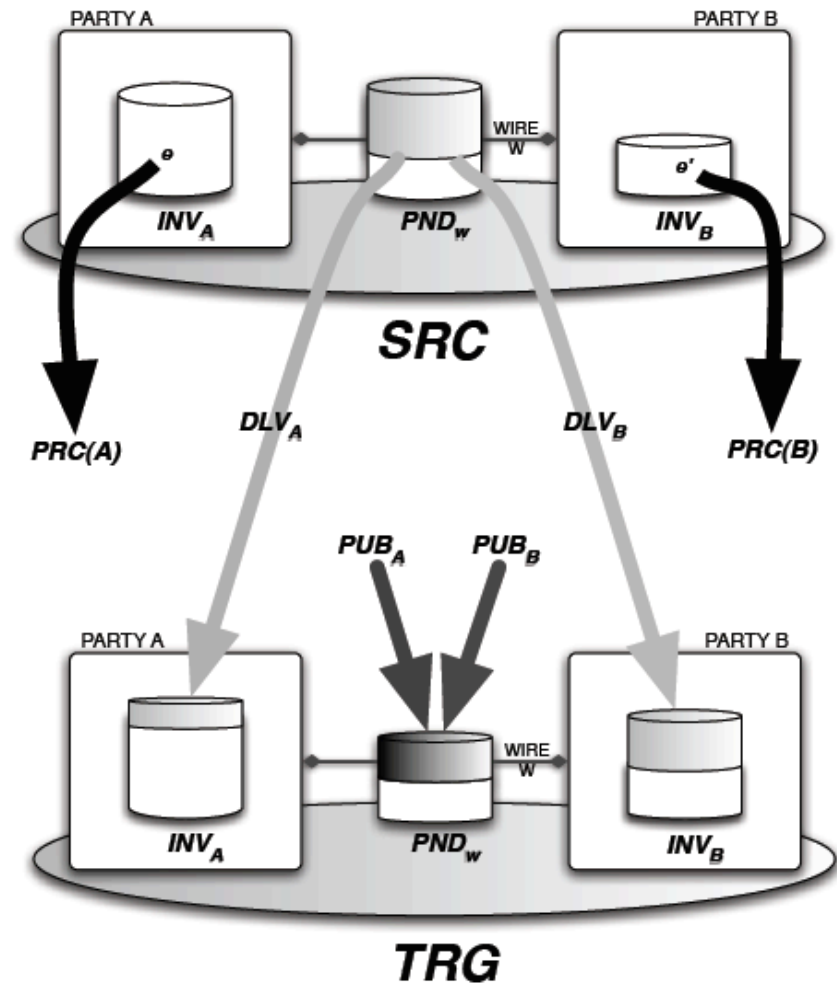
computation steps

A computation step is a tuple $\langle \text{SRC}, \text{TRG}, \text{DLV}, \text{EXC}, \text{DSC}, \text{PUB} \rangle$ where:

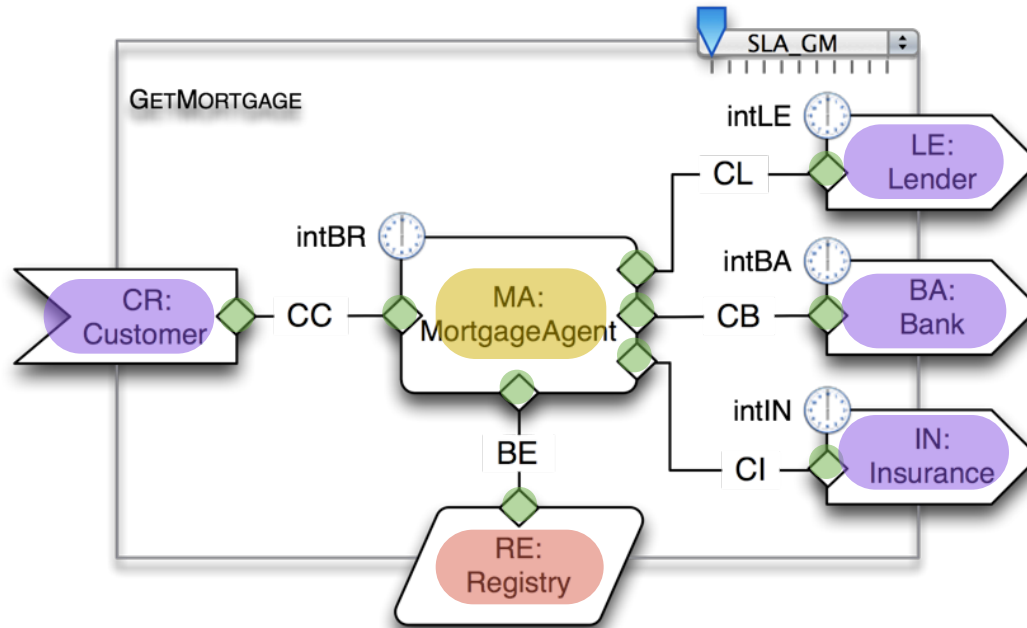
- SRC, TRG – source and the target
- DLV – events that are *delivered*
- EXC – events that are *executed*
- DSC – events that are *discarded*
- PUB – events that are *published*

furthermore

- $\text{PRC} = \text{EXC} + \text{DSC}$ – events that are processed
- $\text{PND}^{\text{TRG}} = (\text{PND}^{\text{SRC}} \setminus \text{DLV}) \cup \text{PUB}$
- $\text{INV}^{\text{TRG}} = (\text{INV}^{\text{SRC}} \setminus \text{PRC}) \cup \text{DLV}$



the Languages of SRML



Business
Roles

Interactions
+
Orchestration

Business
Protocols

Interactions
+
Behaviour

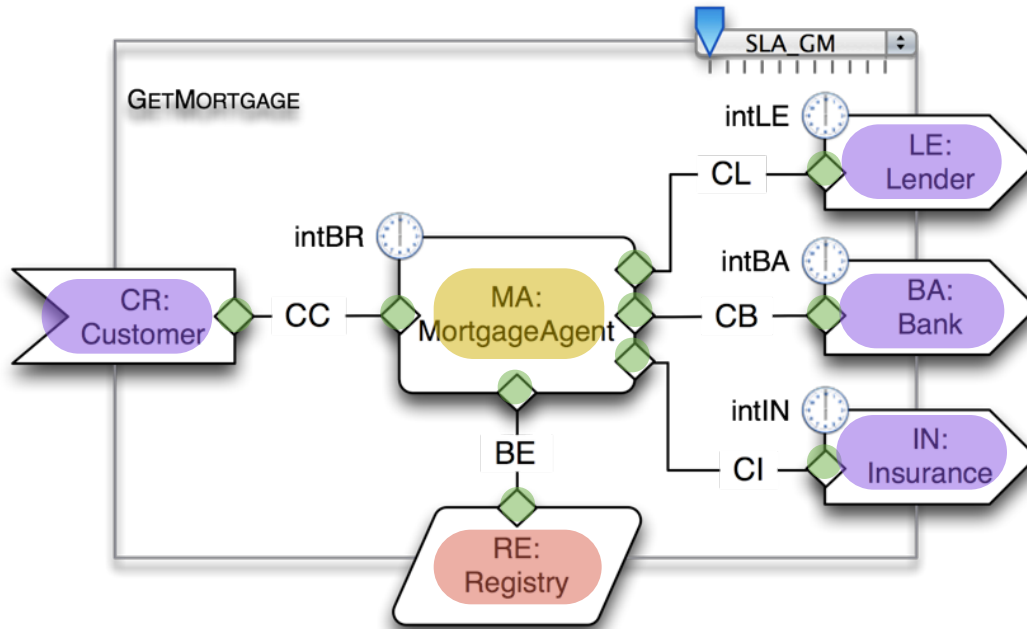
Layer
Protocols

Interactions
+
Behaviour

Interaction
Protocols

Interactions
+
Coordination

the Languages of SRML



**Business
Roles**

**Business
Protocols**

**Layer
Protocols**

**Interaction
Protocols**

Interactions
+
Orchestration

Interactions
+
Behaviour

Interactions
+
Behaviour

Interactions
+
Coordination

logics of interactions

signatures

signatures

- Each party defines a signature – the interactions in which it can be involved

signatures

- Each party defines a signature – the interactions in which it can be involved
- For example, the signature of the business role *MortgageAgent* is defined as follows

```
INTERACTIONS  
r&s getProposal  
  🔔 idData:usrdata,  
    income: moneyvalue,  
    preferences:prefdata  
  ✉ proposal:mortgageproposal,  
    cost:moneyvalue  
r&s askProposal  
  🔔 idData:usrdata,  
    income: moneyvalue,  
  ✉ proposal:mortgageproposal,  
    loanData:loandata,  
    accountIncluded:bool,  
    insuranceRequired:bool  
...  
snd confirmation  
  🔔 contract:loancontract  
ask getLenders(prefdata):setids  
tll regContract(loandata,loancontract)
```


signatures

- Each party defines a signature – the interactions in which it can be involved
- For example, the signature of the business role *MortgageAgent* is defined as follows

**stateful
interaction type** ← **INTERACTIONS** → **interaction name**

```
r&s getProposal
  🔔 idData:usrdata,
  income: moneyvalue,
  preferences:prefdata
  ✉ proposal:mortgageproposal,
  cost:moneyvalue

r&s askProposal
  🔔 idData:usrdata,
  income: moneyvalue,
  ✉ proposal:mortgageproposal,
  loanData:loandata,
  accountIncluded:bool,
  insuranceRequired:bool

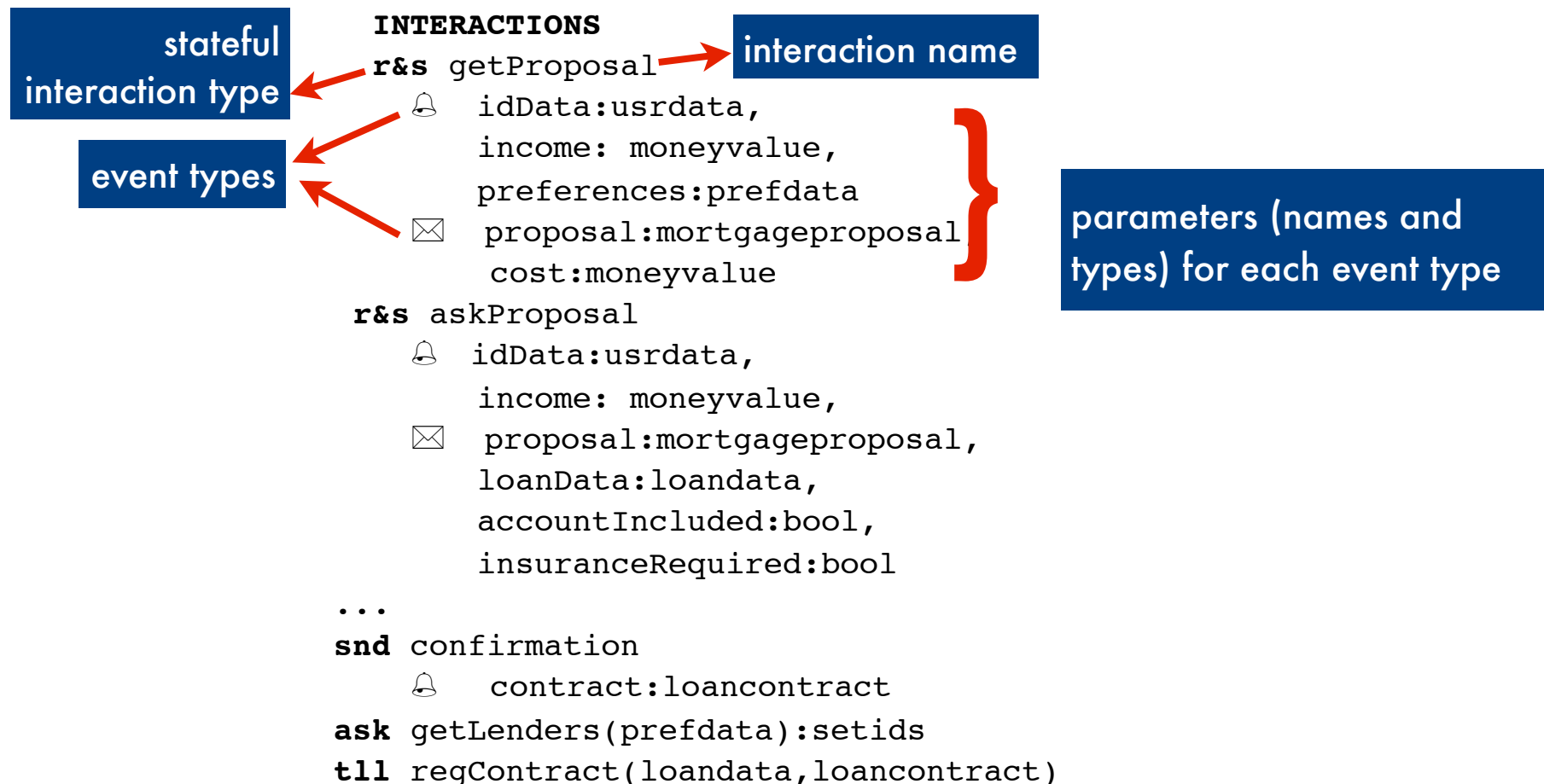
...
snd confirmation
  🔔 contract:loancontract

ask getLenders(prefdata):setids

tll regContract(loandata,loancontract)
```

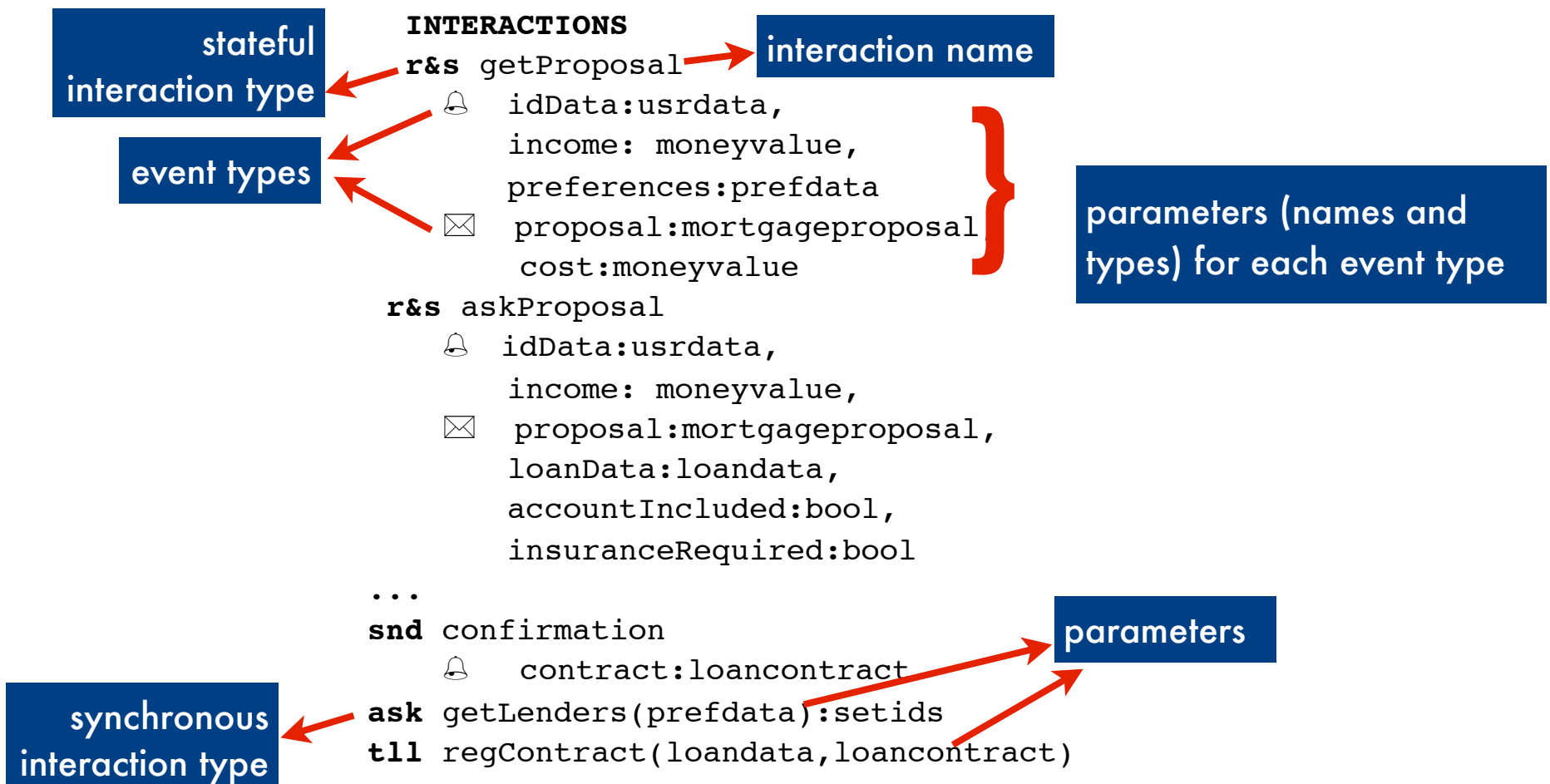
signatures

- Each party defines a signature – the interactions in which it can be involved
- For example, the signature of the business role *MortgageAgent* is defined as follows

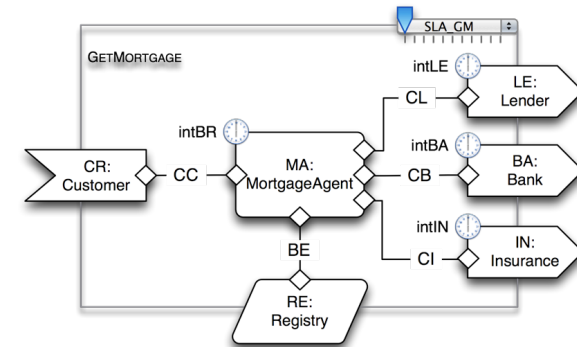
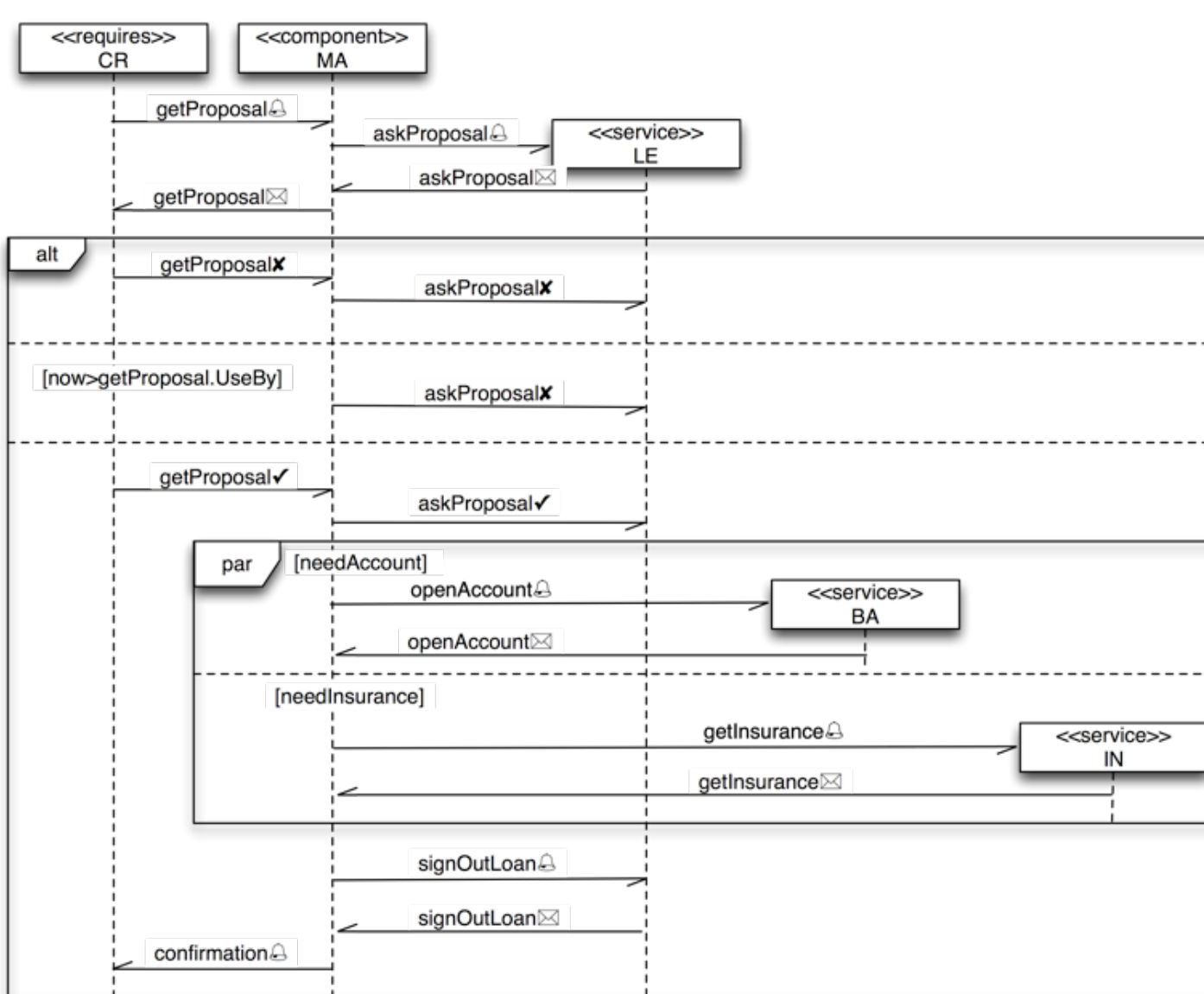


signatures

- Each party defines a signature – the interactions in which it can be involved
- For example, the signature of the business role *MortgageAgent* is defined as follows



orchestration



business roles – local state

BUSINESS ROLE MortgageAgent is

INTERACTIONS

...

ORCHESTRATION

```
local s:[INITIAL, WAIT_PROPOSAL, WAIT_DECISION,  
        PROPOSAL_ACCEPTED, SIGNING, FINAL],  
lenders:setids  
needAccount, needInsurance:bool  
insuranceData:insurancedata,  
accountData:accountdata
```

business roles – local state

BUSINESS ROLE MortgageAgent is


INTERACTIONS

...

ORCHESTRATION

```
local s:[INITIAL, WAIT_PROPOSAL, WAIT_DECISION,  
        PROPOSAL_ACCEPTED, SIGNING, FINAL],  
lenders:setids  
needAccount, needInsurance:bool  
insuranceData:insurancedata,  
accountData:accountdata
```

state variables are
used for storing data
that may be needed
for the orchestration



business roles – local state

BUSINESS ROLE MortgageAgent is

INTERACTIONS

...

ORCHESTRATION

```
local s:[INITIAL, WAIT_PROPOSAL, WAIT_DECISION,  
        PROPOSAL_ACCEPTED, SIGNING, FINAL],  
lenders:setids  
needAccount, needInsurance:bool  
insuranceData:insurancedata,  
accountData:accountdata
```

state variables are used for storing data that may be needed for the orchestration

s is used for control flow (i.e. for encoding an internal state machine)

business roles – local state

BUSINESS ROLE MortgageAgent is

INTERACTIONS

...

ORCHESTRATION

```
local s:[INITIAL, WAIT_PROPOSAL, WAIT_DECISION,  
        PROPOSAL_ACCEPTED, SIGNING, FINAL],  
lenders:setids  
needAccount, needInsurance:bool  
insuranceData:insurancedata,  
accountData:accountdata
```

state variables are used for storing data that may be needed for the orchestration

`s` is used for control flow (i.e. for encoding an internal state machine)

other variables may be used for storing data received during interactions

business roles: transitions

BUSINESS ROLE MortgageAgent is

INTERACTIONS

...

ORCHESTRATION

```
local s:[INITIAL, WAIT_PROPOSAL, WAIT_DECISION,  
        PROPOSAL_ACCEPTED, SIGNING, FINAL],  
    lenders:setids  
    needAccount, needInsurance:bool  
    insuranceData:insurancedata, accountData:accountdata
```

transition GetClientRequest

triggeredBy getProposal🔔

guardedBy s=INITIAL

effects lenders'=getLenders(prefdata)

$\wedge \neg \text{empty}(\text{lenders}') \supset s'=\text{WAIT_PROPOSAL}$

$\wedge \text{empty}(\text{lenders}') \supset s'=\text{FINAL}$

sends $\neg \text{empty}(\text{lenders}') \supset \text{askProposal}$ 🔔

$\wedge \text{askProposal.idData}=\text{getProposal.idData}$

$\wedge \text{askProposal.income}=\text{getProposal.income}$

$\wedge \text{empty}(\text{lenders}') \supset \text{getProposal}$ ✉

$\wedge \text{getProposal.Reply}=\text{false}$

business roles: transitions

The orchestration is defined by a number of transitions

BUSINESS ROLE MortgageAgent **is**

INTERACTIONS

...

ORCHESTRATION

```
local s:[INITIAL, WAIT_PROPOSAL, WAIT_DECISION,
          PROPOSAL_ACCEPTED, SIGNING, FINAL],
      lenders:setids
      needAccount, needInsurance:bool
      insuranceData:insurancedata, accountData:accountdata
```

transition GetClientRequest

triggeredBy getProposal🔔

guardedBy s=INITIAL

effects lenders'=getLenders(prefdata)

$\wedge \neg \text{empty}(\text{lenders}') \supset s'=\text{WAIT_PROPOSAL}$

$\wedge \text{empty}(\text{lenders}') \supset s'=\text{FINAL}$

sends $\neg \text{empty}(\text{lenders}') \supset \text{askProposal}$ 🔔

$\wedge \text{askProposal.idData}=\text{getProposal.idData}$

$\wedge \text{askProposal.income}=\text{getProposal.income}$

$\wedge \text{empty}(\text{lenders}') \supset \text{getProposal}$ ✉

$\wedge \text{getProposal.Reply}=\text{false}$

business roles: transitions

The orchestration is defined by a number of transitions

A trigger is either an interaction event or a state condition

BUSINESS ROLE MortgageAgent **is**

INTERACTIONS

...

ORCHESTRATION

```
local s:[INITIAL, WAIT_PROPOSAL, WAIT_DECISION,
        PROPOSAL_ACCEPTED, SIGNING, FINAL],
    lenders:setids
    needAccount, needInsurance:bool
    insuranceData:insurancedata, accountData:accountdata
```

transition GetClientRequest

triggeredBy getProposal🔔

guardedBy s=INITIAL

effects lenders'=getLenders(prefdata)

$\wedge \neg \text{empty}(\text{lenders}') \supset s'=\text{WAIT_PROPOSAL}$

$\wedge \text{empty}(\text{lenders}') \supset s'=\text{FINAL}$

sends $\neg \text{empty}(\text{lenders}') \supset \text{askProposal}$ 🔔

$\wedge \text{askProposal.idData}=\text{getProposal.idData}$

$\wedge \text{askProposal.income}=\text{getProposal.income}$

$\wedge \text{empty}(\text{lenders}') \supset \text{getProposal}$ ✉

$\wedge \text{getProposal.Reply}=\text{false}$

business roles: transitions

BUSINESS ROLE MortgageAgent **is**

INTERACTIONS

...

ORCHESTRATION

```
local s:[INITIAL, WAIT_PROPOSAL, WAIT_DECISION,  
        PROPOSAL_ACCEPTED, SIGNING, FINAL],  
    lenders:setids  
    needAccount, needInsurance:bool  
    insuranceData:insurancedata, accountData:accountdata
```

transition GetClientRequest

triggeredBy getProposal🔔

guardedBy s=INITIAL

effects lenders'=getLenders(prefdata)

$\wedge \neg \text{empty}(\text{lenders}') \supset s'=\text{WAIT_PROPOSAL}$

$\wedge \text{empty}(\text{lenders}') \supset s'=\text{FINAL}$

sends $\neg \text{empty}(\text{lenders}') \supset \text{askProposal}$ 🔔

$\wedge \text{askProposal.idData}=\text{getProposal.idData}$

$\wedge \text{askProposal.income}=\text{getProposal.income}$

$\wedge \text{empty}(\text{lenders}') \supset \text{getProposal}$ ✉

$\wedge \text{getProposal.Reply}=\text{false}$

The orchestration is defined by a number of transitions

A trigger is either an interaction event or a state condition

A guard identifies the states in which the transition can take place

business roles: transitions

BUSINESS ROLE MortgageAgent **is**

INTERACTIONS

...

ORCHESTRATION

```
local s:[INITIAL, WAIT_PROPOSAL, WAIT_DECISION,
          PROPOSAL_ACCEPTED, SIGNING, FINAL],
lenders:setids
needAccount, needInsurance:bool
insuranceData:insurancedata, accountData:accountdata
```

transition GetClientRequest

triggeredBy getProposal🔔

guardedBy s=INITIAL

effects lenders'=getLenders(prefdata)

$\wedge \neg \text{empty}(\text{lenders}') \supset s'=\text{WAIT_PROPOSAL}$

$\wedge \text{empty}(\text{lenders}') \supset s'=\text{FINAL}$

sends $\neg \text{empty}(\text{lenders}') \supset \text{askProposal}$ 🔔

$\wedge \text{askProposal.idData}=\text{getProposal.idData}$

$\wedge \text{askProposal.income}=\text{getProposal.income}$

$\wedge \text{empty}(\text{lenders}') \supset \text{getProposal}$ ✉

$\wedge \text{getProposal.Reply}=\text{false}$

The orchestration is defined by a number of transitions

A trigger is either an interaction event or a state condition

Effects on the local state (*lenders'* denotes the value of *lenders* after the transition)

A guard identifies the states in which the transition can take place

getLenders is a synchronous interaction. The returned value is stored in the variable *lenders*

business roles: transitions

BUSINESS ROLE MortgageAgent **is**

INTERACTIONS

...

ORCHESTRATION

```
local s:[INITIAL, WAIT_PROPOSAL, WAIT_DECISION,
          PROPOSAL_ACCEPTED, SIGNING, FINAL],
lenders:setids
needAccount, needInsurance:bool
insuranceData:insurancedata, accountData:accountdata
```

transition GetClientRequest

triggeredBy getProposal🔔

guardedBy s=INITIAL

effects lenders'=getLenders(prefdata)

$\wedge \neg \text{empty}(\text{lenders}') \supset s'=\text{WAIT_PROPOSAL}$

$\wedge \text{empty}(\text{lenders}') \supset s'=\text{FINAL}$

sends $\neg \text{empty}(\text{lenders}') \supset \text{askProposal}$ 🔔

$\wedge \text{askProposal.idData}=\text{getProposal.idData}$

$\wedge \text{askProposal.income}=\text{getProposal.income}$

$\wedge \text{empty}(\text{lenders}') \supset \text{getProposal}$ ✉

$\wedge \text{getProposal.Reply}=\text{false}$

A guard identifies the states in which the transition can take place

getLenders is a synchronous interaction. The returned value is stored in the variable *lenders*

Reply is a default parameter...

The orchestration is defined by a number of transitions

A trigger is either an interaction event or a state condition

Effects on the local state (*lenders'* denotes the value of *lenders* after the transition)





Events published during the transitions and values taken by their parameters

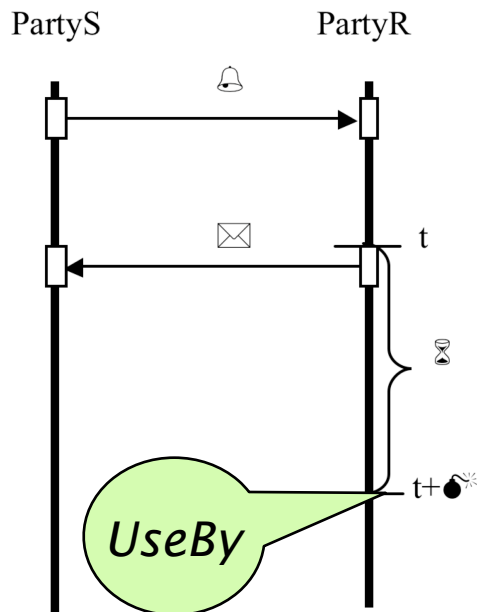
default parameters

default parameters





- Each reply event e  has two default parameters
 - *Reply*: boolean
 - *UseBy*: time

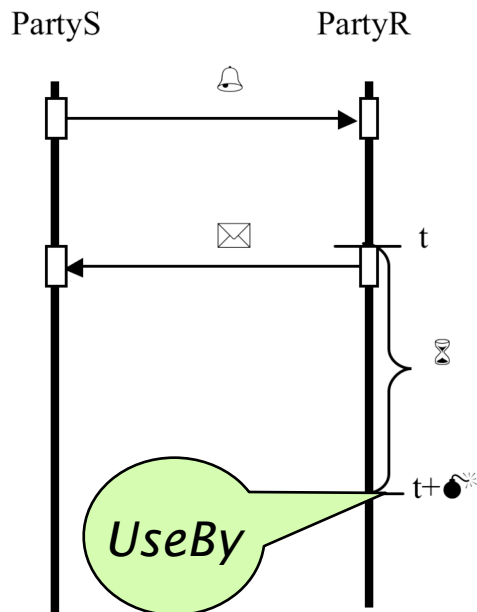
default parameters

- Each reply event e  has two default parameters
 - Reply*: boolean
 - UseBy*: time
- If $e.Reply$ is true, the co-party ensures the pledge a  until $e.UseBy$, and enables a  and a .



default parameters

- Each reply event e  has two default parameters
 - Reply*: boolean
 - UseBy*: time
- If $e.Reply$ is true, the co-party ensures the pledge a  until $e.UseBy$, and enables a  and a .



BUSINESS ROLE MortgageAgent **is**

INTERACTIONS

...

ORCHESTRATION

...

transition TimeOutProposal

triggeredBy now>getProposal.UseBy

guardedBy s=WAIT_DECISION

effects s'=FINAL

sends askProposal~~X~~

now is a system function that returns the current time

using UML state machines

using UML state machines

- the language of business roles is declarative and permits **under-specification**, leaving room for **stepwise refinement**

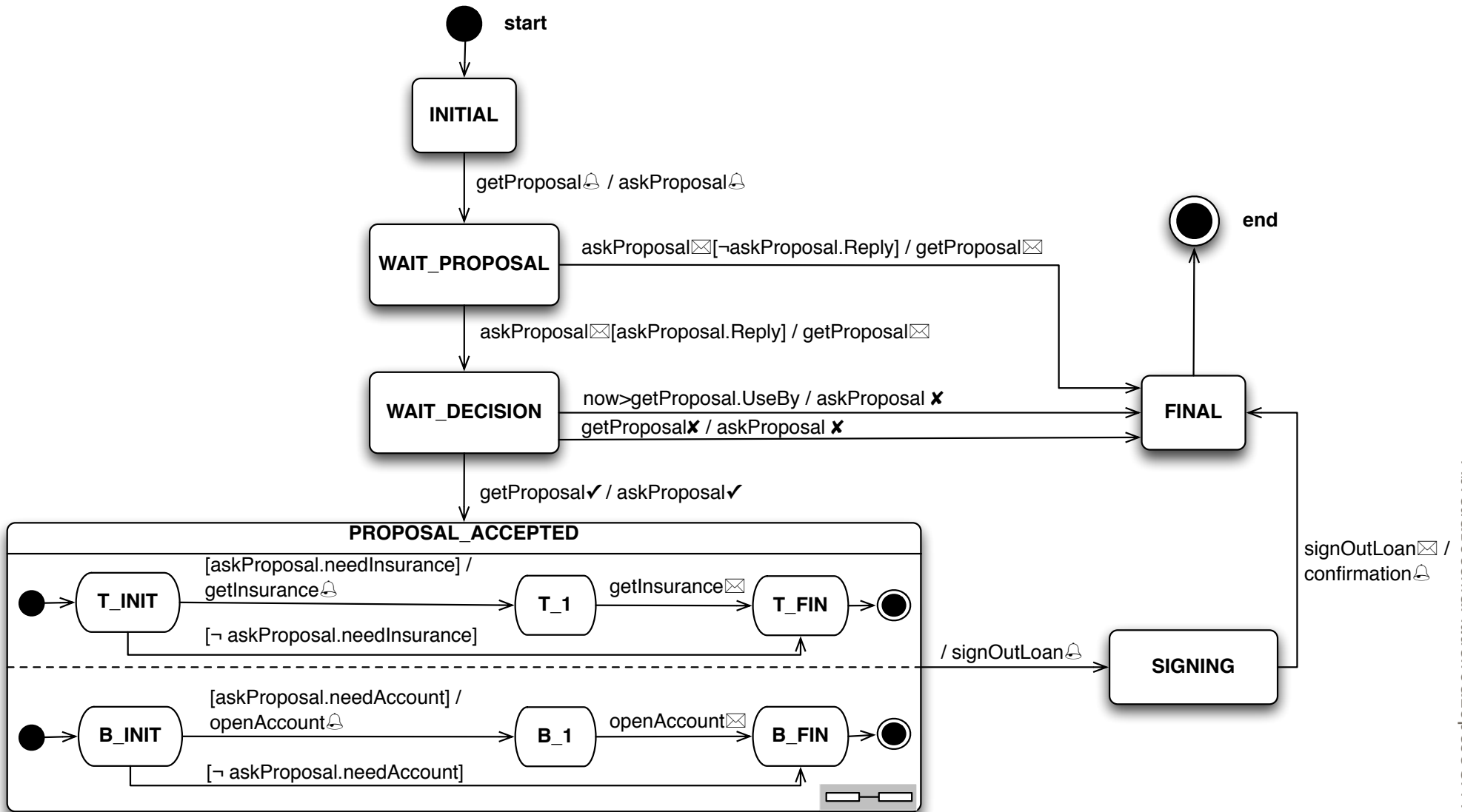
using UML state machines

- the language of business roles is declarative and permits **under-specification**, leaving room for **stepwise refinement**
- other notations can be used (such as UML state machines) when the orchestration is fully specified or one wishes to **reuse** existing specifications

using UML state machines

- the language of business roles is declarative and permits **under-specification**, leaving room for **stepwise refinement**
- other notations can be used (such as UML state machines) when the orchestration is fully specified or one wishes to **reuse** existing specifications
- UML state machines are also used when we want to **analyse** behavioural properties of services through model checkers such as UMC

using UML state machines



other languages

other languages

- business roles can also be extracted from *BPEL* processes

other languages

- business roles can also be extracted from *BPEL* processes
- and from *StPowla workflows* dynamically reconfigured by policies

other languages

- business roles can also be extracted from **BPEL processes**
- and from **StPowla workflows** dynamically reconfigured by policies



From BPEL to SRML: a formal transformational approach
Bocchi, Hong, Lopes and Fiadeiro, WSFM 2008

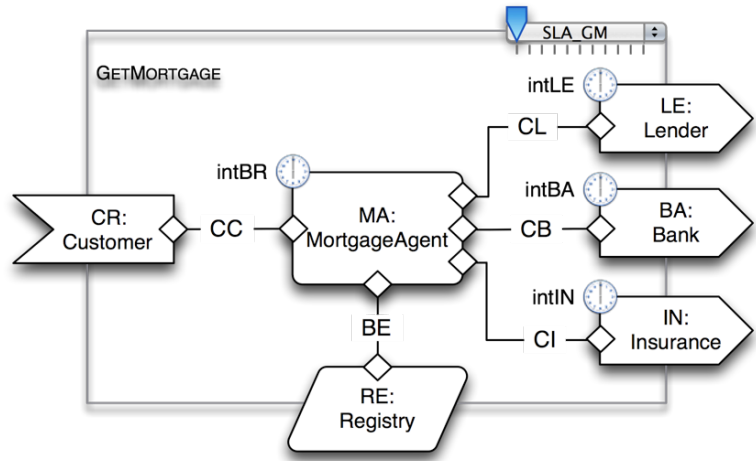


StPowla: SOA, Policies and Workflows. Gorton, Montangero, Reiff-Marganiec and Semini. Engineering Service-Oriented Applications: Analysis, Design and Composition 2007

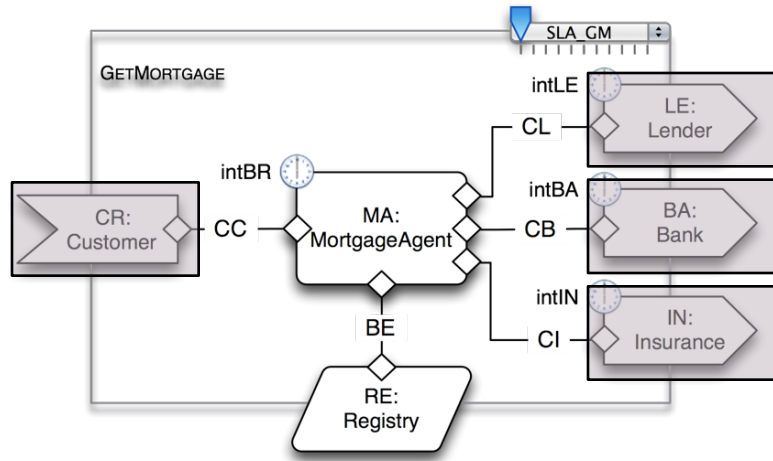


From StPowla processes to SRML models. Bocchi, Gorton and Reiff-Marganiec, Formal Aspects of Computing (FASE 2008)

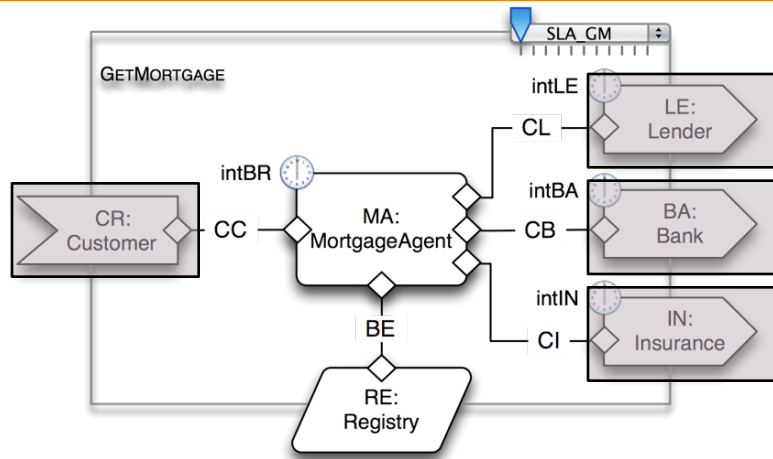
business protocols



business protocols



business protocols



G. Alonso, F. Casati, H. Kuno, V. Machiraju (2004) Web Services. Springer

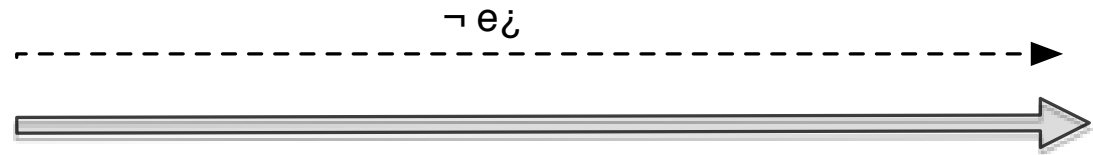
■ *“In particular, a trend that is gathering momentum is that of including, as part of the service description, not only the service interface, but also the business protocol supported by the service, i.e. the specification of which message exchange sequences are supported by the service, for example expressed in terms of constraints on the order in which service operations should be invoked”*

patterns

patterns

initiallyEnabled e

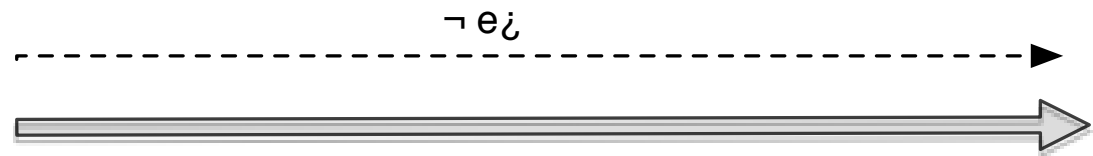
*"e is never discarded
until it is executed"*



patterns

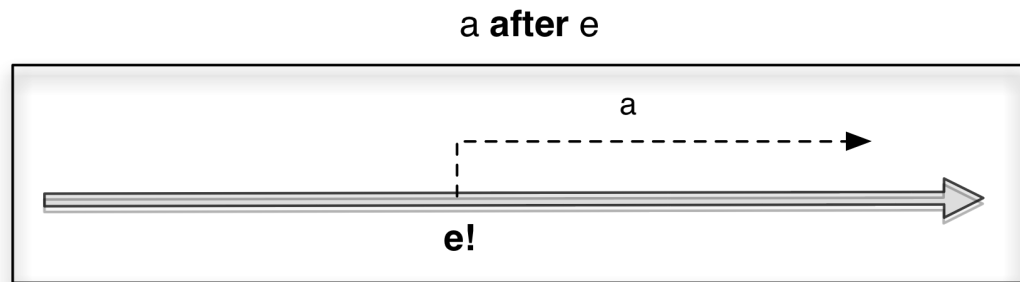
initiallyEnabled e

"e is never discarded until it is executed"



a after e

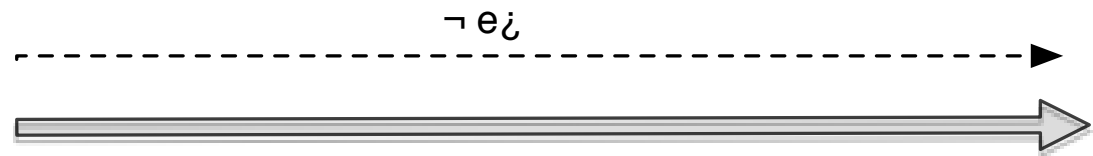
"a holds forever after e is executed"



patterns

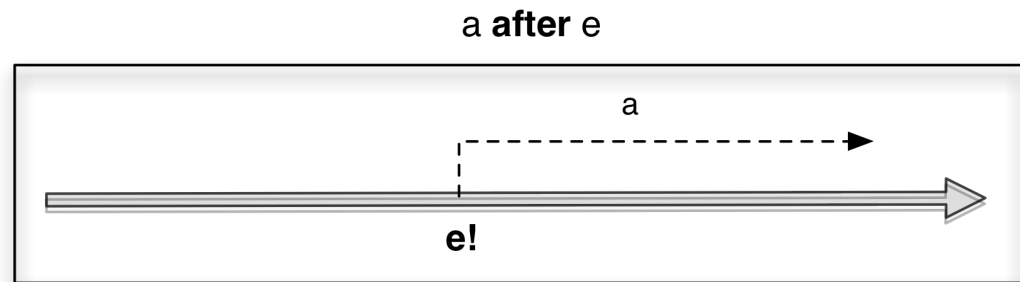
initiallyEnabled e

"e is never discarded until it is executed"



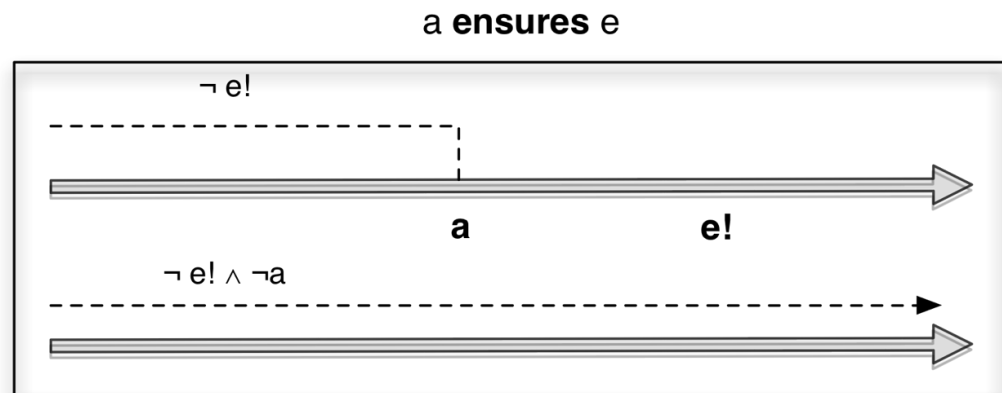
a after e

"a holds forever after e is executed"



a ensures e

"e is not published before a holds, and it is published sometimes after a becomes true"

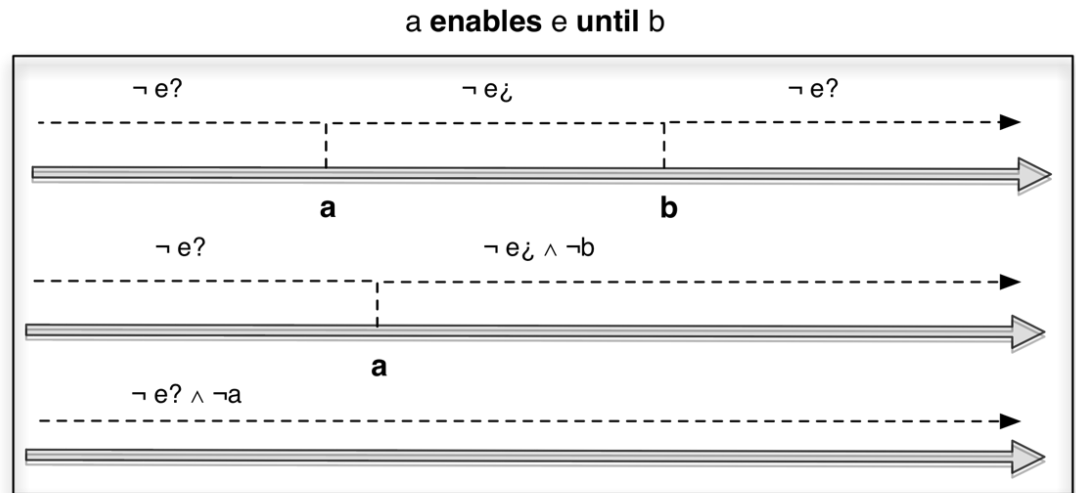


patterns

patterns

a enables e until b

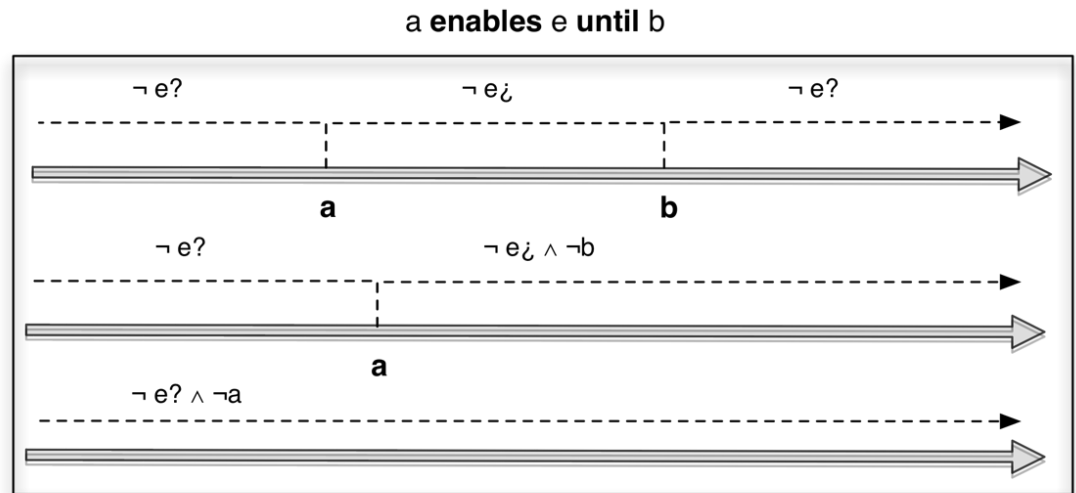
“The event e cannot be executed before a holds and remains enabled after a becomes true until it is either executed or b becomes true (if ever)”



patterns

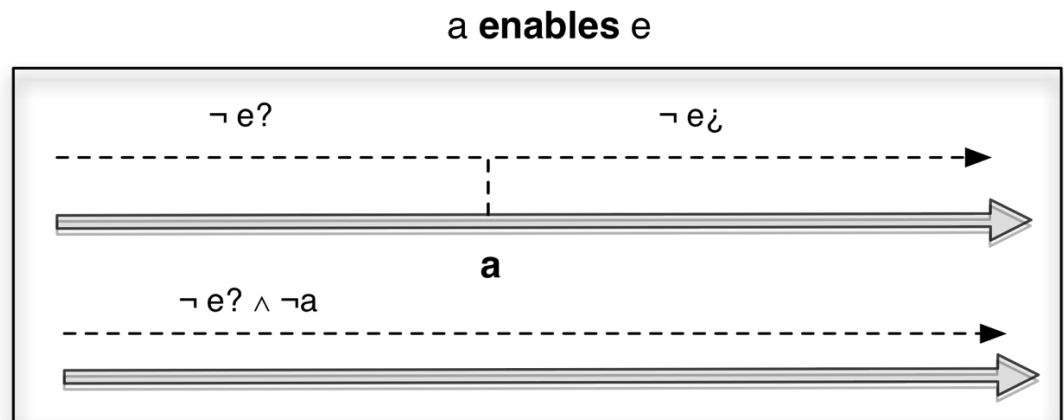
a enables e until b

“The event e cannot be executed before a holds and remains enabled after a becomes true until it is either executed or b becomes true (if ever)”



a enables e

“The event e cannot be executed before a holds and remains enabled after a becomes true until it is executed”



patterns

patterns

- patterns have a translation in **temporal logic** (UCTL) so that they can be model-checked

<i>initiallyEnabled e</i>	$A[true_{\{\neg e_i\}}W_{\{e?\}}true].$
a enables e	$\left(AG[a]\neg EF \langle e_i \rangle true \right) \wedge \left(A[true_{\{\neg e?\}}W_{\{a\}}true \right)$
a ensures e	$\left(AG[a]AF[e!]true \right) \wedge \left(A[true_{\{\neg e!\}}W_{\{a\}}true \right)$



A model-checking approach for service-component architectures.
Abreu, Mazzanti, Fiadeiro, Gnesi. FMOODS 2009

a business protocol

BUSINESS PROTOCOL Customer **is**

INTERACTIONS

r&s getProposal

🔔 idData:usrdata,
income: moneyvalue,
preferences:prefdata

✉ proposal:mortgageproposal,
cost:moneyvalue

snd confirmation

🔔 contract:loancontract

SLA VARIABLES

CHARGE:[0..100]

BEHAVIOURS

initiallyEnabled getProposal🔔?

getProposal.cost $\leq 750 * (CHARGE / 100 + 1)$ **after** getProposal✉! \wedge getProposal.Reply

getProposal✓? **ensures** confirmation🔔!

a business protocol

BUSINESS PROTOCOL Customer **is**

INTERACTIONS

r&s getProposal

🔔 idData:usrdata,
income: moneyvalue,
preferences:prefdata

✉ proposal:mortgageproposal,
cost:moneyvalue

snd confirmation

🔔 contract:loancontract

SLA VARIABLES

CHARGE:[0..100]

BEHAVIOURS

initiallyEnabled getProposal🔔?

getProposal.cost ≤ 750*(CHARGE/100+1) **after** getProposal✉! ∧ getProposal.Reply

getProposal✓? **ensures** confirmation🔔!

A request for *getProposal* is enabled
when the service is activated



a business protocol

BUSINESS PROTOCOL Customer **is**

INTERACTIONS

r&s getProposal

🔔 idData:usrdata,
income: moneyvalue,
preferences:prefdata
✉ proposal:mortgageproposal,
cost:moneyvalue

snd confirmation

🔔 contract:loancontract

SLA VARIABLES

CHARGE:[0..100]

BEHAVIOURS

initiallyEnabled getProposal🔔?

getProposal.cost $\leq 750 * (CHARGE / 100 + 1)$ **after** getProposal✉! \wedge getProposal.Reply

getProposal✓? **ensures** confirmation🔔!

A request for *getProposal* is enabled when the service is activated

The service brokerage has a base price that can be subject to an extra charge, subject to negotiation.

a business protocol

BUSINESS PROTOCOL Customer **is**

INTERACTIONS

r&s `getProposal`

🔔 `idData:usrdata,`
`income: moneyvalue,`
`preferences:prefdata`
✉ `proposal:mortgageproposal,`
`cost:moneyvalue`

snd `confirmation`

🔔 `contract:loancontract`

SLA VARIABLES

`CHARGE:[0..100]`

BEHAVIOURS

initiallyEnabled `getProposal`🔔?

`getProposal.cost ≤ 750*(CHARGE/100+1)` **after** `getProposal`✉! \wedge `getProposal.Reply`

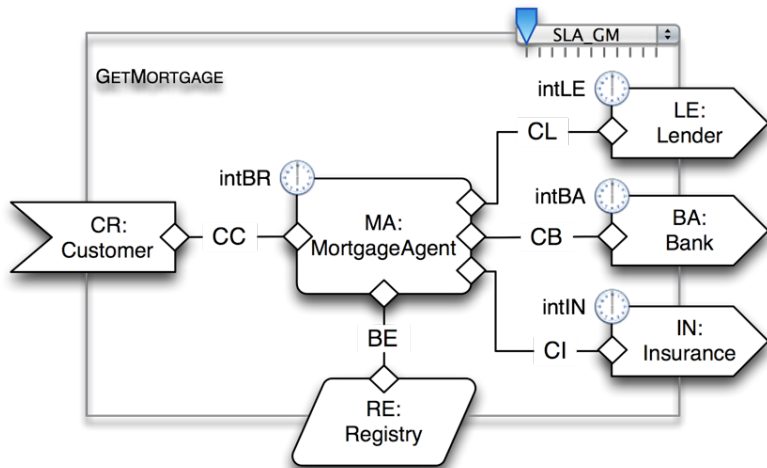
`getProposal`✓? **ensures** `confirmation`🔔!

A request for `getProposal` is enabled when the service is activated

The service brokerage has a base price that can be subject to an extra charge, subject to negotiation.

A confirmation carrying the loan contract will be issued upon receipt of the commit to `getProposal`

layer protocols



LAYER PROTOCOL Registry **is**

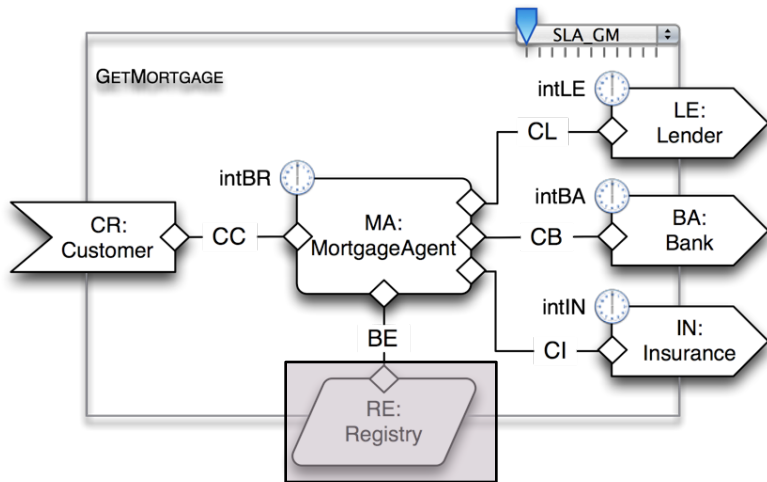
INTERACTIONS

rpl getLenders(prefdata):setids

prf registerContract(loanData,loanContract)

BEHAVIOUR

layer protocols



- Layer Protocols involve persistent components, typically through synchronous blocking interactions

LAYER PROTOCOL Registry **is**

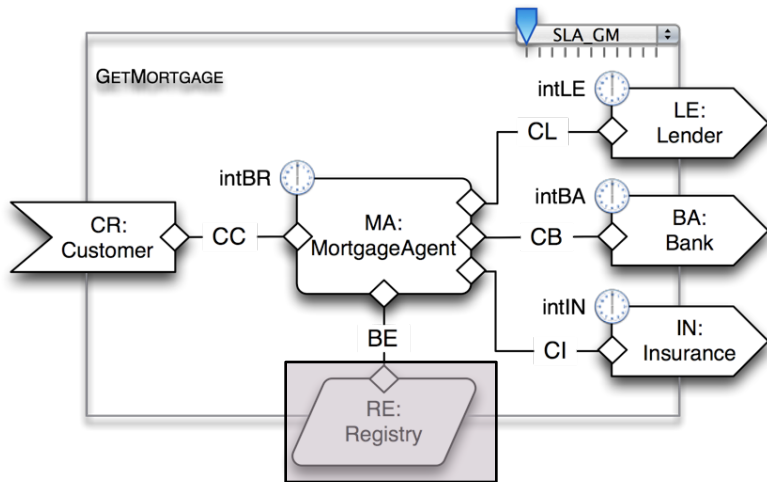
INTERACTIONS

```
rpl getLenders(prefdata):setids
```

```
prf registerContract(loanData,loanContract)
```

BEHAVIOUR

layer protocols



- Layer Protocols involve persistent components, typically through synchronous blocking interactions

The registry can be queried about the registered lenders that meet given users preferences

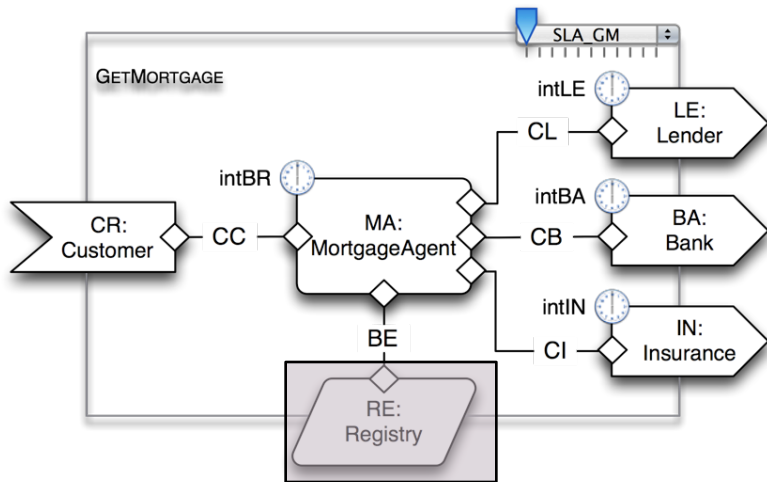
LAYER PROTOCOL Registry is

INTERACTIONS

```
rp1 getLenders(prefdata):setids  
prf registerContract(loanData,loanContract)
```

BEHAVIOUR

layer protocols



- Layer Protocols involve persistent components, typically through synchronous blocking interactions

The registry can be queried about the registered lenders that meet given users preferences

LAYER PROTOCOL Registry is

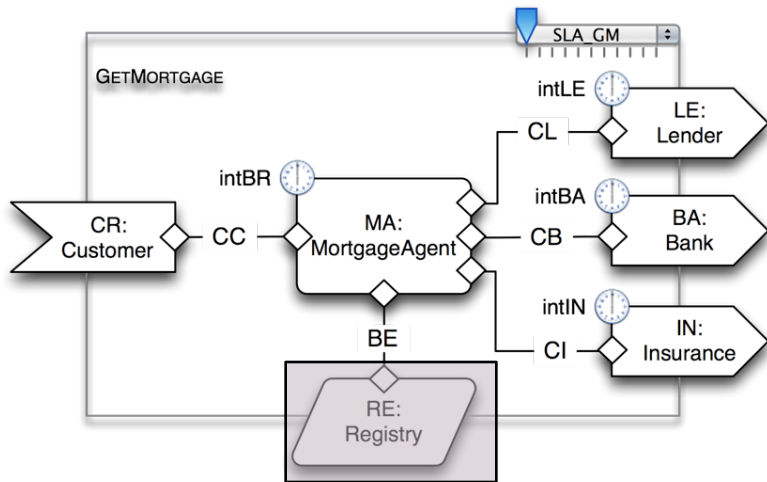
INTERACTIONS

```
rpl getLenders(prefdata):setids
prf registerContract(loanData, loanContract)
```

BEHAVIOUR

The registry is able to register new contracts

layer protocols



Layer Protocols involve persistent components, typically through synchronous blocking interactions

The registry can be queried about the registered lenders that meet given users preferences

LAYER PROTOCOL Registry is

INTERACTIONS

```

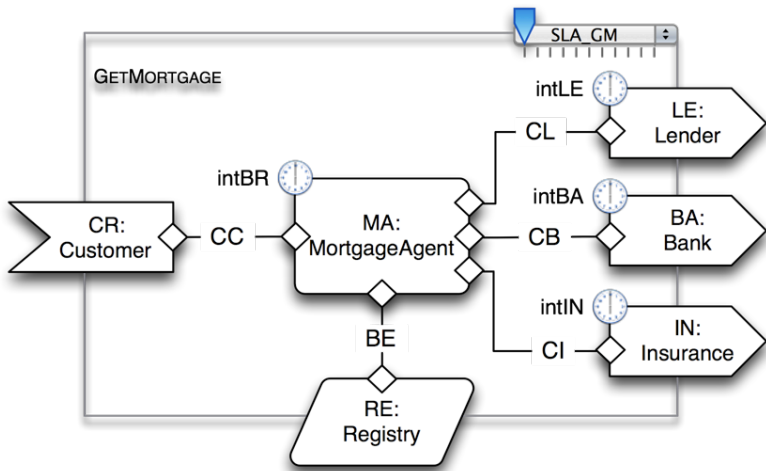
rpl getLenders(prefdata):setids
prf registerContract(loanData,loanContract)
    
```

BEHAVIOUR

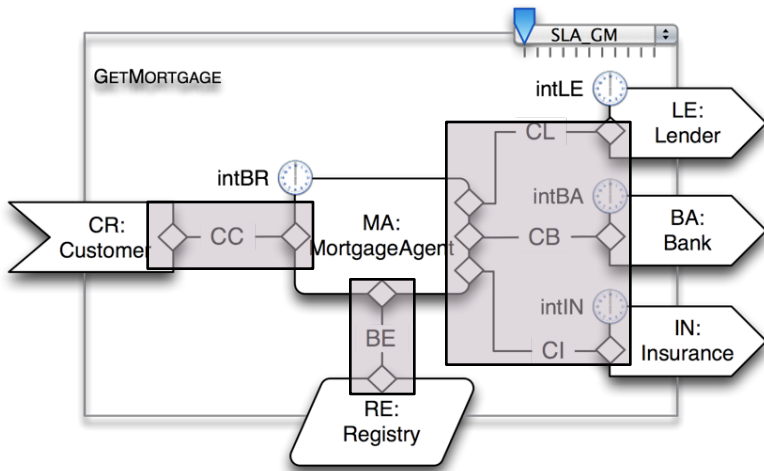
The properties of synchronous interactions are typically in the style of pre/post-condition specifications

The registry is able to register new contracts

interaction protocols

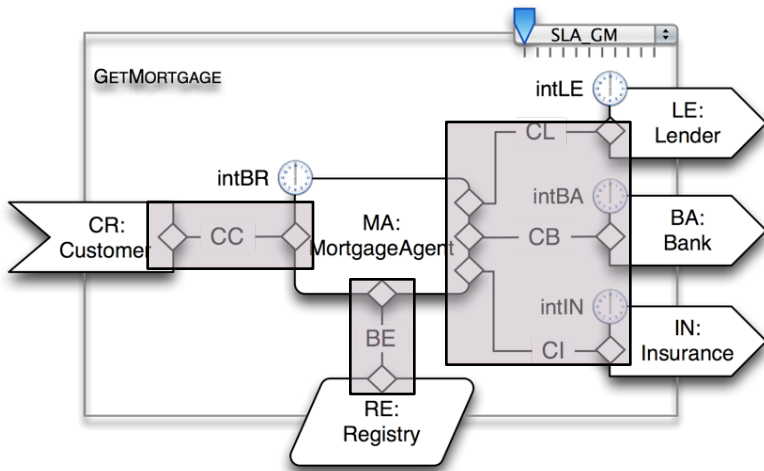


interaction protocols



- Wires are typed with (binary) connectors
- Interaction protocols describe how the interactions between two parties (*ROLE A* and *ROLE B*) are coordinated

interaction protocols



- Wires are typed with (binary) connectors
- Interaction protocols describe how the interactions between two parties (*ROLE A* and *ROLE B*) are coordinated

INTERACTION PROTOCOL $\text{Straight.I}(d_1, d_2)\text{O}(d_3)$ is

ROLE A

s&r S_1

$\text{🔔 } i_1:d_1, i_2:d_2$

$\text{✉ } o_1:d_3$

ROLE B

r&s R_1

$\text{🔔 } i_1:d_1, i_2:d_2$

$\text{✉ } o_1:d_3$

COORDINATION

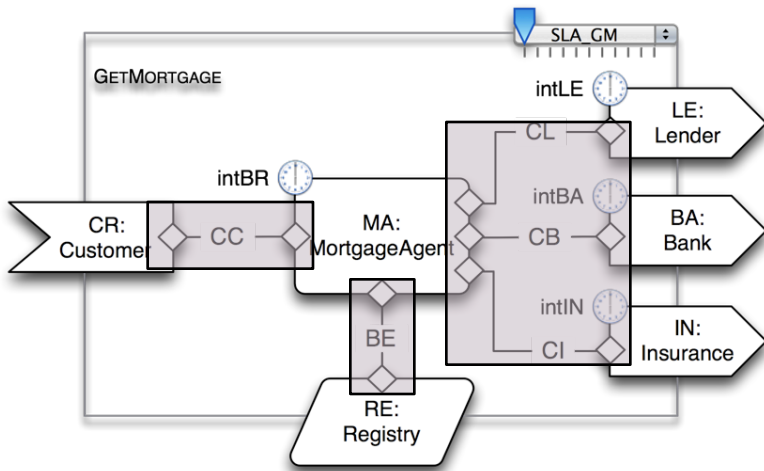
$S_1 \equiv R_1$

$S_1 \cdot i_1 = R_1 \cdot i_1$

$S_1 \cdot i_2 = R_1 \cdot i_2$

$S_1 \cdot o_1 = R_1 \cdot o_1$

interaction protocols



- Wires are typed with (binary) connectors
- Interaction protocols describe how the interactions between two parties (*ROLE A* and *ROLE B*) are coordinated

INTERACTION PROTOCOL $\text{Straight}.I(d_1, d_2)O(d_3)$ is

ROLE A

s&r S_1

🔔 $i_1:d_1, i_2:d_2$

✉ $o_1:d_3$

ROLE B

r&s R_1

🔔 $i_1:d_1, i_2:d_2$

✉ $o_1:d_3$

COORDINATION

$S_1 \equiv R_1$

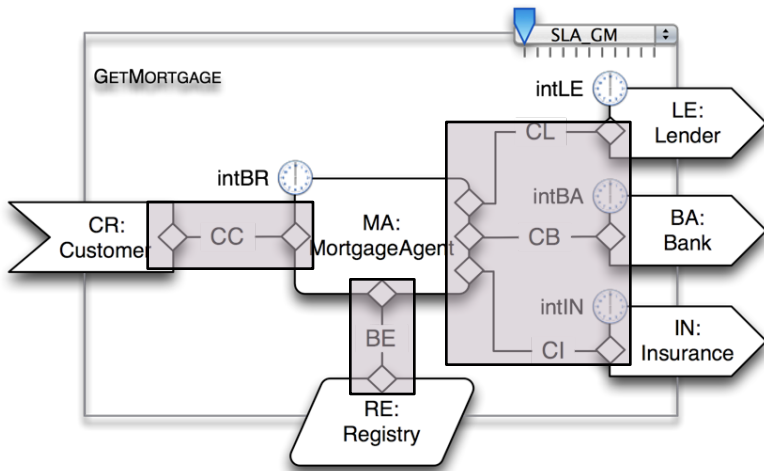
$S_1.i_1 = R_1.i_1$

$S_1.i_2 = R_1.i_2$

$S_1.o_1 = R_1.o_1$

The interaction protocol *Straight* defines simple transmission of events between the corresponding parties

interaction protocols



- Wires are typed with (binary) connectors
- Interaction protocols describe how the interactions between two parties (*ROLE A* and *ROLE B*) are coordinated

INTERACTION PROTOCOL $\text{Straight}.I(d_1, d_2)O(d_3)$ is

ROLE A

s&r S_1

$\text{🔔 } i_1:d_1, i_2:d_2$

$\text{✉ } o_1:d_3$

ROLE B

r&s R_1

$\text{🔔 } i_1:d_1, i_2:d_2$

$\text{✉ } o_1:d_3$

COORDINATION

$S_1 \equiv R_1$

$S_1.i_1 = R_1.i_1$

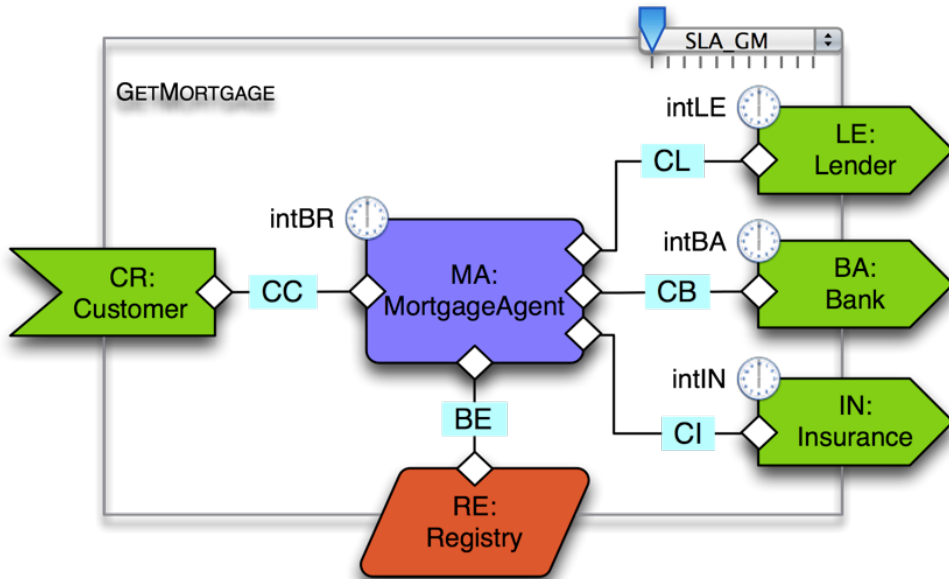
$S_1.i_2 = R_1.i_2$

$S_1.o_1 = R_1.o_1$

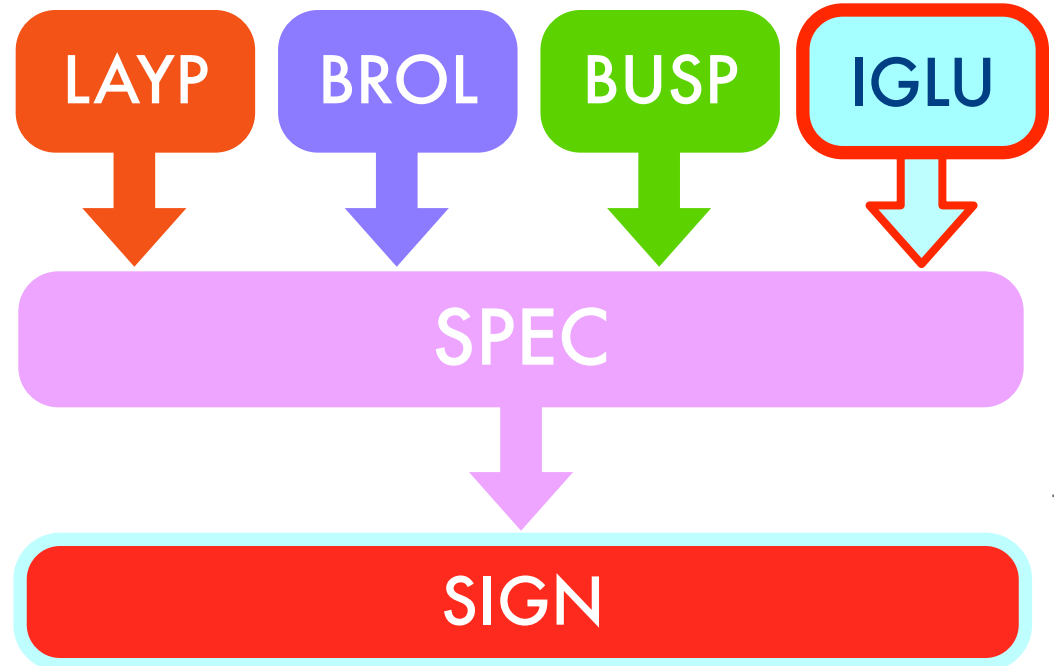
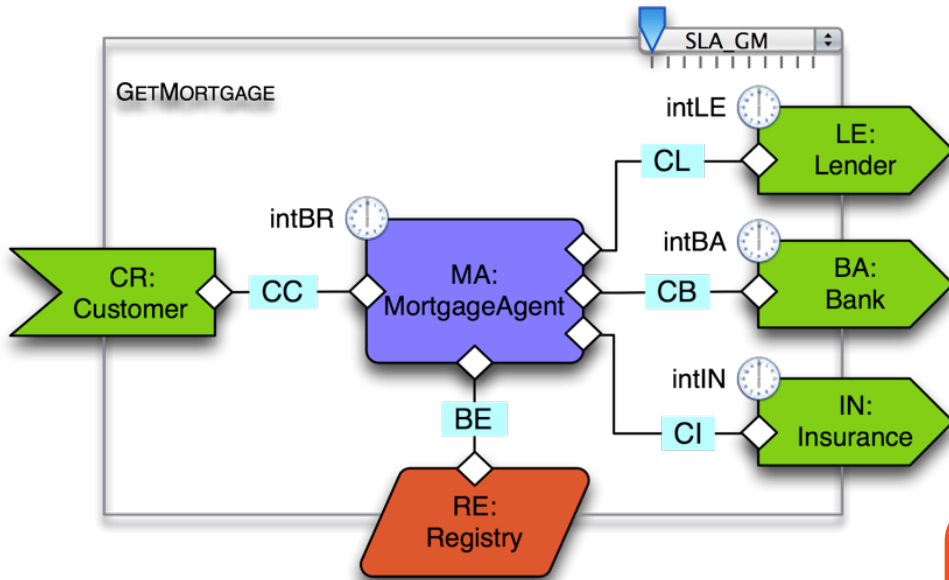
To allow reuse, we parametrise *Straight* with the types of the interaction parameters

The interaction protocol *Straight* defines simple transmission of events between the corresponding parties

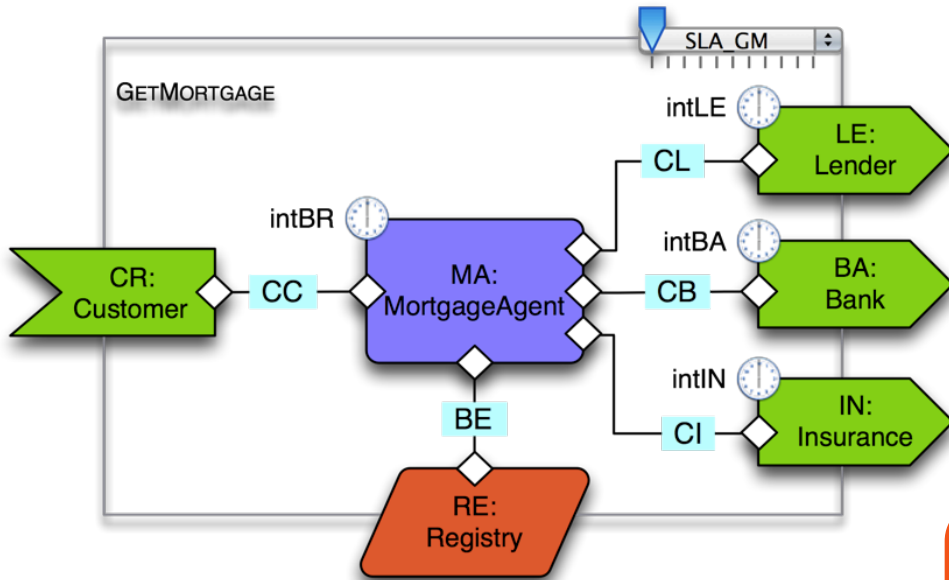
the formal domains



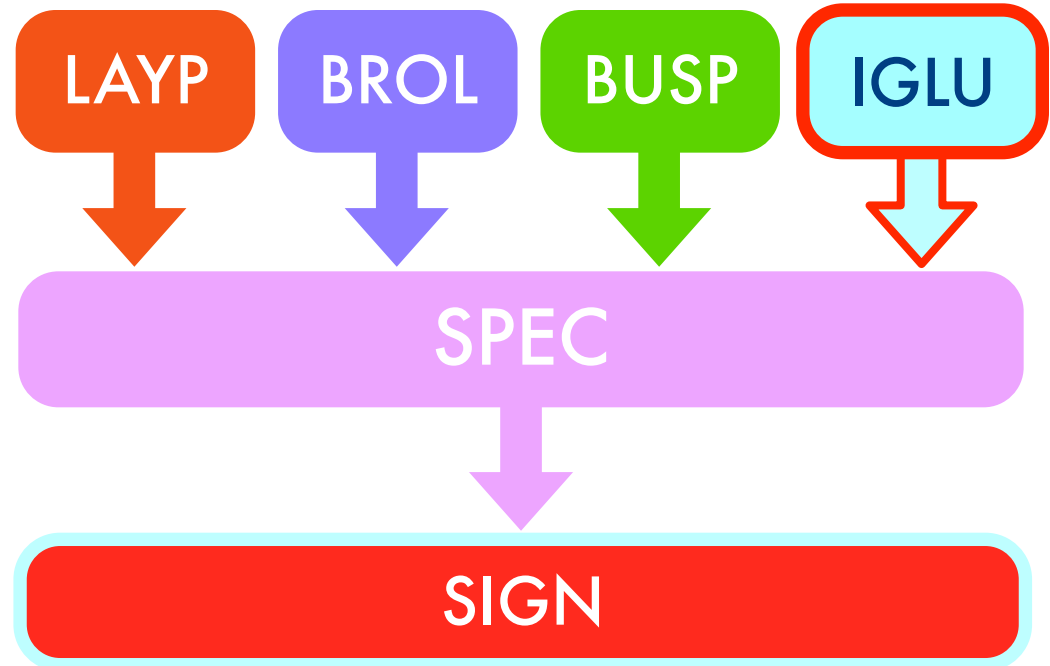
the formal domains



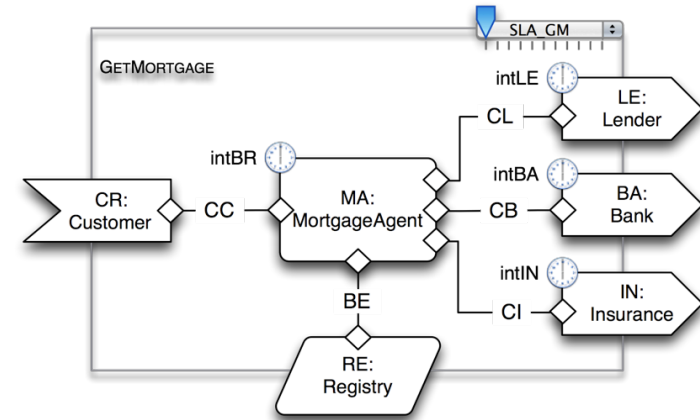
the formal domains



an entailment system \vdash

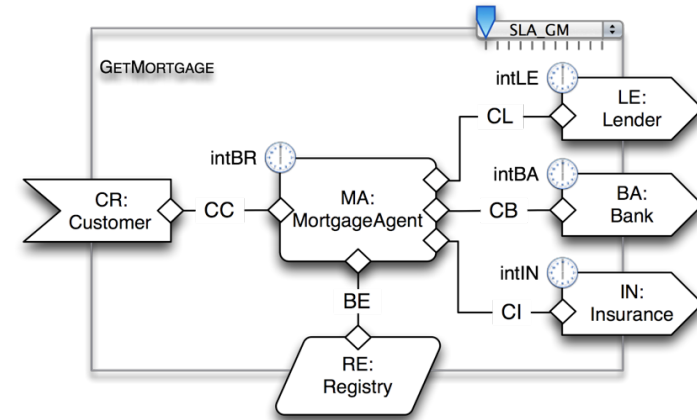


modules as graphs



modules as graphs

A service module M consists of:



modules as graphs

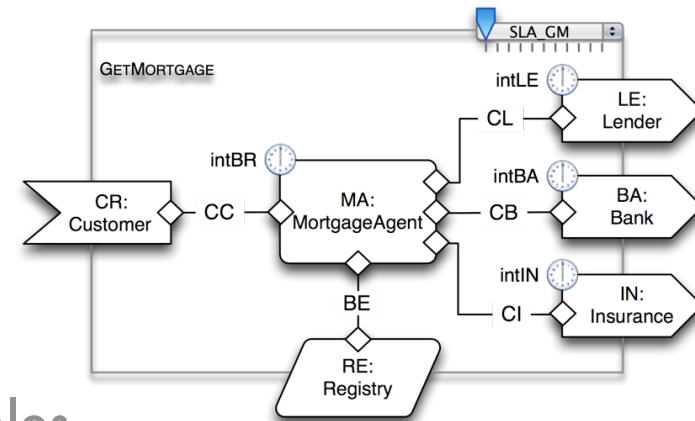
A service module M consists of:

■ A labelled graph:

● Nodes are classified as:

- $\text{components}(M)$, labelled with business roles
- $\text{uses}(M)$, labelled with layer protocols
- $\text{requires}(M)$, labelled with business protocols
- $\{\text{provides}(M)\}$, labelled with a business protocol

● Edges (wires) are labelled with connectors
(interaction glue and attachments)



modules as graphs

A service module M consists of:

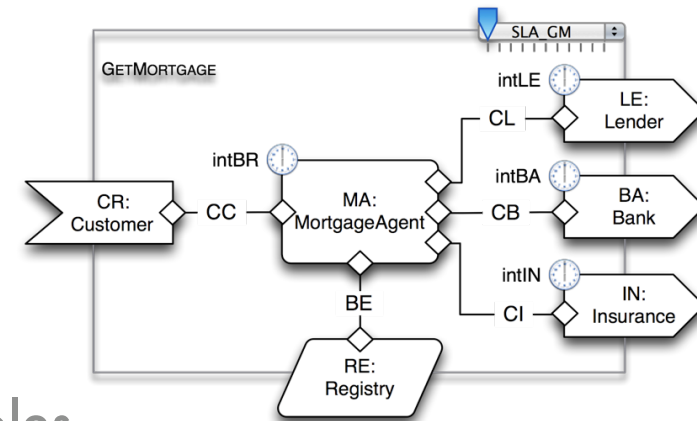
■ A labelled graph:

☀ Nodes are classified as:

- components(M), labelled with business roles
- uses(M), labelled with layer protocols
- requires(M), labelled with business protocols
- {provides(M)}, labelled with a business protocol

Body

☀ Edges (wires) are labelled with connectors
(interaction glue and attachments)



modules as graphs

A service module M consists of:

■ A labelled graph:

☀ Nodes are classified as:

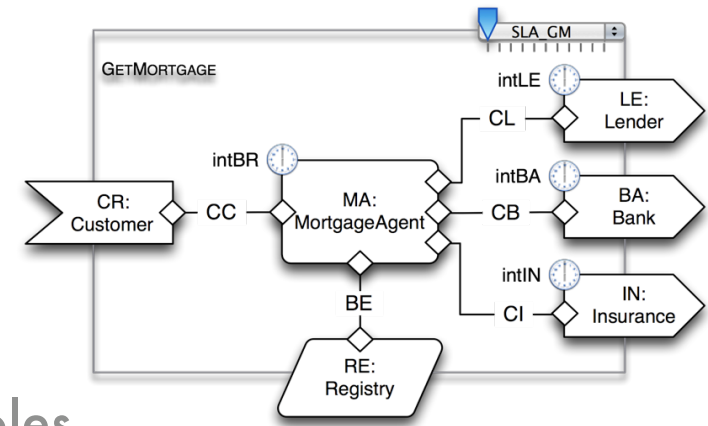
- components(M), labelled with business roles
- uses(M), labelled with layer protocols
- requires(M), labelled with business protocols
- {provides(M)}, labelled with a business protocol

Body

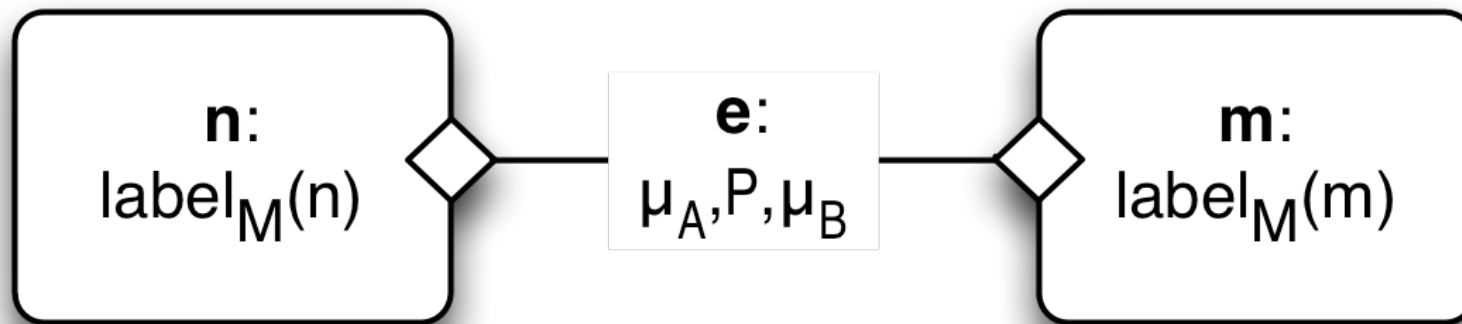
☀ Edges (wires) are labelled with connectors (interaction glue and attachments)

■ An internal configuration policy

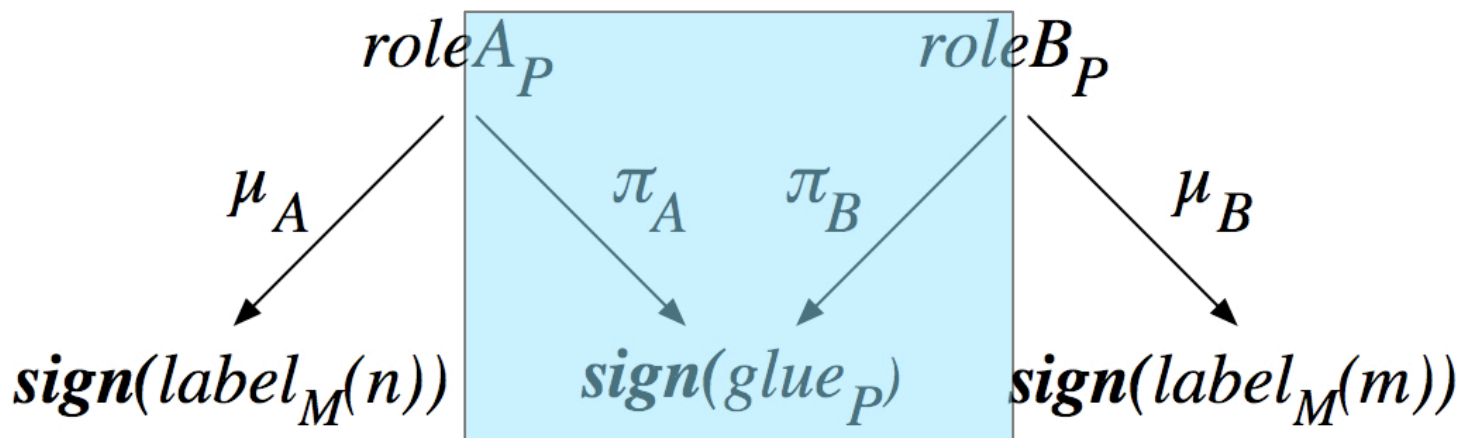
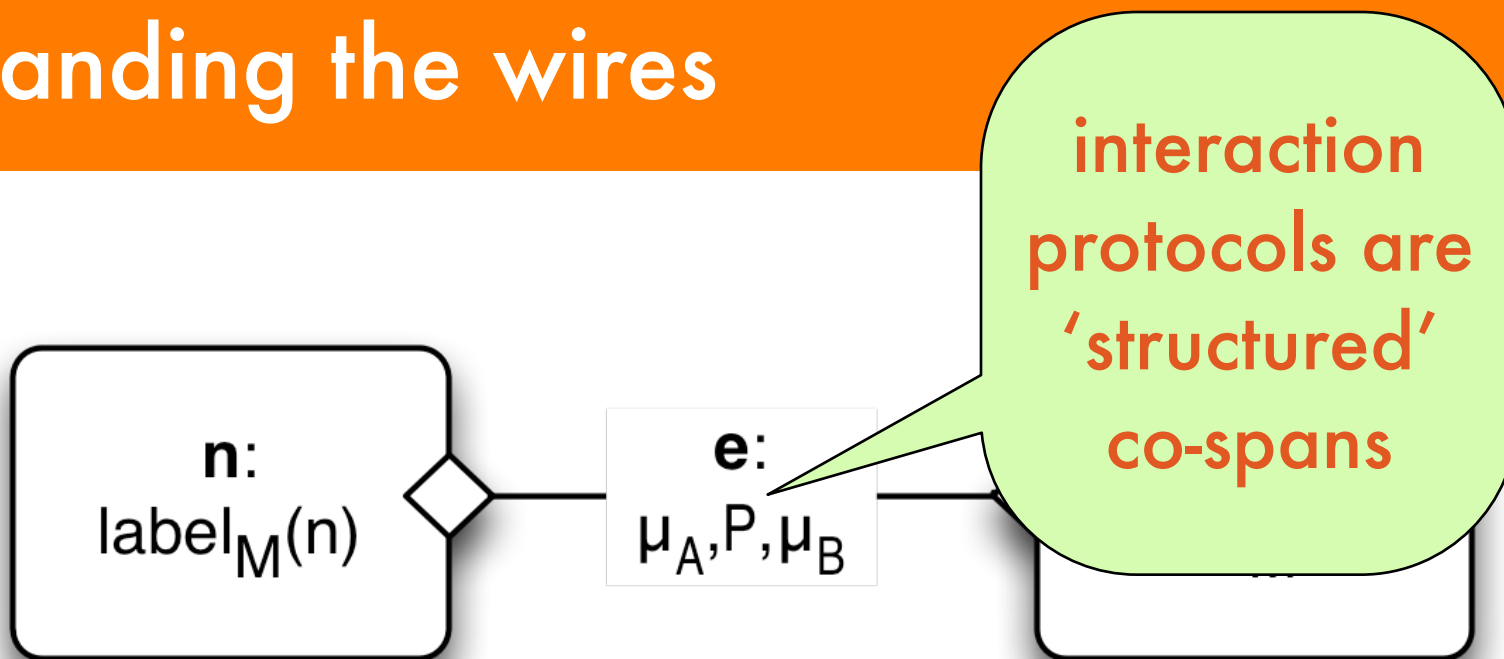
■ An external configuration policy



expanding the wires



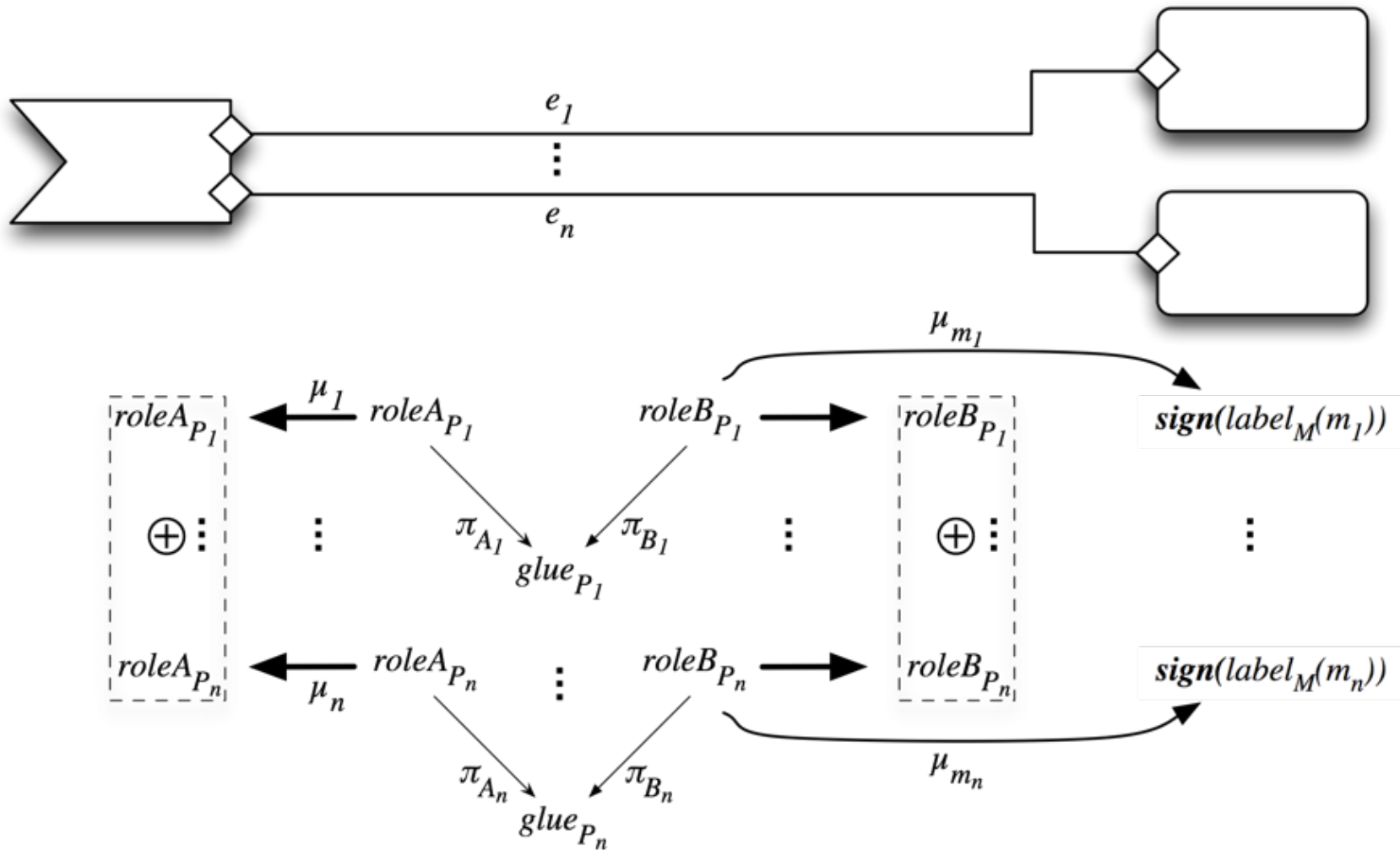
expanding the wires



Structured co-spans: an algebra of interaction protocols. Fiadeiro and Schmitt. LNCS 4624 (CALCO 2007)

expanding the wires

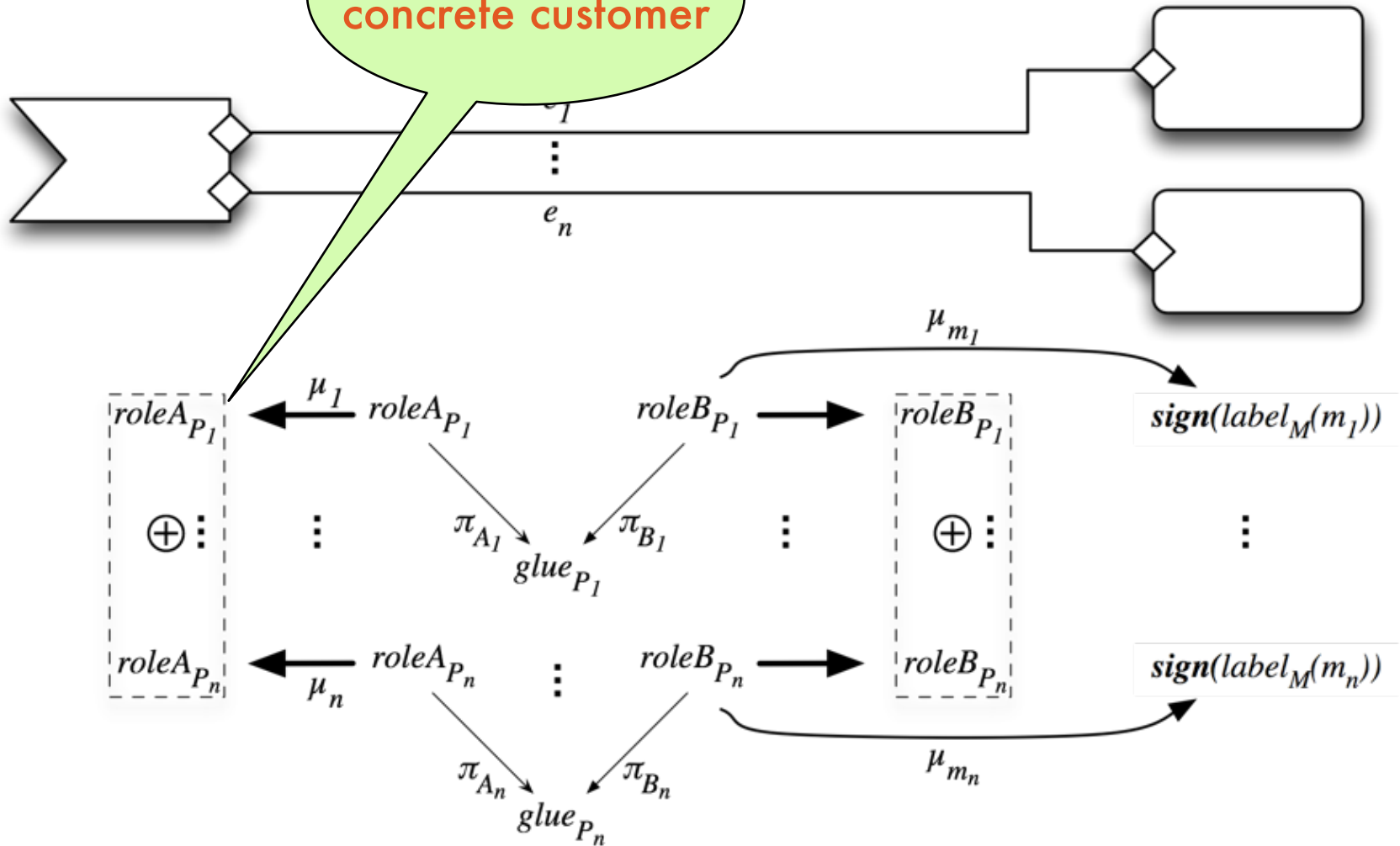
- for wires involving the provides interface



expanding the wires

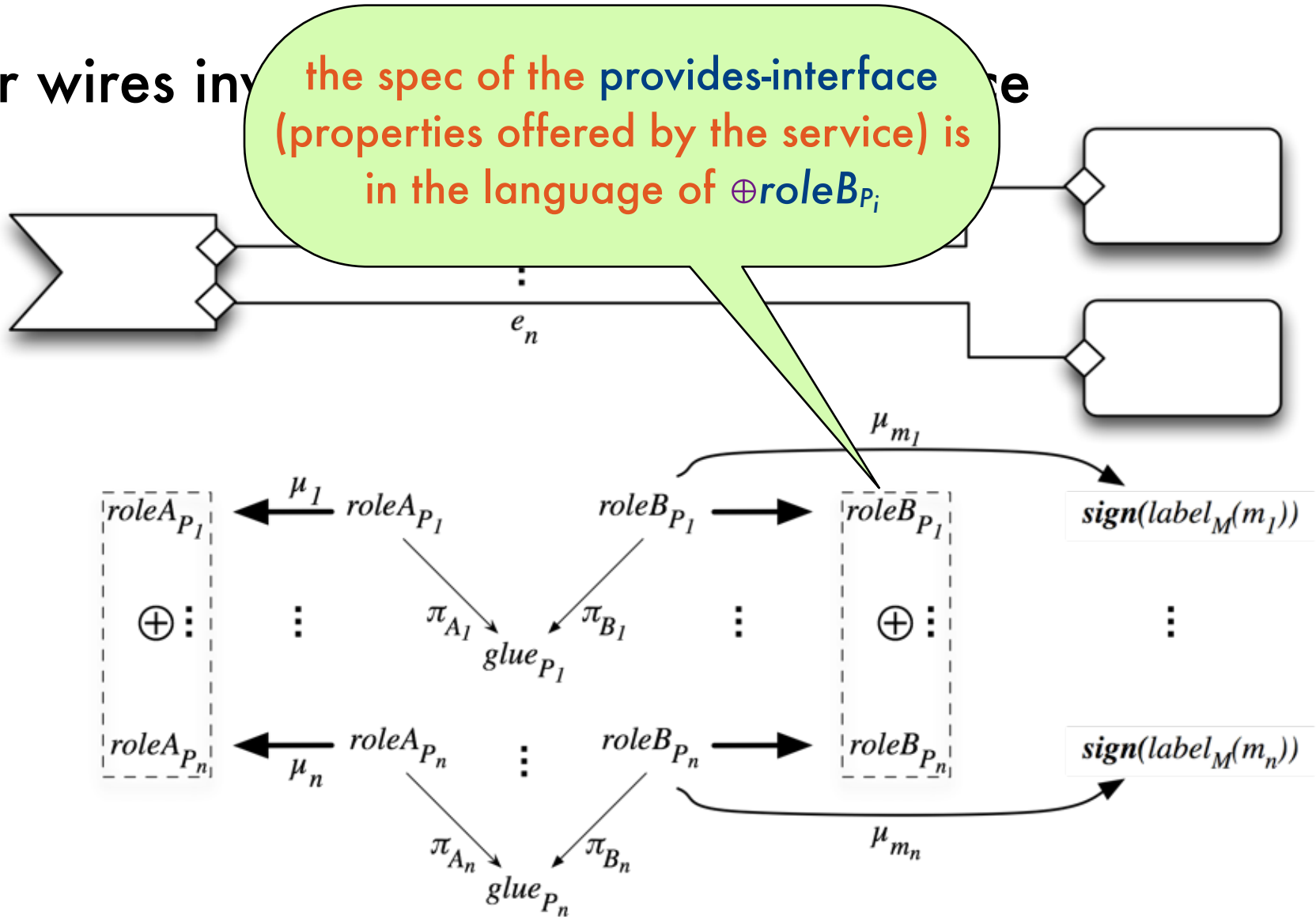
- for wires involved in a side interface

there is no concrete customer



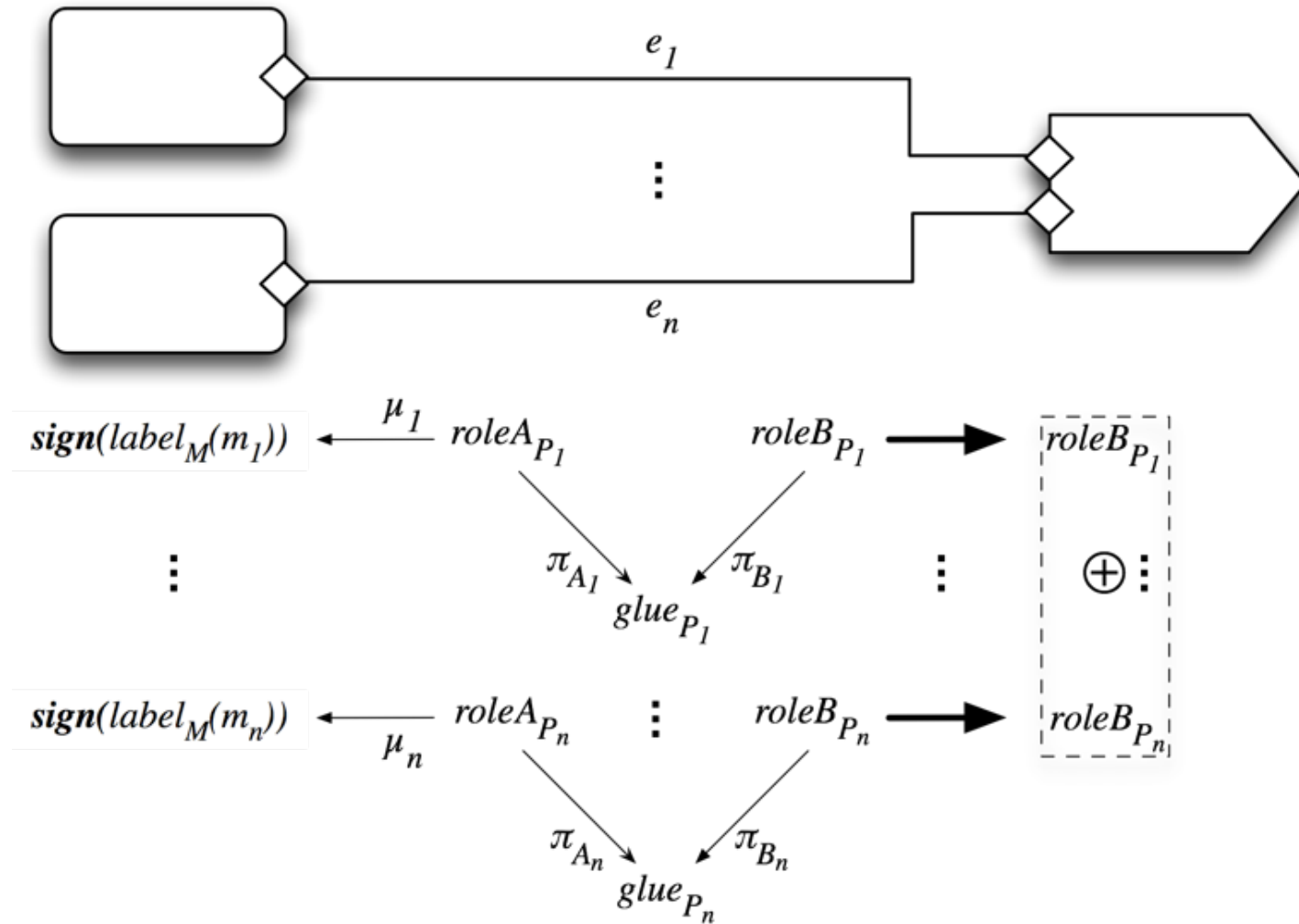
expanding the wires

- for wires in \mathcal{P} , the spec of the provides-interface (properties offered by the service) is in the language of $\oplus \text{role}B_{P_i}$



expanding the wires

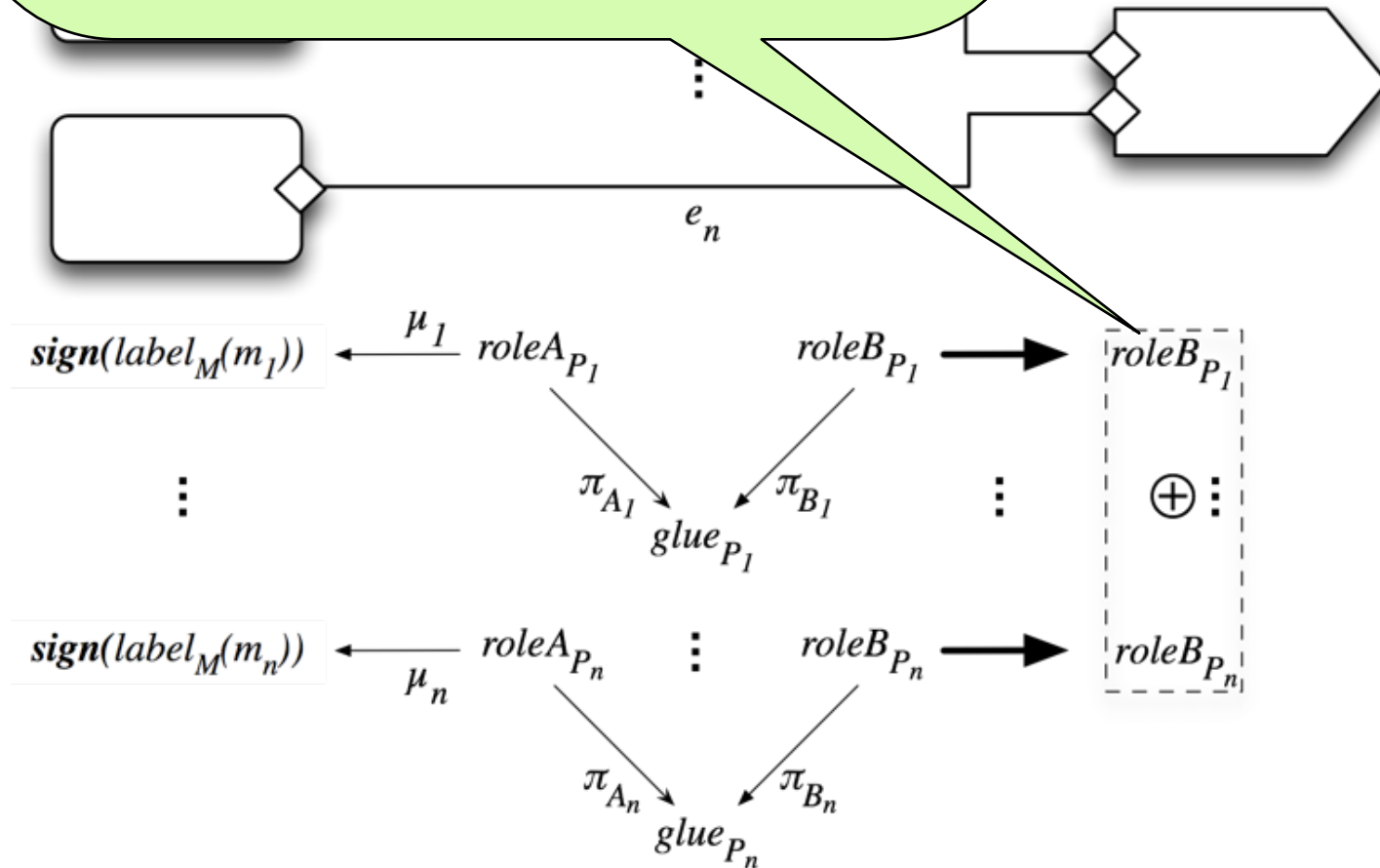
- for wires involving the requires interface



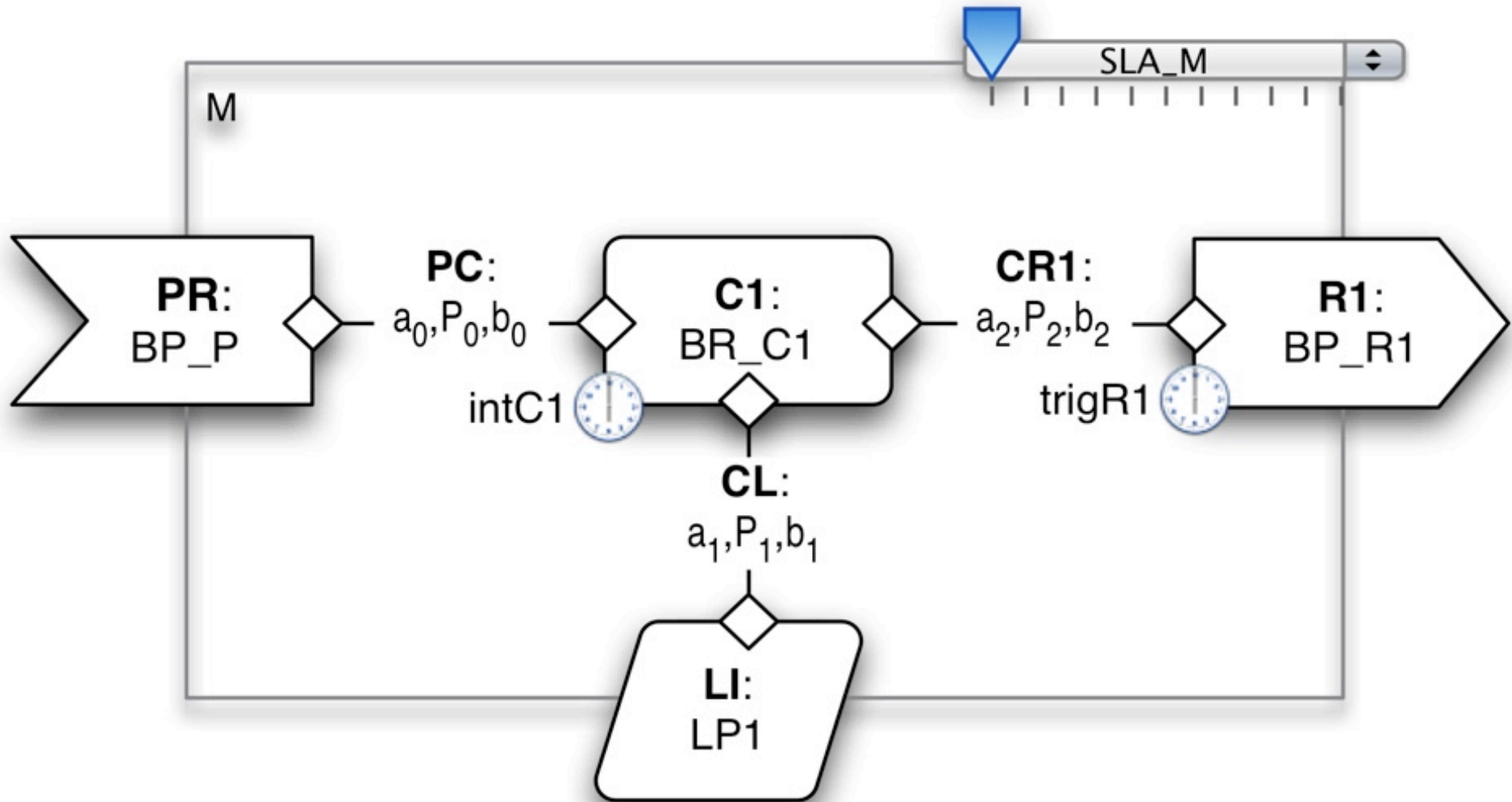
expanding the wires

for the spec of the requires-interface

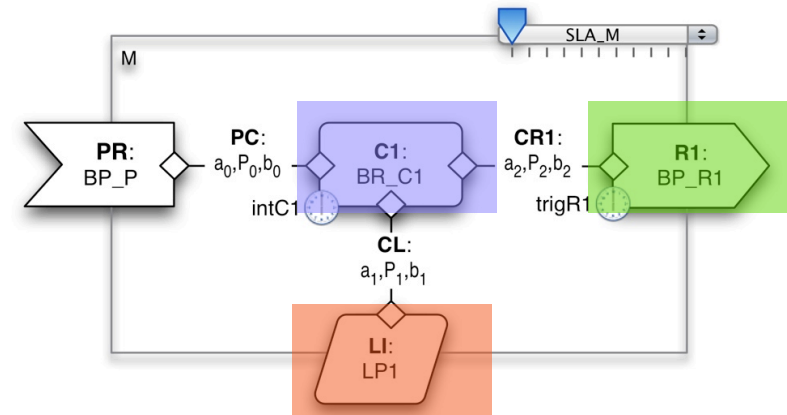
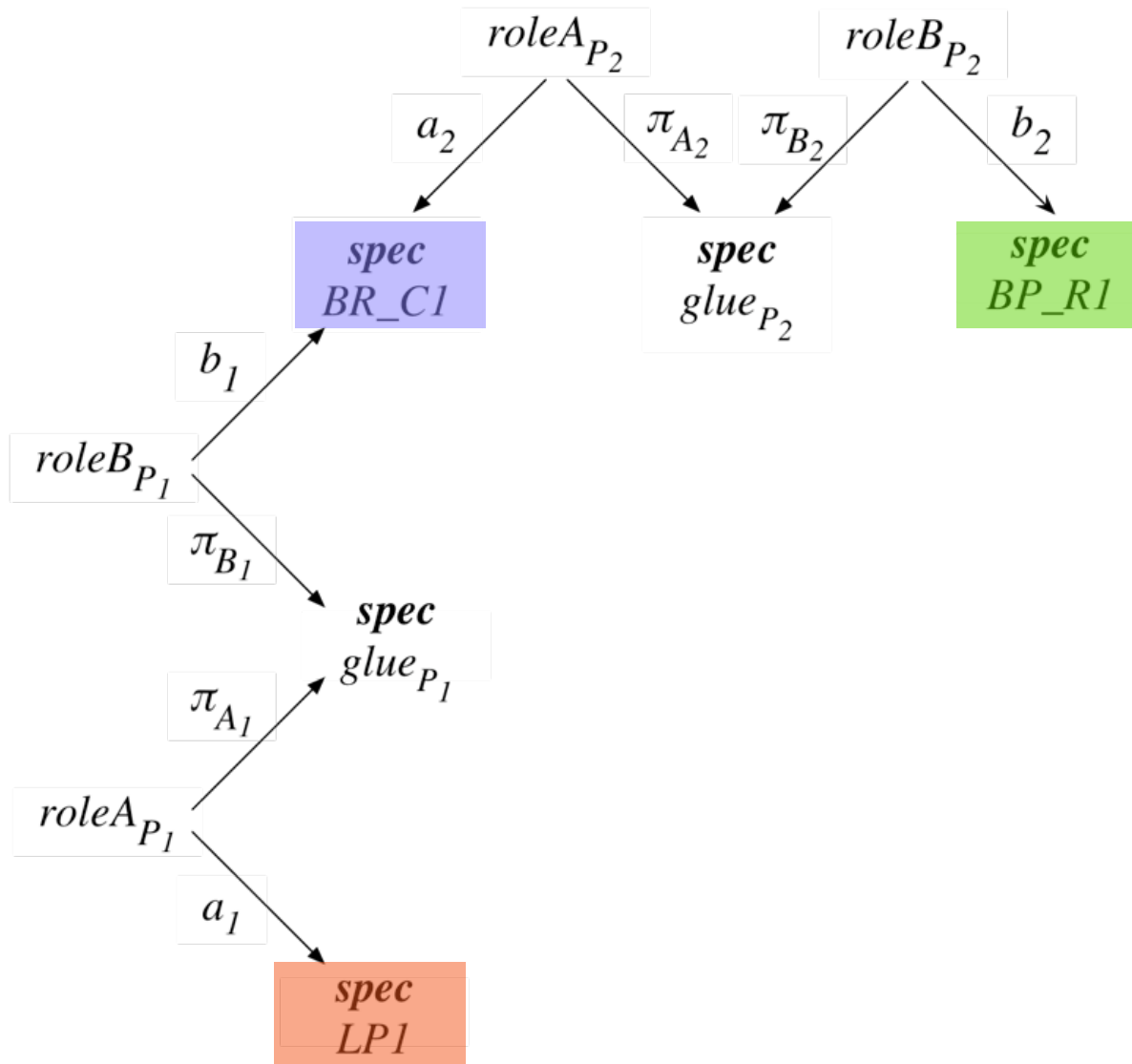
(properties required of the external service)
is in the language of $\oplus \text{roleB}_{P_i}$



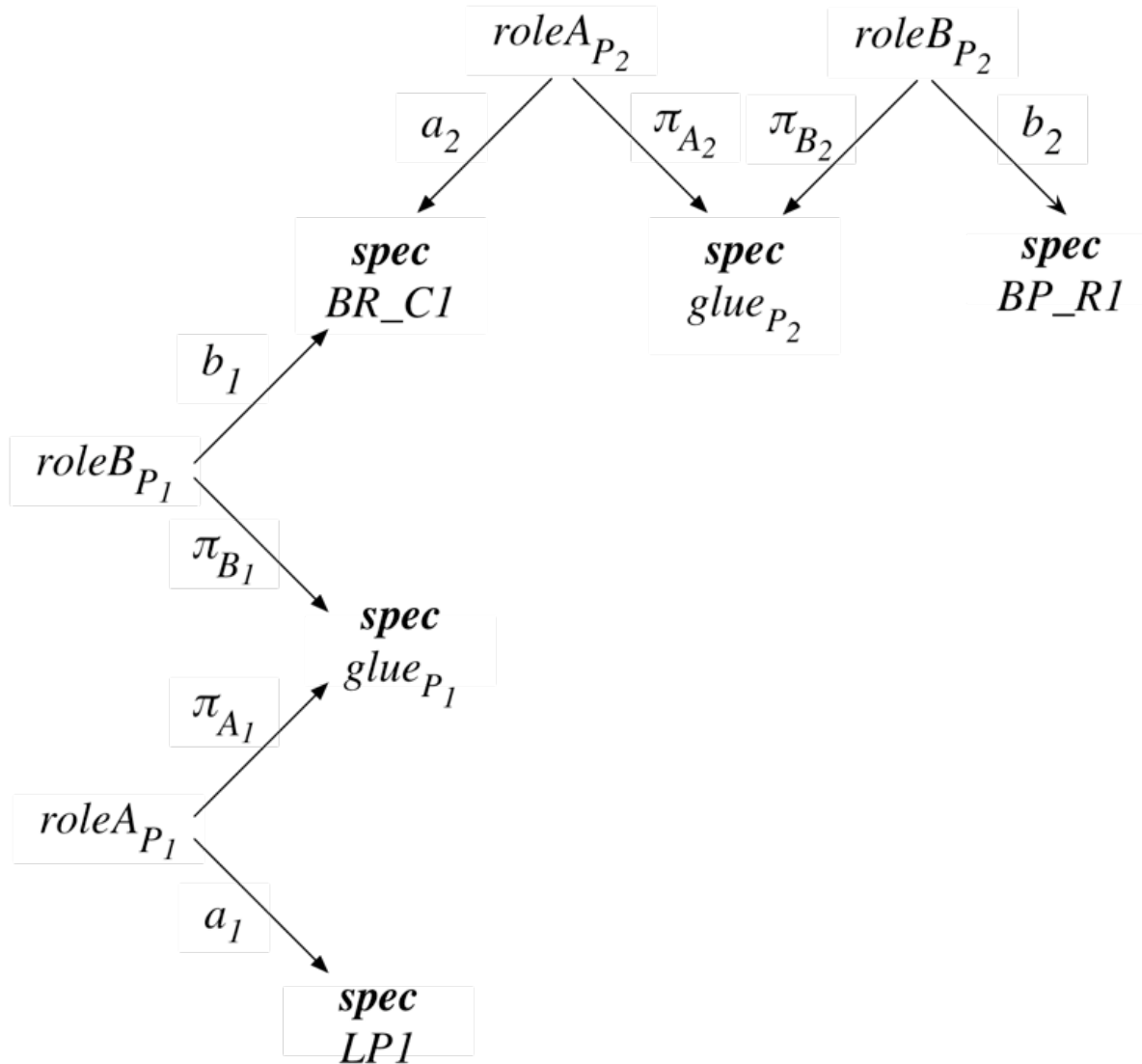
expanding the modules



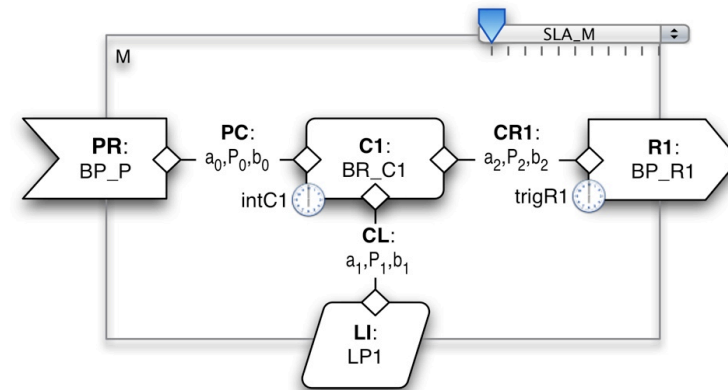
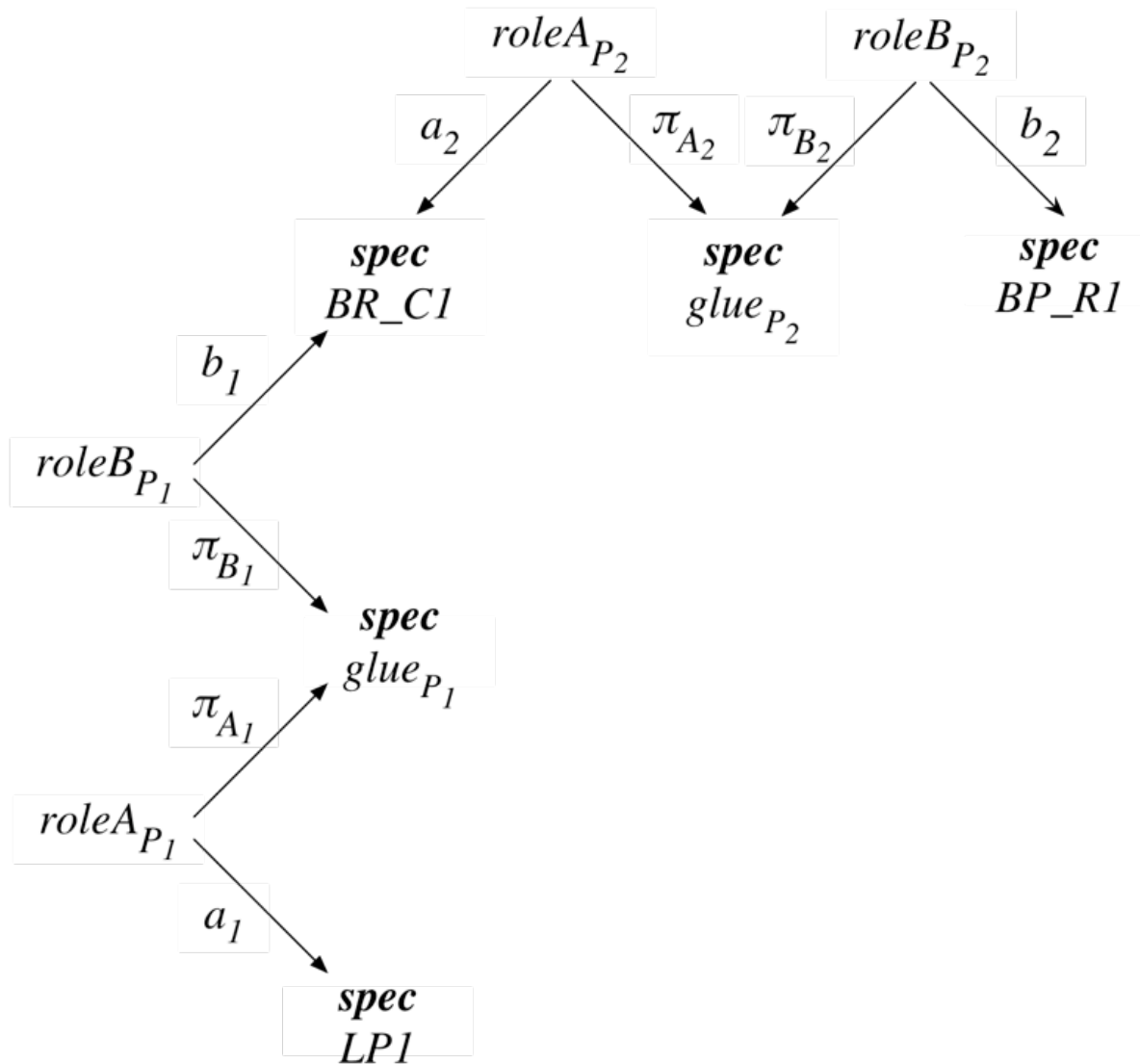
expanding the modules



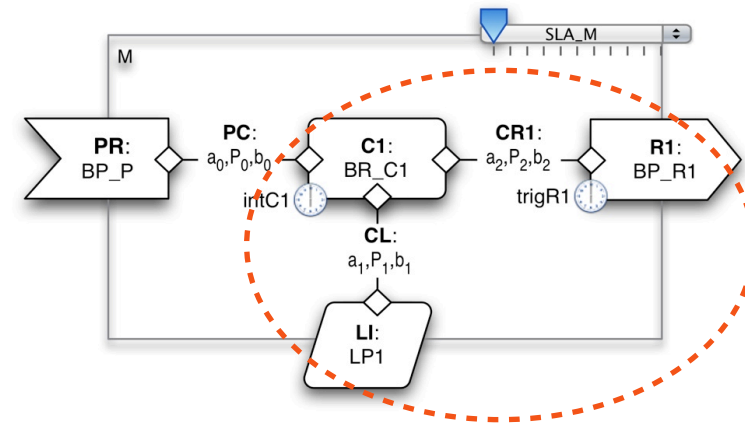
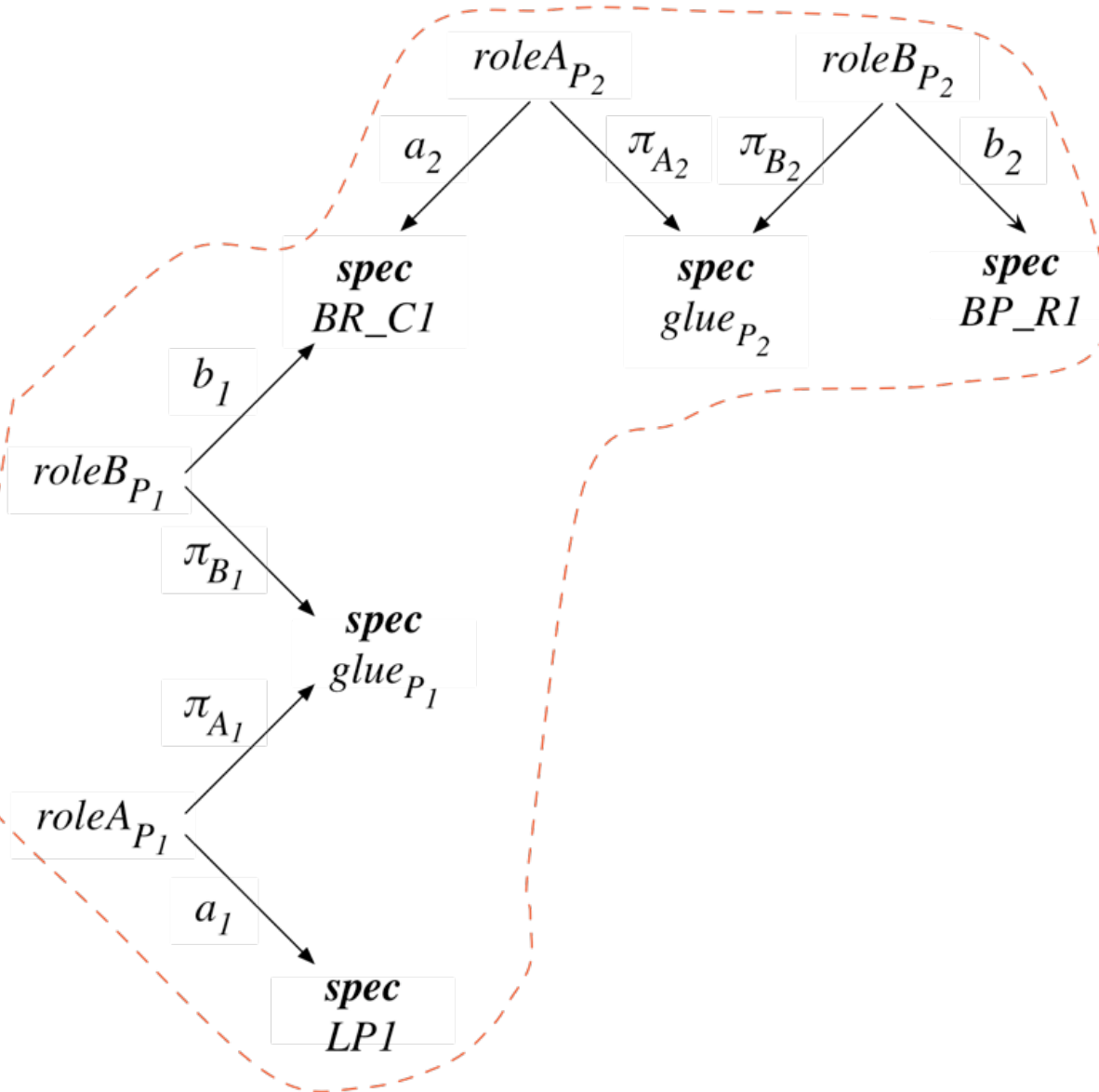
correctness property



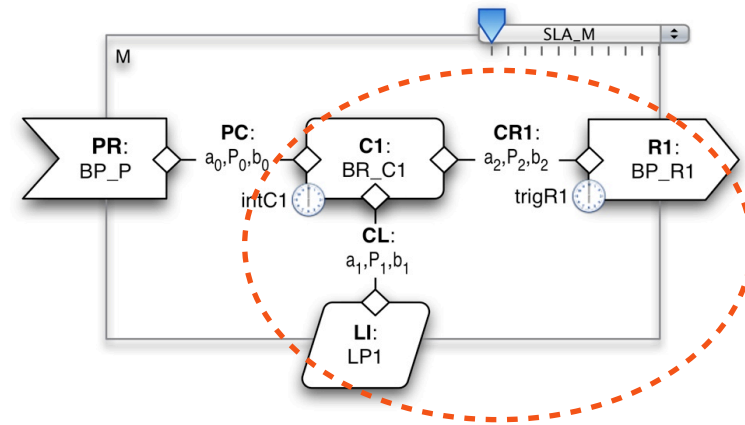
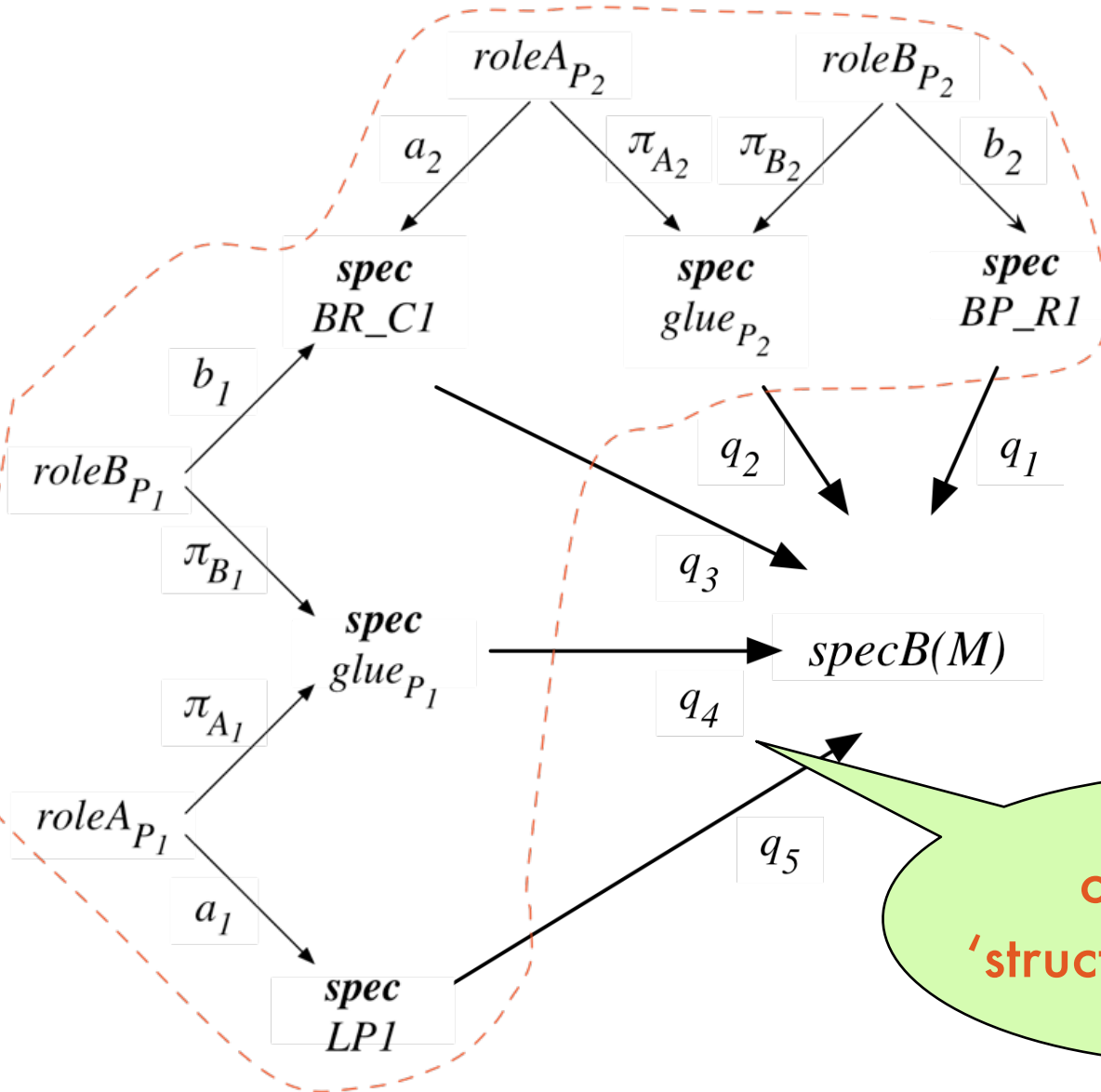
correctness property



correctness property

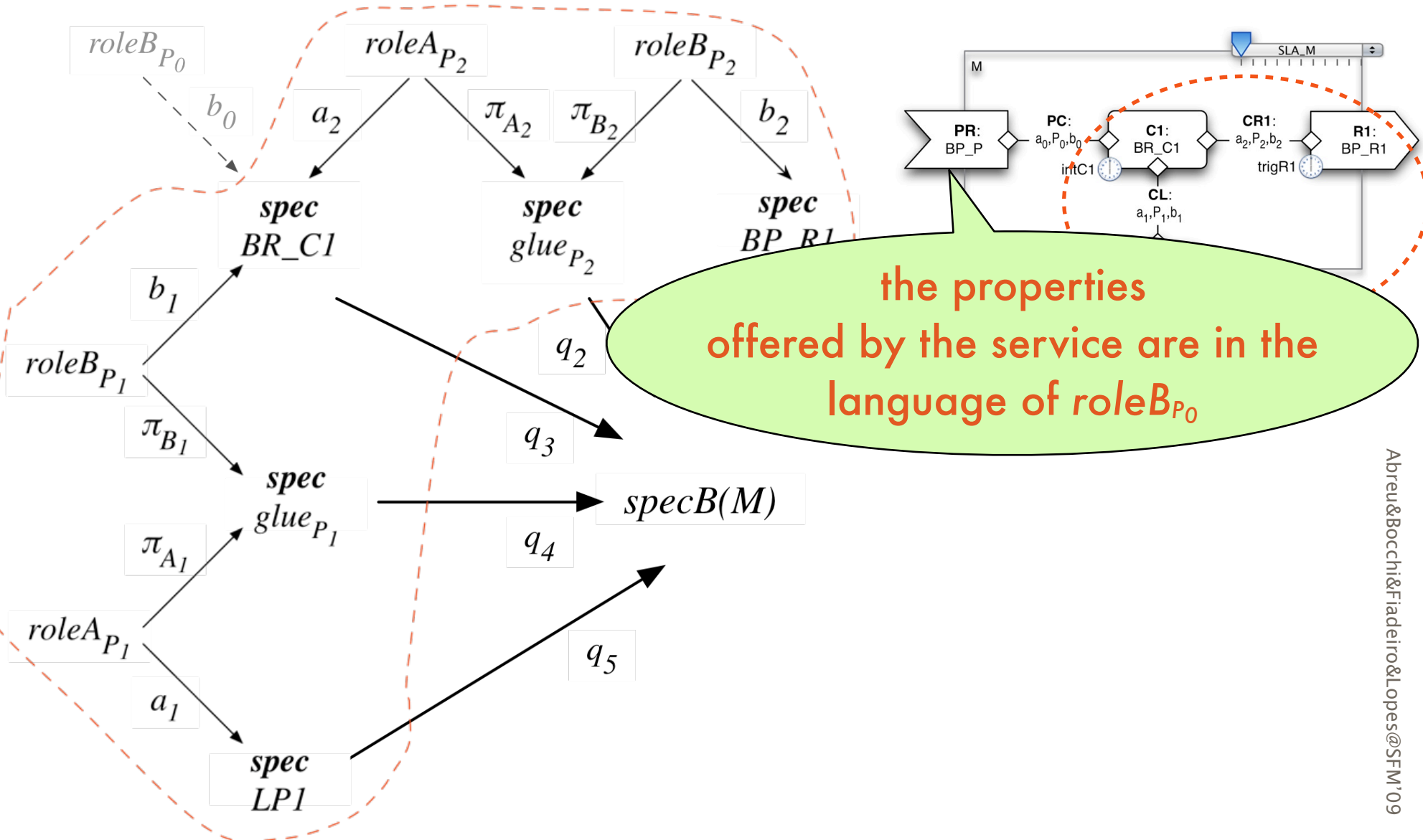


correctness property

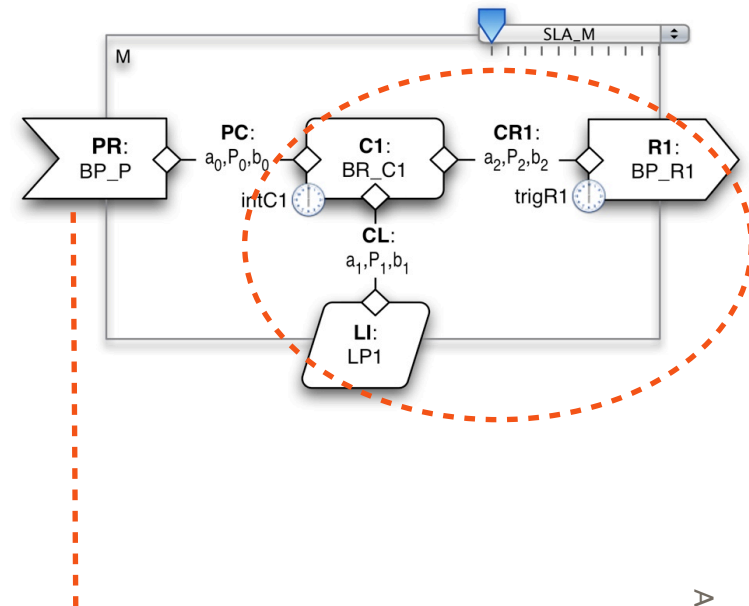
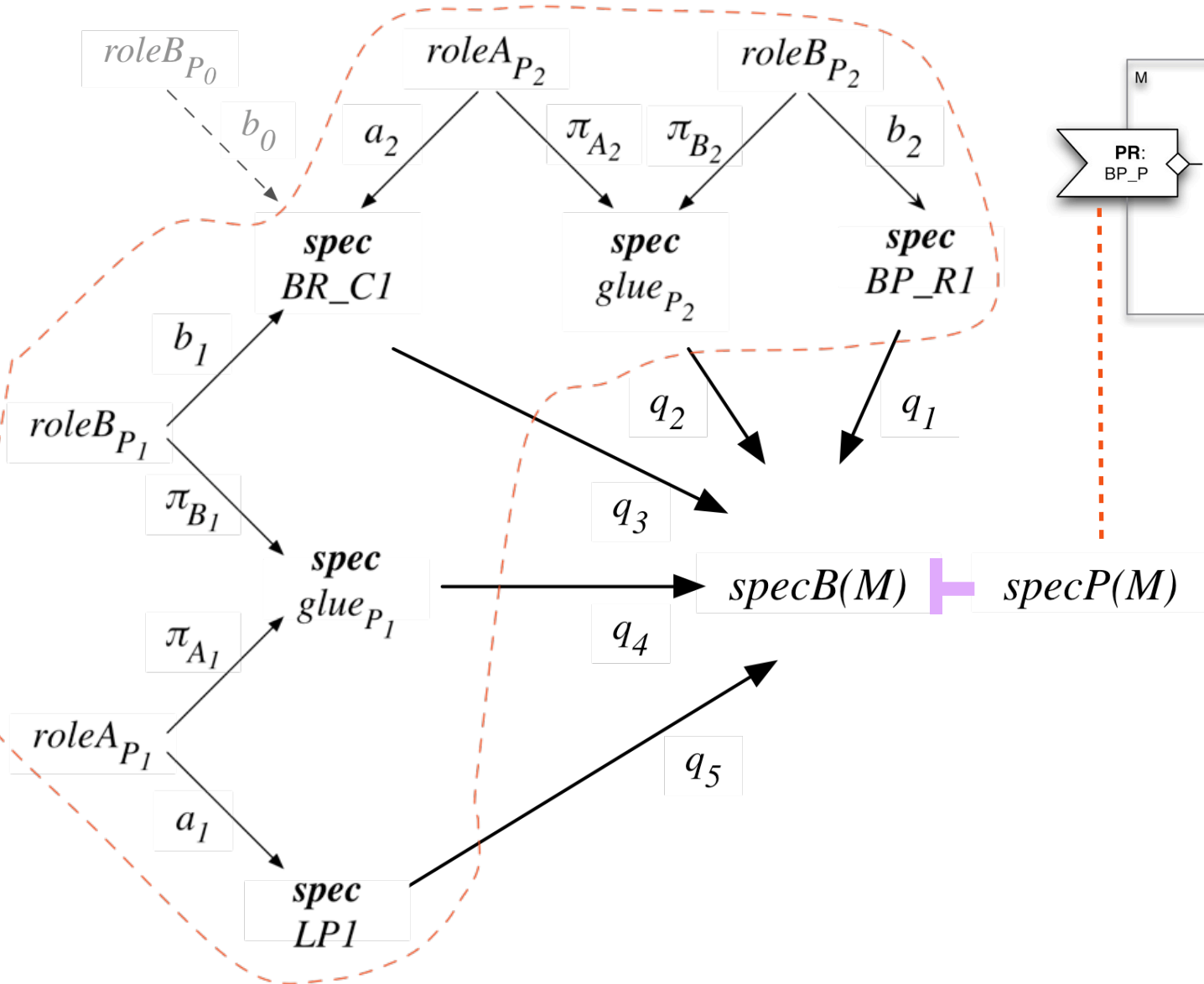


colimit of a 'structured' diagram

correctness property



correctness property



configuration policies

Configuration policies model the **dynamic** aspects of services

configuration policies

Configuration policies model the **dynamic** aspects of services

- **Internal** configuration policies – concern service instantiation:

configuration policies

Configuration policies model the **dynamic** aspects of services

- **Internal** configuration policies – concern service instantiation:
 - the initialisation/termination of the components that instantiate business roles

```
MA: MortgageAgent
  intBR ⌚ init: S=INITIAL
  intBR ⌚ term: S=FINAL
```

configuration policies

Configuration policies model the **dynamic** aspects of services

■ **Internal** configuration policies – concern service instantiation:

- the initialisation/termination of the components that instantiate business roles

```
MA: MortgageAgent
    intBR 🕒 init: S=INITIAL

    intBR 🕒 term: S=FINAL
```

- the triggering of the discovery of required services

```
LE: Lender
    intLE 🕒 trigger: getproposal 🔔?

BA: Bank
    intBA 🕒 trigger: default

IN: Insurance
    intIN 🕒 trigger: default
```


configuration policies

Configuration policies model the **dynamic** aspects of services

■ **Internal** configuration policies – concern service instantiation:

- the initialisation/termination of the components that instantiate business roles

```
MA: MortgageAgent
  intBR ⌚ init: S=INITIAL

  intBR ⌚ term: S=FINAL
```

- the triggering of the discovery of required services

```
LE: Lender
  intLE ⌚ trigger: getproposal 🔔?

BA: Bank
  intBA ⌚ trigger: default

IN: Insurance
  intIN ⌚ trigger: default
```

■ **External** configuration policies – concern service discovery and selection (service-level agreements)

SLAs

SLAs

- In SRML we adopt a **c-semiring-based approach to constraint satisfaction and optimisation** that can express classical, fuzzy, weighted,..., constraint satisfaction problems



S.Bistarelli, U. Montanari, F. Rossi (1997)
Semiring-based constraint satisfaction and optimization.
Journal of the ACM (JACM) 44(2): 201-236

SLAs

- In SRML we adopt a **c-semiring-based approach to constraint satisfaction and optimisation** that can express classical, fuzzy, weighted,..., constraint satisfaction problems



S.Bistarelli, U. Montanari, F. Rossi (1997)
Semiring-based constraint satisfaction and optimization.
Journal of the ACM (JACM) 44(2): 201-236

- A **c-semiring** is an algebraic structure $\langle A, +, \times, 0, 1 \rangle$ where:
 - A is a set of values such that $\{0, 1\} \in A$
 - $+$ is a binary operation on A that is commutative, associative, idempotent and with unit element 0
 - \times is another binary operation on A that is commutative, associative with unit element 1 and absorbing element 0
 - \times distributes over $+$

SLAs

- In SRML we adopt a **c-semiring-based approach to constraint satisfaction and optimisation** that can express classical, fuzzy, weighted,..., constraint satisfaction problems

A is the domain of the degree of satisfaction
 $\langle \{0,1\}, \vee, \wedge, 0, 1 \rangle$ for yes/no
 $\langle [0,1], \max, \min, 0, 1 \rangle$ for intermediate degrees

- A **c-semiring** is an algebraic structure $\langle A, +, \times, 0, 1 \rangle$ where:
 - A is a set of values such that $\{0,1\} \in A$
 - $+$ is a binary operation on A that is commutative, associative, idempotent and with unit element 0
 - \times is another binary operation on A that is commutative, associative with unit element 1 and absorbing element 0
 - \times distributes over $+$

SLAs

- In SRML we adopt a **c-semiring-based approach to constraint satisfaction and optimisation** that can express classical, fuzzy, weighted,..., constraint satisfaction problems

A is the domain of the degree of satisfaction
 $\langle \{0,1\}, \vee, \wedge, 0, 1 \rangle$ for yes/no
 $\langle [0,1], \max, \min, 0, 1 \rangle$ for intermediate degrees

- A **c-semiring** is an algebraic structure $\langle A, +, \times, 0, 1 \rangle$ where:

- A is a set of values such that $\{0,1\} \in A$

- $+$ is a binary operation on A that is commutative, associative, idempotent and with unit element 0

$+$ is a comparison primitive
 $a < b \Leftrightarrow a + b = b$
(b is better than a)

- \times is another binary operation on A that is commutative, associative with unit element 1 and absorbing element 0

- \times distributes over $+$

SLAs

- In SRML we adopt a **c-semiring-based approach to constraint satisfaction and optimisation** that can express classical, fuzzy, weighted,..., constraint satisfaction problems

A is the domain of the degree of satisfaction
 $\langle \{0,1\}, \vee, \wedge, 0, 1 \rangle$ for yes/no
 $\langle [0,1], \max, \min, 0, 1 \rangle$ for intermediate degrees

- A **c-semiring** is an algebraic structure $\langle A, +, \times, 0, 1 \rangle$ where:

- A is a set of values such that $\{0,1\} \in A$

- + is a binary operation on A that is commutative, associative, idempotent and with unit element 0

+ is a comparison primitive
 $a < b \Leftrightarrow a + b = b$
(b is better than a)

- \times is another binary operation on A that is commutative, associative with unit element 1 and absorbing element 0

\times is a composition primitive

- \times distributes over +

SLAs

- A constraint system is a triple $\langle S, D, V \rangle$ where
 - S is a C-semiring
 - D is a finite set (domain of possible elements taken by the variables)
 - V is a totally ordered set (of variables)
- A constraint is a pair $\langle \text{def}, \text{con} \rangle$ where
 - $\text{con} \subseteq V$ is called the type of the constraint
 - $\text{def} : D|\text{con}| \rightarrow A$ is the value (mapping) of the constraint

$\langle a_1, a_2, \dots, a_{|\text{con}|} \rangle$



degree of satisfaction

SLA variables

SLA variables

■ standard configuration variables include

● external interfaces

- *availability*, *responseTime*
- *ServiceId* – service identifiers (e.g., URI's).

● for wires

- *wire.Delay* – the maximum delivery delay for events sent over *wire*

● for interactions

- *interaction*[⚡] for every interaction of type *r&s* – the length of time the pledge is valid after *interaction*_✉ is issued

an example

EXTERNAL POLICY

$\langle [0..1], \max, \min, 0, 1 \rangle$

SLA VARIABLES

MA.CHARGE, MA.getProposal^{*},

LE.ServiceID,

LE.requestMortgage^{*}

CONSTRAINTS

$C_1: \{c: \text{MA.CHARGE}, t: \text{MA.getProposal}^*\}$

$$\begin{cases} 1 & \text{if } t \leq 10 * c \\ 1 + 2 * c - 0.2 * t & \text{if } 10 * c < t \leq 5 + 10 * c \\ 0 & \text{otherwise} \end{cases}$$

$C_2: \{s: \text{LE.ServiceId}\}$

$$\begin{cases} 1 & \text{if } s \in \text{BR.lenders} \\ 0 & \text{otherwise} \end{cases}$$

$C_3: \{t_1: \text{MA.getProposal}^*, t_2: \text{LE.requestMortgage}^*\}$

$$\begin{cases} 1 & \text{if } t_2 > t_1 + \text{CC.Delay} + \text{CL.Delay} \\ 0 & \text{otherwise} \end{cases}$$

an example

EXTERNAL POLICY


$\langle [0..1], \max, \min, 0, 1 \rangle$

SLA VARIABLES

MA.CHARGE, MA.getProposal^{*},
LE.ServiceID,
LE.requestMortgage^{*}

CONSTRAINTS

$C_1: \{c:MA.CHARGE, t:MA.getProposal[*]\}$

$$\begin{cases} 1 & \text{if } t \leq 10 * c \\ 1 + 2 * c - 0.2 * t & \text{if } 10 * c < t \leq 5 + 10 * c \\ 0 & \text{otherwise} \end{cases}$$


$C_2: \{s:LE.ServiceId\}$

$$\begin{cases} 1 & \text{if } s \in BR.lenders \\ 0 & \text{otherwise} \end{cases}$$

$C_3: \{t_1:MA.getProposal[*], t_2:LE.requestMortgage[*]\}$

$$\begin{cases} 1 & \text{if } t_2 > t_1 + CC.Delay + CL.Delay \\ 0 & \text{otherwise} \end{cases}$$

the greater the CHARGE applied to the base price of the brokerage service, the longer the interval during which the proposal is valid

an example

EXTERNAL POLICY

$\langle [0..1], \text{max}, \text{min}, 0, 1 \rangle$

SLA VARIABLES

MA.CHARGE, MA.getProposal^{*},

LE.ServiceID,

LE.requestMortgage^{*}

CONSTRAINTS

$C_1: \{c: \text{MA.CHARGE}, t: \text{MA.getProposal}^*\}$

$$\begin{cases} 1 & \text{if } t \leq 10 * c \\ 1 + 2 * c - 0.2 * t & \text{if } 10 * c < t \leq 5 + 10 * c \\ 0 & \text{otherwise} \end{cases}$$

$C_2: \{s: \text{LE.ServiceId}\}$

$$\begin{cases} 1 & \text{if } s \in \text{BR.lenders} \\ 0 & \text{otherwise} \end{cases}$$

$C_3: \{t_1: \text{MA.getProposal}^*, t_2: \text{LE.requestMortgage}^*\}$

$$\begin{cases} 1 & \text{if } t_2 > t_1 + \text{CC.Delay} + \text{CL.Delay} \\ 0 & \text{otherwise} \end{cases}$$

the greater the CHARGE applied to the base price of the brokerage service, the longer the interval during which the proposal is valid

the selected lender must belong to the set BR.lenders

an example

EXTERNAL POLICY

$\langle [0..1], \text{max}, \text{min}, 0, 1 \rangle$

SLA VARIABLES

MA.CHARGE, MA.getProposal^{*},
LE.ServiceID,
LE.requestMortgage^{*}

CONSTRAINTS

$C_1: \{c: \text{MA.CHARGE}, t: \text{MA.getProposal}^*\}$

$$\begin{cases} 1 & \text{if } t \leq 10 * c \\ 1 + 2 * c - 0.2 * t & \text{if } 10 * c < t \leq 5 + 10 * c \\ 0 & \text{otherwise} \end{cases}$$

$C_2: \{s: \text{LE.ServiceId}\}$

$$\begin{cases} 1 & \text{if } s \in \text{BR.lenders} \\ 0 & \text{otherwise} \end{cases}$$

$C_3: \{t_1: \text{MA.getProposal}^*, t_2: \text{LE.requestMortgage}^*\}$

$$\begin{cases} 1 & \text{if } t_2 > t_1 + \text{CC.Delay} + \text{CL.Delay} \\ 0 & \text{otherwise} \end{cases}$$

the validity of the loan proposal offered by the lender must be greater than the sum of the validity offered to the customer and the delays of the wires

the greater the CHARGE applied to the base price of the brokerage service, the longer the interval during which the proposal is valid

the selected lender must belong to the set *BR.lenders*

recalling...

recalling...

■ Static aspects:

- How can we account for the behaviour of services provided by collections of interconnected parties? – orchestration, conversation protocols (pledges, compensations, ...)

■ Dynamic aspects:

- How can we account for the run-time aspects of service-oriented systems that result from the SOA middleware mechanisms of service discovery, instantiation and binding?

the dynamic aspect of services

the dynamic aspect of services

- Services add a '**business**' layer of abstraction over a component infrastructure in sense that they structure the evolution of software applications seen as systems of interconnected components

the dynamic aspect of services

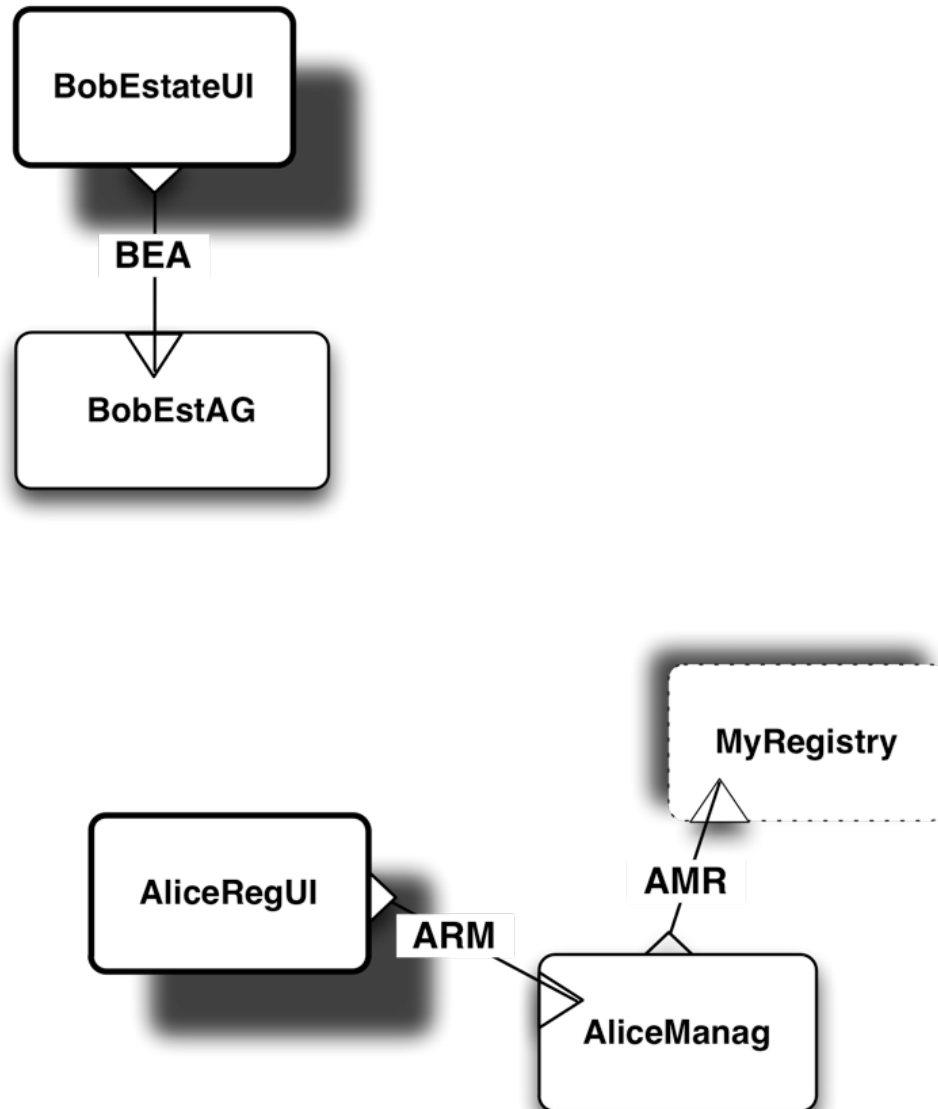
- Services add a **'business'** layer of abstraction over a component infrastructure in sense that they structure the evolution of software applications seen as systems of interconnected components
 - From structured programming to **'structured interactions'**
 - Services address the **'social'** complexity of software-intensive systems

states as configurations

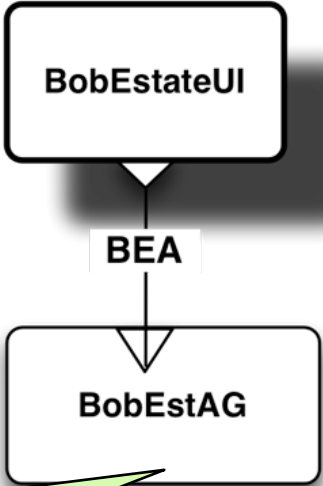
A configuration \mathcal{F} consists of:

- A simple graph whose **nodes are components**, and **edges are wires**.
- A labelling function that assigns a **state** to every node and edge
 - states are as discussed earlier on...

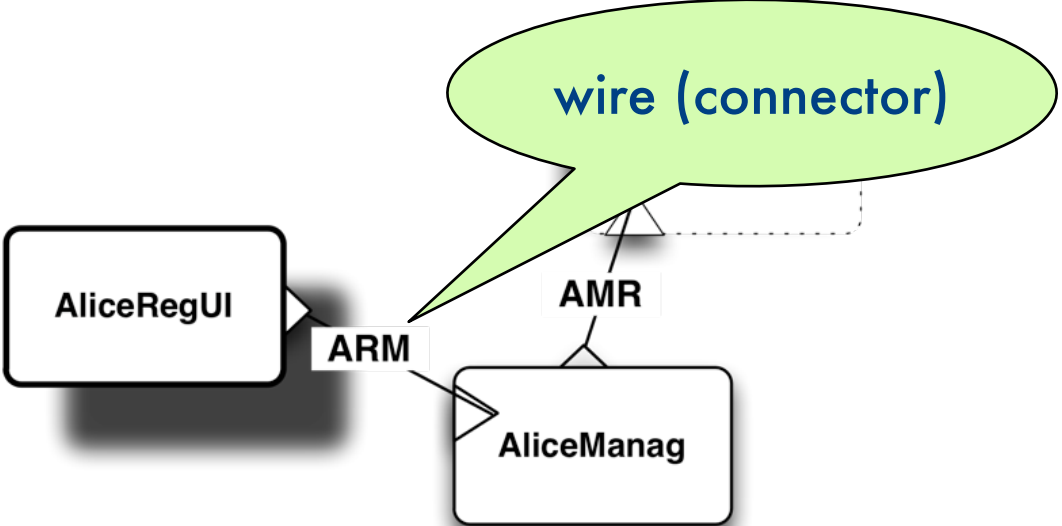
an example



an example

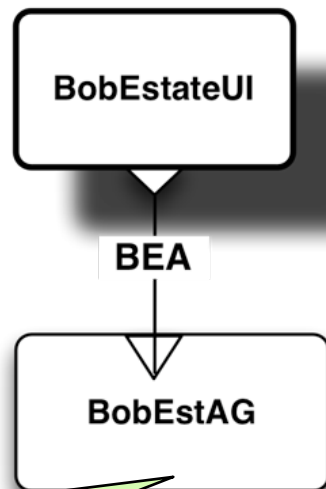


component



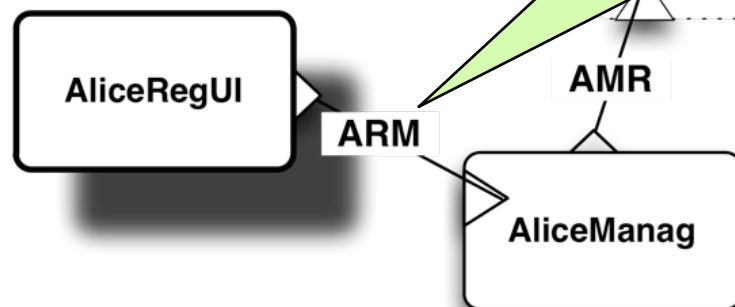
wire (connector)

an example



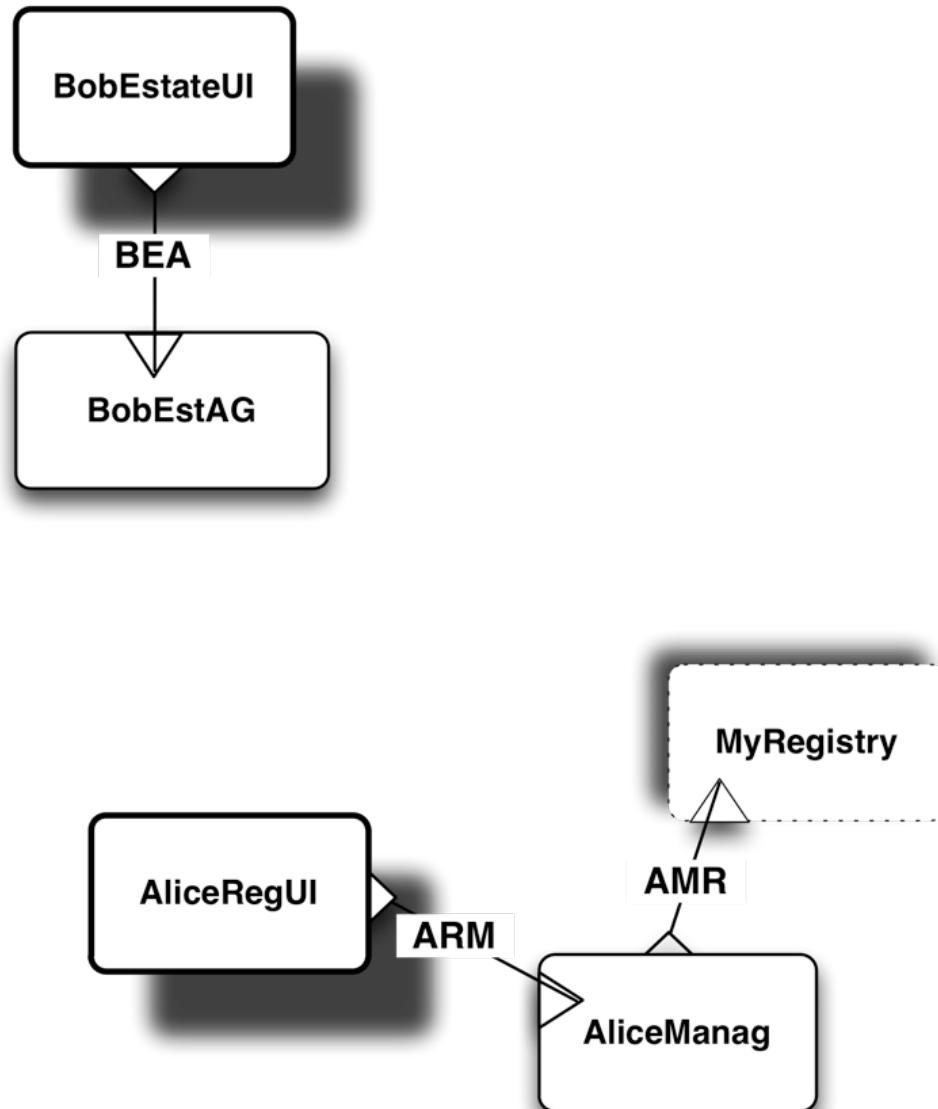
Although we use the same icons,
these are instances,
not specifications

component

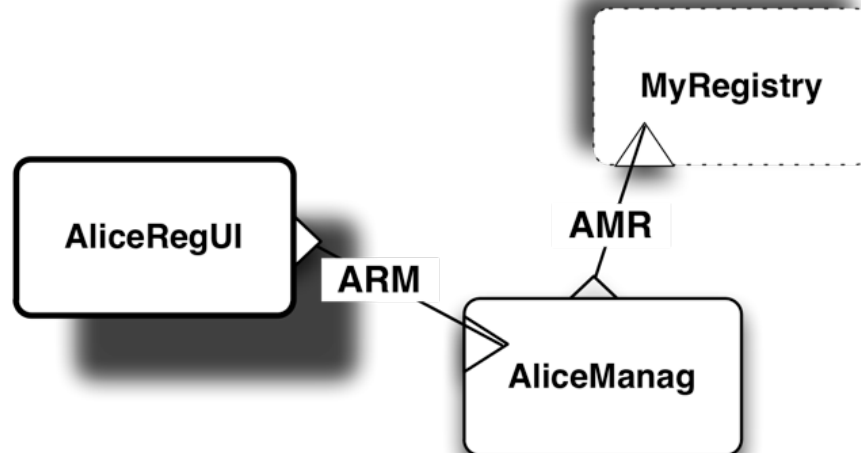
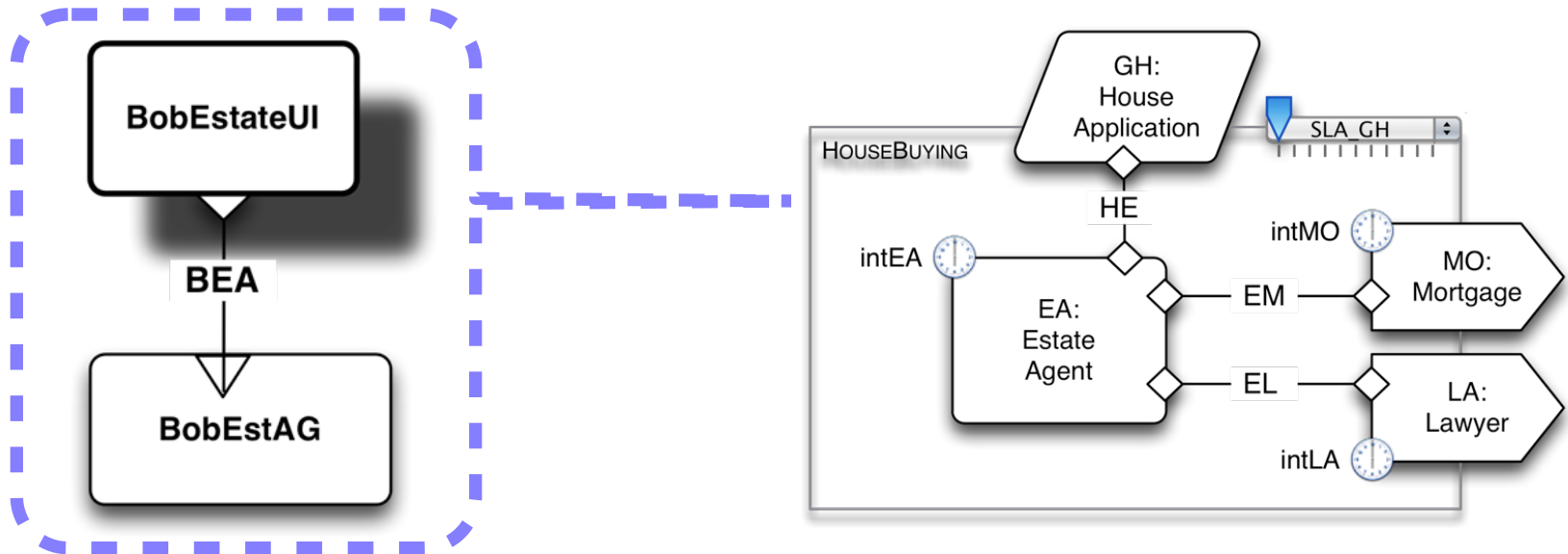


wire (connector)

configurations are typed

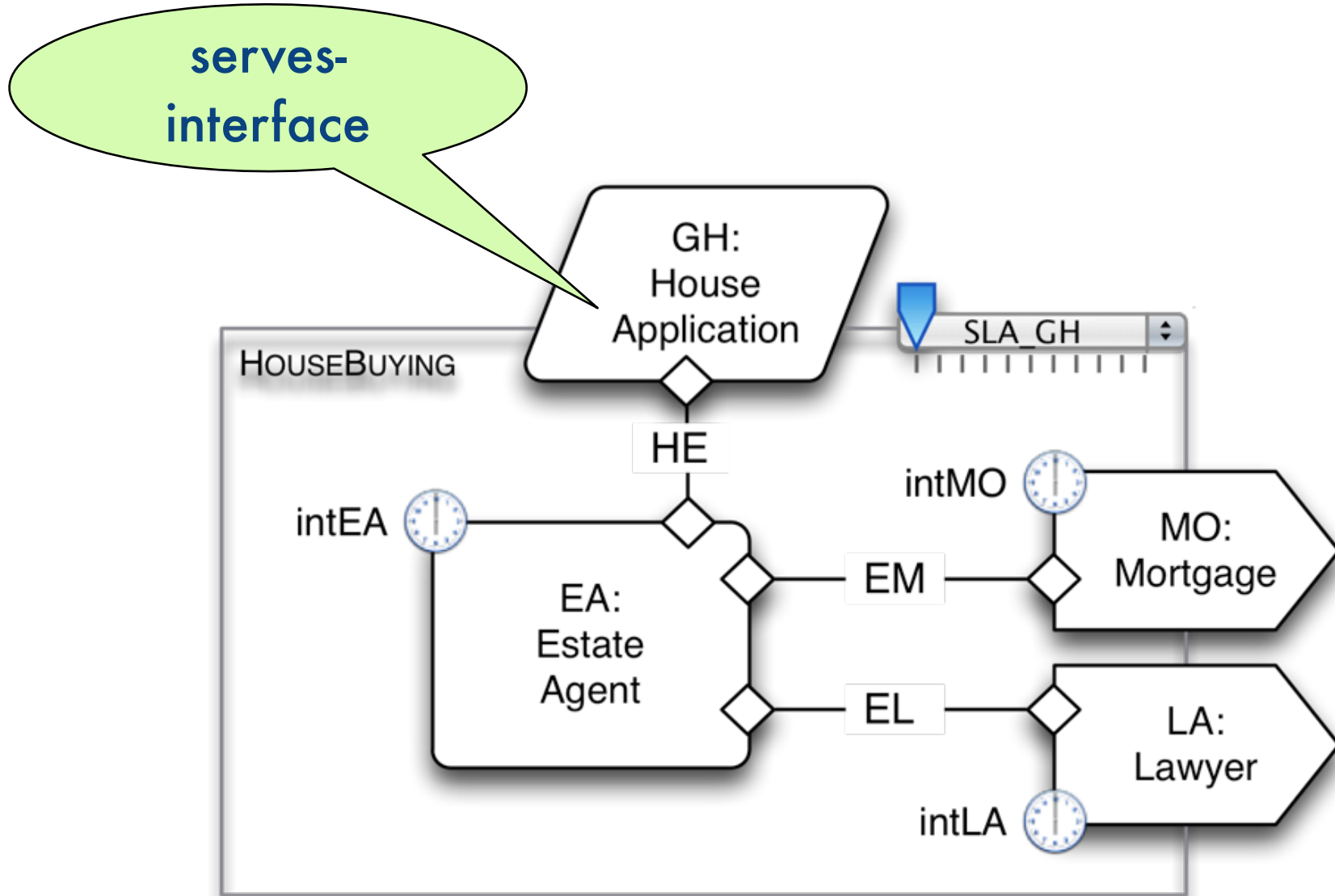


configurations are typed



activity modules

activity modules



modules as graphs

modules as graphs

An activity module M consists of:

modules as graphs

An activity module M consists of:

■ A labelled graph:

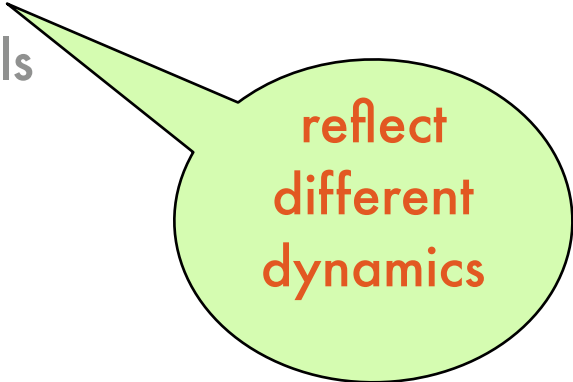
- Nodes are classified as:
 - $\text{components}(M)$, labelled with business roles
 - $\text{uses}(M)$, labelled with layer protocols
 - $\text{requires}(M)$, labelled with business protocols
 - $\{\text{serves}(M)\}$, labelled with a layer protocol
- Edges (wires) are labelled with connectors (interaction glue and attachments)

modules as graphs

An activity module M consists of:

■ A labelled graph:

- Nodes are classified as:
 - $\text{components}(M)$, labelled with business roles
 - $\text{uses}(M)$, labelled with layer protocols
 - $\text{requires}(M)$, labelled with business protocols
 - $\{\text{serves}(M)\}$, labelled with a layer protocol
- Edges (wires) are labelled with connectors (interaction glue and attachments)



reflect
different
dynamics

modules as graphs

An activity module M consists of:

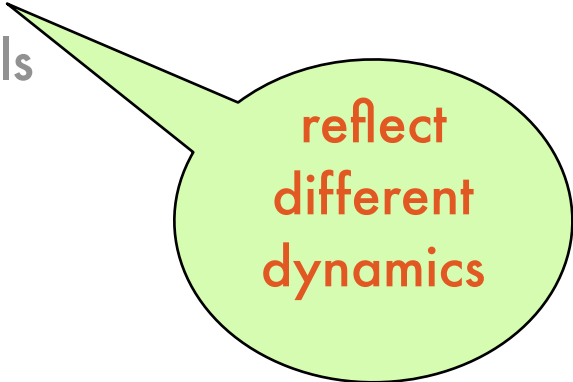
■ A labelled graph:

● Nodes are classified as:

- $\text{components}(M)$, labelled with business roles
- $\text{uses}(M)$, labelled with layer protocols
- $\text{requires}(M)$, labelled with business protocols
- $\{\text{serves}(M)\}$, labelled with a layer protocol

● Edges (wires) are labelled with connectors
(interaction glue and attachments)

■ An internal configuration policy



reflect
different
dynamics

modules as graphs

An activity module M consists of:

■ A labelled graph:

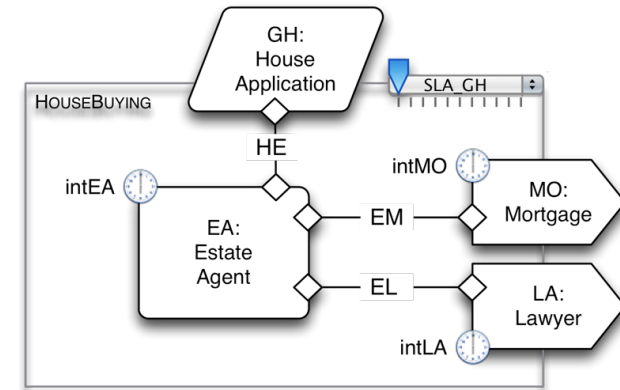
● Nodes are classified as:

- $\text{components}(M)$, labelled with business roles
- $\text{uses}(M)$, labelled with layer protocols
- $\text{requires}(M)$, labelled with business protocols
- $\{\text{serves}(M)\}$, labelled with a layer protocol

● Edges (wires) are labelled with connectors
(interaction glue and attachments)

■ An internal configuration policy

■ An external configuration policy



reflect
different
dynamics

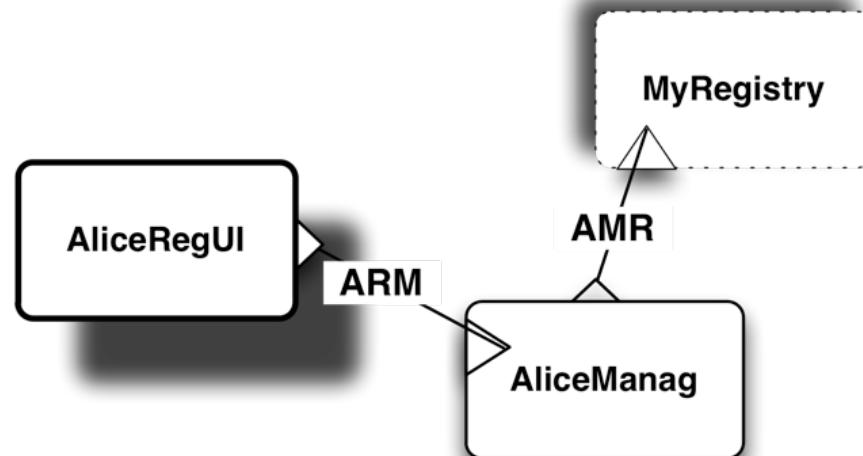
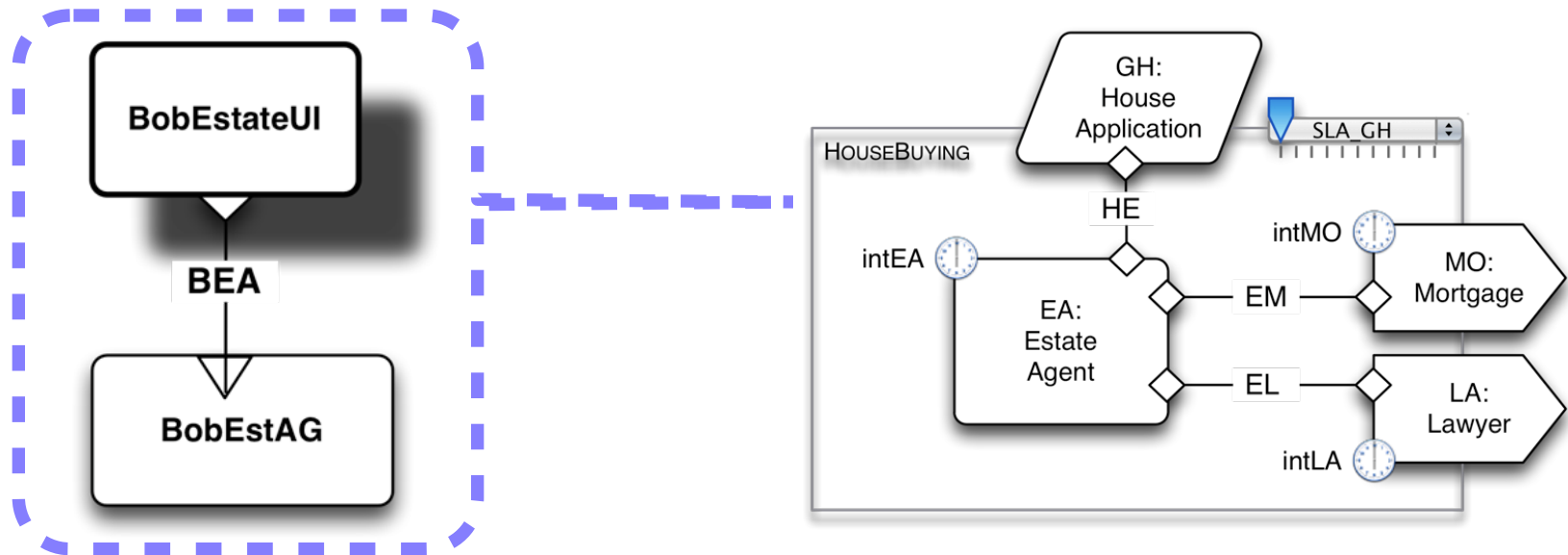
business configurations

A business configuration consists of:

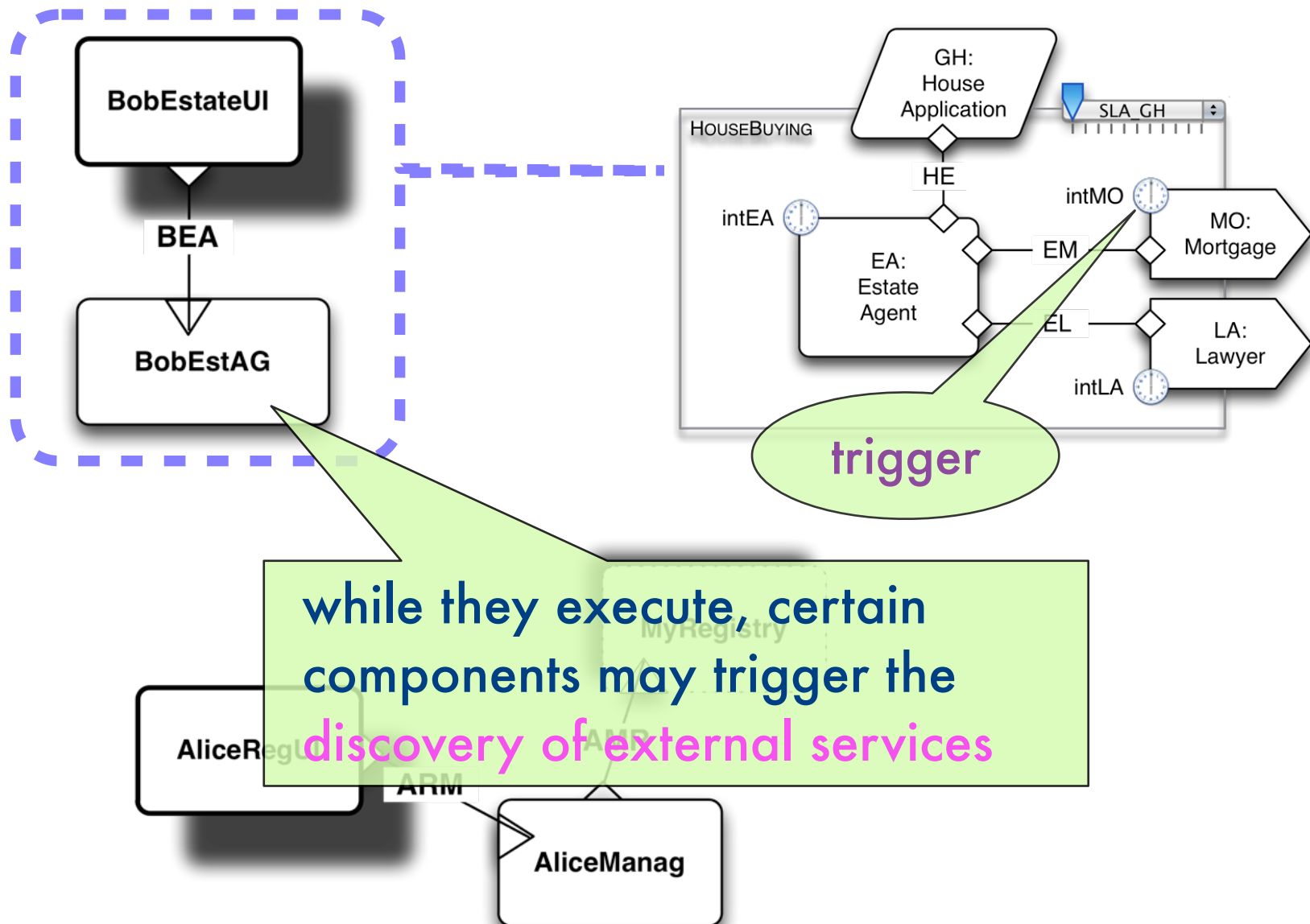
- A set of activity names (chosen from a domain)
- A state configuration \mathcal{F}
- A mapping \mathcal{B} that assigns a module $\mathcal{B}(a)$ to every activity a – the workflow performed by a in \mathcal{F}
- For every activity a , a homomorphism $\mathcal{B}(a)$ of graphs between the body of $\mathcal{B}(a)$ and \mathcal{F}

This homomorphism makes configurations **reflective**

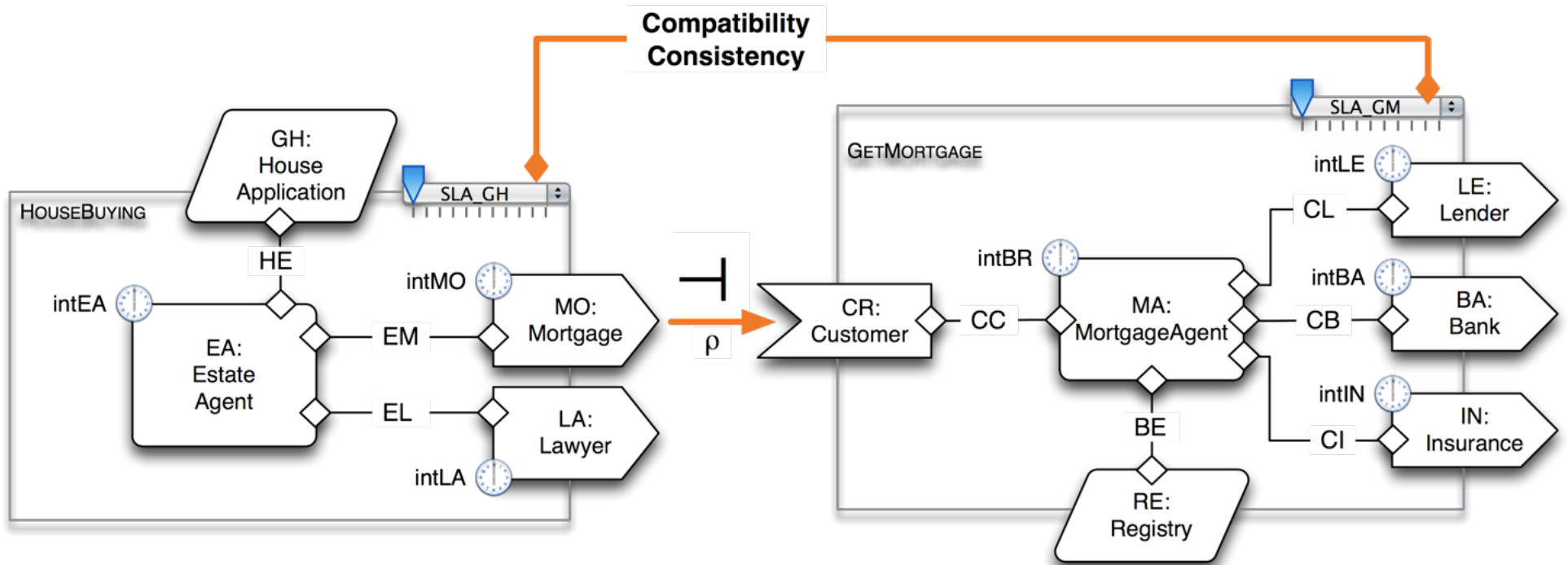
the state transitions



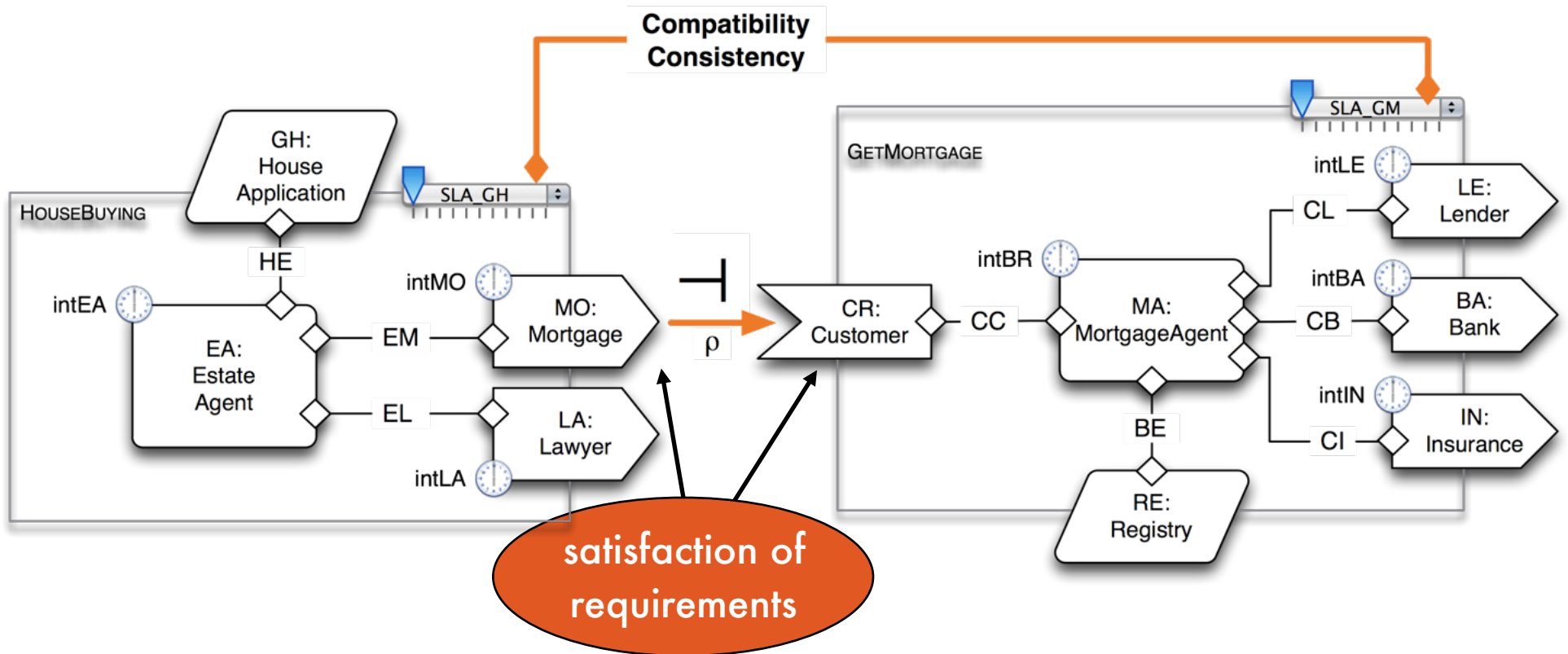
the state transitions



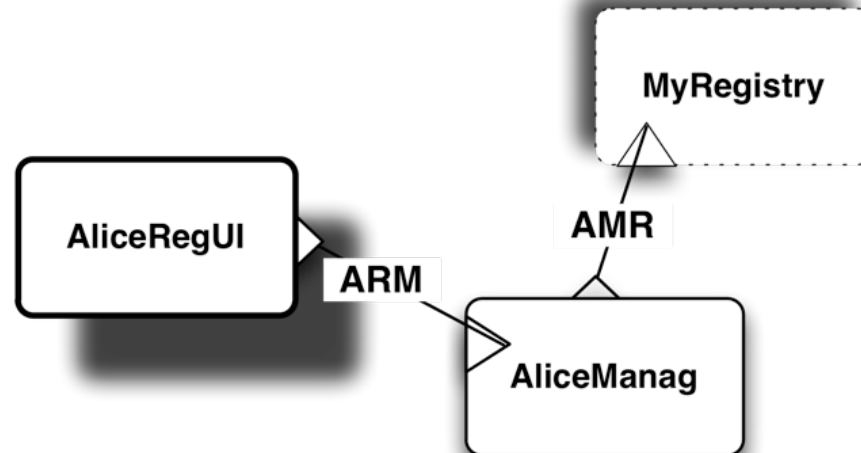
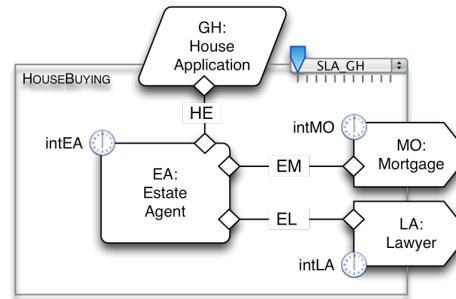
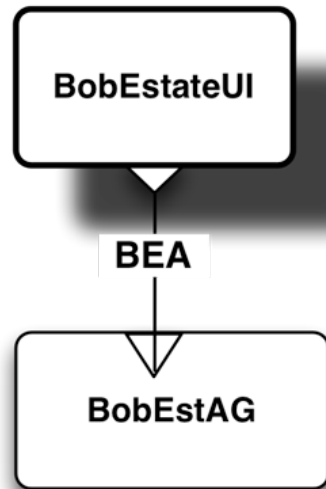
discovery and selection



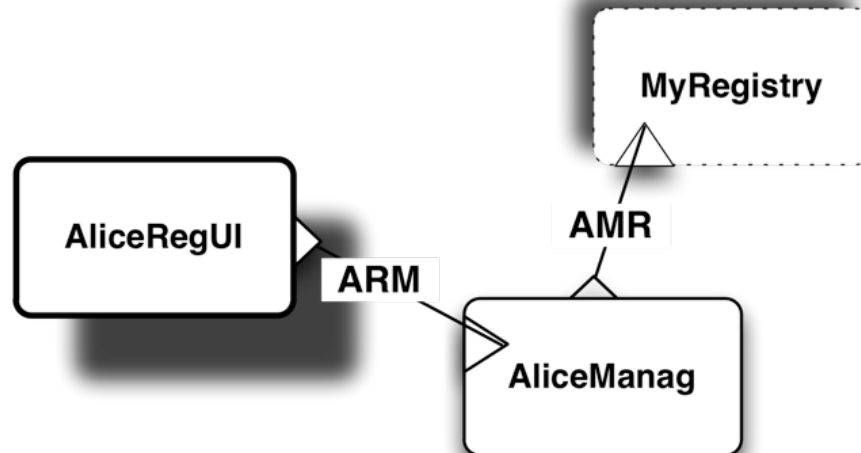
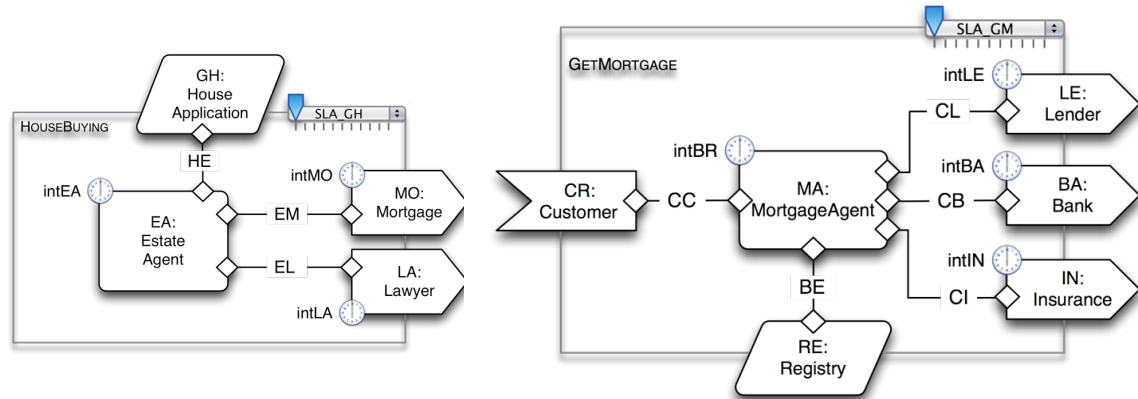
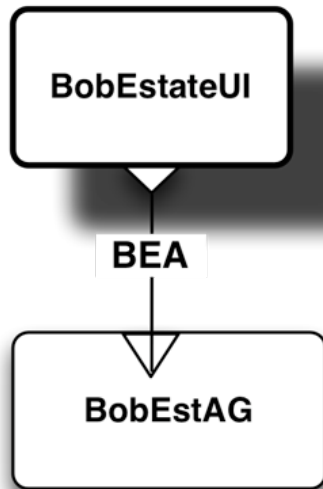
discovery and selection



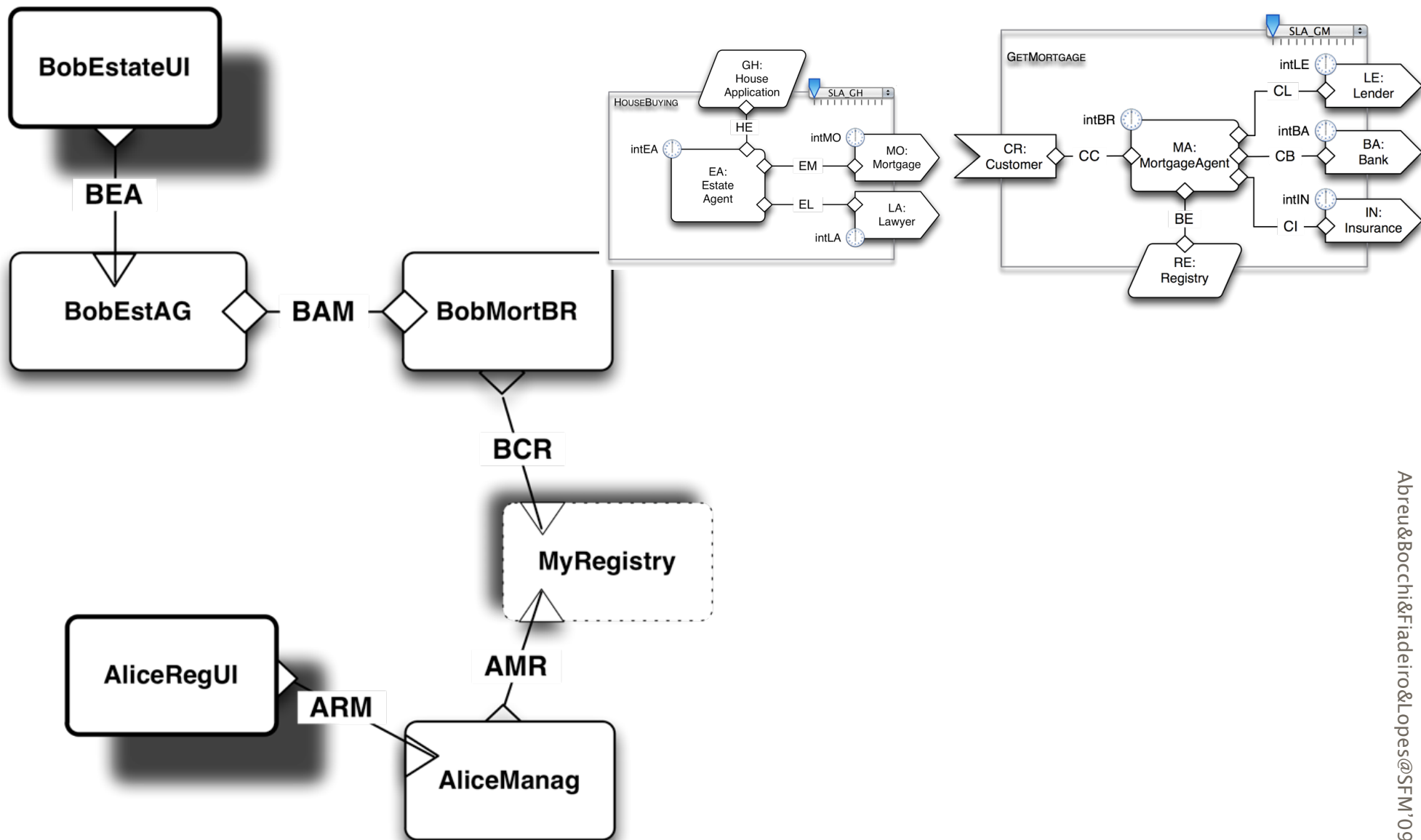
instantiation and binding



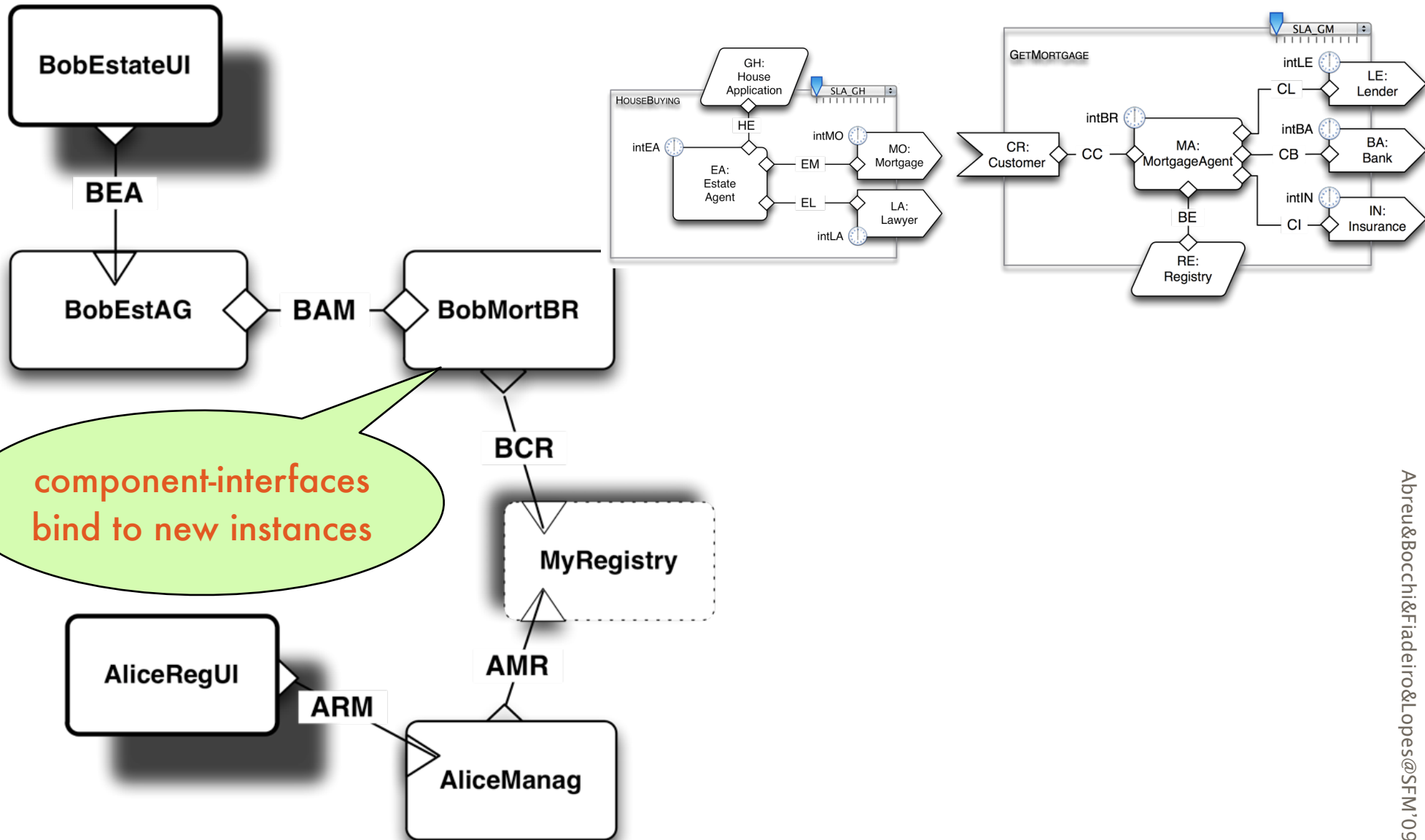
instantiation and binding



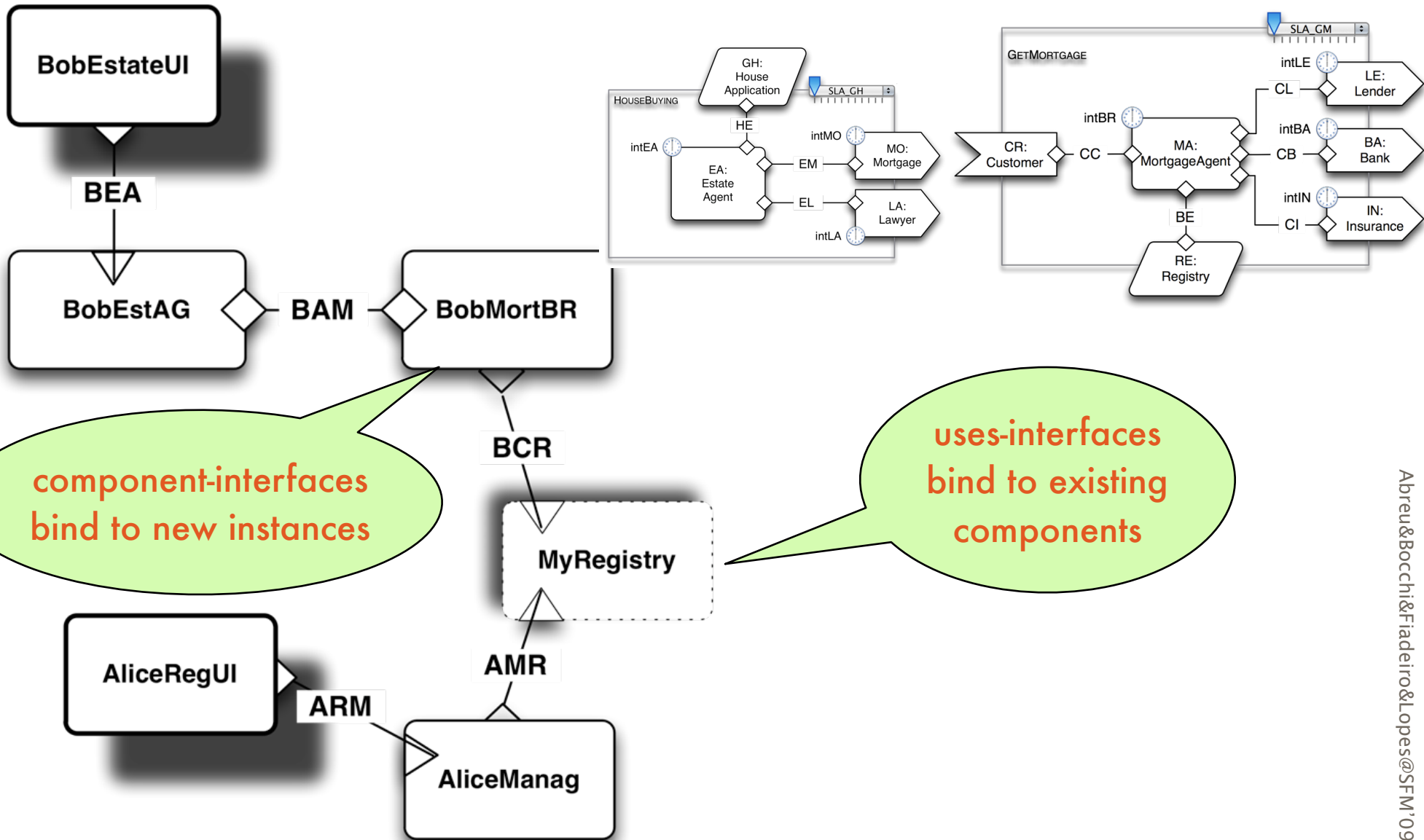
instantiation and binding



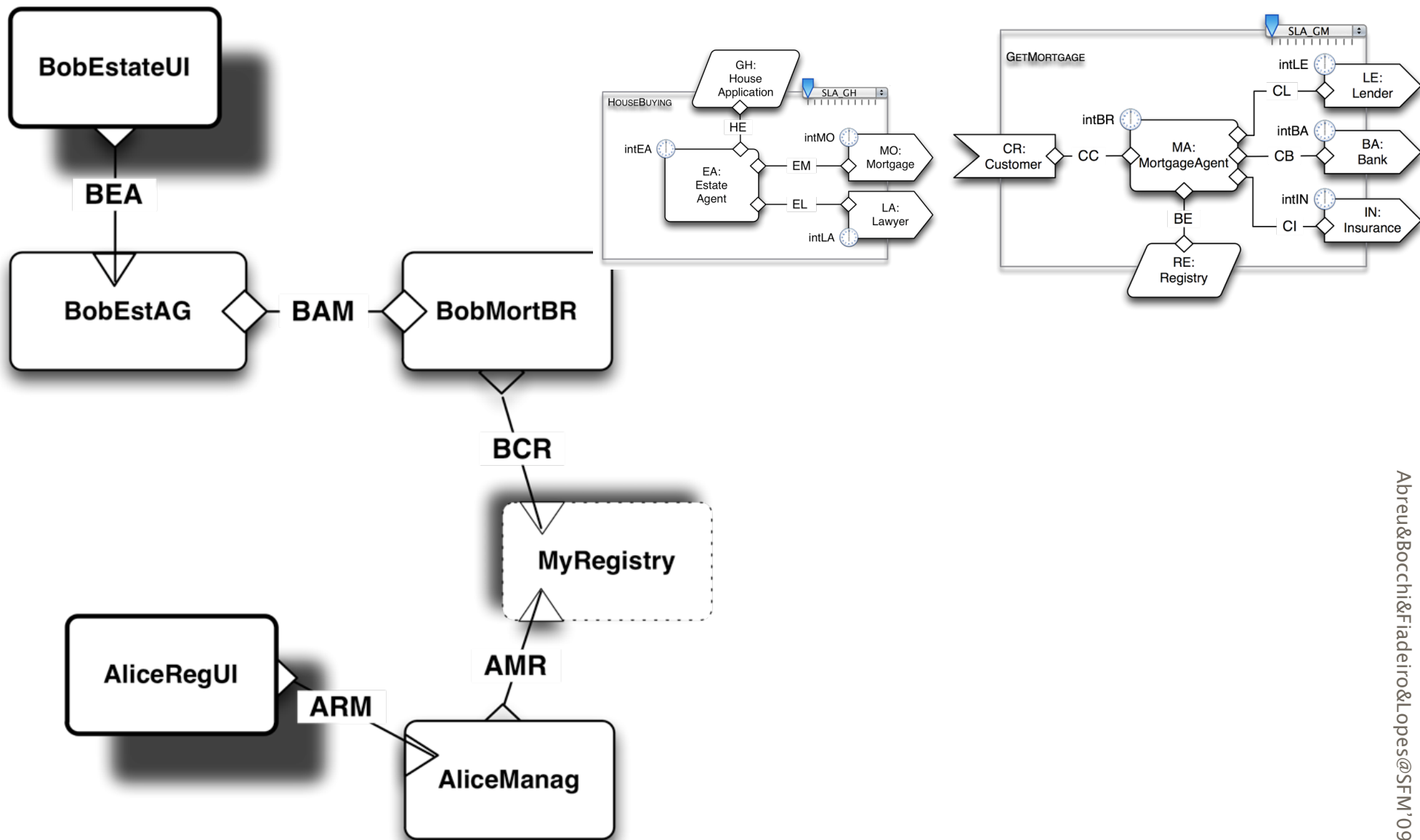
instantiation and binding



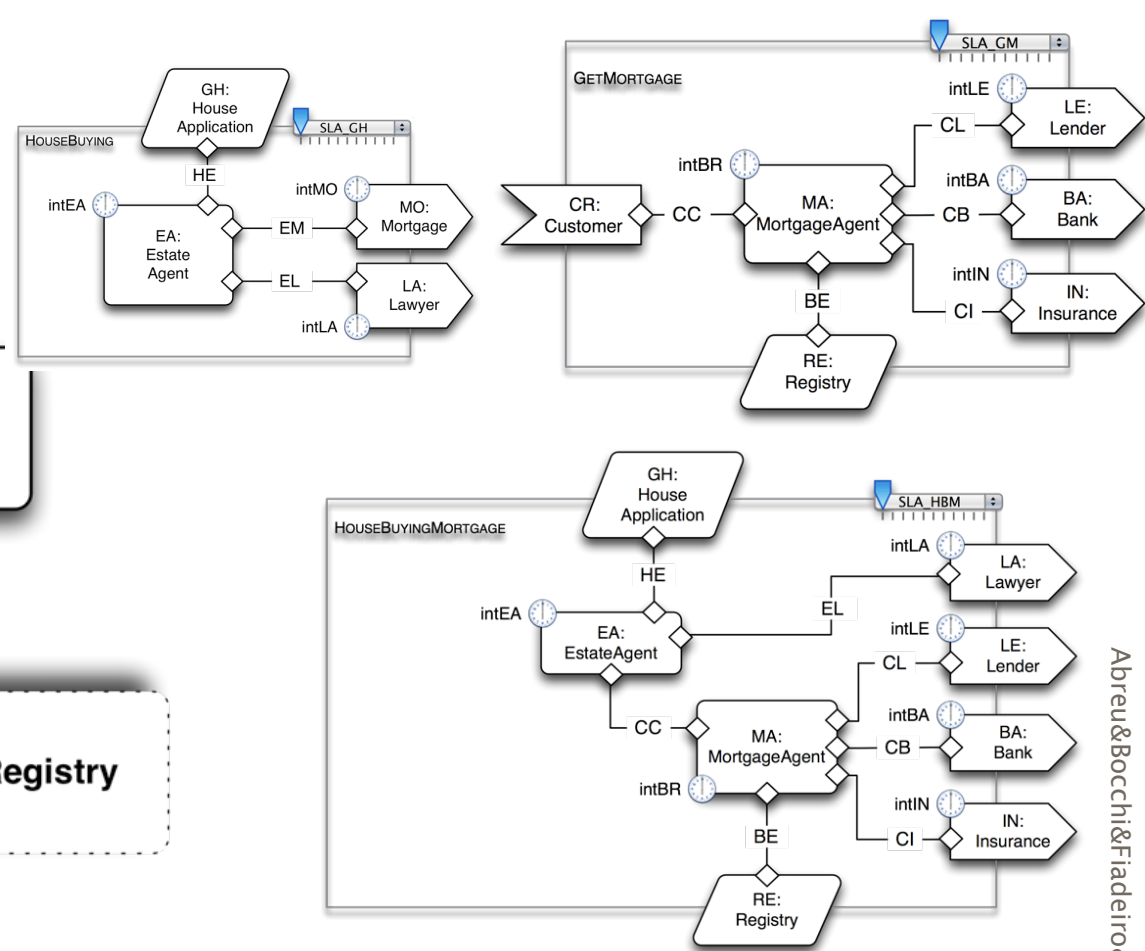
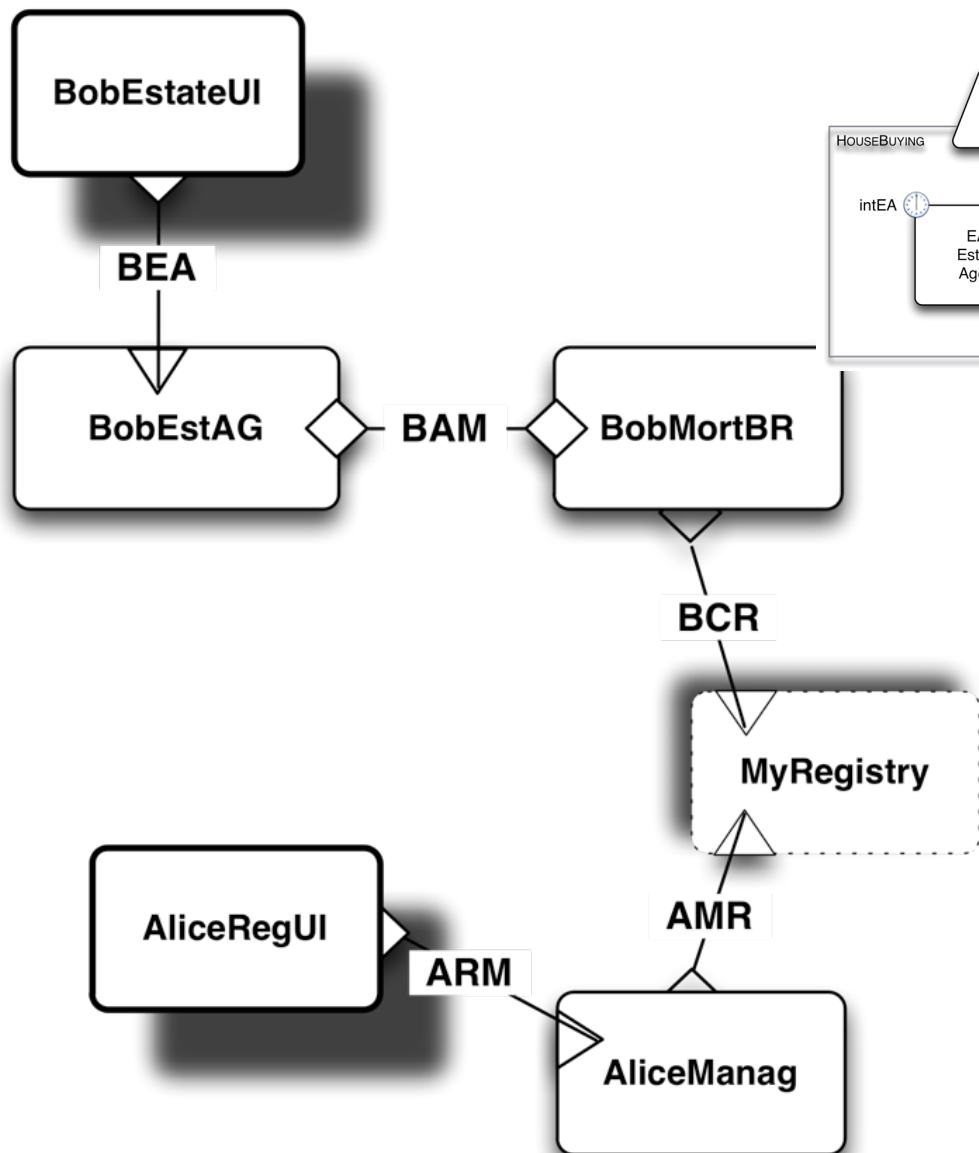
instantiation and binding



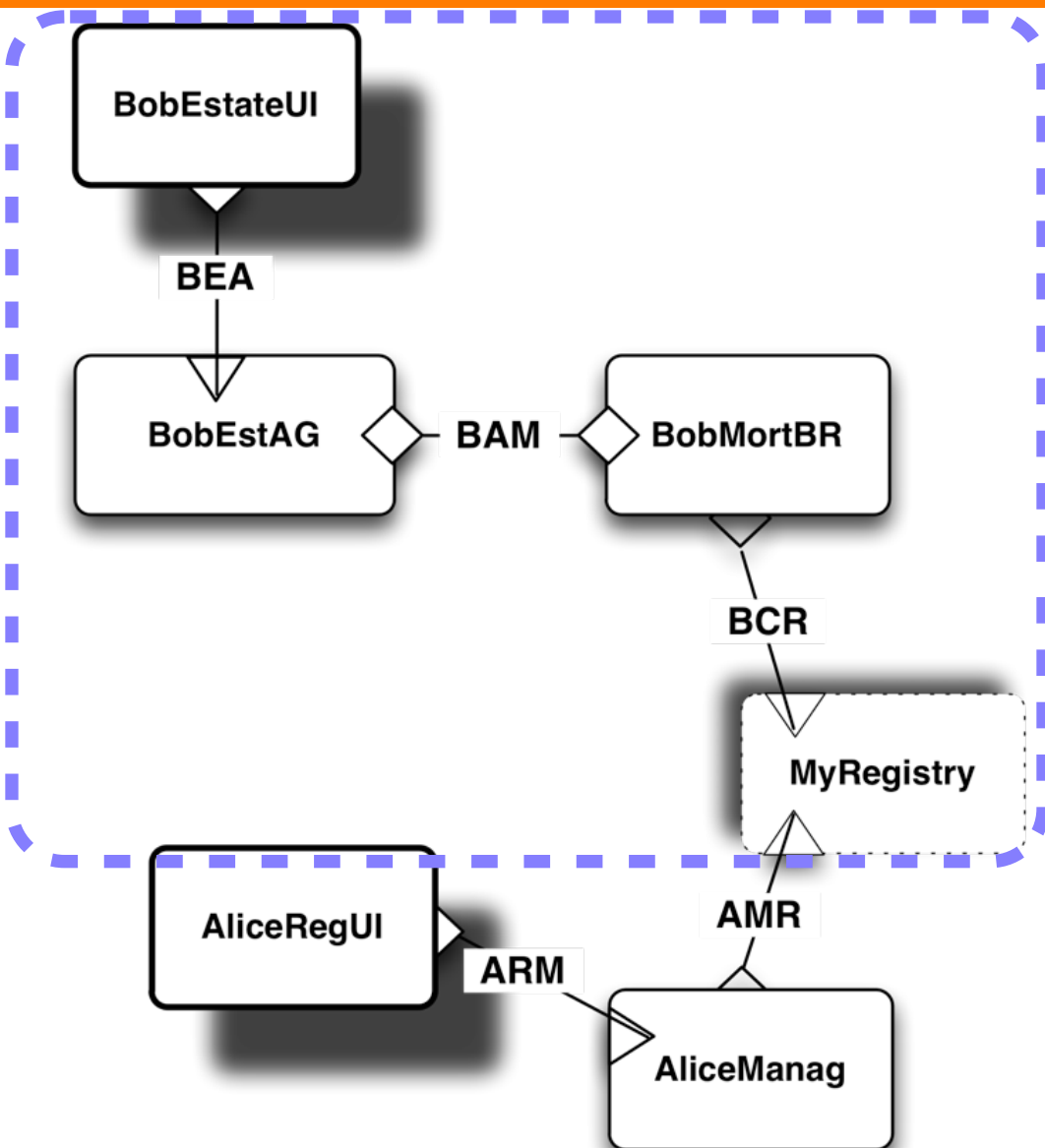
instantiation and binding



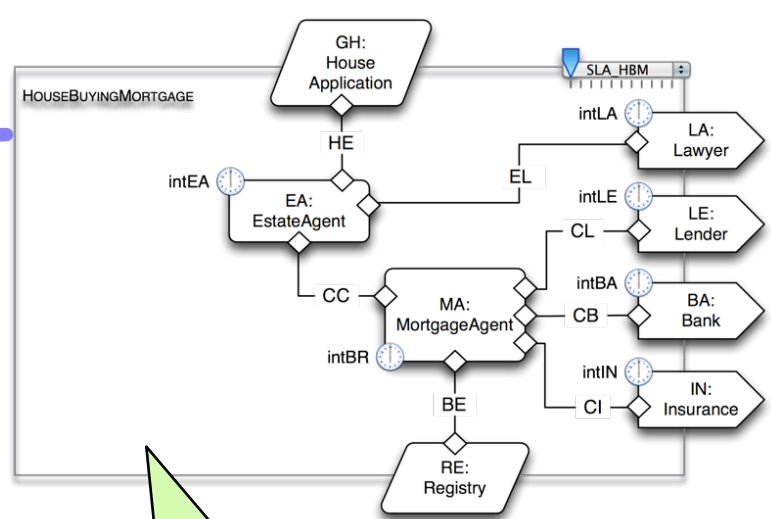
instantiation and binding



instantiation and binding

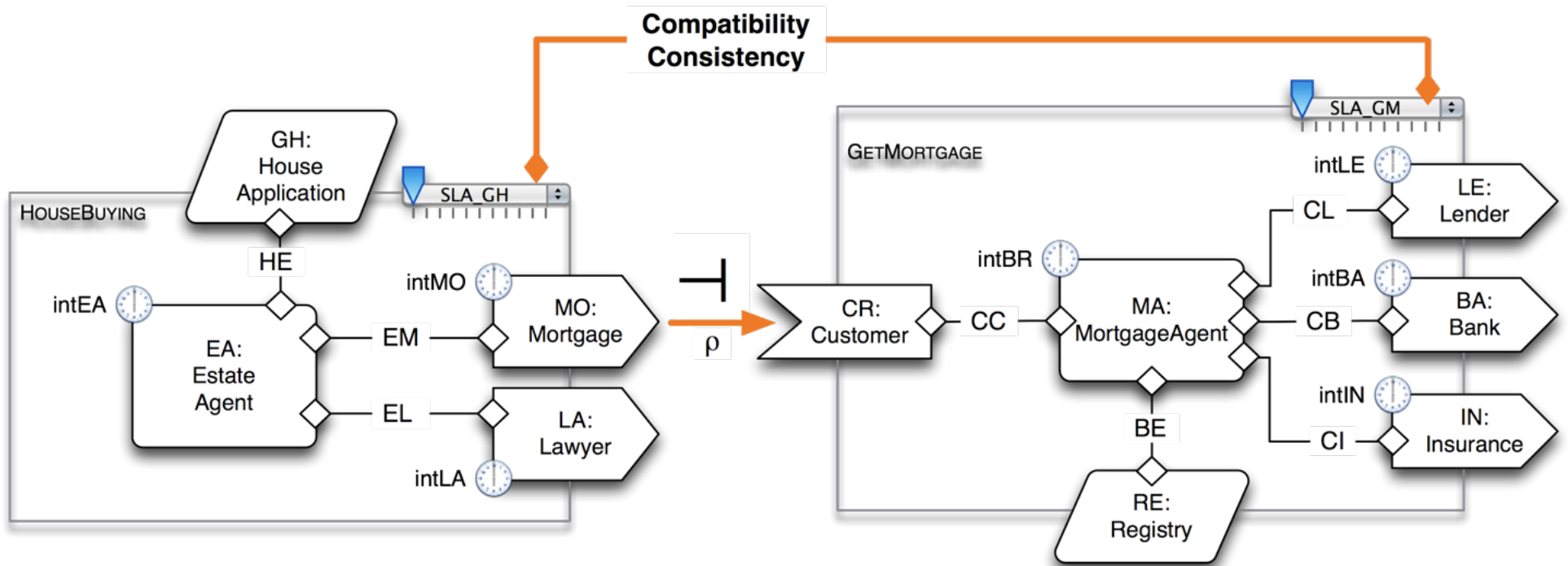


...and a new configuration

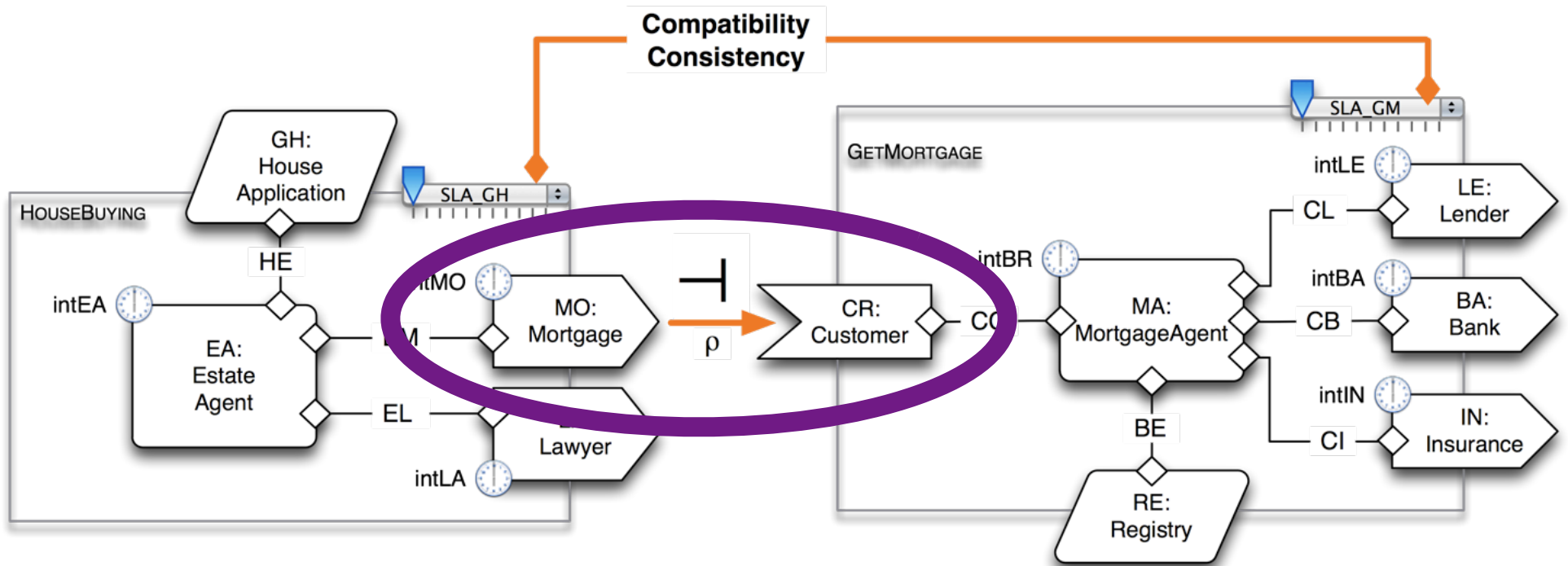


the activity has a new type...

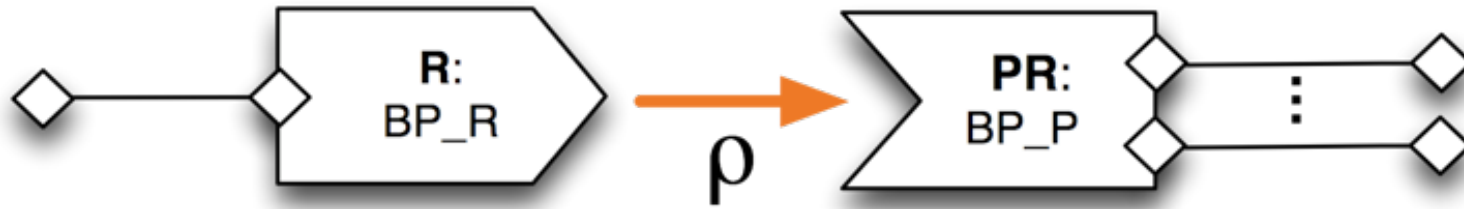
semantics of matching and selection



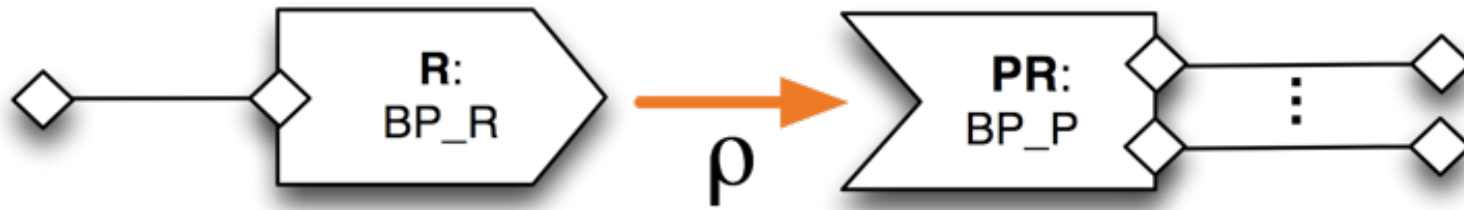
semantics of matching and selection



matching the wires

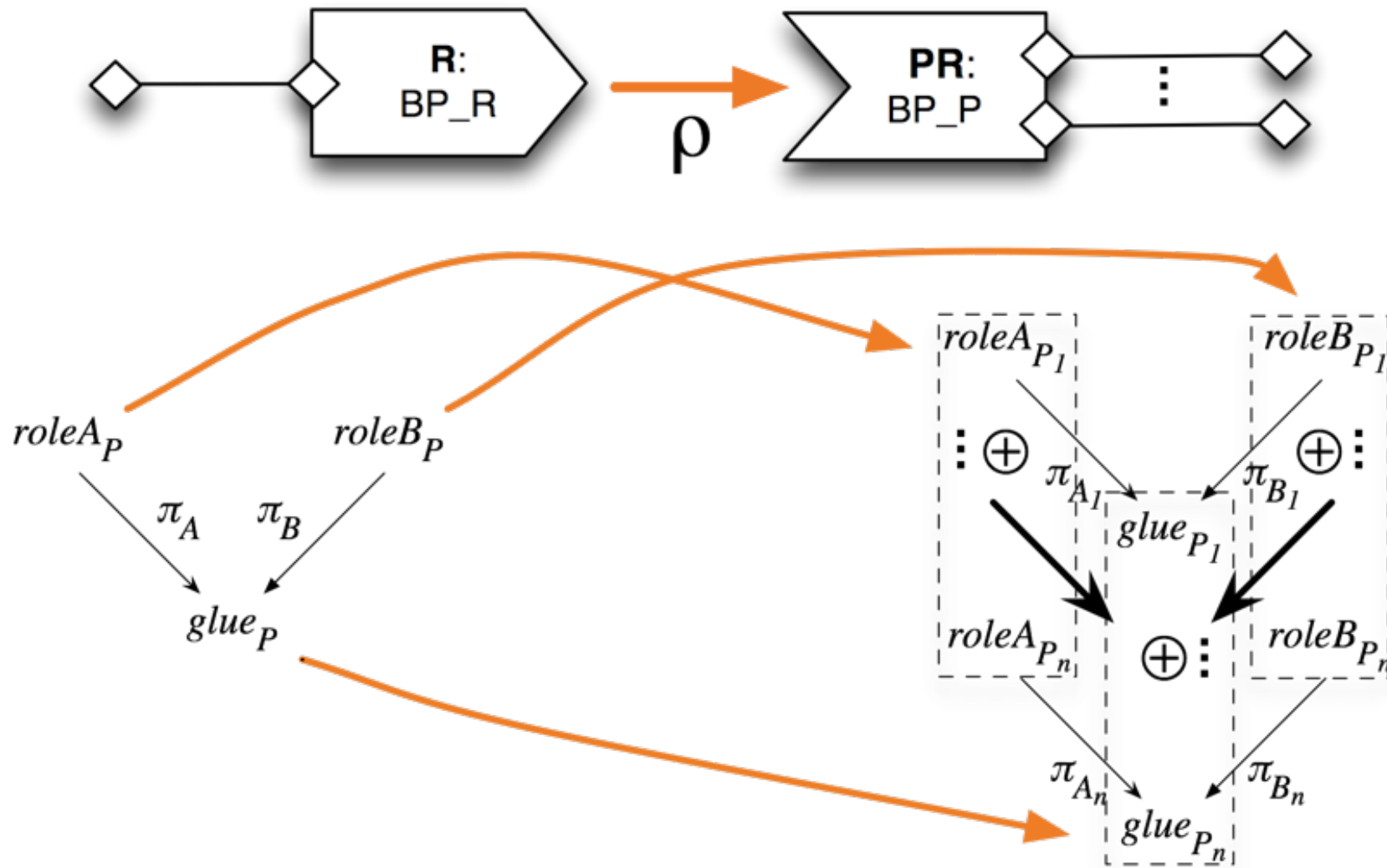


matching the wires

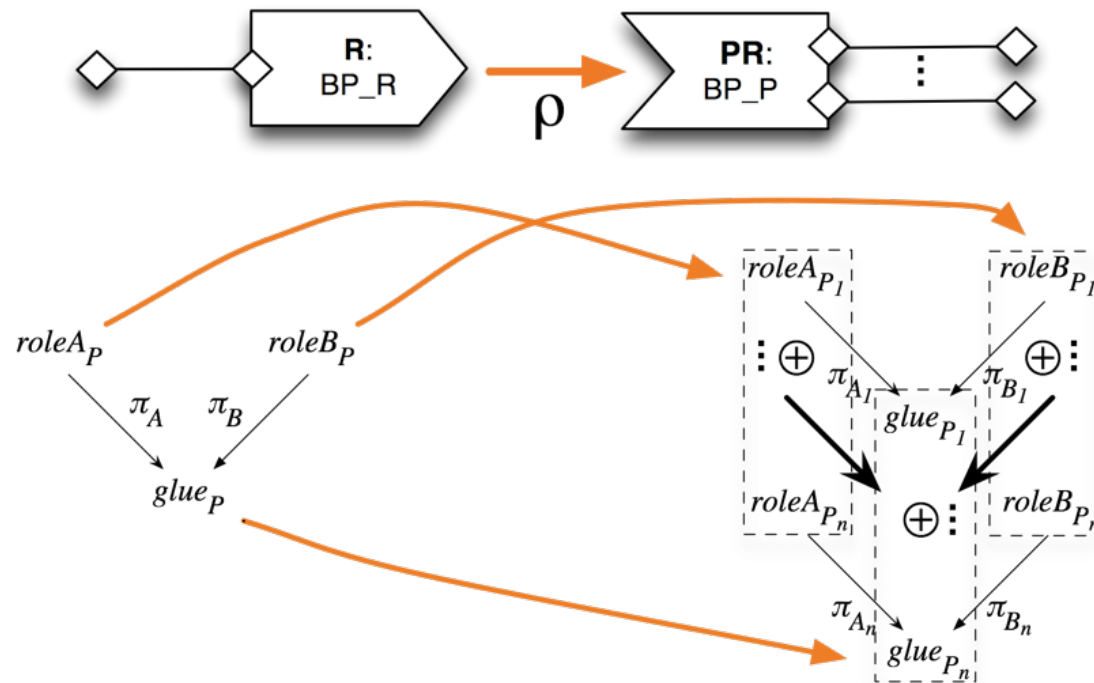


a morphism of
structured co-spans

matching the wires

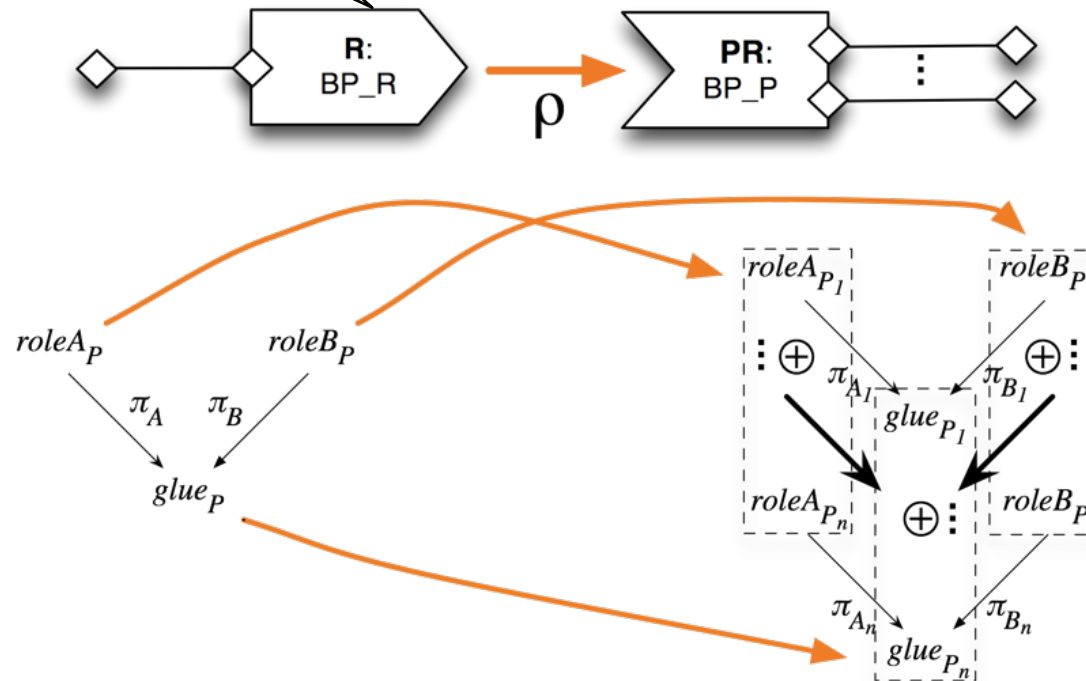


matching the specifications



matching the specifications

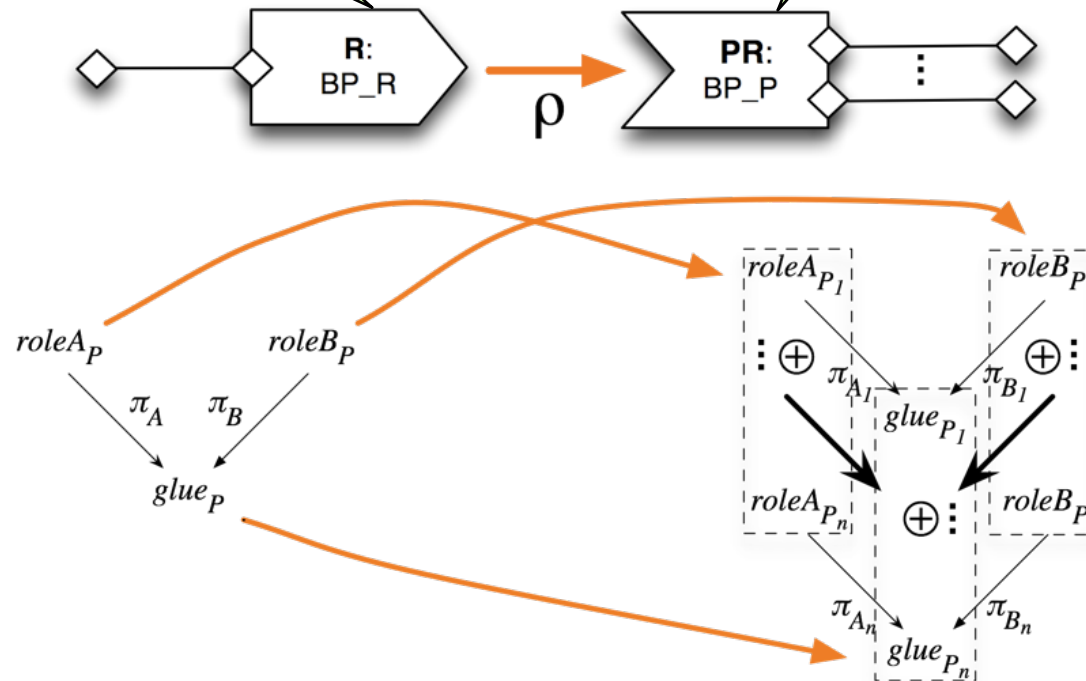
the spec of R (properties required by the activity) is in the language of $roleB_P$



matching the specifications

the spec of **R** (properties required by the activity) is in the language of $roleB_P$

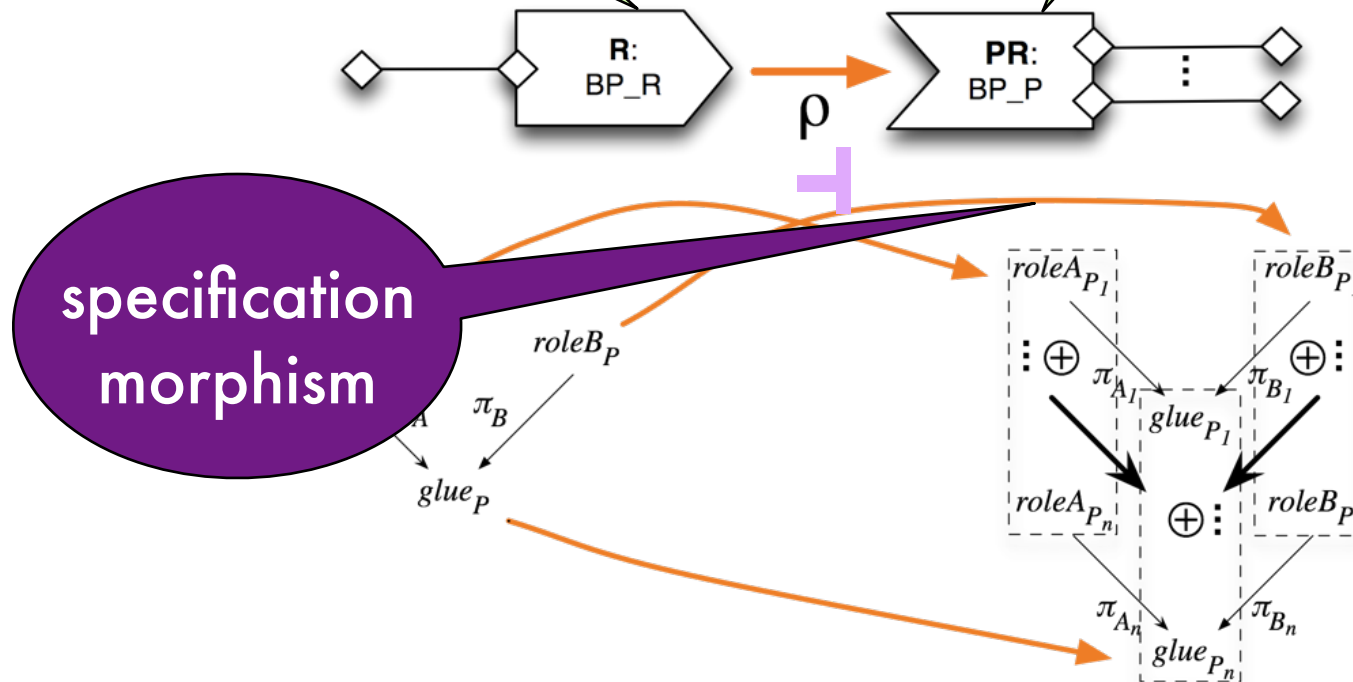
the spec of **PR** (properties offered by the service) is in the language of $\oplus roleB_{P_i}$



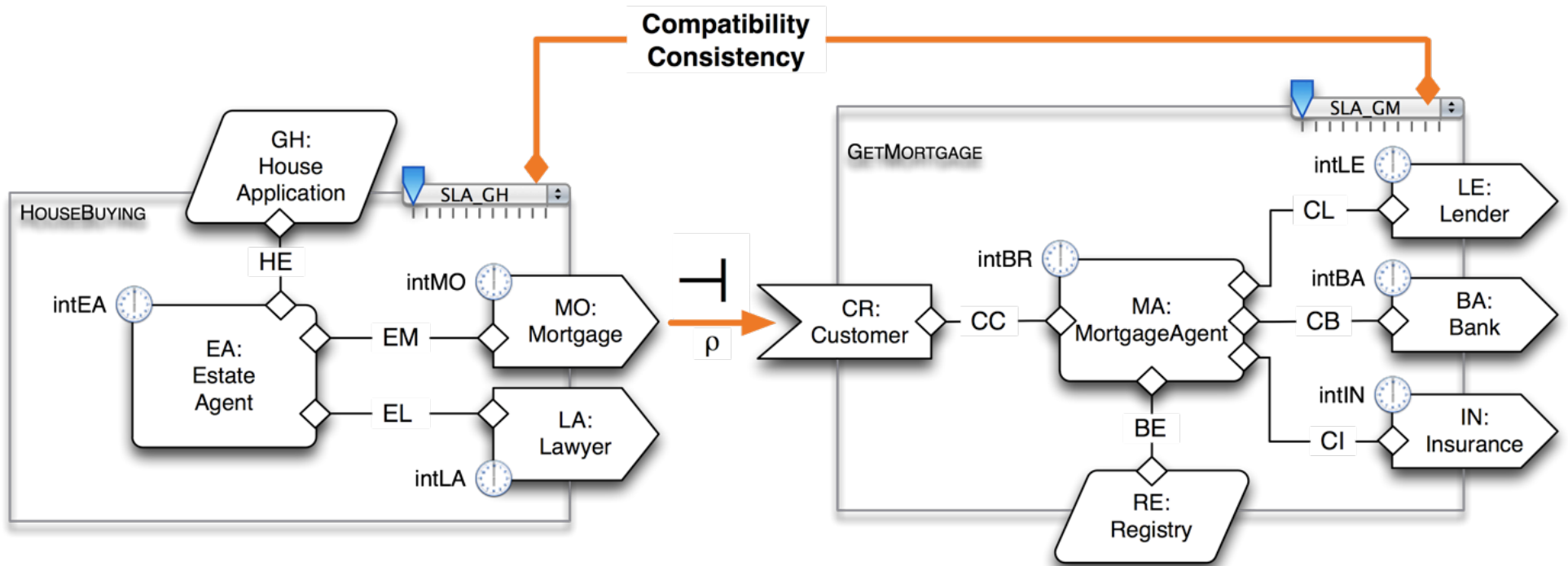
matching the specifications

the spec of **R** (properties required by the activity) is in the language of $roleB_P$

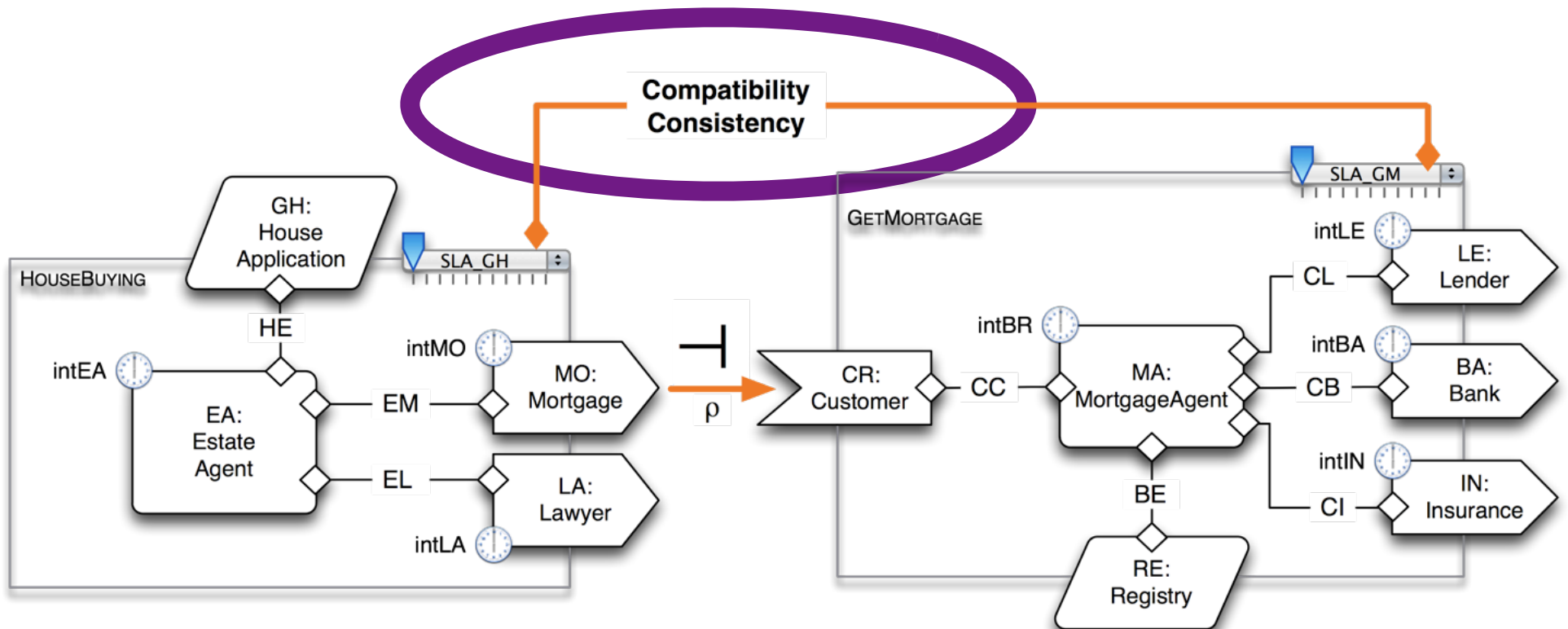
the spec of **PR** (properties offered by the service) is in the language of $\oplus roleB_{P_i}$



algebraic semantics of matching



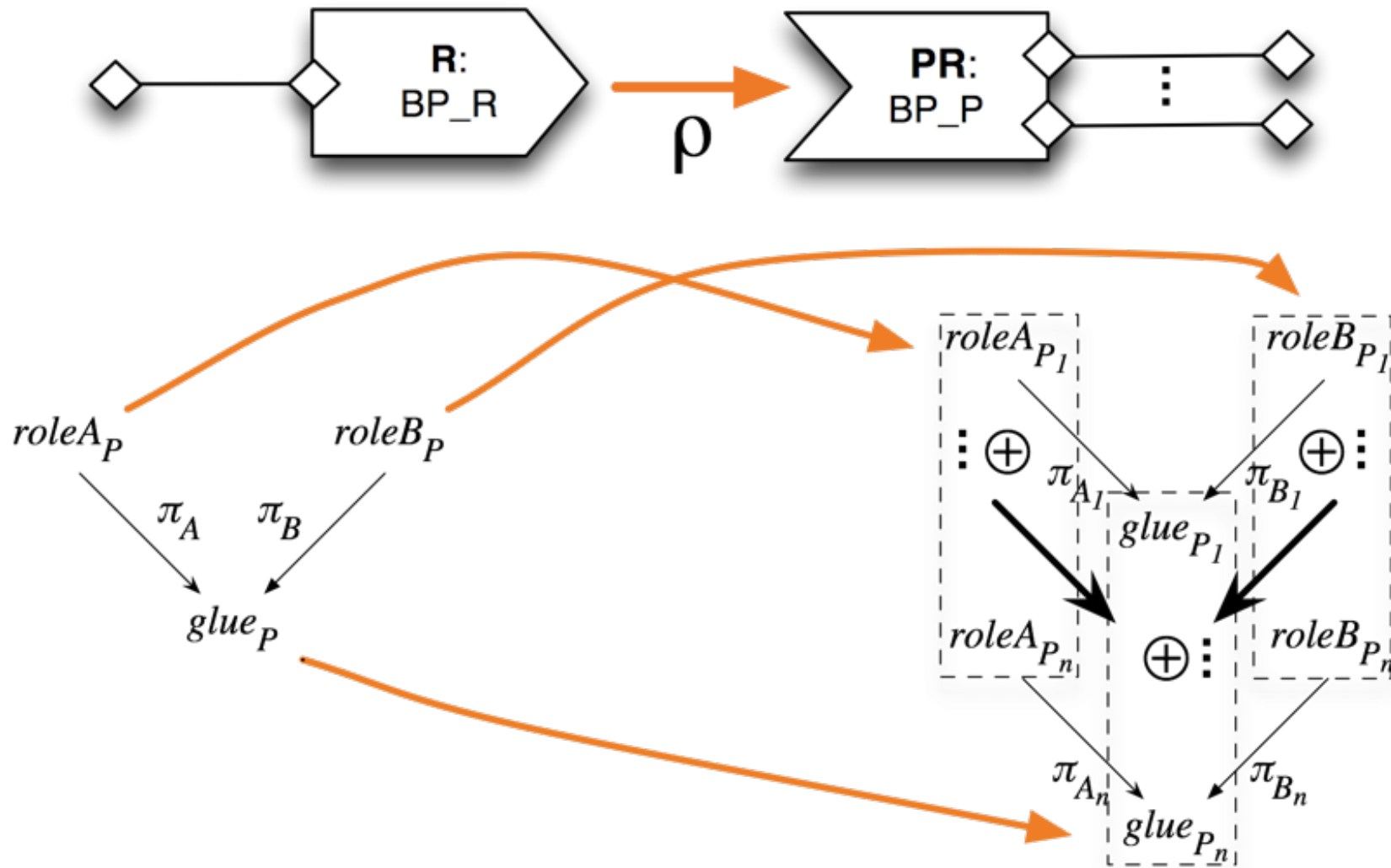
algebraic semantics of matching



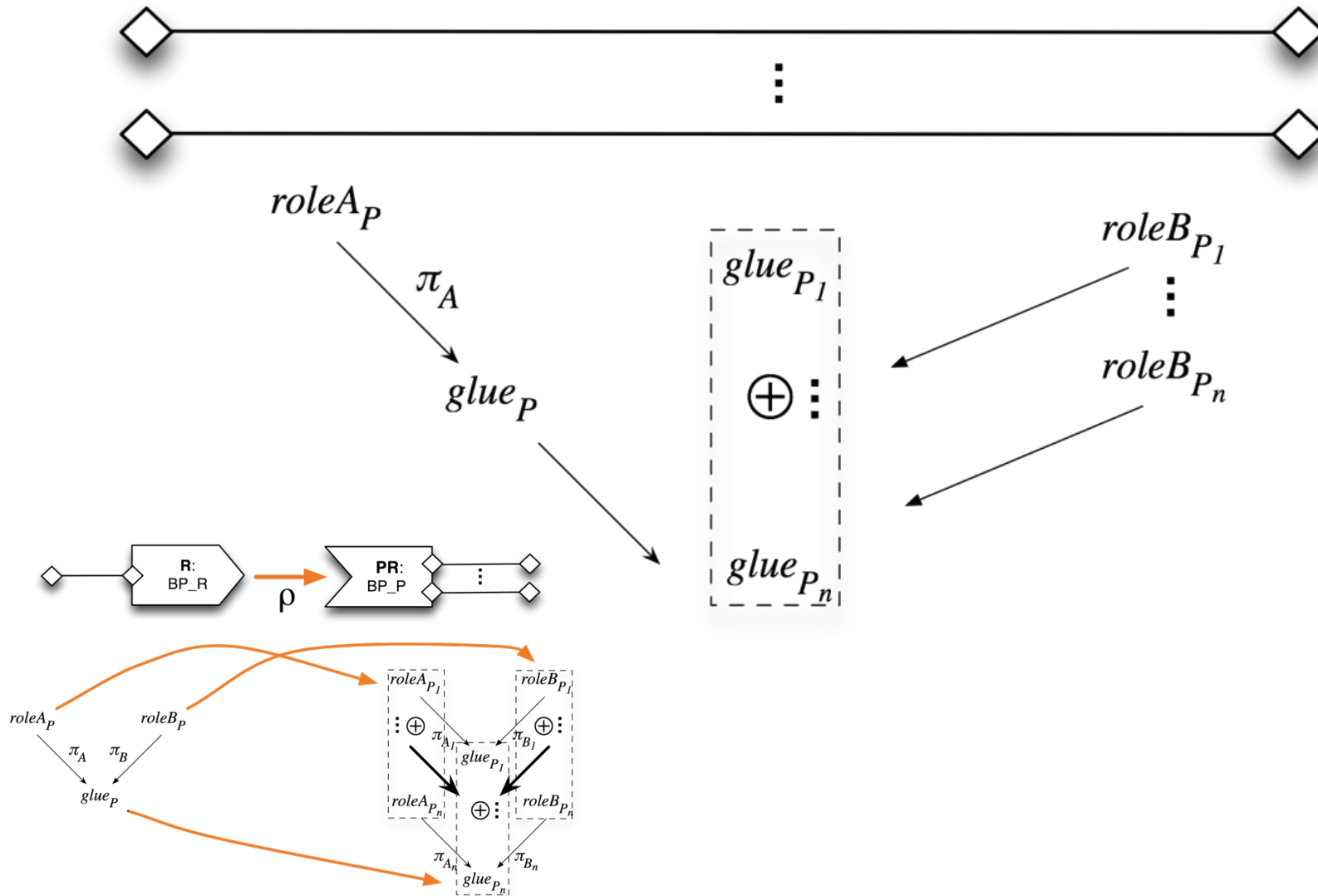
■ matching, ranking and selection involve:

- finding services whose constraint systems are compatible with that of the activity and lead to a consistent combination of constraints
- maximising the degree of satisfaction of the combined set of constraints

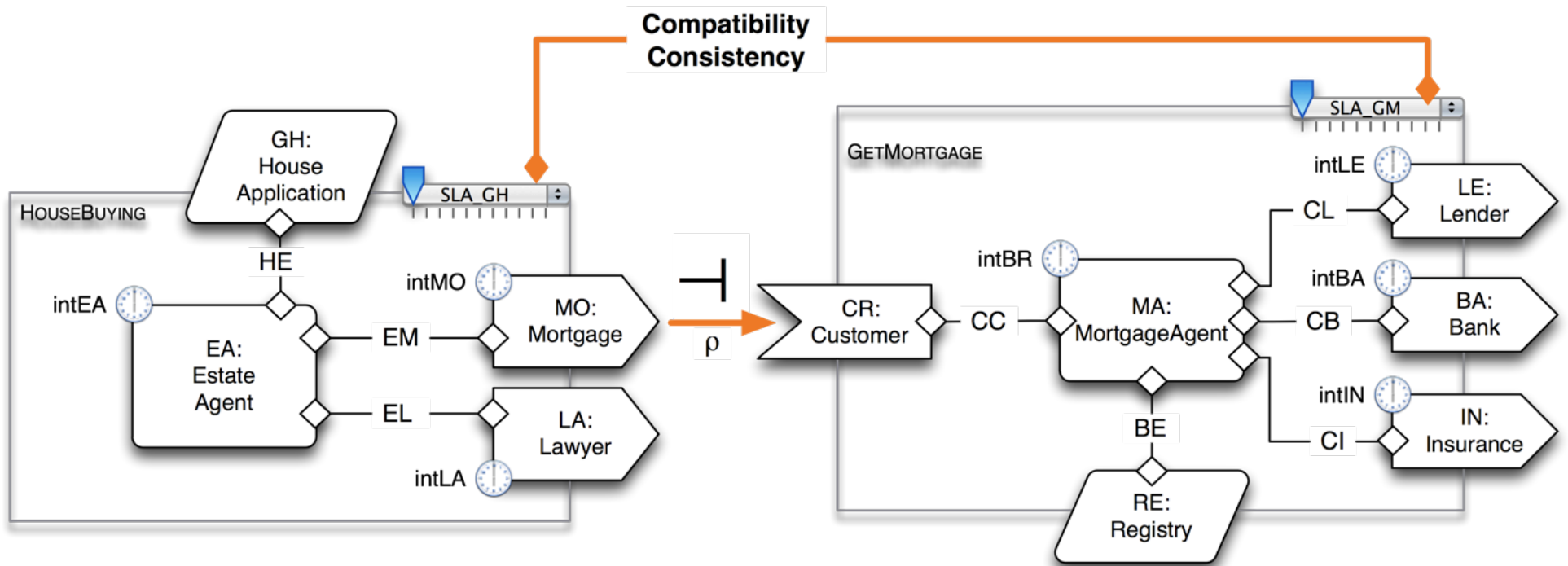
algebraic semantics of composition



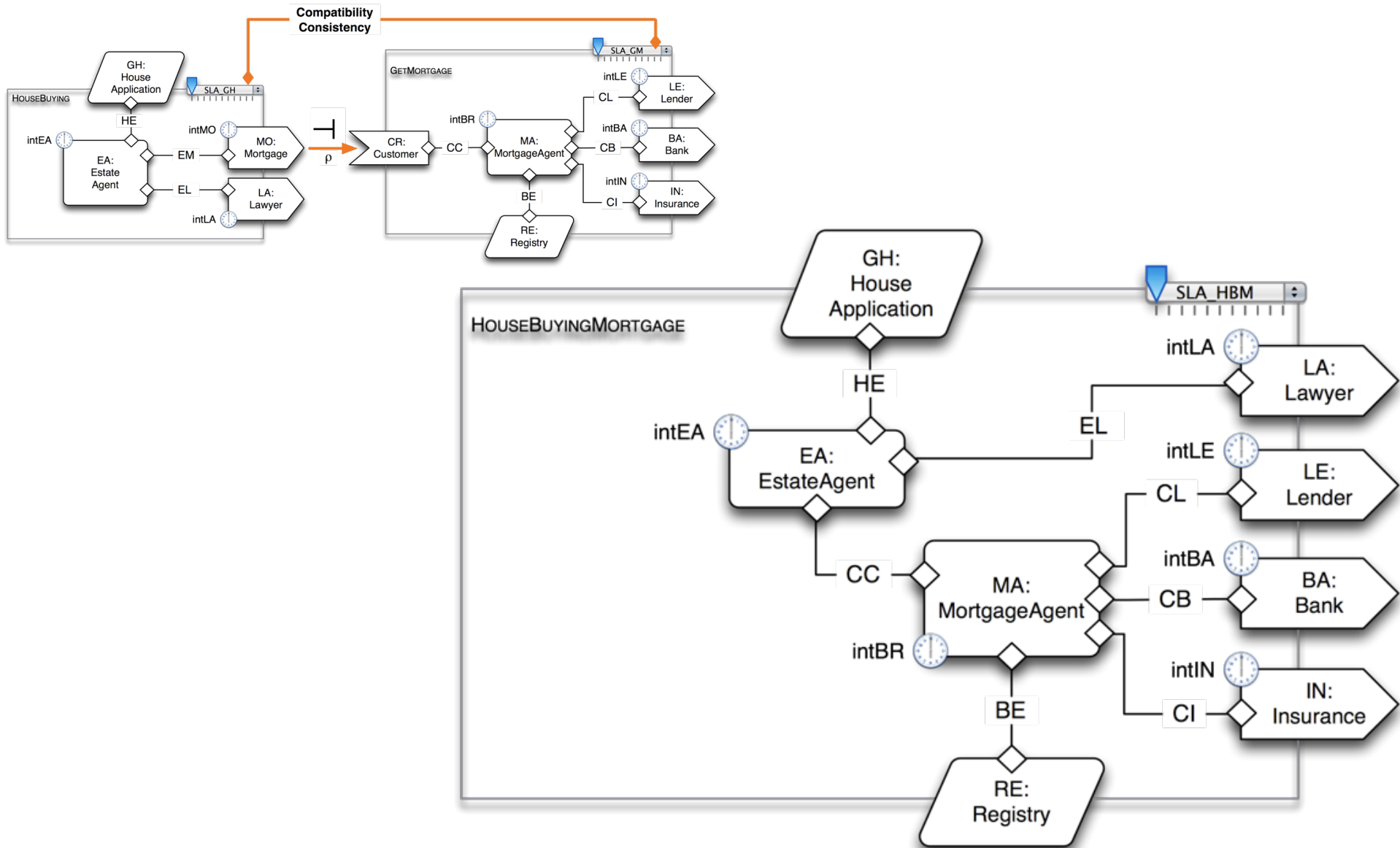
algebraic semantics of composition



algebraic semantics of composition



algebraic semantics of composition



what else?

what else?

■ Qualitative analysis

- Doubly-labelled transition systems and temporal logic
- Model-checking using UMC

what else?

■ Qualitative analysis

- Doubly-labelled transition systems and temporal logic
- Model-checking using UMC

■ Analysis of timing properties using PEPA

what else?

■ Qualitative analysis

- Doubly-labelled transition systems and temporal logic
- Model-checking using UMC

■ Analysis of timing properties using PEPA

■ A number of case studies

- Travel booking
- Procurement
- Automotive
- Telco

future work

future work

■ Back to SCA...

future work

- Back to SCA...
- SRML4People
 - Team automata
 - Deontic logic

future work

- Back to SCA...
- SRML4People
 - Team automata
 - Deontic logic
- Run-time verification and monitoring

future work

- Back to SCA...
- SRML4People
 - Team automata
 - Deontic logic
- Run-time verification and monitoring
- Requirements and coreography