An overview

Laura Bocchi
bocchi@mcs.le.ac.uk

Agenda

- Formal Methods and Engineering Practice
  (and some motivation)
- An overview of SRML as related with other languages
  - SCA (Service Deployment)
  - PEPA (Quantitative analysis)
  - StPowla (Workflows and Business Policies)
  - BPEL (Web services orchestration)

Formal Methods

- Different languages and levels of abstractions for different purposes
  - e.g., Petri-Nets is a graphical formal language to model/analyze concurrent processes
  - e.g., Pi-calculus is a process calculus to model/analyze concurrent communicating processes (it has primitives for modelling reconfiguration)
  - e.g., Z is a logic notation to specify systems ...
- Why many? Because each targets a specific aspect and aim. Focusing on a specific target and aim limits the complexity of models.
- Why formal methods? Specification, transformation, analysis, understanding the semantics of "real" languages or real problems or real scenarios

One Example: the semantics of BPEL

- BPEL is a standard for Web Service orchestration inspired by pi-calculus. It does not come with a formal semantics (although some proposals have been done in academia)
- A number of engines support BPEL. Each BPEL engine may implement a slightly different semantics
- E.g., if you have two parallel threads in a transaction and one fails: do you have to interrupt the other one? do you have to wait for its completion and then compensate it? In which order should the compensations be executed?
- Background knowledge about orchestration semantics enables BPEL programmers to update and adapt to evolving technologies (e.g., update or change).
- In general, the understanding of the key problems of a scenario is a know-how which makes it easier to familiarise and cope with evolving technologies

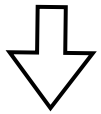
What about SRML?

- Service-oriented modelling is an emerging discipline and issue, supported and encouraged by major companies (Microsoft, IBM, BEA, etc)
- There are a number of formal languages that address, very specifically, the different aspects of service-oriented engineering
- There is no standard language for the modelling of service oriented systems (yet)
- SRML is the only language (at the moment) that address architectural and behavioural modelling in a whole formal framework
  - algebraic semantics of reconfiguration, logics of interaction, c-semirings
- Will I use SRML in industry? Probably not, but you will possibly use (or even define!) some language that models some of the features modelled by SRML.
- Learning an reference language which is general enough, because defined for being such, provides a basis to cope with an evolving set of technologies in the context of distributed dynamically reconfiguring systems

A view of ensemble

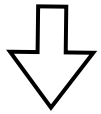
use case diagrams

Define requirements



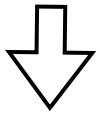
from scratch
use case -> SRML

Define modules + EX-Is



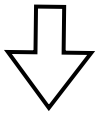
from scratch
CBD

Define internal structure

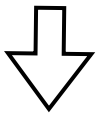


from scratch
reuse
refinement

Define behaviours



Analyse

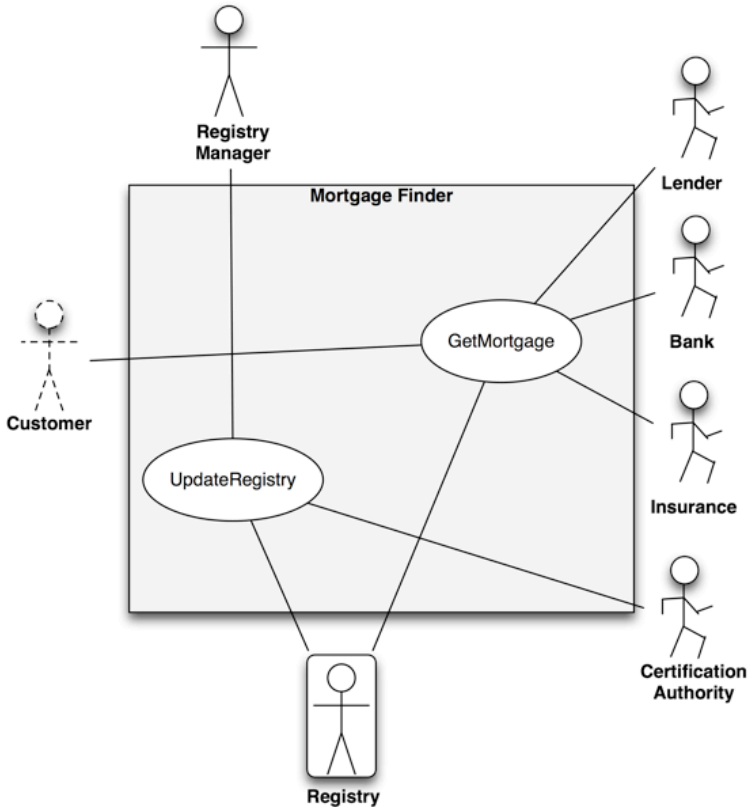


Deploy

The mortgage example

use case diagrams

Define requirements

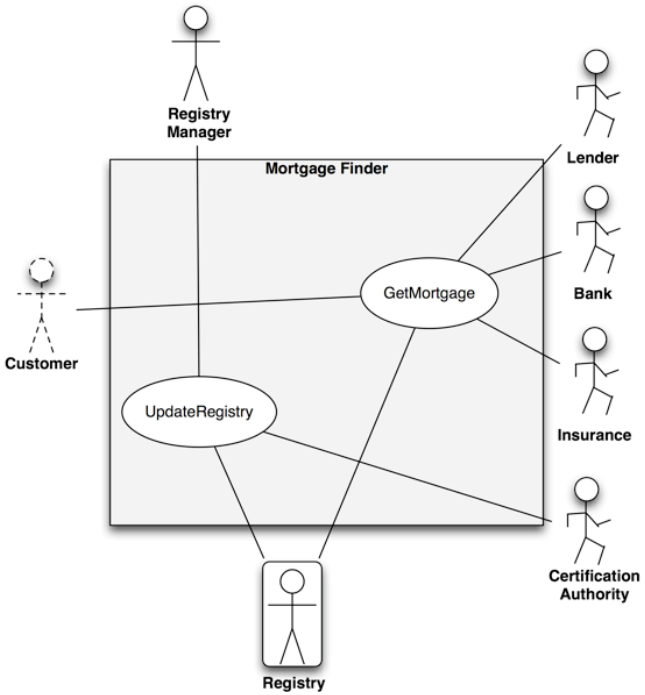


the Mortgage scenario

The mortgage example

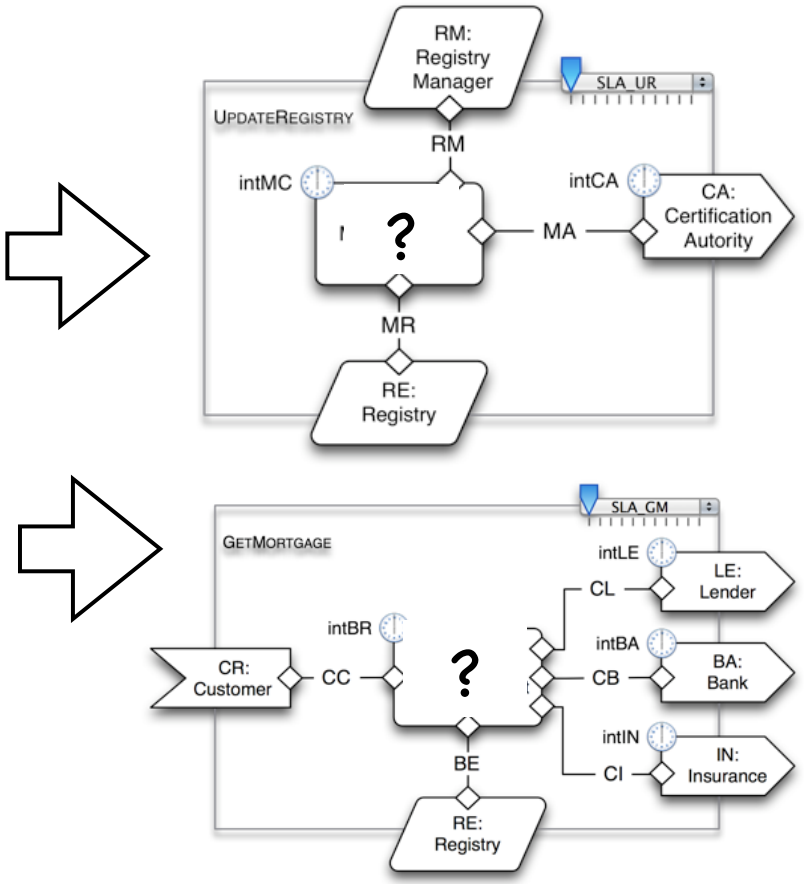
use case diagrams

from scratch
use case -> SRML



Define requirements

Define modules + EX-Is



A view of ensemble

use case diagrams

Define requirements

⇩

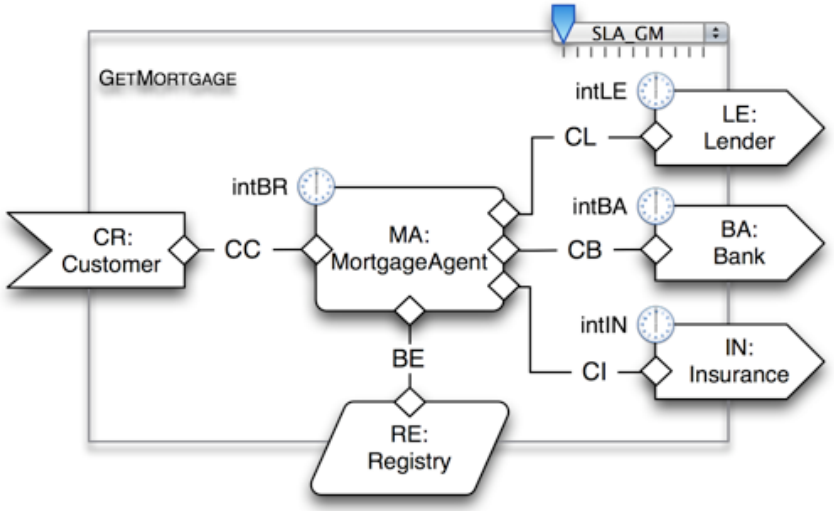
from scratch
use case -> SRML

Define modules + EX-Is

⇩

from scratch
CBD

Define internal structure

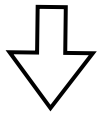


more component could be defined
for modular development, reuse,
limiting the complexity etc.

A view of ensemble

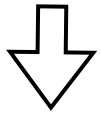
use case diagrams

Define requirements



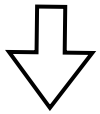
from scratch
use case -> SRML

Define modules + EX-Is



from scratch
CBD

Define internal structure

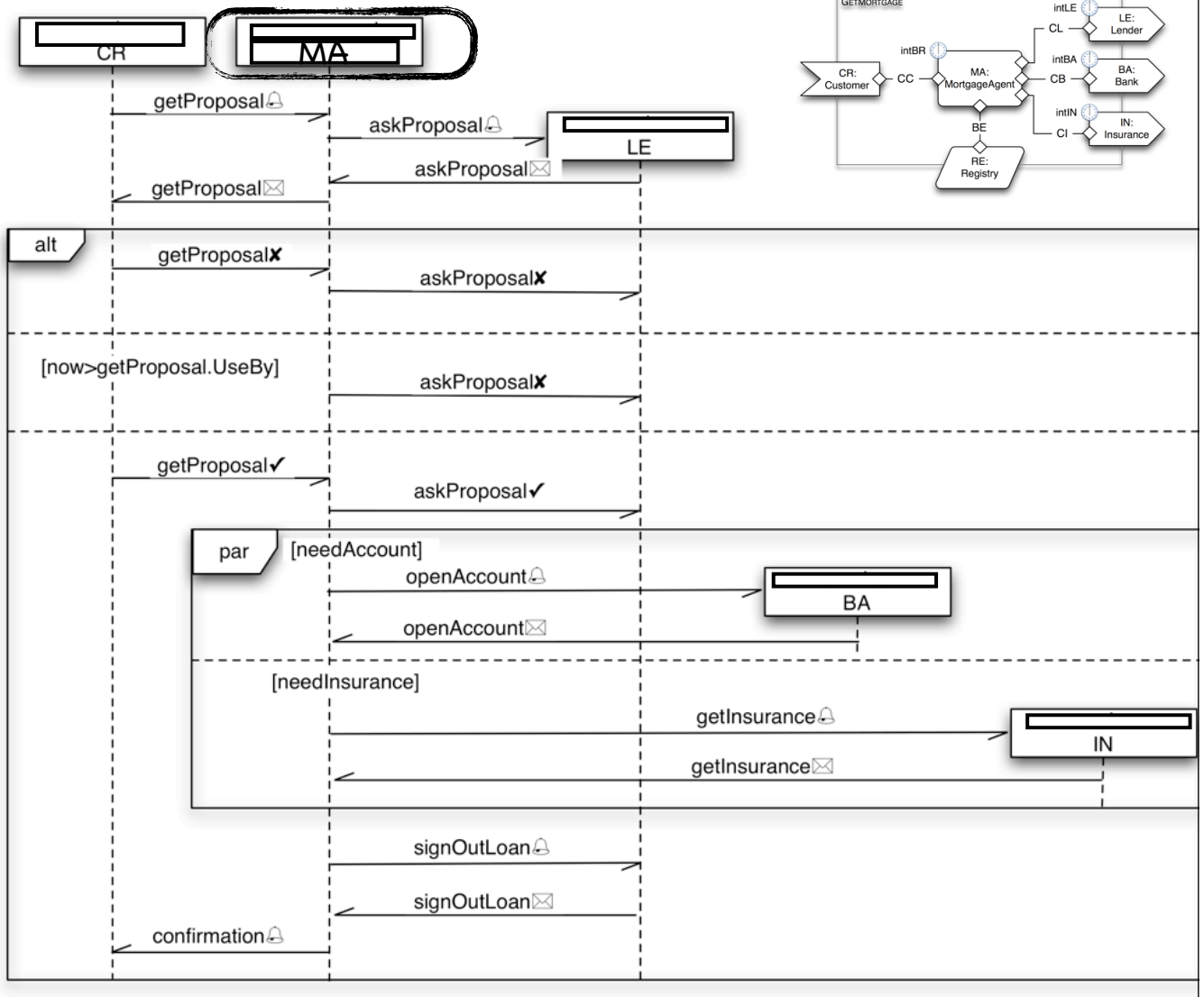


from scratch
reuse
refinement

Define behaviours

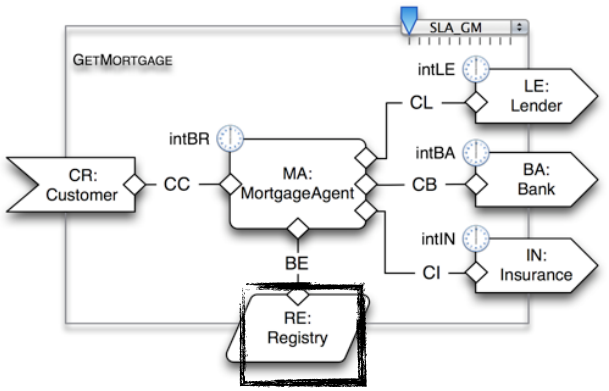
The specifications (interactions)

```
BUSINESS ROLE MortgageAgent is
    INTERACTIONS
        r&s getProposal
            🔔 idData:usrdata,
               income:moneyvalue,
               preferences:prefdata,
            ✉ proposal:mortgageproposal
               cost:moneyvalue
        s&r askProposal
            🔔 idData:usrdata,
               income:moneyvalue,
            ✉ proposal:mortgageproposal
               loanData:loandata,
               accountIncluded:bool,
               insuranceRequired:bool
        s&r getInsurance
            🔔 idData:usrdata,
               loanData:loandata,
            ✉ insuranceData:insurancedata
        s&r openAccount
            🔔 idData:usrdata,
               loanData:loandata,
            ✉ accountData:accountdata
        s&r signOutLoan
            🔔 insuranceData:insurancedata,
               accountData:accountdata,
            ✉ contract:loancontract
        snd confirmation
            🔔 contract:loancontract
        ask getLenders(prefdata):setids
        tll regContract(loandata,loancontract
```

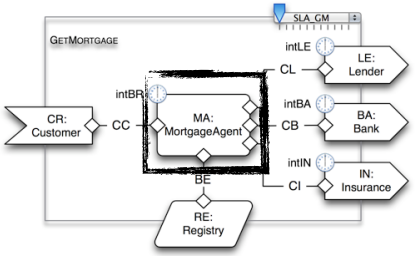


The specifications (textual notation)

```
SPECIFICATIONS

    LAYER PROTOCOL Registry is

        INTERACTIONS
            rpl getLenders(prefdata):setids
            prf registerContract(loandata,loancontract)
        BEHAVIOUR
```



The orchestration as a statechart



start

INITIAL

getProposal /

GetClientRequest

/ askProposal

WAIT_PROPOSAL

askProposal /

GetProposal

[askProposal.Reply] / getProposal

WAIT_DECISION

[askProposal.Reply] / getProposal

FINAL

end

now>getProposal.UseBy /

TimeoutProposal

/ askProposal ✘

getProposal✓ [now<askProposal.UseBy]/

getProposal ✘ /

ProposalNotAccepted

/ askProposal ✘

/ confirmation

ProposalAccepted

[¬ needAccount ∧ ¬ needInsurance]/ / signOutLoan

Conclude

[needAccount ∨ needInsurance]/

signOutLoan /

PROPOSAL_ACCEPTED

[askProposal.needInsurance] / getInsurance

...

PA1

getInsurance /

GetInsurance

[¬ askProposal.needInsurance]

SIGNING

/ signOutLoan

[askProposal.needAccount] / openAccount

...

PA2

openAccount /

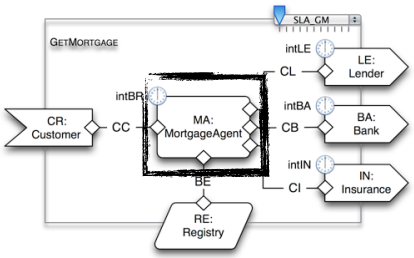
GetAccount

[¬ askProposal.needAccount]

The specifications (the business role - interactions)

```
BUSINESS ROLE MortgageAgent is

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

```
  SLA VARIABLES
      CHARGE:[0..100]

  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 s'=WAIT_PROPOSAL
         ∧ lenders'= getLenders(prefdata)
      sends askProposal🔔
         ∧ askProposal.idData=getProposal.idData
         ∧ askProposal.income=getProposal.income

    transition GetProposal
      triggeredBy askProposal✉
      guardedBy s=WAIT_PROPOSAL
      effects needAccount'=askProposal.accountIncluded
         ∧ needInsurance'=askProposal.insuranceRequired
         ∧ askProposal.Reply ⊃ s'=WAIT_DECISION
         ∧ ¬askProposal.Reply ⊃ s'=FINAL
      sends getProposal✉
         ∧ getProposal.Reply=askProposal.Reply
         ∧ getProposal.proposal=askProposal.proposal
         ∧ getProposal.cost=(CHARGE/100+1)*750

    transition TimeoutProposal
      triggeredBy now>getProposal.UseBy
      guardedBy s=WAIT_DECISION
      effects s'=FINAL
      sends askProposal×
```



The specifications (the business role - interactions)

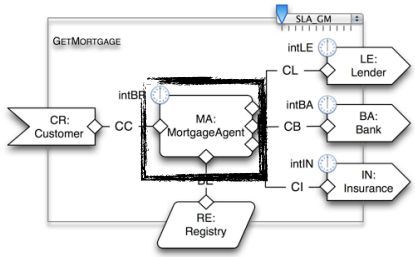
```
BUSINESS ROLE MortgageAgent is
  INTERACTIONS
    r&s getProposal
      🔔 idData:usrdata,
         income:moneyvalue,
         preferences:prefdata,
      ✉ proposal:mortgageproposal
         cost:moneyvalue
    s&r askProposal
      🔔 idData:usrdata,
         income:moneyvalue,
      ✉ proposal:mortgageproposal
         loanData:loandata,
         accountIncluded:bool,
         insuranceRequired:bool
    s&r getInsurance
      🔔 idData:usrdata,
         loanData:loandata,
      ✉ insuranceData:insurancedata
    s&r openAccount
      🔔 idData:usrdata,
         loanData:loandata,
      ✉ accountData:accountdata
    s&r signOutLoan
      🔔 insuranceData:insurancedata,
         accountData:accountdata,
      ✉ contract:loancontract
    snd confirmation
      🔔 contract:loancontract
    ask getLenders(prefdata):setids
    tll regContract(loandata,loancontract)
```

```
  SLA VARIABLES
      CHARGE:[0..100]

  ORCHESTRATION

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

...

```
transition ProposalNotAccepted
  triggeredBy getProposal✖
  guardedBy s=WAIT DECISION
     ∧ now<askProposal.UseBy
  effects s'=FINAL
  sends askProposal✖
```

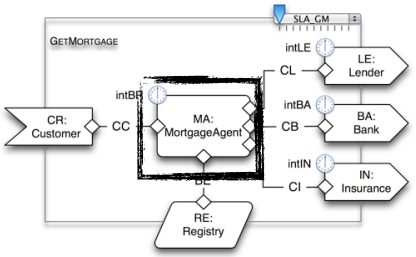
```
transition ProposalAccepted
  triggeredBy getProposal✓
  guardedBy s=WAIT DECISION
     ∧ now<deadline
  effects needAccount ∨ needInsurance ⊃ s'=PROPOSAL ACCEPTED
     ∧ ¬needAccount ∧ ¬needInsurance ⊃ s'=SIGNING
  sends askProposal✓
     ∧ needAccount ⊃ openAccount🔔
             ∧ openAccount.idData=getProposal.idData
             ∧ openAccount.loanData=getProposal.loanData
     ∧ needInsurance ⊃ getInsurance🔔
             ∧ getInsurance.idData=getProposal.idData
             ∧ getInsurance.loanData=getProposal.loanData
     ∧ ¬needAccount ∧ ¬needInsurance ⊃ signOutLoan🔔
             ∧ signOutLoan.insuranceData=insuranceData
             ∧ signOutLoan.accountData=accountData
```



The specifications (the business role - interactions)

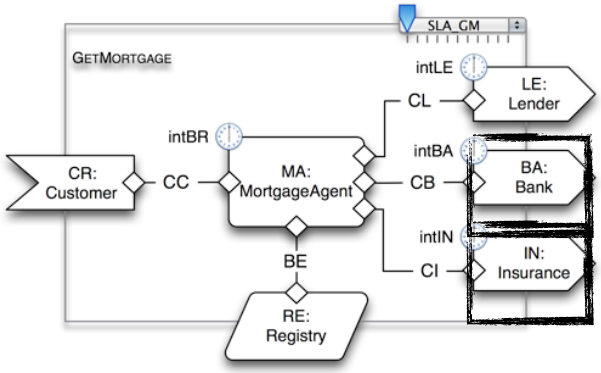
```
BUSINESS ROLE MortgageAgent is

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

```
SLA VARIABLES
    CHARGE:[0..100]

ORCHESTRATION
    local    s:[INITIAL, WAIT_PROPOSAL, WAIT_DECISION,
                PROPOSAL_ACCEPTED, SIGNING, FINAL],
    ...
```




```
transition GetAccount
    triggeredBy openAccount✉
    guardedBy s=PROPOSAL_ACCEPTED
    effects needAccount'=false
       ∧ ¬needInsurance ⊃ s'=SIGNINING
       ∧ accountData=openAccount.accountData
    sends ¬needInsurance ⊃ signOutLoan🔔
                ∧ signOutLoan.insuranceData=insuranceData
                ∧ signOutLoan.accountData=accountData

transition GetInsurance
    triggeredBy getInsurance✉
    guardedBy s=PROPOSAL_ACCEPTED
    effects needInsurance'=false
       ∧ ¬needAccount ⊃ s'=SIGNING
       ∧ insuranceData=getInsurance.insuranceData
    sends ¬needAccount ⊃ signOutLoan🔔
                ∧ signOutLoan.insuranceData=insuranceData
                ∧ signOutLoan.accountData=accountData

transition Conclude
    triggeredBy signOutLoan✉
    guardedBy s=SIGNING
    effects s'=FINAL
    sends confirmation🔔
       ∧ confirmation.contract=signOutLoan.contract
       ∧ regContract(askProposal.loanData,signOutLoan.contract)
```

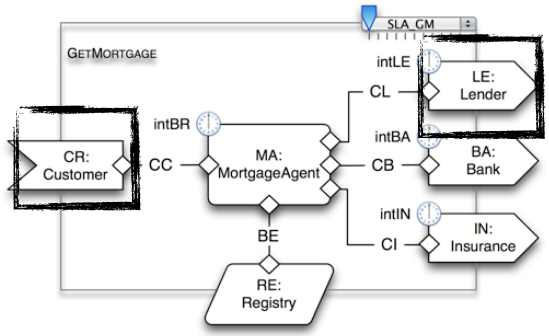
The specifications (the business protocols)




```
BUSINESS PROTOCOL Bank is
    INTERACTIONS
        r&s newMortgageAccount
        🔔 idData:usrdata,
           loanData:loandata,
        ✉ accountData:accountdata
    BEHAVIOUR
        initiallyEnabled newMortgageAccount🔔?
        newMortgageAccount.Reply after newMortgageAccount✉!
```

```
BUSINESS PROTOCOL Insurance is
    INTERACTIONS
        r&s newMortgageInsurance
        🔔 idData:usrdata,
           loanData:loandata,
        ✉ insuranceData:insurancedata
    BEHAVIOUR
        initiallyEnabled newMortgageInsurance🔔?
        newMortgageInsurance.Reply after newMortgageInsurance✉!
```

The specifications (the business protocols)



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]

BEHAVIOUR

initiallyEnabled getProposal🔔?
getProposal.cost≤750*(CHARGE/100+1) **after** getProposal✉!
getProposal✓? **ensures** confirmation🔔!

BUSINESS PROTOCOL Lender **is**

INTERACTIONS

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

r&s requestSignOut
- 🔔 insuranceData:insurancedata,
  accountData:accountdata,
- ✉ contract:loancontract

BEHAVIOUR

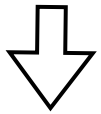
initiallyEnabled requestMortgage🔔?
requestMortgage✓? **enables** requestSignOut🔔?
requestSignOut✉.Reply **after** requestSignOut✉?

END SPECIFICATIONS

A view of ensemble

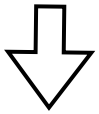
use case diagrams

Define requirements



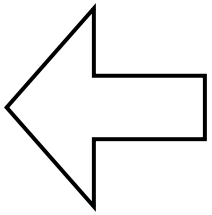
from scratch
use case -> SRML

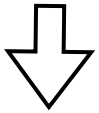
Define modules + EX-Is



from scratch
CBD

Define internal structure
(textual notation)





from scratch
reuse
refinement

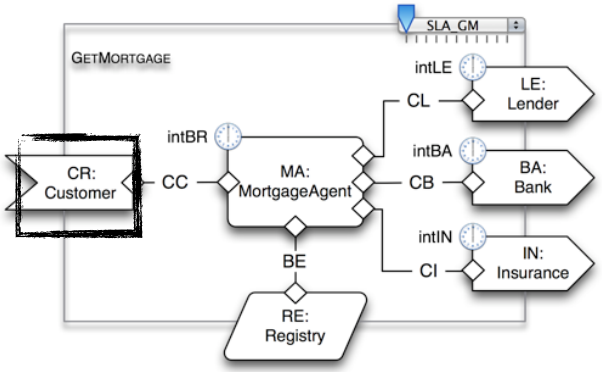
Define behaviours

The textual definition (nodes)

MODULE GETMORTGAGE **is**

DATATYPES

sorts: usrdata, prefdata, moneyvalue, mortgageproposal, loandata, loancontract, insurancedata, accountdata, setids, bool, nat



PROVIDES

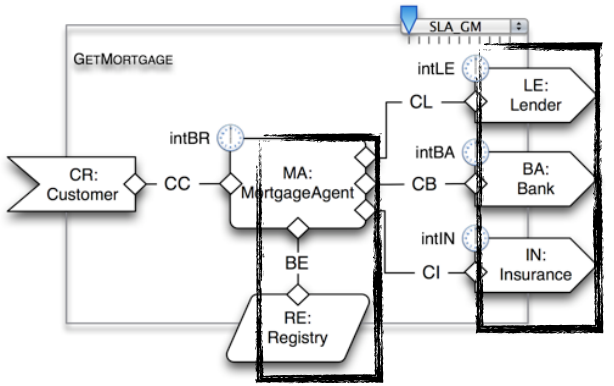
CR: Customer

CR Customer	**MA** MortgageAgent
r&s getProposal 🔔 idData income preferences ✉ proposal cost	**r&s** getProposal 🔔 idData income preferences ✉ proposal cost
snd confirmation 🔔 contract	**snd** confirmation 🔔 contract
SLA VARIABLES CHARGE	**SLA VARIABLES** CHARGE

...

The textual definition (nodes and internal policies)

```
REQUIRES
   LE: Lender
         intLE  trigger: getproposal?
   BA: Bank
         intBA  trigger: default
   IN: Insurance
         intIN  trigger: default

COMPONENTS
   MA: MortgageAgent
         intMA  init: s=INITIAL
         intMA  term: s=FINAL

USES
   RE: Registry

   ...
```



The textual definition (external policies)

EXTERNAL POLICY

SLA VARIABLES

MA.CHARGE, MA.getProposal💣,
LE.ServiceId, LE.COST, LE.requestMortgage💣

CONSTRAINTS

C1 relates the charge with the deadline of getProposal

C_1: {MA.CHARGE, MA.getProposal💣}

$$def(c,t)= \begin{cases} 1 \text{ if } t \le 10*c \\ 1+2*c-0.2*t \text{ if } 10*c < t \le 5+10*c \\ 0 \text{ otherwise} \end{cases}$$

e.g., if I give 2 hours and the charge is more than 20 then I am happy

e.g., if I give 2 hours and the charge is between 10 and 20 then the satisfaction depends on c and t

C2 enforces that the chosen lender is in the set of trusted lenders stored in the registry

C_2: {LE.ServiceId}

$$def(s)= \begin{cases} 1 \text{ if } s \in MA.lenders \\ 0 \text{ otherwise} \end{cases}$$

C3 ensures that the deadline for proposal can be met wrt the delays

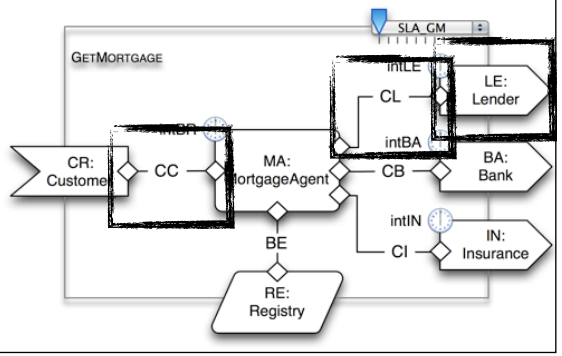
C_3: {MA.getProposal💣, LE.requestMortgage💣},

$$def(t1,t2)= \begin{cases} 1 \text{ if } t2 > t1 + CC.Delay + CL.Delay \\ 0 \text{ otherwise} \end{cases}$$

C4 ensures that the cost is inversely proportional to the satisfaction and directly proportional to the deadline of requestMortgage

C_4: {LE.COST, LE.requestMortgage💣}

$$def(c,t)= \begin{cases} \frac{1}{c}+\frac{t}{100} \text{ if } c < 500 \\ 0 \text{ otherwise} \end{cases}$$



The textual definition (wires part1)

WIRES

MA MortgageAgent	c_4	**BE**	d_4	**RE** Registry
ask getLenders	S_1	Straight. A(prefdata)R(setids)	R_1	**rpl** getLenders
tll regContract	S_1	Straight. T(loandata,loancontract)	R_1	**prf** registerContract

MA MortgageAgent	c_1	**CB**	d_1	**BA** Bank
s&r openAccount	S_1	Straight.	R_1	**r&s** newMortgageAccount
🔔 idData	i_1	I(usrdata,	i_1	🔔 idData
loanData	i_2	loandata)	i_2	loanData
✉ accountData	o_1	O(accountdata)	o_1	✉ accountData

MA MortgageAgent	c_1	**CI**	d_1	**IN** Insurance
s&r getInsurance	S_1	Straight.	R_1	**r&s** newMortgageInsurance
🔔 idData	i_1	I(usrdata,	i_1	🔔 idData
loanData	i_2	loandata)	i_2	loanData
✉ insuranceData	o_1	O(insurancedata)	o_1	✉ insuranceData

...

The textual definition (wires part2)

c_1	**CC**	d_1	**MA** MortgageAgent
S_1 i_1 i_2 i_3 o_1 o_2	Straight. I(usrdata, moneyvalue,prefdata) O(mortageproposal, moneyvalue)	R_1 i_1 i_2 i_3 o_1 o_2	**r&s** getProposal 🔔 idData income preferences ✉ proposal cost
R_1 i_1	Straight O(loancontract)	S_1 i_1	**snd** confir-mation 🔔 contract

END MODULE

This was the end of the module
The module refers to a number of specifications (as seen before)

A view of ensemble

use case diagrams

from scratch
use case -> SRML

from scratch
CBD

from scratch
reuse
refinement

PEPA
Logics of Interactions

Define requirements

⇩

Define modules + EX-Is

⇩

Define internal structure

⇩

Define behaviours

⇩

Analyse

⇩

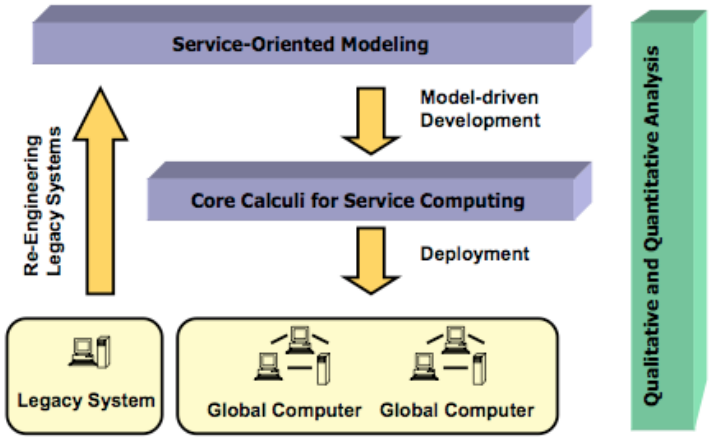
Deploy

SRML2PEPA

A Formal Approach to Model Time Properties in Service-Oriented Systems
Bocchi, Fiadeiro, Gilmore, Abreu, Solanki, Vankayala
http://www.cs.le.ac.uk/people/jfiadeiro/Papers/SRML-T.pdf

PEPA

SRML



- PEPA is a process algebra for stochastic quantitative analysis whose building entities are (1) components and (2) activities (a,r) where a is an action and r is a rate
- There are tools for PEPA (Eclipse Plugin) that allow to make quantitative analysis
- We provided an encoding from SRML modules to PEPA terms to perform quantitative analysis on SRML modules

SRML -> PEPA

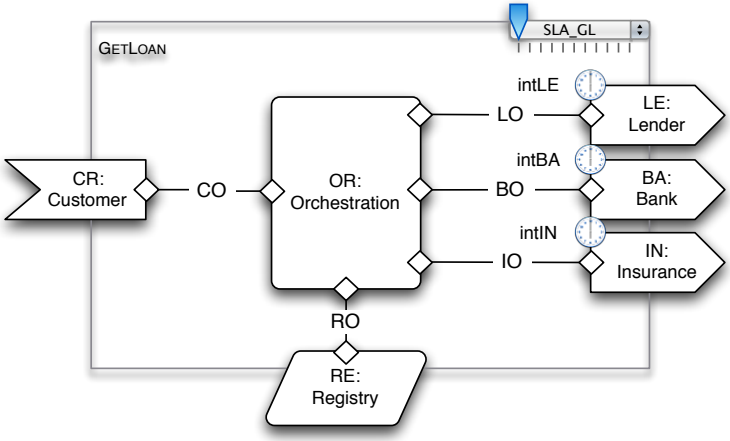
- The aim is to enable quantitative analysis of SRML modules
- SRML describes behaviours terms of interaction events
- We want to determine the delay between couple of interaction events

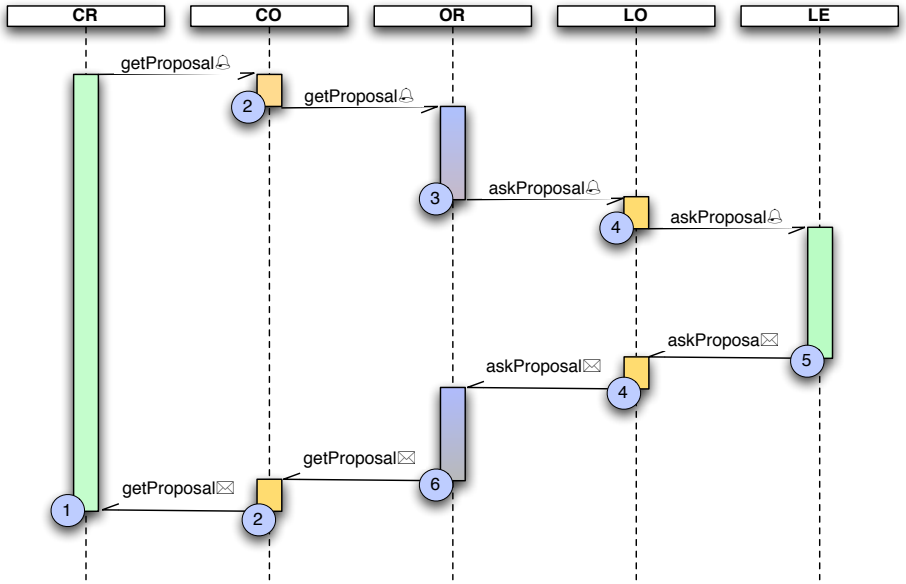
"is the reply to the event getProposal, in the 80% of the cases, received in 7s?"

- We want to determine how the single rates influence the overall delay between couples of interaction events

"which rates worth to be improved?"

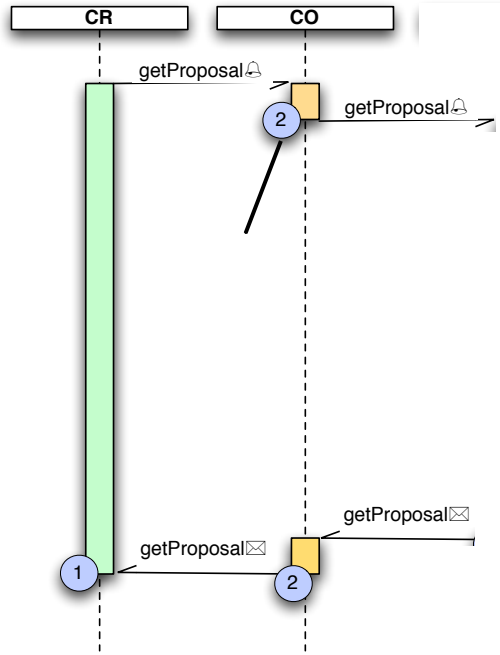
The aim of the encoding

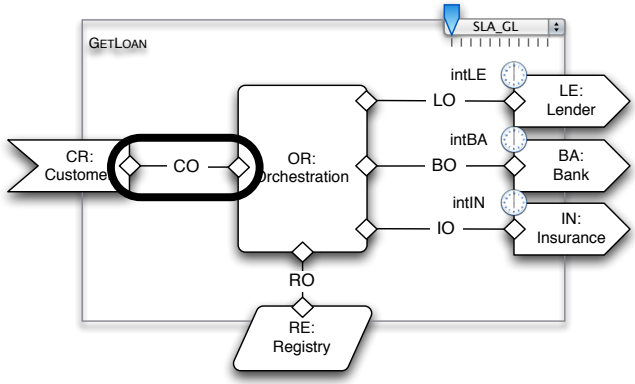




- We want to determine the delay between couples of events in the provides-interface
- For example, we want to analyze the delay between getProposal🔔 and getProposal✉ in CR ((1) in the sequence diagram)
- In order to analyze time-related properties in a SRML module:
  - we determine which delays occur in a SRML module
  - we encode SRML into PEPA

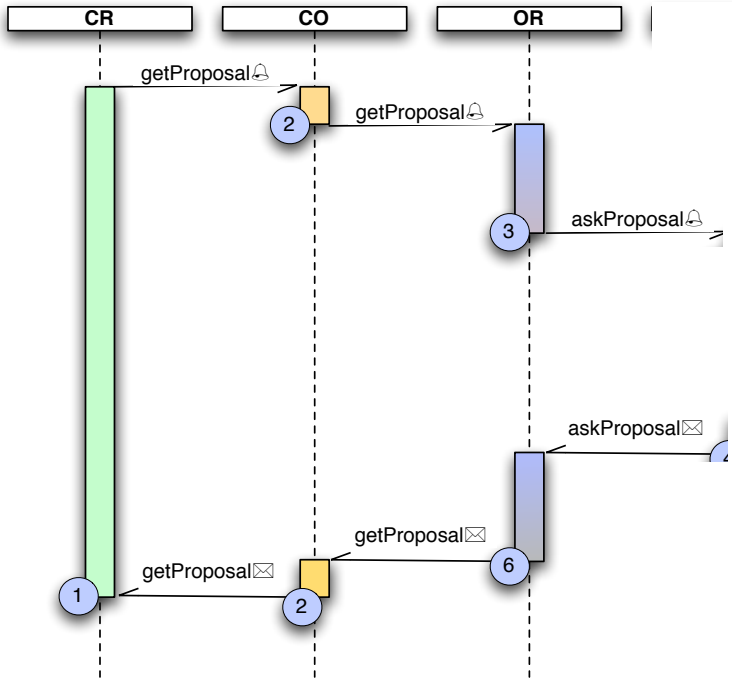
Delays in SRML

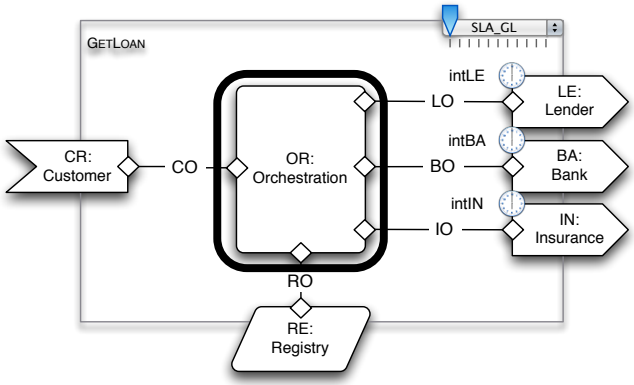




- CO transfers getProposal🔔 to from CR to OR
- Wires can take some amount of time to transfer events between nodes
  - because of bandwidth/capacity
  - because of the execution of an interaction protocol
- Each wire has its own TRANSFER RATE
- For example, the delay of wire CO is CO.transferRate

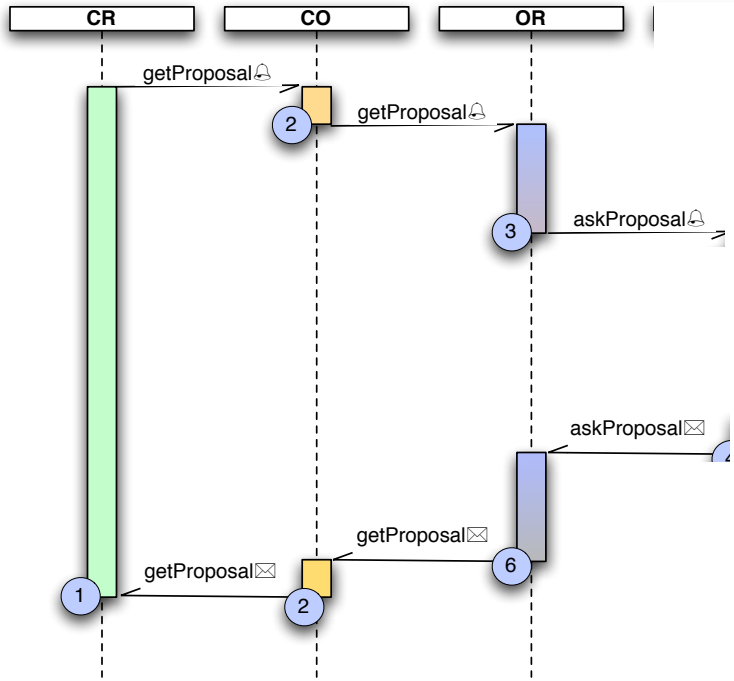
Delays and dependencies



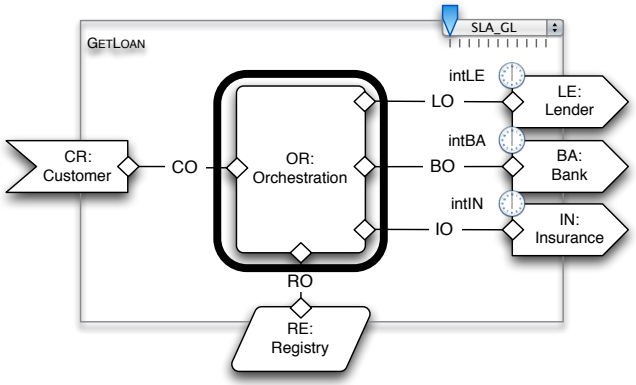


- The event getProposal🔔 is stored in OR's buffer, waiting to be processed
- Each component is associated to a PROCESSING RATE, which represents the delay of processing a received event
- All the events received by a specific component are affected by the same processing rate
- For example the delay of OR for processing events is processingRate(OR)

Delays and dependencies

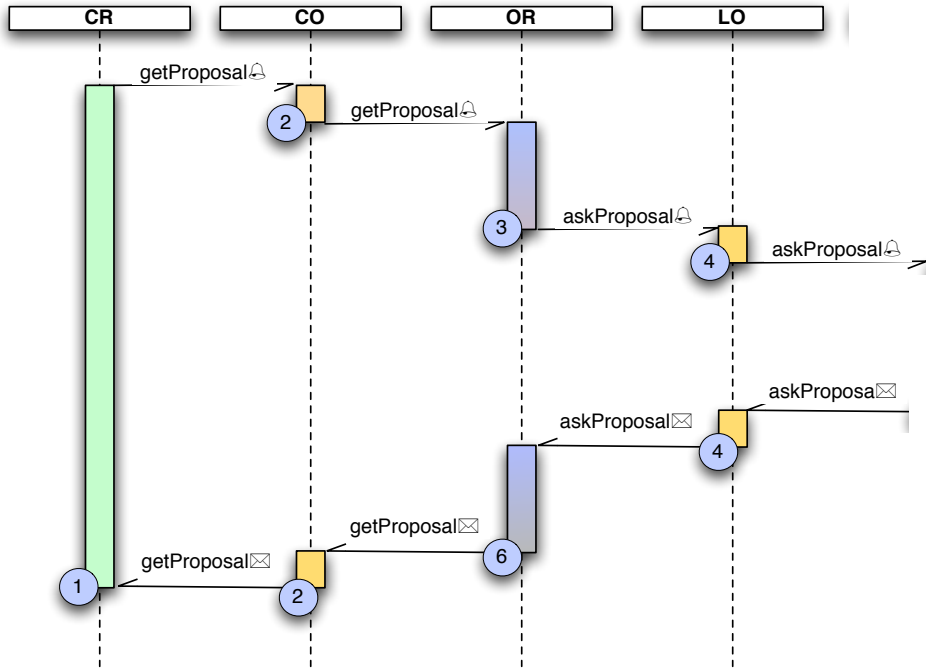


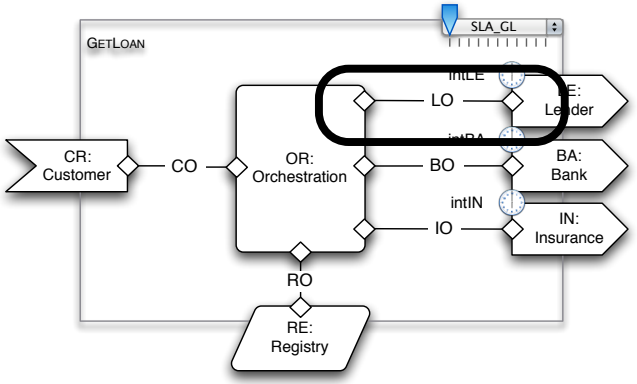

```
transition P1
    triggeredBy getProposal
    guardedBy s=INITIAL
    effects s'=WAIT_PROPOSAL
    sends askProposal
```



- When the event is processed it can be either executed or discarded
- The execution of an event corresponds to the execution of a transition.
- Each transition is associated to an EXECUTION RATE which represents the time taken to compute the reaction to an event, for example to execute transition P2
- The execution rate is different for any transition of the same component (e.g., executionRate(OR)(P1))

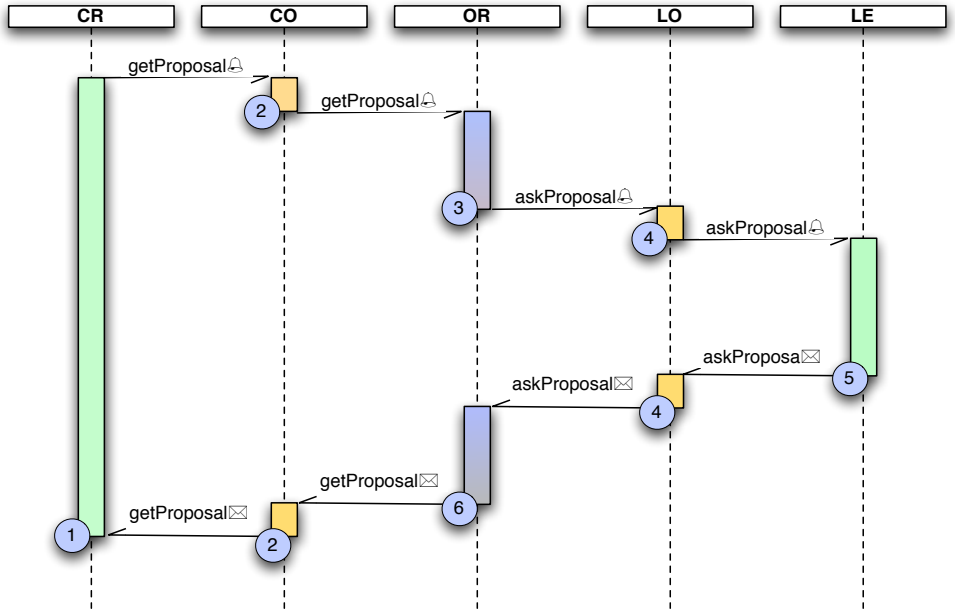
Delays and dependencies

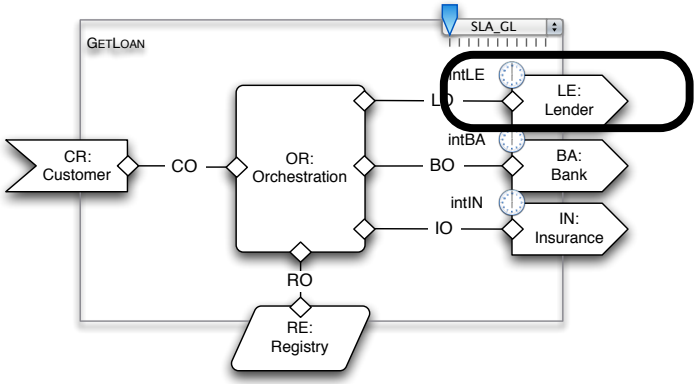




- Again there is a delay due to a wire
- (4) represents transferRate(LO)

Delays and dependencies

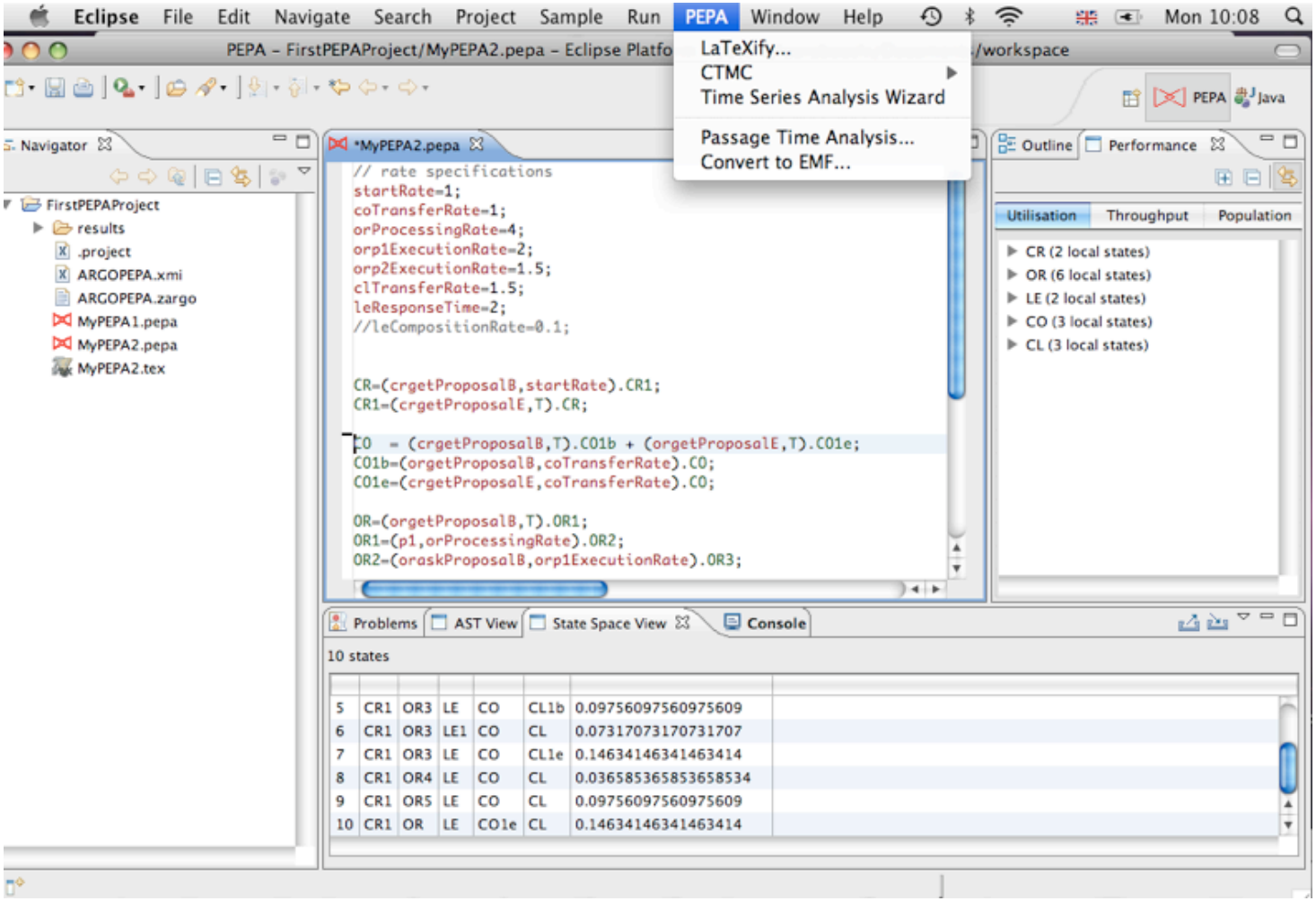




- when a requires interface is discovered and bound at run time, we have a delay which we call COMPOSITION RATE
- For example, the composition rate of LE is compositionRate(LE)
- Also, each requires-interface is associated to a RESPONSE RATE representing the time taken by an external interface to reply to an event
- e associate a responseRate to every r&s (delay between 🔔-event and ✉-event)
- (5) is compositionRate(LE) + responseRate(LE)(askProposal🔔,askProposal✉)

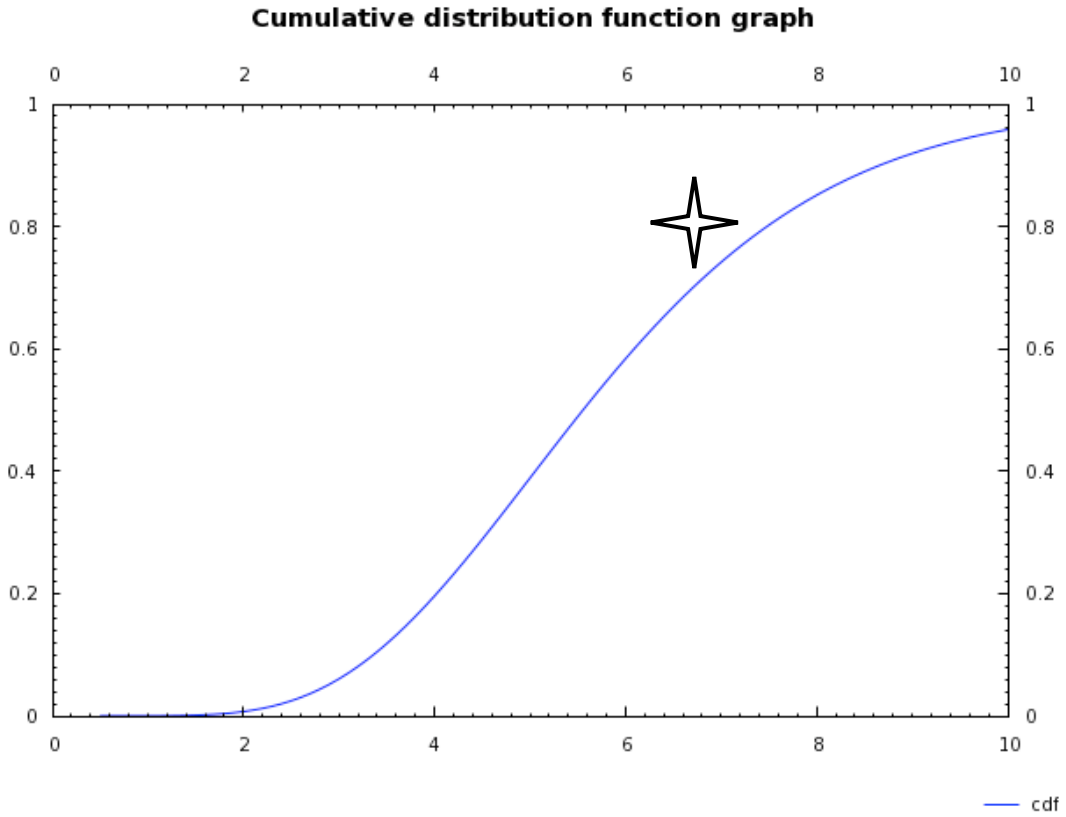
PEPA Eclipse Plugin

- After having extended a SRML module with delays, associating one or more rate to each component, EX-I, and wire, we can encode the module into a PEPA term

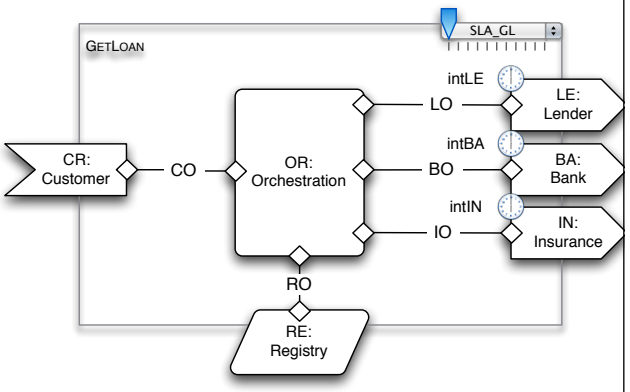


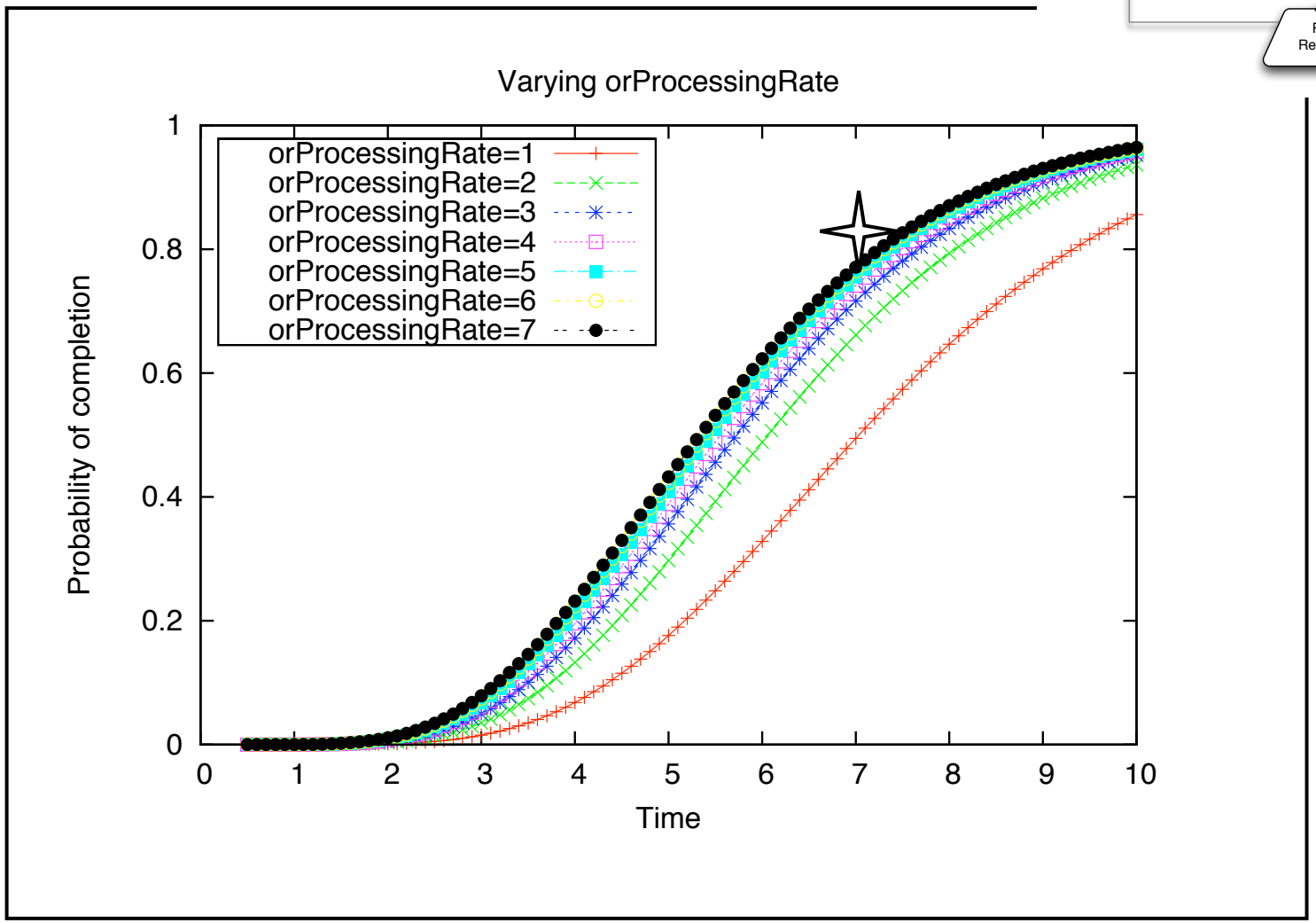
Passage Time Analysis

- Using the Passage Time Analysis we analyse for which rates the following holds: "In 80% of the cases, the delay between getProposal🔔 and getProposal✉ has an upper bound of 7s".

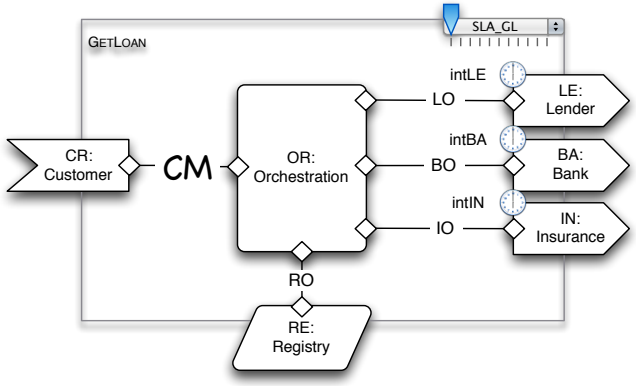


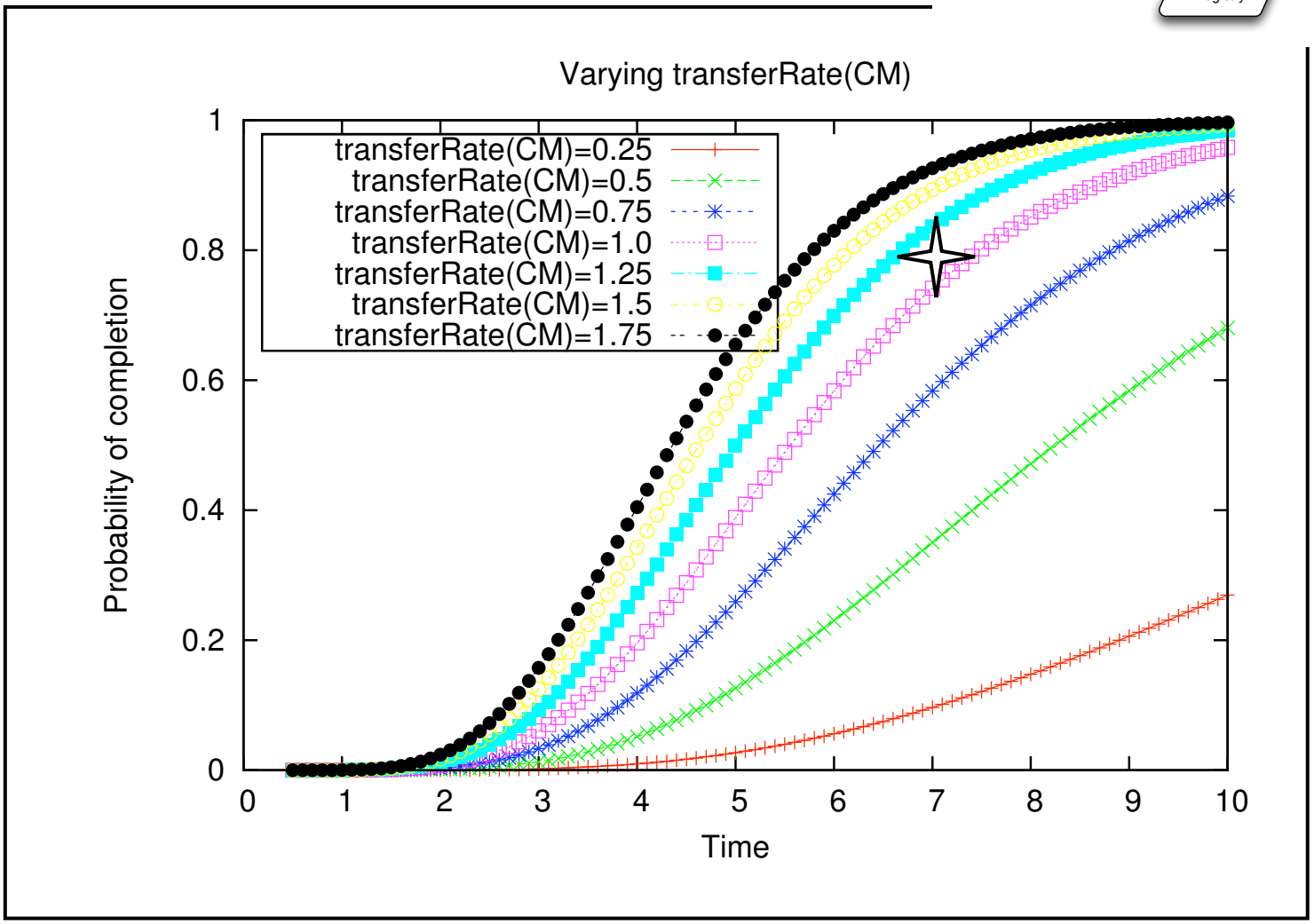
Passage Time Analysis





Passage Time Analysis





A view of ensemble

use case diagrams

from scratch
use case -> SRML

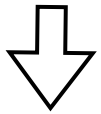
from scratch
CBD

from scratch
reuse
refinement

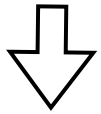
PEPA
Logics of Interactions

SCA

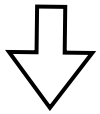
Define requirements



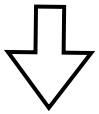
Define modules + EX-Is



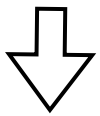
Define internal structure



Define behaviours



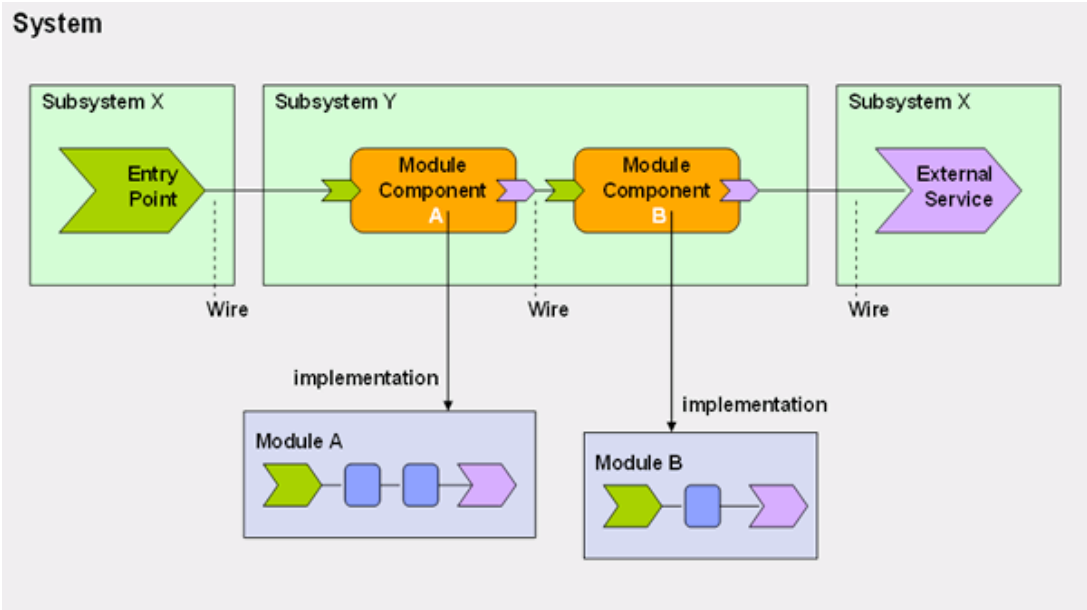
Analyse



Deploy

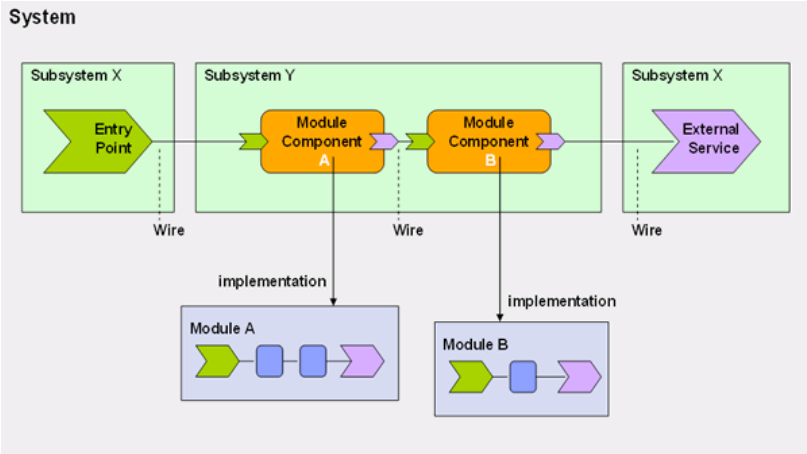
SCA

- The Service Component Architecture (SCA) is a recent set of specifications, proposed by an industrial consortium including major vendors like IBM, ORACLE, BEA, etc.
- SCA describe a middleware-independent model for building over SOAs.
- Similarly to SCA, SRML provides primitives for modelling, in a technology agnostic way, business processes as assemblies of
  - (1) tightly coupled components that may be implemented using different technologies (including wrapped-up legacy systems, BPEL, Java, etc.)
  - (2) loosely coupled, dynamically discovered services.



SCA

- Differently from SRML, SCA is not a modelling language but a framework for modelling the structure of a service-oriented software artefact and for its deployment.
- SCA abstracts from the business logic provided by components in the sense that it does not provide a means to model the behavioural aspects of services.
- SRML is, instead, a modelling language that
  - provides the primitives to specify such behavioural aspects.
  - relies on a mathematical framework for reconfiguration, behavioural interfaces and SLA



A view of ensemble

use case diagrams

from scratch
use case -> SRML

from scratch
CBD

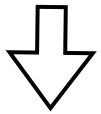
from scratch
reuse
refinement

BPEL
StPowla

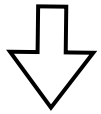
PEPA
Logics of Interactions

SCA

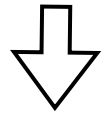
Define requirements



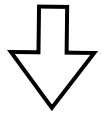
Define modules + EX-Is



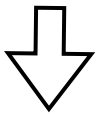
Define internal structure



Define behaviours



Analyse



Deploy

BPEL2SRML

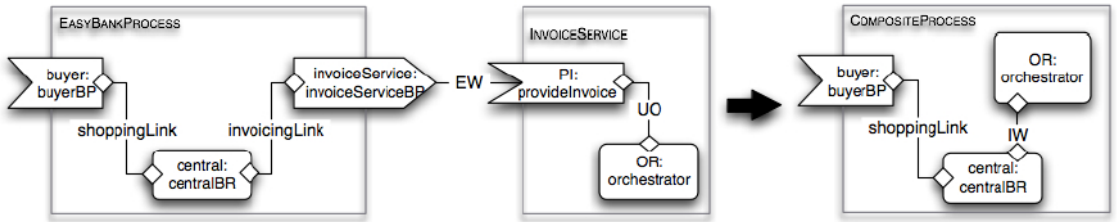
From BPEL to SRML: a formal transformational approach
Bocchi, Hong, Lopes and Fiadeiro, WSFM 2008
http://www.cs.le.ac.uk/people/jfiadeiro/Papers/BPEL2SRML.pdf

- WS-BPEL is an OASIS specification for defining business processes of Web services
- We defined an encoding to extract SRML models out of existing BPEL processes
- In fact we defined an encoding of (part of) BPEL into SRML
- Aspects of session/fault/configuration management still have to be added

BPEL Tag/Construct	Tool	Encoding
Invoke , Receive, Reply, Assign (BA)	✓	✓
Wait, Empty, Exit (BA)	✗	✓
Throw (BA)	✗	✗
Sequence, Switch (SA)	✓	✓
Flow, While (SA)	✗	✓
Control Links, Scopes, Correlation Sets	✗	✗

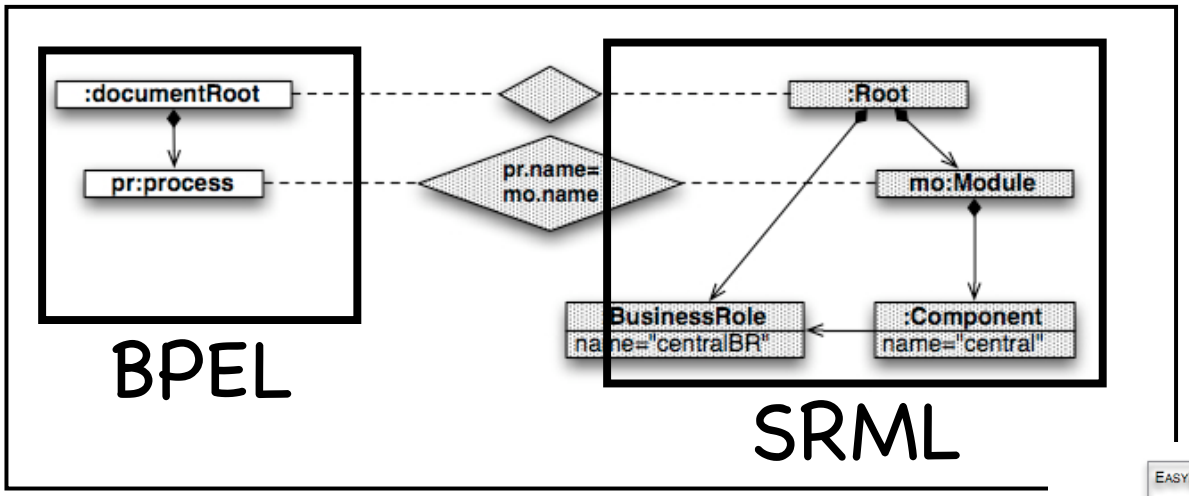
Model Extraction
BPEL and SRML: Advantages

- The main aim is not to provide BPEL with a formal semantics
- The aim is
  - to enable extraction of models,
  - to provide a library of models (SRML components),
  - to allow the models deriving from existing processes to be used to define other SRML models. The components may be more than one and they can be
    - derived by BPEL processes,
    - defined from scratch,
    - derived from any language for which an encoding into SRML exists.
  - To allow the ensemble to be analysed within one formal framework.

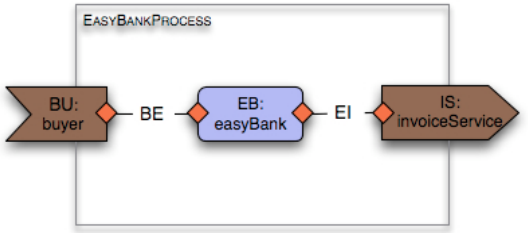


The encoding BPEL2SRML

- The encoding has been done as follows...
  - EMF tree for WSDL/BPEL derived from XSD and Eclipse BPEL project
  - EMF tree for SRML being refined (while implementing the SRML editor)
  - Design of transformation rules (structure and behaviour)



- A BPEL process is encoded into a SRML module with one component



The encoding BPEL2SRML

- For every transition A we define a variable ra ("a is ready") and fa ("a has finished"), which are initially false.
- The encoding of a simple activity (receive)

```
<receive partnerLink="pl"
   portType="pt"
    operation="op"
   variable="v"
    createInstance=…/>
```

→

```
transition transition_A
triggeredBy pt.op🔔?
guardedBy ra ∧ ¬exit
effects ¬ra' ∧ fa'
  ∧ v.p1'=pt.op🔔.v.p1 ∧ …
  ∧ v.pn'=pt.op🔔.v.pn
```

- The encoding of a structured activity (sequence)

```
<sequence
   name="X">
   activity  A
   activity  B
</sequence>
```

The transition is executed the first time when X is triggered

The transition is executed the second time when A terminates

```
transition transitio_
triggeredBy(rx ∨ fa ∨ fb)  ∧ ¬exit
guardedBy (rx ∨ fa ∨ fb)  ∧ ¬exit
effects (rx ⊃ ra'∧¬rx) ∧ (fa ⊃ rb'∧¬fa') ∧ (fb ⊃ fx'∧¬fb')
```

The effect is to enable the execution of A

The effect is to trigger B

When the last activity terminates, the end of X is notified

A view of ensemble

use case diagrams

from scratch
use case -> SRML

from scratch
CBD

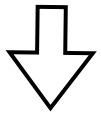
StPowla
BPEL

from scratch
reuse
refinement

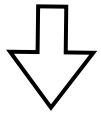
PEPA
Logics of Interactions

SCA

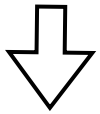
Define requirements



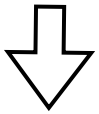
Define modules + EX-Is



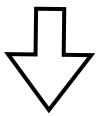
Define internal structure



Define behaviours



Analyse



Deploy

StPowla2SRML

- StPowla

[S Gorton, C Montangero, S Reiff-Marganiec and L Semini. StPowla: SOA, Policies and Workflows. WESOA 2007]

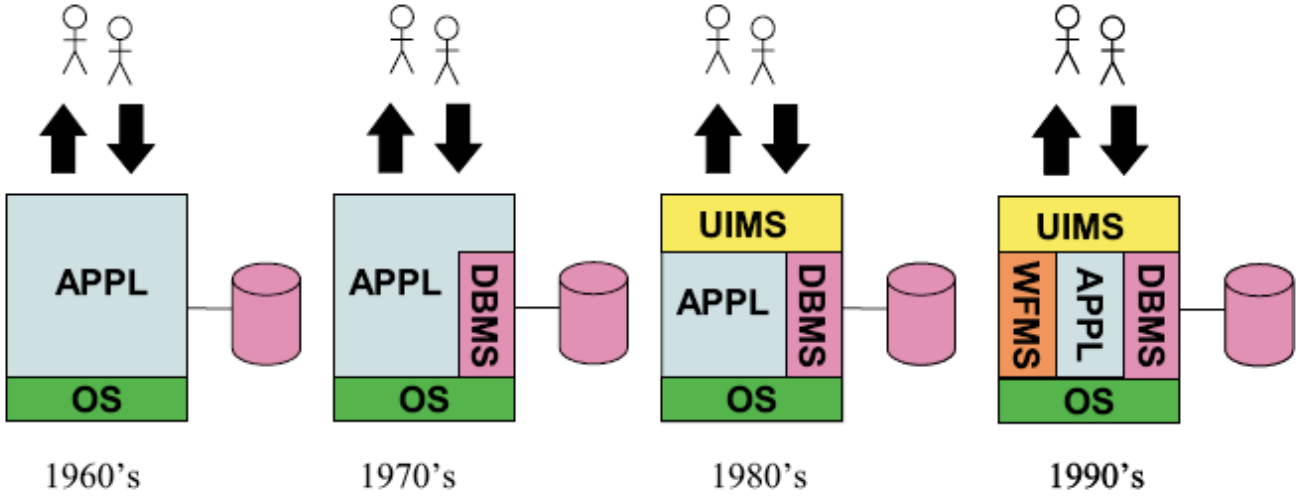
is a service-targeted, policy-oriented, workflow approach

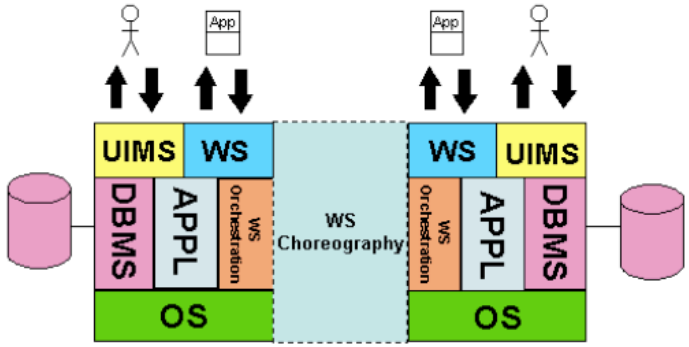
- workflows
- reconfiguration through policies

- StPowla has been encoded into SRML in order to
  - provide StPowla with a formal framework
  - add a higher level of modelling in SRML

Engineering Service Oriented Applications: From StPowla Processes to SRML Models
Bocchi, Gorton, Reiff-Marganiec, FASE 2008
http://www.cs.le.ac.uk/people/srm13/publications/fase08.pdf

A bit of history...





Workflow

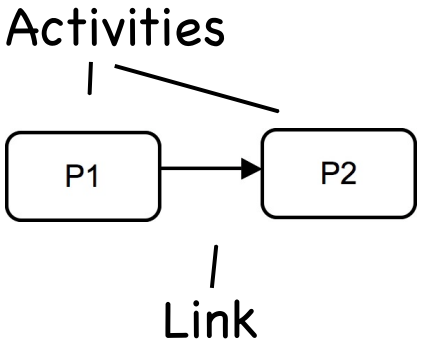
- The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.

Business Process

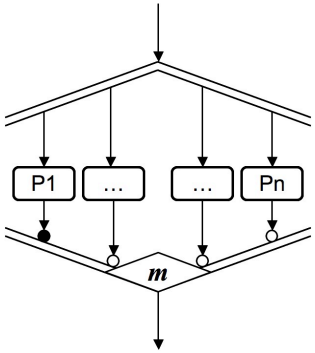
- A set of one or more **linked** procedures or **activities** which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships.

from WFMC (Workflow Management Coalition) Glossary
(http://www.aiai.ed.ac.uk/project/wfmc/ARCHIVE/DOCS/glossary/glossary.html)

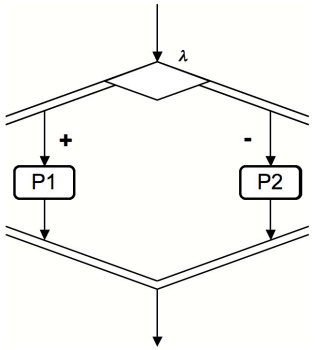
A sample notation...



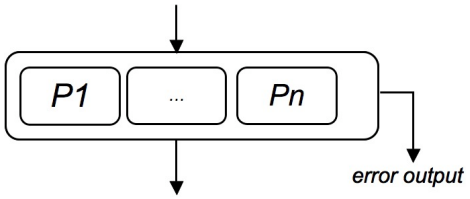
sequence



split join

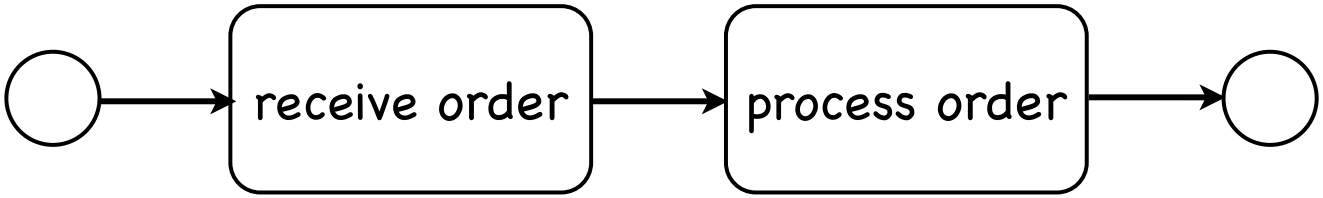


condition join



random preference

Example



- Business Modelling
  - concerned with the ordering of tasks in an execution model
  - kept at a high-level for end-users to create

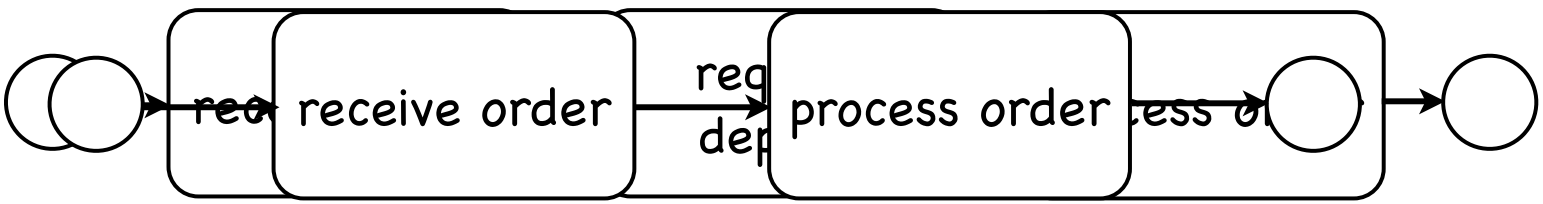
Business Policies

- There exists a need for flexibility: customization of a core model to handle variability in domain

- Business models are subjected to overarching constraints (e.g., business rules, global or enterprise constraints) expressed as business policies

```
getDepositIfLargeOrder
   appliesTo receiveOrder
      when task_completion
         if receiveOrder.orderValue > £250000 do
         insert(requestDeposit, receiveOrder, true)
```

request deposit



StPowla: Reconfiguration Functions

- fail() -
declares the current task to have failed (i.e., discard further task processing and generates the task_failure event)

- abort() -
aborts the current task and progresses to the next task, generating the task_abort event

- block(s,p) -
waits until predicate p is true before commencing scope s

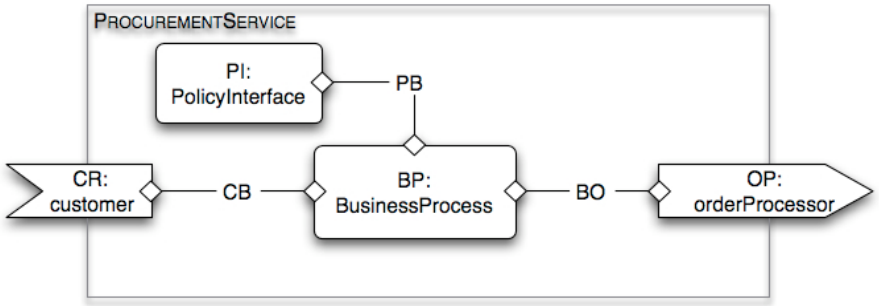
- insert(x,y,z) -
inserts task or scope y into the current workflow instance after task x if z is true, or in parallel with x is z is false

- delete(x) -
deletes task/scope x from the current workflow instance

StPowla2SRML

- The internal structure of the SRML module is organised in two components: one implementing the business process and one implementing the policy interface

- The policy interface determines when a policy requires a reconfiguration and notifies the business process component



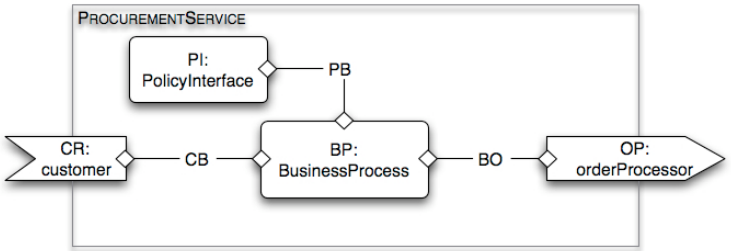
- BP has one interaction for each of the reconfiguration functions...

```
BUSINESS ROLE BusinessProcess is
INTERACTIONS
 r&s delete[i:natural]
   🔔  task:taskId
 r&s insert[i:natural]
   🔔  task:taskId
       newTask:taskId
       c:condition
 ...
 r&s fail[i:natural]
   🔔  task:taskId
 r&s abort[i:natural]
   🔔  task:taskId
 r&s block[i:natural]
   🔔  task:taskId
       c:condition
```

StPowla2SRML

- The business role of BP has one or more transition that model the reaction to each reconfiguration function

- The the transitions for the delete task reconfiguration function are presented below:



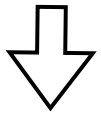

```
transition policyHandler_delete_1
     triggeredBy delete[i]🔔?
     guardedBy state[delete[i]🔔.task]=toStart
     effects policy[delete[i]🔔.task]'

transition policyHandler_delete_2
     triggeredBy start[x]
     guardedBy P_delete[i]🔔? ∧ P_delete[i]🔔.task=x
     effects ¬start[x]' ∧ done[x]' ∧ state[x]'=done
     sends delete[i]✉!
```

- The encoding of the workflow constructs is similar to the one of the encoding from BPEL

A view of ensemble

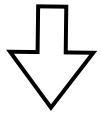
use case diagrams

Define requirements



from scratch
use case -> SRML

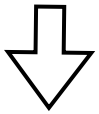
Define modules + EX-Is



from scratch
CBD

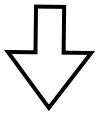
StPowla
BPEL

Define internal structure



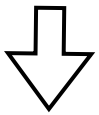
from scratch
reuse
refinement

Define behaviours



PEPA
Logics of Interactions

Analyse



SCA

Deploy

A view of ensemble