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)
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
  - From scratch
  - Use case -> SRML
- Define requirements
- Define modules + EX-Is
- Define internal structure
- 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

Define modules + EX-Is

Define internal structure

more component could be defined for modular development, reuse, limiting the complexity etc.
A view of ensemble

use case diagrams

from scratch
use case -> SRML

from scratch
CBD

from scratch
reuse
refinement

Define requirements

Define modules + EX-Is

Define internal structure

Define behaviours
The specifications (interactions)

**BUSINESS ROLE** MortgageAgent

**INTERACTIONS**

- r&s getProposal
  - idData: usrdatalnt
  - income: moneyvalue
  - preferences: prefdata
  - proposal: mortgageproposal
  - cost: moneyvalue

- s&r askProposal
  - idData: usrdatalnt
  - income: moneyvalue
  - proposal: mortgageproposal
  - loanData: loandata
  - accountIncluded: bool
  - insuranceRequired: bool

- s&r getInsurance
  - idData: usrdatalnt
  - loanData: loandata
  - insuranceData: insuredatalnt

- s&r openAccount
  - idData: usrdatalnt
  - loanData: loandata
  - accountData: accountdata

- s&r signOutLoan
  - insuranceData: insuredatalnt
  - accountData: accountdata
  - contract: loancontract

- snd confirmation
  - contract: loancontract

- ask getLenders(prefdata): setIds

- ttl 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
The specifications
(the business role - interactions)
The specifications
(the business role - interactions)

BUSINESS ROLE MortgageAgent is

INTERACTIONS
r&s getProposal
  idData:usrdta, income:moneyvalue
  proposal:mortgageproposal
  cost:moneyvalue
s&r askProposal
  idData:usrdta, income:moneyvalue
  proposal:mortgageproposal
  loanData:loandata, account:Included:boolean
  insurance:Required:boolean
s&r getInsurance
  idData:usrdta, loanData:loandata,
  insuranceData:insurancedata
s&r openAccount
  idData:usrdta, loanData:loandata,
  accountData:accountdata
s&r signOutLoan
  insuranceData:insurancedata,
  accountData:accountdata,
  contract:loancontract
end confirmation
  contract:loancontract
  ask Lenders(prefdata):setids
til regContract(loandata,loancontract)

|E AL VARIABLES
  Charge:[0..100]

ORCHESTRATION
local

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 v 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**

- **getProposal**
  - `idData:usrdata, income:moneyvalue, preferences:prefdata, proposal:mortgageproposal`
  - `cons:moneyvalue`

- **askProposal**
  - `idData:usrdata, income:moneyvalue, proposal:mortgageproposal`
  - `loanData:loanData, account:Included:bool, insurance Required:bool`

- **getInsurance**
  - `idData:usrdata, loanData:loanData, insuranceData:insuranceData`

- **openAccount**
  - `idData:usrdata, loanData:loanData, accountData:accountData`

- **signOutLoan**
  - `insuranceData:insuranceData, accountData:accountData, contract:loancontract`

- **confirmation**
  - `contract:loancontract`

**SLA VARIABLES**

- `Charge:[0..100]`

**ORCHESTRATION**

- `local s:[INITIAL, WAIT_PROPOSAL, WAIT_DECISION, PROPOSAL_ACCEPTED, SIGNING, FINAL],`
The specifications (the business protocols)

**BUSINESS PROTOCOL** Bank is

**INTERACTIONS**

r&s newMortgageAccount  
 applaud idData:userdata, loanData:loandata,  
 applause accountData:accountdata

**BEHAVIOUR**

- **initiallyEnabled** newMortgageAccount??
- newMortgageAccount.Reply after newMortgageAccount!!

**BUSINESS PROTOCOL** Insurance is

**INTERACTIONS**

r&s newMortgageInsurance  
 applause idData:userdata, loanData:loandata,  
 applause insuranceData:insurancedata

**BEHAVIOUR**

- **initiallyEnabled** newMortgageInsurance??
- newMortgageInsurance.Reply after newMortgageInsurance!!
The specifications
(the business protocols)

**BUSINESS PROTOCOL** Customer is

**INTERACTIONS**

- r&s getProposal
  - ? idData:userdata,
  - ? income:moneyvalue,
  - ? preferences:prefdata,
  - ? proposal:mortgageproposal
  - ? cost:moneyvalue
- snd confirmation
- ? contract:loancontract

**SLA VARIABLES**

- charge:[0..100]

**BEHAVIOUR**

- initiallyEnabled getProposal?
- getProposal.costs750*(charge/100+1) after getProposal!
- getProposal->? ensures confirmation!!

**BUSINESS PROTOCOL** Lender is

**INTERACTIONS**

- r&s requestMortgage
- ? idData:userdata,
- ? 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

from scratch
use case -> SRML

from scratch
CBD

from scratch
reuse
refinement

Define requirements

Define modules + EX-Is

Define internal structure
(textual notation)

Define behaviours
The textual definition (nodes)

**MODULE** GetMortgage **is**

**DATATYPES**

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

**PROVIDES**

<table>
<thead>
<tr>
<th>CR: Customer</th>
<th>MA: MortgageAgent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>r&amp;x</strong></td>
<td>getProposal</td>
</tr>
<tr>
<td></td>
<td>idData</td>
</tr>
<tr>
<td></td>
<td>income</td>
</tr>
<tr>
<td></td>
<td>preferences</td>
</tr>
<tr>
<td></td>
<td>proposal</td>
</tr>
<tr>
<td></td>
<td>cost</td>
</tr>
<tr>
<td><strong>snd</strong></td>
<td>confirmation</td>
</tr>
<tr>
<td></td>
<td>contract</td>
</tr>
<tr>
<td><strong>SLA VARIABLES</strong></td>
<td><strong>SLA VARIABLES</strong></td>
</tr>
<tr>
<td>Charge</td>
<td>Charge</td>
</tr>
</tbody>
</table>
The textual definition (nodes and internal policies)

**REQUIRES**

- **LE**: Lender
  - intLE\(\text{trigger}\): getproposal\(\oplus\)?
- **BA**: Bank
  - intBA\(\text{trigger}\): default
- **IN**: Insurance
  - intIN\(\text{trigger}\): default

**COMPONENTS**

- **MA**: MortgageAgent
  - intMA\(\text{init}\): s=INITIAL
  - intMA\(\text{term}\): s=FINAL

**USES**

- **RE**: Registry

...
The textual definition (external policies)

**EXTERNAL POLICY**

<table>
<thead>
<tr>
<th>SLA variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA.Charge, MA.getProposal, LE.ServiceId, LE.Cost, LE.requestMortgage</td>
</tr>
</tbody>
</table>

**Constraints**

- **C1** relates the charge with the deadline of getProposal:
  
  \[
  C_1: \{ \text{MA.Charge, MA.getProposal} \} \]

- **C2** enforces that the chosen lender is in the set of trusted lenders stored in the registry:
  
  \[
  C_2: \{ \text{LE.ServiceId} \} \]

- **C3** ensures that the deadline for proposal can be met with the delays:
  
  \[
  C_3: \{ \text{MA.getProposal, LE.requestMortgage} \} \]

- **C4** ensures that the cost is inversely proportional to the satisfaction and directly proportional to the deadline of requestMortgage:
  
  \[
  C_4: \{ \text{LE.Cost, LE.requestMortgage} \} \]

**Example Constraints**

- If I give 2 hours and the charge is more than 20, then I am happy:
  
  \[
  \text{def}(c,t)=\begin{cases} 1 \text{ if } t \leq 10 \times c \\ 0 \text{ otherwise} \end{cases} \]

- If I give 2 hours and the charge is between 10 and 20, then the satisfaction depends on \( c \) and \( t \):
  
  \[
  \text{def}(c,t)=\begin{cases} 1+2\times c-0.2\times t \text{ if } 10 \times c < t \leq 5 + 10 \times c \\ 0 \text{ otherwise} \end{cases} \]

- If I give 2 hours and the charge is between 10 and 20, then the satisfaction depends on \( c \) and \( t \):
  
  \[
  \text{def}(c,t)=\begin{cases} 1 \text{ if } s \in \text{MA.lenders} \\ 0 \text{ otherwise} \end{cases} \]

- If \( t \geq t_1 + CC.Delay + CL.Delay \), otherwise:
  
  \[
  \text{def}(t_1,t_2)=\begin{cases} 1 \text{ if } t_2 > t_1 + CC.Delay + CL.Delay \\ 0 \text{ otherwise} \end{cases} \]

- If \( c < 500 \), otherwise:
  
  \[
  \text{def}(c,t)=\begin{cases} \frac{1}{c} + \frac{t}{100} \text{ if } c < 500 \\ 0 \text{ otherwise} \end{cases} \]
The textual definition (wires part1)

<table>
<thead>
<tr>
<th>WIRES</th>
<th>MA</th>
<th>BE</th>
<th>d_4</th>
<th>RE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MortgageAgent</td>
<td>c_4</td>
<td></td>
<td>d_4</td>
<td>Registry</td>
</tr>
<tr>
<td>ask getLenders</td>
<td>s_1</td>
<td>Straight.</td>
<td>R_1</td>
<td>rpl getLenders</td>
</tr>
<tr>
<td>tll regContract</td>
<td>s_1</td>
<td>Straight.</td>
<td>R_1</td>
<td>prf registerContract</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MA</th>
<th>c_1</th>
<th>CB</th>
<th>d_1</th>
<th>BA</th>
</tr>
</thead>
<tbody>
<tr>
<td>MortgageAgent</td>
<td></td>
<td></td>
<td>d_1</td>
<td>Bank</td>
</tr>
<tr>
<td>s&amp;r openAccount</td>
<td>s_1</td>
<td>Straight.</td>
<td>R_1</td>
<td>r&amp;s newMortgageAccount</td>
</tr>
<tr>
<td>@ idData</td>
<td>i_1</td>
<td>I(usrdata, loandata)</td>
<td>i_1</td>
<td>@ idData</td>
</tr>
<tr>
<td>loanData</td>
<td>i_2</td>
<td></td>
<td>i_2</td>
<td>loanData</td>
</tr>
<tr>
<td>☑ accountData</td>
<td>o_1</td>
<td>O(accountdata)</td>
<td>o_1</td>
<td>☑ accountData</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MA</th>
<th>c_1</th>
<th>CI</th>
<th>d_1</th>
<th>IN</th>
</tr>
</thead>
<tbody>
<tr>
<td>MortgageAgent</td>
<td></td>
<td></td>
<td>d_1</td>
<td>Insurance</td>
</tr>
<tr>
<td>s&amp;r getInsurance</td>
<td>s_1</td>
<td>Straight.</td>
<td>R_1</td>
<td>r&amp;s newMortgageInsurance</td>
</tr>
<tr>
<td>@ idData</td>
<td>i_1</td>
<td>I(usrdata, loandata)</td>
<td>i_1</td>
<td>@ idData</td>
</tr>
<tr>
<td>loanData</td>
<td>i_2</td>
<td></td>
<td>i_2</td>
<td>loanData</td>
</tr>
<tr>
<td>☑ insuranceData</td>
<td>o_1</td>
<td>O(insuranceData)</td>
<td>o_1</td>
<td>☑ insuranceData</td>
</tr>
</tbody>
</table>
The textual definition (wires part2)

<table>
<thead>
<tr>
<th>CI</th>
<th>CC</th>
<th>d1</th>
<th>MA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_i$</td>
<td>Straight,</td>
<td>$R_i$</td>
<td>MortgageAgent</td>
</tr>
<tr>
<td></td>
<td>I(usrdata,</td>
<td></td>
<td>getProposal</td>
</tr>
<tr>
<td></td>
<td>moneyvalue, prefdata)</td>
<td></td>
<td>idData</td>
</tr>
<tr>
<td></td>
<td>O(mortgageproposal,</td>
<td></td>
<td>income</td>
</tr>
<tr>
<td></td>
<td>moneyvalue)</td>
<td></td>
<td>preferences</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>proposal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cost</td>
</tr>
<tr>
<td>$O_i$</td>
<td></td>
<td>$S_i$</td>
<td>send confirmation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>contract</td>
</tr>
</tbody>
</table>

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

Define requirements
Define modules + EX-Is
Define internal structure
Define behaviours
Analyse
Deploy

PEPA
Logics of Interactions
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.

---

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
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\(\bullet\) and getProposal\(\bigcirc\) 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).
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)
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 $\text{compositionRate}(\text{LE})$

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 $\text{askProposal}$-event and $\text{askProposal}$-event)

(5) is $\text{compositionRate}(\text{LE}) + \text{responseRate}(\text{LE})(\text{askProposal},\text{askProposal})$
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”.

![Cumulative distribution function graph](image_url)
Passage Time Analysis

Varying orProcessingRate

<table>
<thead>
<tr>
<th>orProcessingRate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

Probability of completion vs Time

- orProcessingRate=1
- orProcessingRate=2
- orProcessingRate=3
- orProcessingRate=4
- orProcessingRate=5
- orProcessingRate=6
- orProcessingRate=7

Diagram showing a flowchart with various nodes labeled as follows:

- CR: Customer
- OR: Orchestration
- LO: Lender
- BA: Bank
- IN: Insurance
- RE: Registry
- CD
- SLA: SLA
- LE: Lender
- BO: Borrower
- intLE: intLE
- intBA: intBA
- intIN: intIN
- CO: Customer
- CR
- RO
- LO
- BO
- IN
- int
- RE: Registry

Legend:

- OR: Orchestration
- CD
- SLA: SLA
- LE: Lender
- BO: Borrower
- IN: Insurance
- RE: Registry
- CO: Customer
Passage Time Analysis

Varying transferRate(CM)

<table>
<thead>
<tr>
<th>transferRate(CM)</th>
<th>Line Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>-</td>
</tr>
<tr>
<td>0.5</td>
<td>--</td>
</tr>
<tr>
<td>0.75</td>
<td>*</td>
</tr>
<tr>
<td>1.0</td>
<td>o</td>
</tr>
<tr>
<td>1.25</td>
<td>-r</td>
</tr>
<tr>
<td>1.5</td>
<td>-g</td>
</tr>
<tr>
<td>1.75</td>
<td>-b</td>
</tr>
</tbody>
</table>

Probability of completion vs. Time

0 1 2 3 4 5 6 7 8 9 10
A view of ensemble

- Define requirements
  - Define modules + EX-Is
  - Define internal structure
  - Define behaviours
  - Analyse
  - Deploy

- Use case diagrams
  - From scratch
  - Use case -> SRML

- From scratch
  - CBD

- From scratch
  - Reuse
  - Refinement

- PEPA
  - Logics of Interactions

- 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.
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

Define requirements

Define modules + EX-Is

Define internal structure

Define behaviours

Analyse

Deploy

use case diagrams

from scratch
use case -> SRML

from scratch
CBD

from scratch
reuse
refinement

PEPA
Logics of Interactions

SCA

BPEL
StPowla
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

<table>
<thead>
<tr>
<th>BPEL Tag/Construct</th>
<th>Tool</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoke, Receive, Reply, Assign (BA)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Wait, Empty, Exit (BA)</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Throw (BA)</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Sequence, Switch (SA)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Flow, While (SA)</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Control Links, Scopes, Correlation Sets</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>
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)

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

transition transition_A
triggeredBy pt.op? 
guardedBy ra ∧ ¬exit
effects ¬ra' ∧ fa'
∧ v.pl'=pt.op<v>.v.pl ∧ ... 
∧ v.pn'=pt.op<v>.v.pn
```

- The encoding of a structured activity (sequence)

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

- The transition is executed the first time when X is triggered

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

- The transition is executed the second time when A terminates

  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

from scratch
reuse
refinement

PEPA
Logics of Interactions

StPowla
BPEL

Define requirements

Define modules + EX-Is

Define internal structure

Define behaviours

Analyse

Deploy
StPowla2SRML

StPowla


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...

Activities

sequence

split join

condition join

random preference

P1 → P2

error output
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)
```

Diagram:

```
receive order -> request deposit
```

```
request deposit
```

```
process order
```
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 if z is false

- **delete(x)** -
  deletes task/scope x from the current workflow instance
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
The business role of BP has one or more transition that model the reaction to each reconfiguration function.

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

\[
\text{transition policyHandler_delete_1} \\
\text{triggeredBy delete[i] \& \&} \\
\text{guardedBy state[delete[i].task]=toStart} \\
\text{effects policy[delete[i].task]'}
\]

\[
\text{transition policyHandler_delete_2} \\
\text{triggeredBy start[x]} \\
\text{guardedBy P_delete[i].? \& P_delete[i].task=x} \\
\text{effects \neg start[x]'} \& \neg done[x]’ \& \neg state[x]'=done \\
\text{sends delete[i]!!}
\]

The encoding of the workflow constructs is similar to the one of the encoding from BPEL.
A view of ensemble

use case diagrams

from scratch
use case \rightarrow 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
A view of ensemble

UML → StPowla → BPEL → WSDL → SRML → ? → analysis → PEPA
   COWS
   SCA
   Java
   BPEL

deployment