Extension of Behaviours with Formal Data Types: Integration and Coordination

Pascal Poizat

Laboratoire de Méthodes Informatiques (LaMI)
UMR 8042 CNRS - University of Évry, GENOPOLE

Invited Lecture, Universities of Málaga and Extremadura

1. Introduction
2. Integration
3. Coordination
4. Conclusions

1. Introduction
2. Integration
3. Coordination
4. Conclusions

1. Introduction
2. Integration
3. Coordination
4. Conclusions

1. Introduction
2. Integration
3. Coordination
4. Conclusions

The problem : complex systems

- expressive structuring needed
  (modules $\leadsto$ objects $\leadsto$ components $\leadsto$ aspects)
- encapsulated datatypes
- behaviours, communication, value-passing
- verification

The problem : complex systems

- expressive structuring needed
  (modules $\rightsquigarrow$ objects $\rightsquigarrow$ components $\rightsquigarrow$ aspects)
- encapsulated datatypes
- behaviours, communication, value-passing
- verification

(Possible) pieces of a solution

- trusted components, ADL: interface, ports, ... concepts

The problem : complex systems

- expressive structuring needed
  (modules $\leadsto$ objects $\leadsto$ components $\leadsto$ aspects)
- encapsulated datatypes
- behaviours, communication, value-passing
- verification

(Possible) pieces of a solution

- trusted components, ADL: interface, ports, ... concepts
- mixed specifications: behaviours + datatypes

The problem : complex systems

- expressive structuring needed
  (modules $\rightsquigarrow$ objects $\rightsquigarrow$ components $\rightsquigarrow$ aspects)
- encapsulated datatypes
- behaviours, communication, value-passing
- verification

(Possible) pieces of a solution

- trusted components, ADL: interface, ports, ... concepts
- mixed specifications: behaviours + datatypes
- formality

The problem : complex systems

- expressive structuring needed
  (modules $\rightsquigarrow$ objects $\rightsquigarrow$ components $\rightsquigarrow$ aspects)
- encapsulated datatypes
- behaviours, communication, value-passing
- verification

Our framework

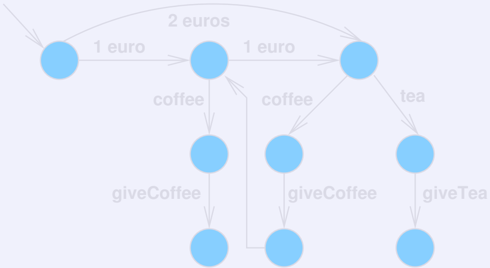
- formal components with BIDL
- expressive gluing mechanisms
- mixity: behaviours + abstract datatypes = STS
- analysis techniques for STS

LTS in everyday life

Labelled Transition Systems

Usual models for behaviours are LTS $< S, s_0, A, T >$
with $s_0 \in S$ and $T \subseteq S \times A \times S$, often, $A = A^{\mathrm{in}} \uplus A^{\mathrm{out}}$ (IOLTS)

Example (Coffee Machine)

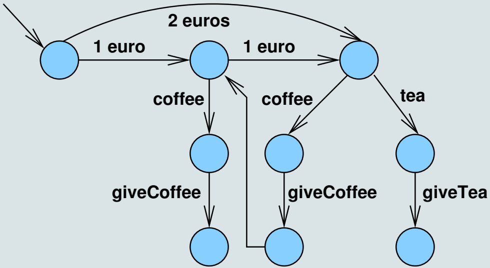


LTS in everyday life

Labelled Transition Systems

Usual models for behaviours are LTS $< S, s_0, A, T >$
with $s_0 \in S$ and $T \subseteq S \times A \times S$, often, $A = A^{\mathrm{in}} \uplus A^{\mathrm{out}}$ (IOLTS)

Example (Coffee Machine)



State explosion

In presence of data ...

The computation of an LTS from a specification may explode !

Example (Buffer)

State explosion

In presence of data ...

The computation of an LTS from a specification may explode !

Example (Buffer)

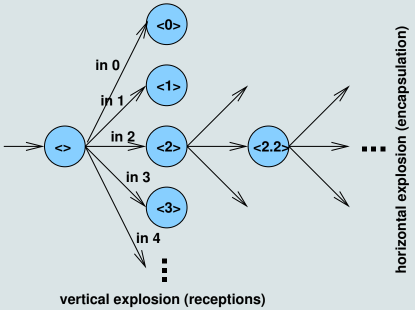
```
Buffer<>     = in  ?a:Nat .  Buffer<a>
Buffer<b.X> = in  ?a:Nat .  Buffer<b.X.a>
             + out !b     .  Buffer<X>
```

State explosion

In presence of data ...

The computation of an LTS from a specification may explode !

Example (Buffer)



The STS Solution

Symbolic Transition Systems

STS abstract the data on states and transitions

[HL-HandbookPA,STS4LOTOS], [CPR00]

e.g., $< D, (\Sigma, Ax), S, s_0, v_0, T >$ [MPR04]

with elements of T of the form:

The STS Solution

Symbolic Transition Systems

STS abstract the data on states and transitions

[HL-HandbookPA,STS4LOTOS], [CPR00]

e.g., $< D, (\Sigma, Ax), S, s_0, v_0, T >$ [MPR04]

with elements of T of the form:

$$s \xrightarrow{[\text{guard}(\text{self}, x_1, \ldots, x_n)]\ \text{event}?x_1 \ldots ?x_n\ !t_1 \ldots !t_m\ /\text{action}(\text{self}, x_1, \ldots, x_n)} s'$$

The STS Solution

Symbolic Transition Systems

STS abstract the data on states and transitions

[HL-HandbookPA,STS4LOTOS], [CPR00]

e.g., $< D, (\Sigma, Ax), S, s_0, v_0, T >$ [MPR04]

with elements of T of the form:

$$s \xrightarrow{[\text{guard}(\text{self},x_1,\ldots,x_n)]\ \text{event}?x_1\ldots?x_n\ !t_1\ldots!t_m\ /\text{action}(\text{self},x_1,\ldots,x_n)} s'$$

The STS Solution

Symbolic Transition Systems

STS abstract the data on states and transitions

[HL-HandbookPA,STS4LOTOS], [CPR00]

e.g., $< D, (\Sigma, Ax), S, s_0, v_0, T >$ [MPR04]

with elements of T of the form:

$$s \xrightarrow{[\text{guard}(\text{self},x_1,\ldots,x_n)]\ \text{event}?x_1\ldots?x_n\ !t_1\ldots!t_m\ /\text{action}(\text{self},x_1,\ldots,x_n)} s'$$

The STS Solution

Symbolic Transition Systems

STS abstract the data on states and transitions

[HL-HandbookPA,STS4LOTOS], [CPR00]

e.g., $< D, (\Sigma, Ax), S, s_0, v_0, T >$ [MPR04]

with elements of T of the form:

$$s \xrightarrow{[\text{guard}(\text{self}, x_1, \ldots, x_n)]\ \text{event}?x_1 \ldots ?x_n\ !t_1 \ldots !t_m\ /\text{action}(\text{self}, x_1, \ldots, x_n)} s'$$

The STS Solution

Symbolic Transition Systems

STS abstract the data on states and transitions

[HL-HandbookPA,STS4LOTOS], [CPR00]

e.g., $< D, (\Sigma, Ax), S, s_0, v_0, T >$ [MPR04]

with elements of T of the form:

$$s \xrightarrow{[\mathrm{guard}(x_1,\ldots,x_n)]\ \mathrm{event}?x_1\ldots?x_n\ !t_1\ldots!t_m\ /\mathrm{action}(x_1,\ldots,x_n)} s'$$

The STS Solution

Symbolic Transition Systems

STS abstract the data on states and transitions

[HL-HandbookPA,STS4LOTOS], [CPR00]

e.g., $< D, (\Sigma, Ax), S, s_0, v_0, T >$ [MPR04]

with elements of T of the form:

$$s \xrightarrow{[\text{guard}(x_1,\ldots,x_n)]\ \text{event}?x_1\ldots?x_n\ !t_1\ldots!t_m} s'$$

Question

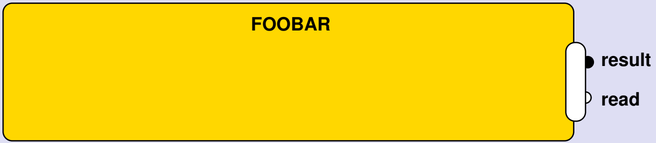
Are BIDL needed anyway ?

Yes

Question

Are BIDL needed anyway ?

Yes
what does this component ?

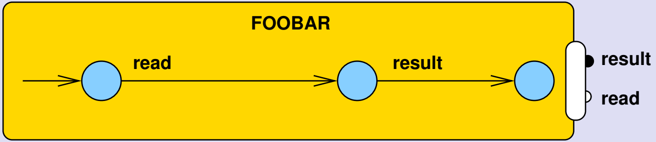


Question

Are BIDL needed anyway ?

Yes

it reads (something) and then outputs a result



Question

Are data needed anyway ?

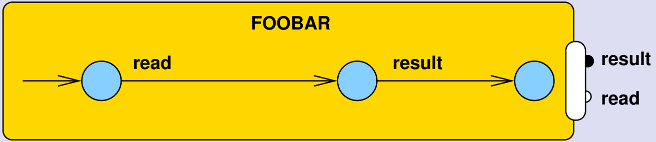
Yes

Question

Are data needed anyway ?

Yes

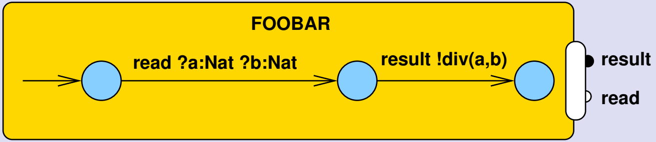
what does this component ?



Question

Are data needed anyway ?

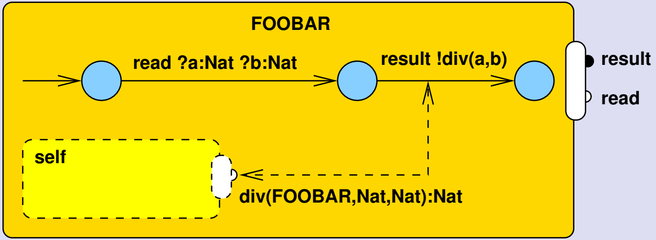
Yes
it reads (at a time) two integers and then outputs the result of the `div` operation applied to the integers



Question

Are data needed anyway ?

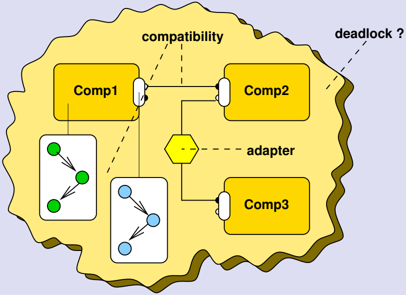
Yes
hidden underlying static type (with usual static signatures)



What does formality bring in ?

Some information in [PRS04]:

- abstract, expressive descriptions for BIDL
- animation
- equivalence checking, deadlock freedom, adaptors



Lots of mixed specification languages

kind	dynamic	static	examples
Heterogeneous	P. Alg.	model	ObjectZ-CSP, CSP-OZ, ZCCS, ZCSP, TCOZ
	P. Alg.	alg.	LOTOS, PSF
	T/S	model	$\mu\mathcal{SZ}$, MaC, Event Calculus
	T/S	alg.	Korrigan, SDL, CASLChart, TAG
	T/S	– spec. –	Estelle, UML, Argos, BDL
	Petri	alg.	OBJSA, Clown, CO-OPN/2
	Petri	– spec. –	CO, OPN
Homogeneous	Algebraic		LTL, Rewriting Logic, ASM
	Logical		TLA, Unity, TRIO, OSL
	Proc. Alg.		CCS+value, CSP, π-calcul

Formal + Semi-Formal

semi-formal

+ graph. notations, readability, expressiveness, structuring
    - UML (formal ?)
- tools, consistency ?
    - ArgoUML, SMW, UMLAut, ...

formal

+ abstraction
    - what not how
+ semantics
    - tools, verification
- not easy to learn and use

Formal + Semi-Formal

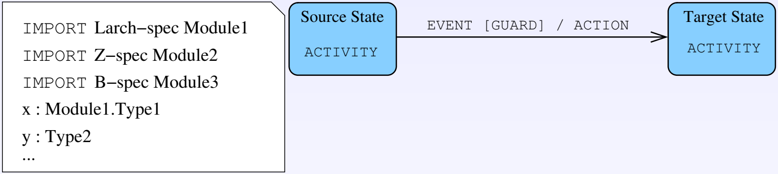
semi-formal

+ graph. notations, readability, expressiveness, structuring
  - UML (formal ?)
- tools, consistency ?
  - ArgoUML, SMW, UMLAut, ...

formal

+ abstraction
  - what not how
+ semantics
  - tools, verification
- not easy to learn and use

Syntactic Extensions



transition part	interaction kind	example
EVENT	reception	$evt\text{-}name(x_1{:}T_1,\ldots,x_n{:}T_n)$
GUARD	guard	*predicate*
ACTION	emission	$receiver\,\hat{}\,evt\text{-}name(t_1,\ldots,t_n)$
ACTION	assignment	$x{:=}t$

Typical Use: Case Study

The Gas Station

- furnishes different gas
- three pumps, three tanks
- credit card payment

Typical Use: Analysis

Static part

- booleans (Z)
- integers, real numbers (Larch)
- gases, pumps, tanks (Z)

Dynamic part

- card manager
- pump manager 3 Extended State Diagrams
- tank manager

Typical Use: Analysis

Static part

- booleans (Z)
- integers, real numbers (Larch)
- gases, pumps, tanks (Z)

Dynamic part

- card manager
- pump manager 3 Extended State Diagrams
- tank manager

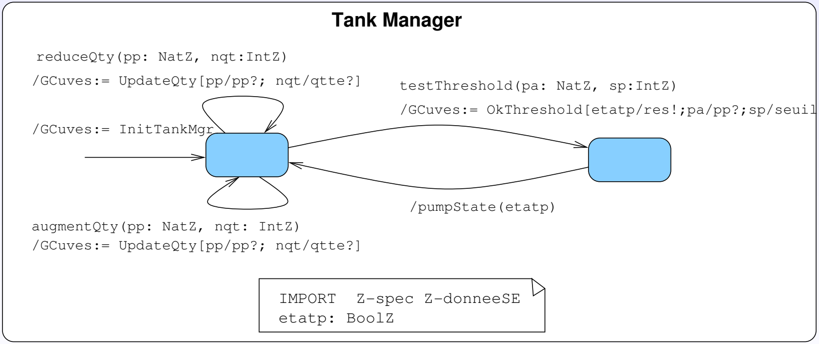
Typical Use: Analysis

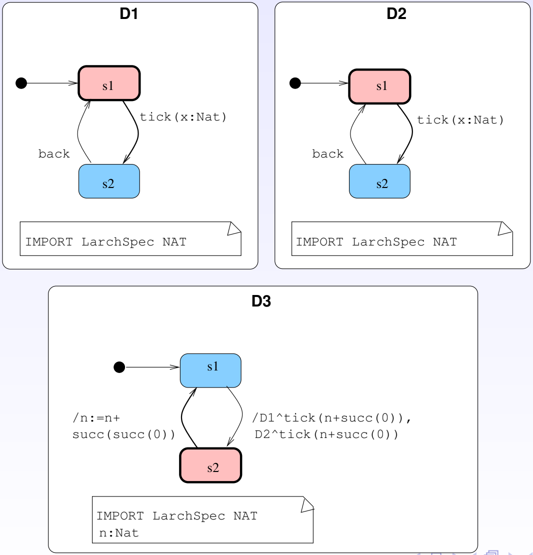
Static part

- booleans (Z)
- integers, real numbers (Larch)
- gases, pumps, tanks (Z)

Dynamic part

- card manager
- pump manager 3 Extended State Diagrams
- tank manager


Tank Manager
reduceQty(pp: NatZ, nqt:IntZ)
/GCuves:= UpdateQty[pp/pp?; nqt/qtte?]
testThreshold(pa: NatZ, sp:IntZ)
/GCuves:= OkThreshold[etatp/res!;pa/pp?;sp/seuil
/GCuves:= InitTankMgr
/pumpState(etatp)
augmentQty(pp: NatZ, nqt: IntZ)
/GCuves:= UpdateQty[pp/pp?; nqt/qtte?]
IMPORT Z-spec Z-donneeSE
etatp: BoolZ


D1
s1
tick(x:Nat)
back
s2
IMPORT LarchSpec NAT
D2
s1
tick(x:Nat)
back
s2
IMPORT LarchSpec NAT
D3
s1
/n:=n+
succ(succ(0))
/D1^tick(n+succ(0)),
D2^tick(n+succ(0))
s2
IMPORT LarchSpec NAT
n:Nat

Operational Semantics – $||.||_{sos}$

On operational semantics ...

- can be used for Transition Systems and Process Algebras
- well suited for animation and equivalence checking

  are the interfaces of C(lient) and S(erver) compatible ?
- refinement

  does the C implementation do what is required in its interface ?
- compositionality

  if I prove that C and C' are compatible, may I replace C with C' in any system ?
- adequacy wrt temporal logic

  if C and C' are equal, may I prove properties on the simplest one ?

Operational Semantics – $||.||_{SOS}$

On operational semantics ...

- can be used for Transition Systems and Process Algebras
- *are the interfaces of C(lient) and S(erver) compatible ?*
  $\mathrm{Interf}(C) = \mathrm{Interf}(S)$, with $= \in \{=_T, \sim, \approx, ...\}$
- *does the C implementation do what is required in its interface ?*
  $\mathrm{Interf}(C) \subseteq \mathrm{Interf}(\mathrm{Impl}(C))$, with $\subseteq \in \{\sqsubseteq_T, \sqsubseteq_F, ...\}$
- *if I prove that C and C' are compatible, may I replace C with C' in any system ?*
  $C \sim C' \Rightarrow (\forall \mathcal{S}[.].\mathcal{S}[C] \sim \mathcal{S}[C'])$
- *if C and C' are equal, may I prove properties on the simplest one ?*
  $C \sim C' \Leftrightarrow (\forall \phi \in \Phi_{HML}.C \models \phi \Leftrightarrow C' \models \phi)$

The [APS03] Semantics

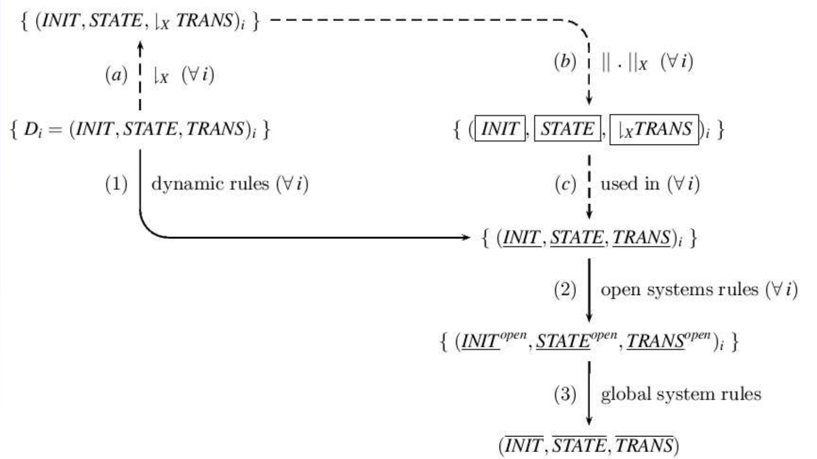
- based on experience with several mixed languages (Korrigan, CCS+ADT, TAG, MaC, ...)
- representative for the definition of a generic approach to integrate static formal specifications (SFS) into dynamic formal specification (DFS)
  - builds on a first proposal for UML state diagrams + SFS + synchronous communication
  - generalizing and asynchronous communication

Principle

Formal rules in 4 layers

- meta-typing
- static evolution
- dynamic evolution and open-systems
- composition

Principle


{ (INIT, STATE, ⌊X TRANS)i }
(a) ⌊X (∀ i)
(b) || . ||X (∀ i)
{ Di = (INIT, STATE, TRANS)i }
{ (INIT, STATE, ⌊XTRANS)i }
(1) dynamic rules (∀ i)
(c) used in (∀ i)
{ (INIT, STATE, TRANS)i }
(2) open systems rules (∀ i)
{ (INIT^open, STATE^open, TRANS^open)i }
(3) global system rules
(INIT, STATE, TRANS)

Remarks

- lots of dynamic semantics
  - use of generic elements, *e.g.*, *event* $\boxed{\in}$ $\mathcal{Q}_{in}$

Constraints

- $||D||_{SOS}$ = LTS ($\boxed{INIT}$, $\boxed{STATE}$, $\boxed{TRANS}$) $\Rightarrow$OK !

Notation

- $\mathcal{D}$, $D = (INIT, STATE, TRANS, DeclImp, DeclVar) \in \mathcal{D}$
- $EVENT = EVENT^{?} \cup EVENT^{!}$, $DeclVar = DeclVar^{?} \cup DeclVar^{!}$
- $\mathcal{S} \subseteq \boxed{STATE} \times \mathcal{E} \times \boxed{\mathcal{Q}} [EVENT^{?}] \times \boxed{\mathcal{Q}} [EVENT^{!}]$

Remarks

- lots of dynamic semantics
  - use of generic elements, *e.g.*, *event* $\boxed{\in}$ $\mathcal{Q}_{in}$

Constraints

- $||D||_{SOS}$ = LTS ($\boxed{INIT}$, $\boxed{STATE}$, $\boxed{TRANS}$) $\Rightarrow$OK !

Notation

- $\mathcal{D}$, $D = (INIT, STATE, TRANS, DeclImp, DeclVar) \in \mathcal{D}$
- $EVENT = EVENT^{?} \cup EVENT^{!}$, $DeclVar = DeclVar^{?} \cup DeclVar^{!}$
- $\mathcal{S} \subseteq \boxed{STATE} \times \mathcal{E} \times \boxed{\mathcal{Q}}[EVENT^{?}] \times \boxed{\mathcal{Q}}[EVENT^{!}]$

Remarks

- lots of dynamic semantics
  - use of generic elements, *e.g.*, *event* $\boxed{\in}$ $\mathcal{Q}_{in}$

Constraints

- $||D||_{SOS}$ = LTS ($\boxed{INIT}$, $\boxed{STATE}$, $\boxed{TRANS}$) $\Rightarrow$OK !

Notation

- $\mathcal{D}$, $D = (INIT, STATE, TRANS, DeclImp, DeclVar) \in \mathcal{D}$
- $EVENT = EVENT^? \cup EVENT^!$, $DeclVar = DeclVar^? \cup DeclVar^!$
- $\mathcal{S} \subseteq \boxed{STATE} \times \mathcal{E} \times \boxed{\mathcal{Q}} [EVENT^?] \times \boxed{\mathcal{Q}} [EVENT^!]$

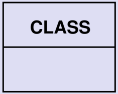
Static Evolution

$$\frac{\forall i \in 1..n \;.\; \exists X_i \;.\; t_i ::_D X_i \quad \exists v_i \;.\; E \vdash t_i \rhd_{X_i} v_i}{act-eval(rec\hat{\ }e(t_1,\ldots,t_n), <\Gamma, E, Q_{in}, Q_{out}>, D) = <\Gamma, E, Q_{in}, Q_{out} \boxed{\uplus} \{rec\hat{\ }e(v_1,\ldots,v_n)\}>}$$

$$\frac{\exists X \;.\; t ::_D X \quad \exists v \;.\; E \vdash t \rhd_X v}{act-eval(x := t, <\Gamma, E, Q_{in}, Q_{out}>, D) = <\Gamma, E\{x \mapsto v\}, Q_{in}, Q_{out}>}$$

Static Evolution

Term evaluation, $\triangleright_X$

- $\triangleright_{Alg}$: rewriting (+ tools : Larch Prover, ELAN)
- $\triangleright_Z$, $\triangleright_B$: LTS construction (+ tools : Z-Eves)
- $\triangleright$ 

: classes formelles, Z

Dynamic Evolution

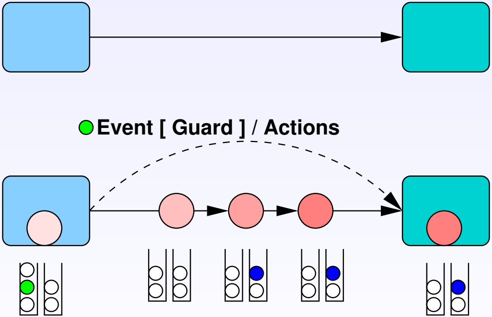
Notation

$EVENT^{?+} = EVENT^{?} \cup \{\varepsilon\}$

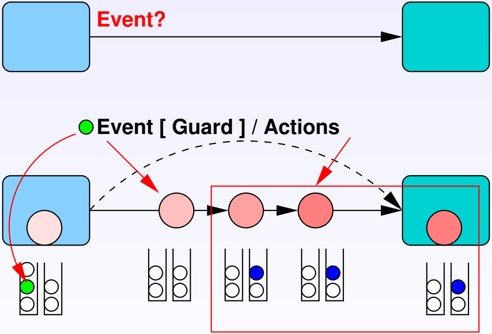
$||D||_{SOS} = LTS(\underline{INIT}, \underline{STATE}, \underline{TRANS})$ with:

- $\underline{STATE} \subseteq \mathcal{S}$
- $\underline{INIT} \subseteq \underline{STATE}$
- $\underline{TRANS} \subseteq \underline{STATE} \times EVENT^{?+} \times \underline{STATE}$

Dynamic Evolution



Dynamic Evolution



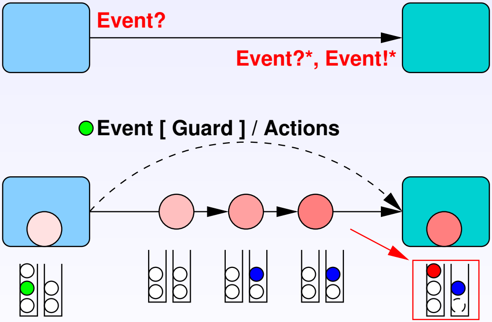
Open Systems

Notation

$||D||^{open}_{SOS} = LTS(\underline{INIT}^{open}, \underline{STATE}^{open}, \underline{TRANS}^{open})$ with:

- $\underline{INIT}^{open} \subseteq \underline{INIT}$
- $\underline{TRANS}^{open} \subseteq \underline{TRANS} \times \boxed{\mathcal{Q}}[EVENT^{?}] \times \boxed{\mathcal{Q}}[EVENT^{!}]$
- $\underline{STATE}^{open} \subseteq SOURCE(\underline{TRANS}^{open}) \cup TARGET(\underline{TRANS}^{open})$

Open Systems



Compositions

Notation

$|| \cup_{i \in 1..n} D_i ||^{open}_{oper} =$
$LTS(\overline{INIT}^{open}(\cup_{i \in 1..n} D_i), \overline{STATE}^{open}(\cup_{i \in 1..n} D_i), \overline{TRANS}^{open}(\cup_{i \in 1..n} D_i))$
with:

- $\overline{INIT}(\cup_{i \in 1..n} D_i) \subseteq \Pi_i \underline{INIT}^{open}(D_i)$
- $\overline{TRANS}(\cup_{i \in 1..n} D_i) \subseteq \{t \in \Pi_i \underline{TRANS}^{open}(D_i) | CC(t)\}$
- $\overline{STATE}(\cup_{i \in 1..n} D_i) \subseteq$ $\overline{INIT}(\cup_{i \in 1..n} D_i) \cup TARGET(\overline{TRANS}(\cup_{i \in 1..n} D_i))$

Compositions

Idea

whenever

something addressed to D_j

is taken out of a given D_k output queue

then

it is put, at the same time, within the D_j input queue

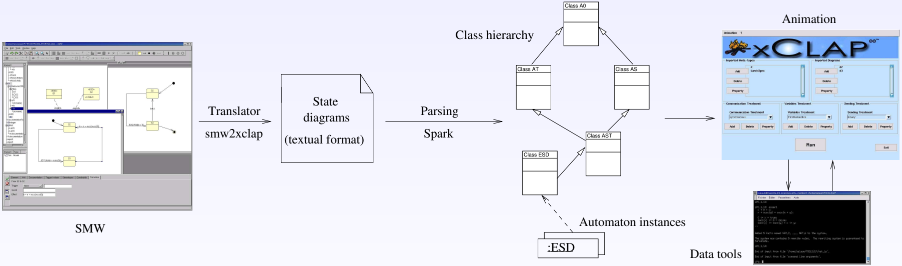
Compositions

Formally ...

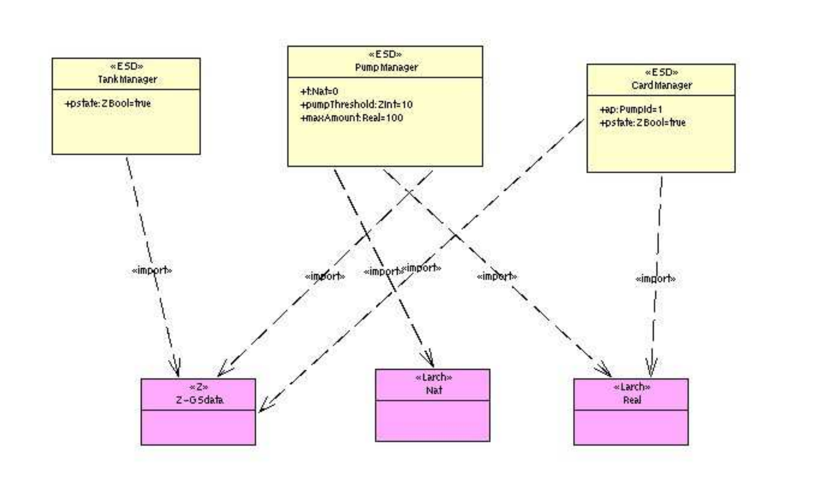
$$CC(S_1 \xrightarrow{l_1}_{E_{in_1},E_{out_1}} S'_1, \ldots, S_n \xrightarrow{l_n}_{E_{in_n},E_{out_n}} S'_n) \Leftrightarrow$$

$$\forall k \in 1 \ldots n \;.\; \forall D_j\hat{\ }e \boxed{\in} E_{out_k} \;.\; D_j \in \cup_{i \in 1..n} D_i \Longrightarrow e \boxed{\in} E_{in_j}$$

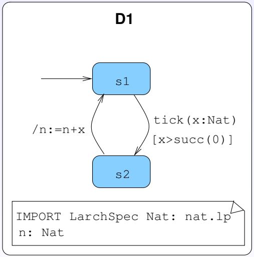
xCLAP - Architecture



xCLAP - Designing



xCLAP - Translation



(a) state diagram (graphical format)

Translator

→

smw2xclap

declare D1

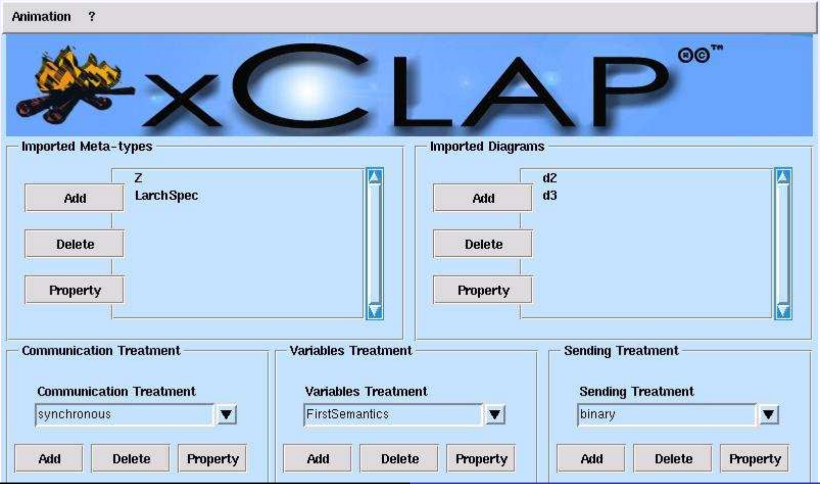
import LarchSpec Nat: nat.lp
n: Nat

esdstate
EsdState s1 {'initial':1, 'final':0}.
EsdState s2 {'initial':0, 'final':0}.
−−
END_STATES

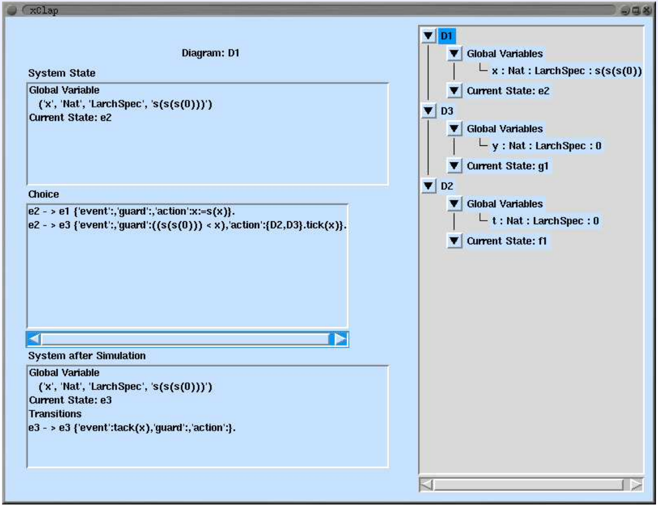
esdtrans
EsdTrans s1 − > s2 {'event':"tick(x:nat)", 'guard':"x>succ(0)", 'action':""}.
EsdTrans s2 − > s1 {'event':"", 'guard':"", 'action':"n:=n+x"}.
−−
END_TRANS

(b) state diagram (textual format)

xCLAP - Configuration



xCLAP - Animation



What do we model ?

Distributed Entities

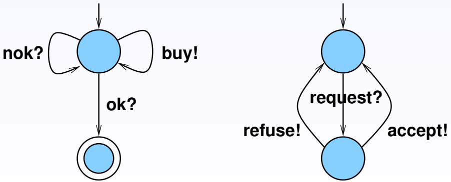
- viewed through interfaces (black-box foundation)
- interfaces have to take into account behavioural information (BIDL)
- goal: quick survey and comparison of formal material to describe coordination/interaction among entities
- remember ?
  formal means enable one to use existing verification tools to ensure correctness of interactions
- applications: web services, genetic regulatory networks

How ?

A Simple Formal Model: LTS

- here: simple yet general formal model of entities: a nondeterministic LTS $< L, S, I, F, T >$
- labels may be emissions `e!` or receptions `r?`
- data information is discarded for simplicity
- running example: one store and several suppliers

Store **Supplier**

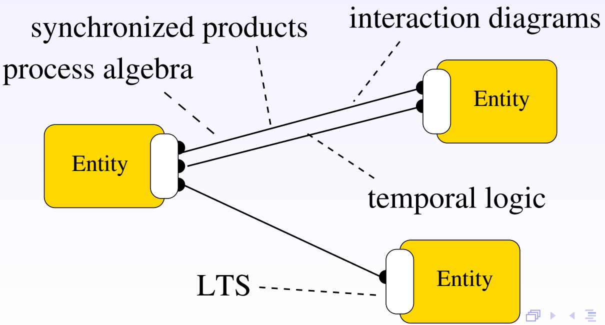


[YellinEtAl-TOPLAS'97]
[deAlfaroHenzinger-ESEC'01]

[ArbabEtAl-FOCLASA'02]

Communication Model

- depends on the means used to compose entities
- implicit means: semantic rules (first part)
- explicit means



Semantics

- basic idea: redefine the *CC* constraint of part I

Semantics

- basic idea: redefine the *CC* constraint of part I

$$CC(S_1 \xrightarrow{l_1}_{E_{in_1}, E_{out_1}} S'_1, \ldots, S_n \xrightarrow{l_n}_{E_{in_n}, E_{out_n}} S'_n) \Leftrightarrow$$

$$\forall k \in 1 \ldots n \,.\, \forall D_j\hat{\ }e \boxed{\in} E_{out_k} \,.\, D_j \in \cup_{i \in 1..n} D_i \Longrightarrow e \boxed{\in} E_{in_j}$$

Semantics

- basic idea: redefine the *CC* constraint of part I

$$CC(S_1 \xrightarrow{l_1}_{E_{in_1}, E_{out_1}} S'_1, \ldots, S_n \xrightarrow{l_n}_{E_{in_n}, E_{out_n}} S'_n, \text{Coord}) \Leftrightarrow$$

$$???$$

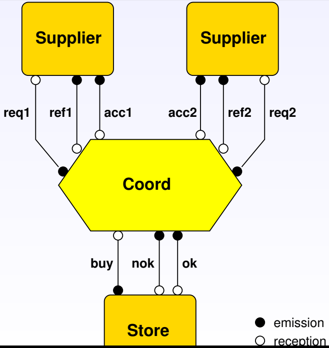
- see [SP04], [JUCS, 2005, submitted]
- here: examples

Process Algebra

- parallel composition operators are way to match inputs and outputs
- may be used as an explicit 1st class coordinator language to take into account more complex coordination protocols

Process Algebra

- parallel composition operators are way to match inputs and outputs
- may be used as an explicit 1st class coordinator language to take into account more complex coordination protocols



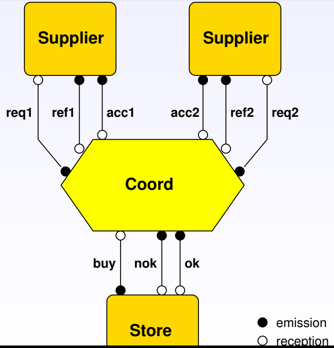
Example (with processes)

```
S = Supplier[req1/request, ref1/refuse, ..]
  | Supplier[req2/request, ref2/refuse, ..]
  | Store
  | Coord
Coord = buy.('req1.Wait1 + 'req2.Wait2)
Wait1 = acc1.'ok .0 + ref1.'nok.Coord
Wait2 = acc2.'ok.0 + ref2.'nok.Coord
```

[SalaünEtAl-IJBPIM]

Process Algebra

- parallel composition operators are way to match inputs and outputs
- may be used as an explicit 1st class coordinator language to take into account more complex coordination protocols



Example (with processes)

```
S = Supplier[req1/request, ref1/refuse, ..]
  | Supplier[req2/request, ref2/refuse, ..]
  | Store
  | Coord
Coord = buy.('req1.Wait1 + 'req2.Wait2)
Wait1 = acc1.'ok .0 + ref1.'nok.Coord
Wait2 = acc2.'ok.0 + ref2.'nok.Coord
```

[SalaünEtAl-IJBPIM]

Synchronized Products

- simple and readable means to define interactions among entities [Arnold94,ArnoldEtAl-FI04]
- extended synchronization vectors [SP04]

Synchronized Products

- simple and readable means to define interactions among entities [Arnold94,ArnoldEtAl-FI04]
- extended synchronization vectors [SP04]

synchronous, one to many: $<a!,\varepsilon,b?,\varepsilon,c?>$

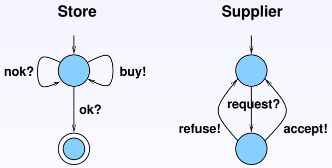
synchronous, matching: $<a!,\varepsilon,b!,\varepsilon,c!>$

synchronous, generation: $<a?,\varepsilon,b?,\varepsilon,c?>$

asynchronous, one to many: $[a!,\varepsilon,b?,\varepsilon,c?]$

Synchronized Products

- simple and readable means to define interactions among entities [Arnold94,ArnoldEtAl-FI04]
- extended synchronization vectors [SP04]



Example (with vectors)

$<$ buy!, request?, request? $>$
$<$ nok?, ε, refuse! $>$
$<$ nok?, refuse!, ε $>$
$<$ ok?, ε, accept! $>$
$<$ ok?, accept!, ε $>$

Interaction Diagrams

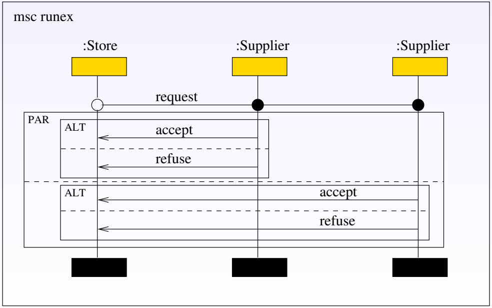
- coordination may be described using interaction diagrams: MSC, or UML sequence and collaboration diagrams
- many formalisations proposed so far

  [ITU-MSC'96,MauwReniers-MSC'96,KrügerEtAl-SFEDL'02]

Interaction Diagrams

- coordination may be described using interaction diagrams: MSC, or UML sequence and collaboration diagrams
- many formalisations proposed so far

[ITU-MSC'96,MauwReniers-MSC'96,KrügerEtAl-SFEDL'02]



Temporal Logic

- numerous: LTL,CTL/CTL*,ACTL,TLA, μ-calculus,...
- expressive means to coordinate entities, *e.g.* in formal ADLs [JUCS, 2005, submitted]
  - first, being able to describe the properties of objects that are to be glued (states and transitions)
  - indexed formulas, then lift the properties of the subcomponents of a composition up to the composition
  - the logic also takes into account coordination using logical conjunction

Example (with logic)

$$\begin{aligned} & \texttt{Store.buy!} \Leftrightarrow \textbf{ALL}(\{i : [1..N]\texttt{Supplier}_i\}).\texttt{request?} \\ \vee\ & \texttt{Store.ok?} \Leftrightarrow \textbf{ONE}(\{i : [1..N]\texttt{Supplier}_i\}).\texttt{accept!} \\ \vee\ & \texttt{Store.nok?} \Leftrightarrow \textbf{ONE}(\{i : [1..N]\texttt{Supplier}_i\}).\texttt{refuse!} \end{aligned}$$

Temporal Logic

- numerous: LTL,CTL/CTL*,ACTL,TLA, μ-calculus,...
- expressive means to coordinate entities, *e.g.* in formal ADLs [JUCS, 2005, submitted]
  - first, being able to describe the properties of objects that are to be glued (states and transitions)
  - indexed formulas, then lift the properties of the subcomponents of a composition up to the composition
  - the logic also takes into account coordination using logical conjunction

Example (with logic)

$$\begin{array}{ll} & \texttt{Store.buy!} \Leftrightarrow \mathbf{ALL}(\{i : [1..N]\texttt{Supplier}_i\}).\texttt{request?} \\ \vee & \texttt{Store.ok?} \Leftrightarrow \mathbf{ONE}(\{i : [1..N]\texttt{Supplier}_i\}).\texttt{accept!} \\ \vee & \texttt{Store.nok?} \Leftrightarrow \mathbf{ONE}(\{i : [1..N]\texttt{Supplier}_i\}).\texttt{refuse!} \end{array}$$

A First Comparison

		Process Algebras	Vectors	Interaction Diagrams	Logics
Communication Expressiveness	1 to 1 1 to N 1 to M in N	+ yes yes extension	++ yes yes yes	++ yes yes yes	++ yes yes yes
	Name matching Data Order	+ no yes yes	+ yes extension no	− no no yes	+ yes yes no
User Friendliness	Tools Executability	++ animation equivalence checking model–checking yes	+ animation equivalence checking model–checking no	+ animation model–checking yes	− embeddings no
	Graphical notations	− no	− no	++ yes	−− no

Conclusions

Overview

- semantics for STS: operational (here), denotational
- partially tool-equipped: animating (xCLAP), PVS embedding
- semantics for different coordination means

Perspectives

- framework for STS (Eclipse)
- implement coordination means
- better verification means
- relations wrt code / code generation

Conclusions

Overview

- semantics for STS: operational (here), denotational
- partially tool-equipped: animating (xCLAP), PVS embedding
- semantics for different coordination means

Perspectives

- framework for STS (Eclipse)
- implement coordination means
- better verification means
- relations wrt code / code generation

Any questions ?

Pascal.Poizat@lami.univ-evry.fr
http://www.lami.univ-evry.fr/˜poizat

Christian Attiogbé, Pascal Poizat, and Gwen Salaün.
Integration of Formal Datatypes within State Diagrams.
In *Fundamental Approaches to Software Engineering (FASE'2003)*, volume 2621 of *Lecture Notes in Computer Science*, pages 344–355. Springer-Verlag, 2003.

Christine Choppy, Pascal Poizat, and Jean-Claude Royer.
A Global Semantics for Views.
In *International Conference on Algebraic Methodology And Software Technology (AMAST'2000)*, volume 1816 of *Lecture Notes in Computer Science*, pages 165–180. Springer-Verlag, 2000.

Olivier Maréchal, Pascal Poizat, and Jean-Claude Royer.
Checking Asynchronously Communicating Components using Symbolic Transition Ssystems.

In *Distributed Objects and Applications (DOA'2004)*, volume 3291 of *Lecture Notes in Computer Science*, pages 1502–1519. Springer-Verlag, 2004.

Pascal Poizat, Jean-Claude Royer, and Gwen Salaün.
Formal methods for component description, coordination and adaptation (organizer position paper).
In Carlos Canal, Juan Manuel Murillo, and Pascal Poizat, editors, *Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'04)*, pages 89–100, 2004.
Held in conjunction with the 18th European Conference on Object-Oriented Programming (ECOOP). Published as a Technical Report of the Universities of Málaga (Spain), Extremadura (Spain) and Évry (France). ISBN 84-688-6782-9. Available at http://wcat04.unex.es/.

Gwen Salaün and Pascal Poizat.
Interacting Extended State Diagrams.
In *Semantic Foundations of Engineering Design Languages (SFEDL'2004)*, volume 115 of *Electronic Notes in Theoretical Computer Science*, pages 49–57, 2004.