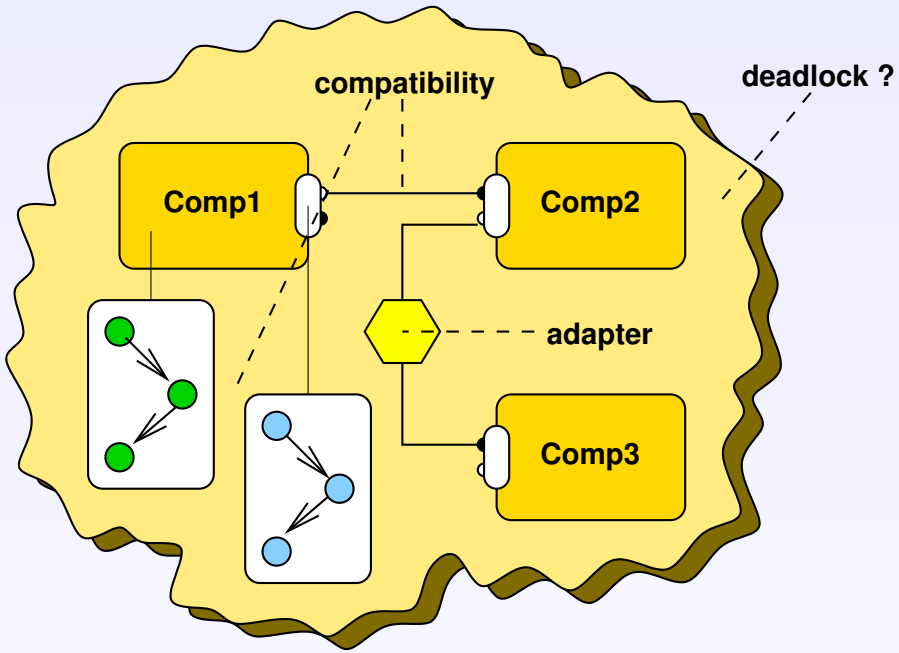
Adaptation Logicielle

Pascal Poizat

Laboratoire Informatique, Biologie Intégrative et Systèmes Complexes
IBISC FRE 2873
CNRS, Université d'Évry, GENOPOLE

collaboration avec C. Canal (Univ. Málaga) & G. Salaün (VASY, INRIA Rhône- Alpes)

Contexte



- complexité des systèmes : composants, architectures
- sécurité : méthodes formelles (vérification, modèles dynamiques)
- interactions : coordination, adaptation

Composant

A software component is a unit of composition with contextually specified interfaces and explicit context dependencies only. A software component can be deployed independentely and is subject to third-party composition
(Szyperski, 1998)

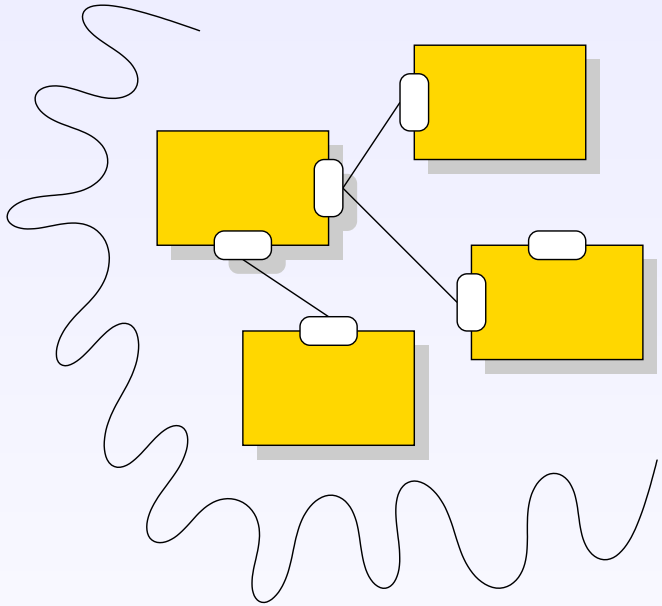
Caractéristiques des composants

- unité dédiée à la composition
- intégrée dans un contexte
- implémentant un ensemble d'interfaces ou rôles
- réutilisable à partir de sa spécification

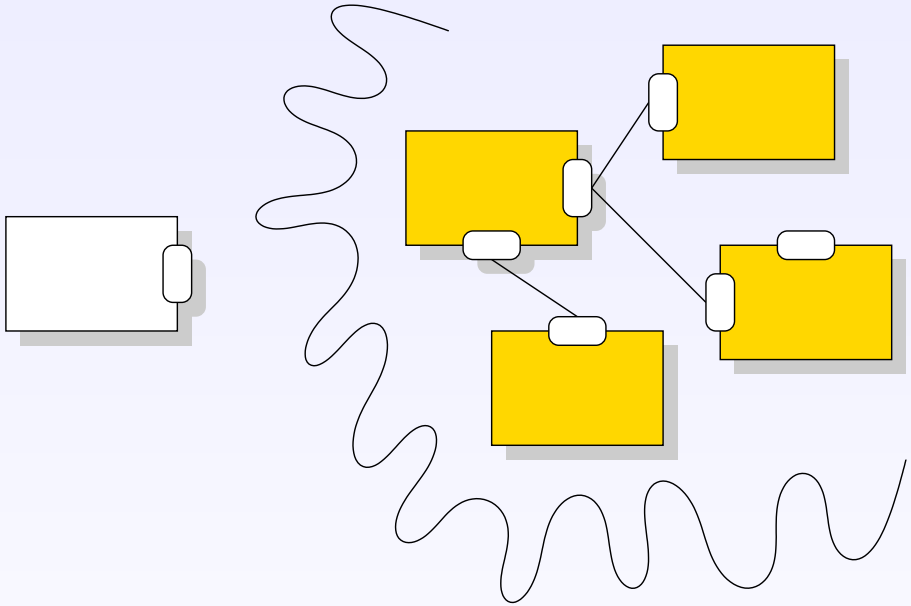
Nouveaux problèmes ?

- composants sont binaires : souvent pas de spécs, pas de code source
- emphase sur la composition (coordination), approche dynamique
- utilisation non prédictible (quand ? où ? dans quel contexte ?)

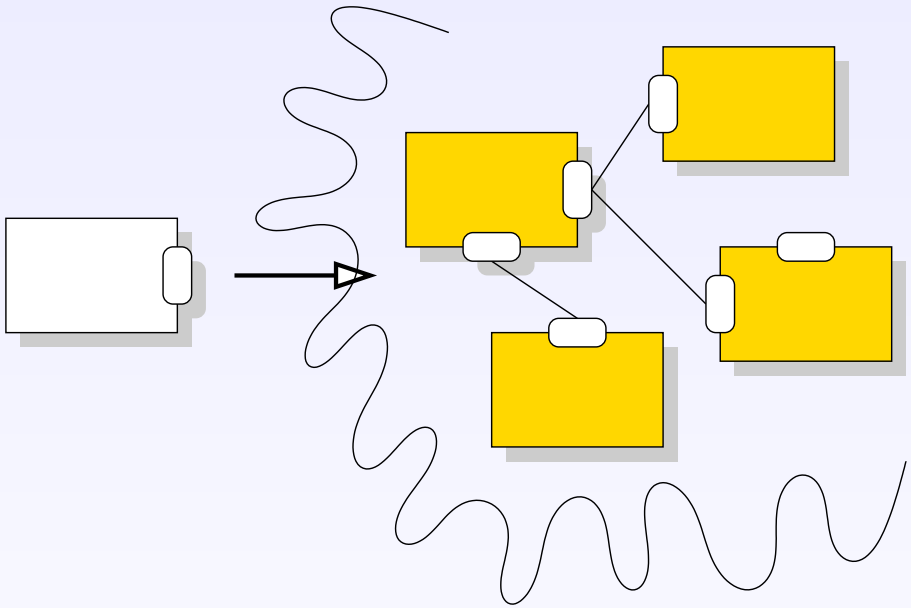
Un scénario de composition



Un scénario de composition

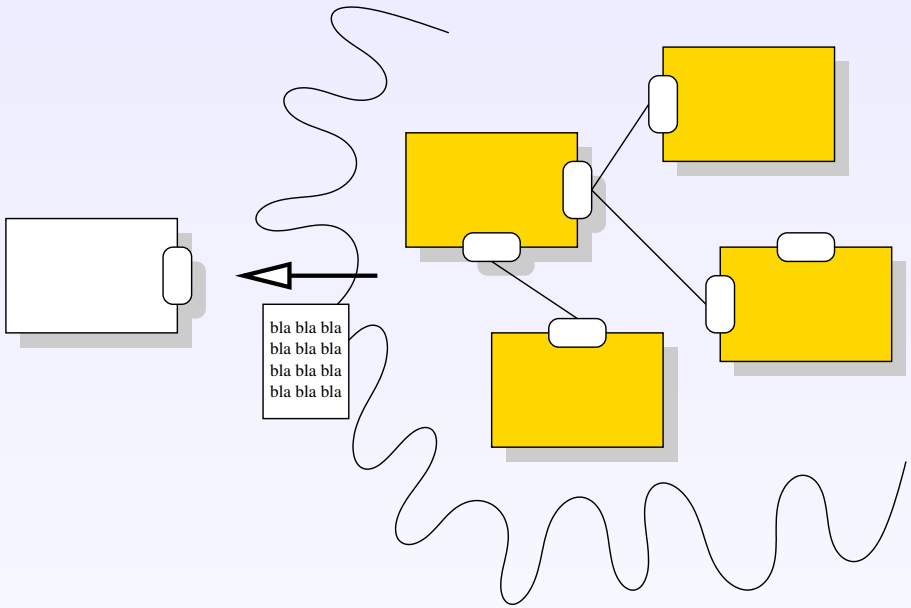


Un scénario de composition



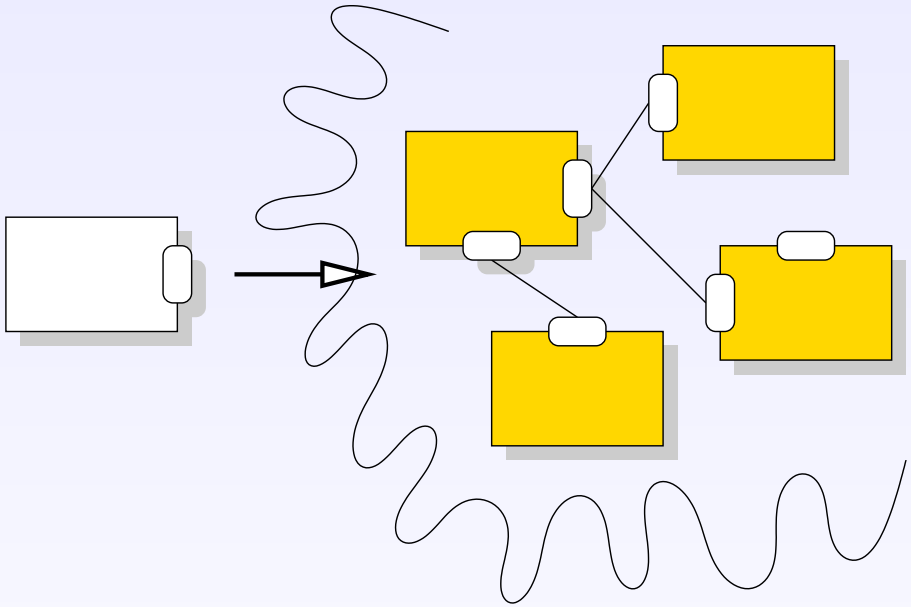
1. le composant demande l'interface du contexte

Un scénario de composition



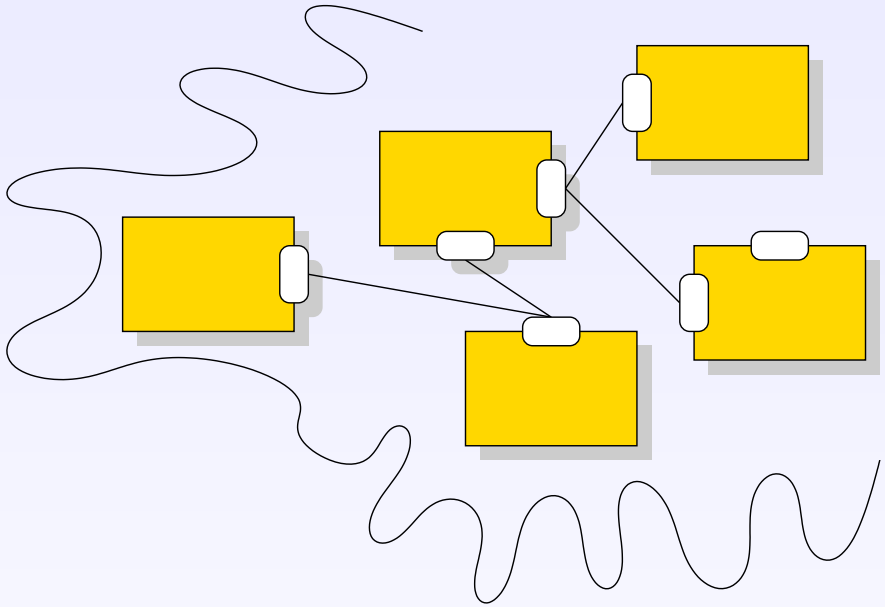
1. le composant demande l'interface du contexte
2. le contexte envoie son interface

Un scénario de composition



1. le composant demande l'interface du contexte
2. le contexte envoie son interface
3. le composant envoie une requète de connexion

Un scénario de composition

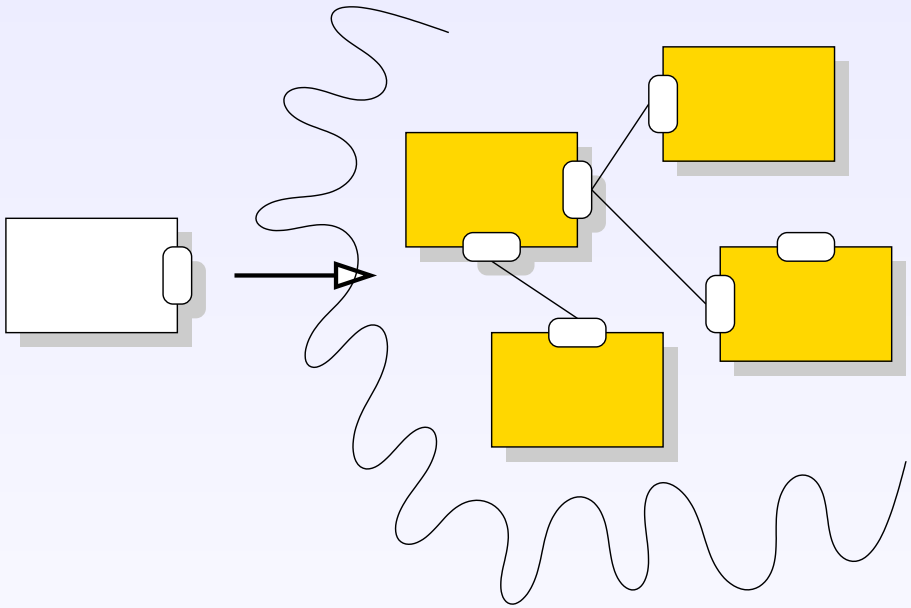


1. le composant demande l'interface du contexte
2. le contexte envoie son interface
3. le composant envoie une requète de connexion
4. le composant se joint au contexte

Nouveaux problèmes ...

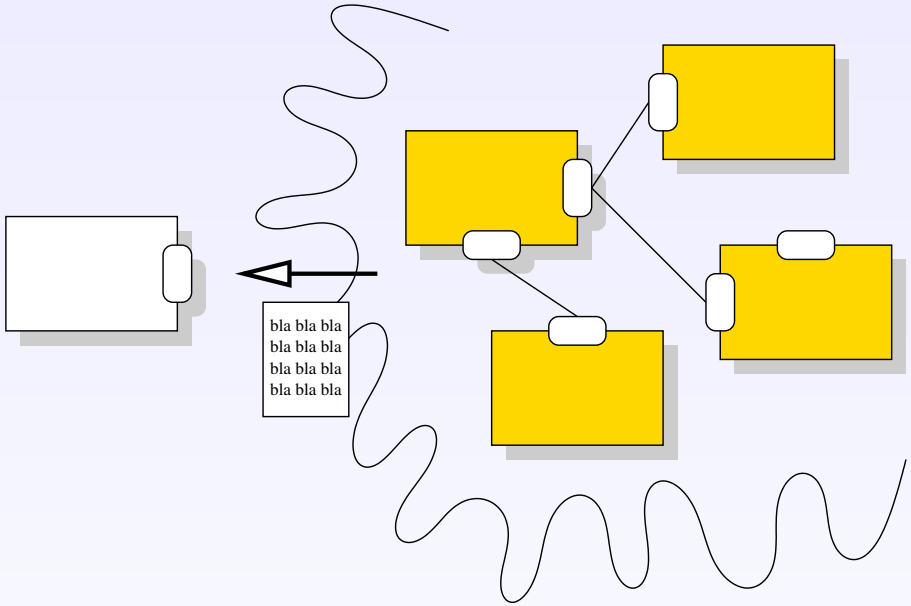
- comment joindre un contexte ?
- comment trouver le bon composant ?
- comment spécifier les caractéristiques des composants ?
- difficulté de réutilisation *as is* !

Retour sur notre scénario



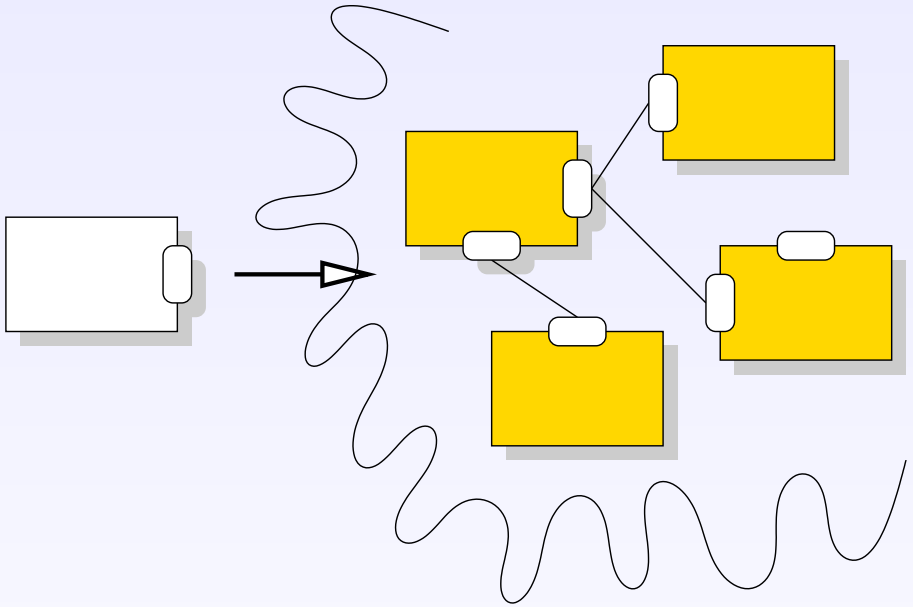
1. le composant demande l'interface du contexte

Retour sur notre scénario



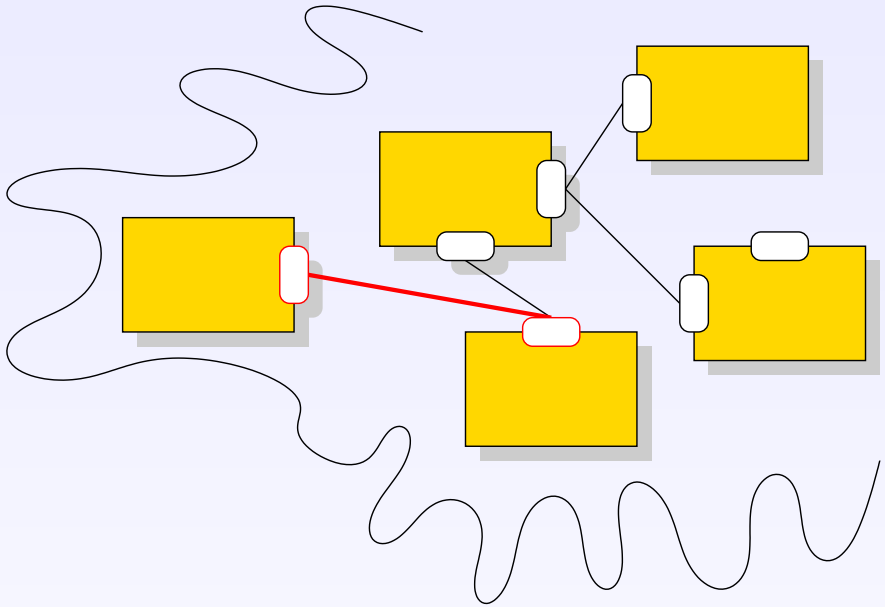
1. le composant demande l'interface du contexte
2. le contexte envoie son interface

Retour sur notre scénario



1. le composant demande l'interface du contexte
2. le contexte envoie son interface
3. le composant envoie une requète de connexion

Retour sur notre scénario

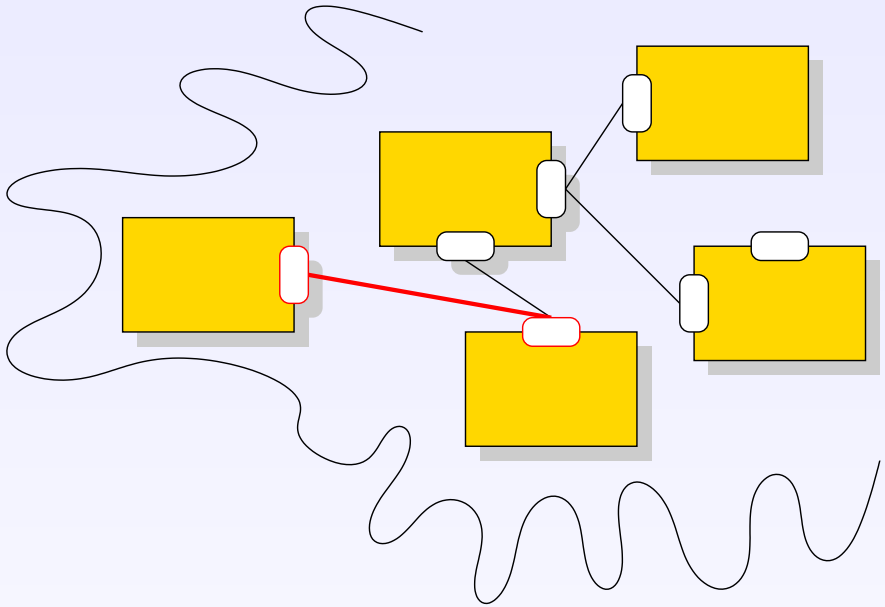


1. le composant demande l'interface du contexte
2. le contexte envoie son interface
3. le composant envoie une requète de connexion
4. le composant ne peut se connecter !
   exemple : $P|C$ bloquant

Niveaux d'interopérabilité

- Niveau signature
  - état de l'art courant
- Niveau comportemental
  - extension interfaces avec des *BIDL*
- Niveau sémantique
  - ce que *fait* le composant (fonctionnel)
- Niveau qualité de service
  - non fonctionnel (temps, sécurité, coût, ...)

Incompatibilités



- signature : noms services incompatibles, problèmes nombre/ordre/type paramètres
- comportemental : non correspondance requis/fournis, reordonnancement
- sémantique : non compatiblité de tâches
- QoS : temps de réponse, tolérance aux fautes

Niveau comportemental : (quelques) exemples

- problèmes de noms
  - `C = requête!URL . réponse?fichier . 0`
  - `Server = query?URL . return!file . Server`
- problèmes de relation 1-N
- problèmes de réordonnancement

Niveau comportemental : (quelques) exemples

- problèmes de noms
- problèmes de relation 1-N
  - `C = imprimer!(fic,qté) . 0`
  - `Printer = setCopies?nb . Printer`
    `+ print?file . Printer`
- problèmes de réordonnancement

Niveau comportemental : (quelques) exemples

- problèmes de noms
- problèmes de relation 1-N
  - `C = user!id . pass!pwd . 0`
  - `Server = login?(id,pwd) . 0`
- problèmes de réordonnancement

Niveau comportemental : (quelques) exemples

- problèmes de noms
- problèmes de relation 1-N
- problèmes de réordonnancement
  - `C = connect!id . (ask! . reply?)* . end!`
  - `Server = (log?id . ask? . reply!)*`

Adaptation, Evolution et Customization

Distinction à faire entre ces trois activités

Adaptation, Evolution et Customization

Distinction à faire entre ces trois activités

- *Evolution* : équipe originale ou programmeurs pour maintenance/extension du composant.
  - compréhension complète du composant
  - modification libre du code
  - nouveau composant accessible pour achat et réutilisation

Adaptation, Evolution et Customization

Distinction à faire entre ces trois activités

- *Evolution* : équipe originale ou programmeurs pour maintenance/extension du composant.
  - compréhension complète du composant
  - modification libre du code
  - nouveau composant accessible pour achat et réutilisation
- *Customization* : utilisateur final joue sur des options

Adaptation, Evolution et Customization

Distinction à faire entre ces trois activités

- *Evolution* : équipe originale ou programmeurs pour maintenance/extension du composant.
  - compréhension complète du composant
  - modification libre du code
  - nouveau composant accessible pour achat et réutilisation
- *Customization* : utilisateur final joue sur des options
- *Adaptation* :
  - architecte logiciel utilise un composant tiers
  - réutilisation limitée
  - utilisateur écrit du nouveau code pour altérer le composant

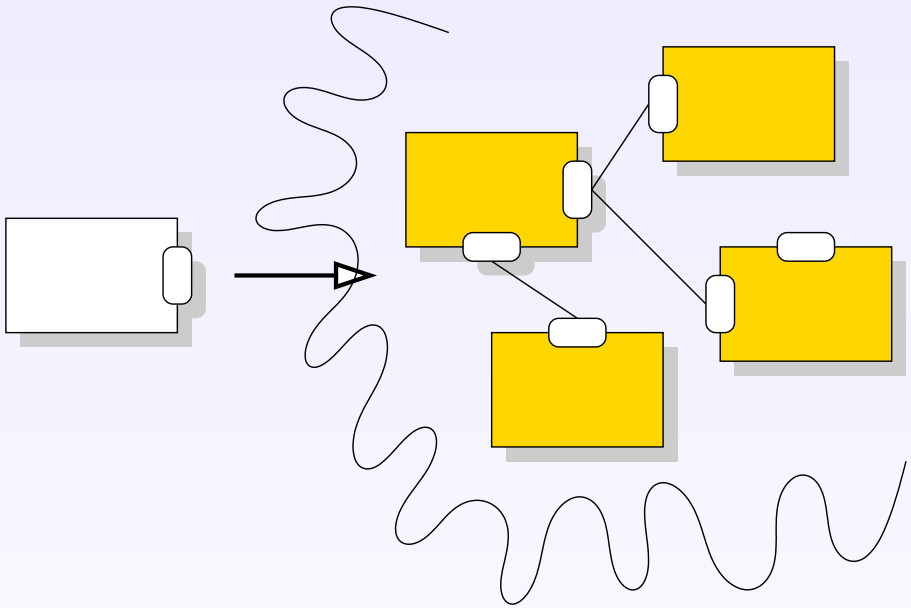
Types d'adaptation

- Statique *vs* dynamique
  conception ou run-time
- Manuelle *vs* automatique
  le plus automatique possible
- Functionnelle *vs* technique
  (ici) restriction aux signatures et au comportement

(Aparté) adaptation et implantation

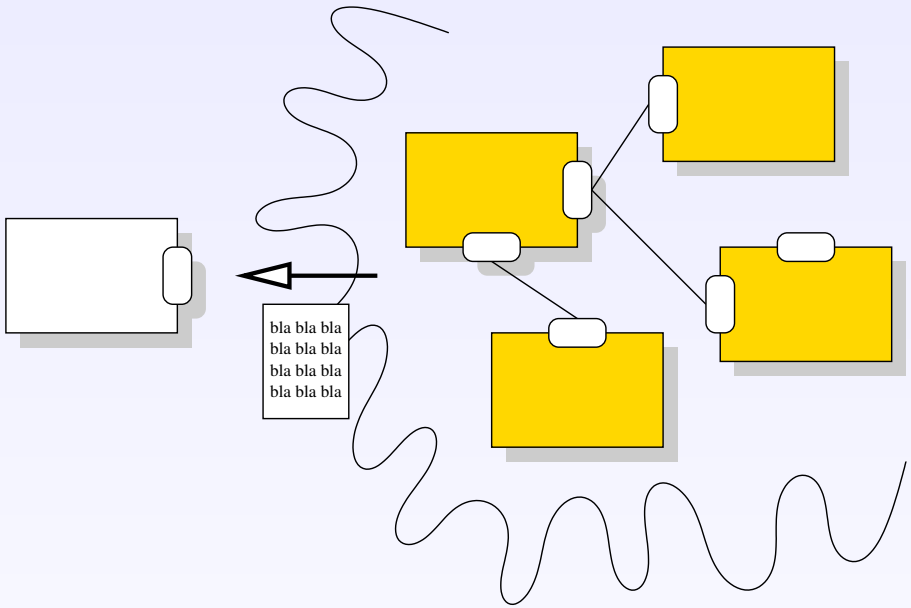
- L'adaptation est souvent présentée au niveau conceptuel. L'implantation peut passer par :
  - (code source), héritage, AOP
  - interfaces actives, wrappers
  - création de connecteurs-adapteurs (ou composants)

Retour (le dernier) sur notre scénario



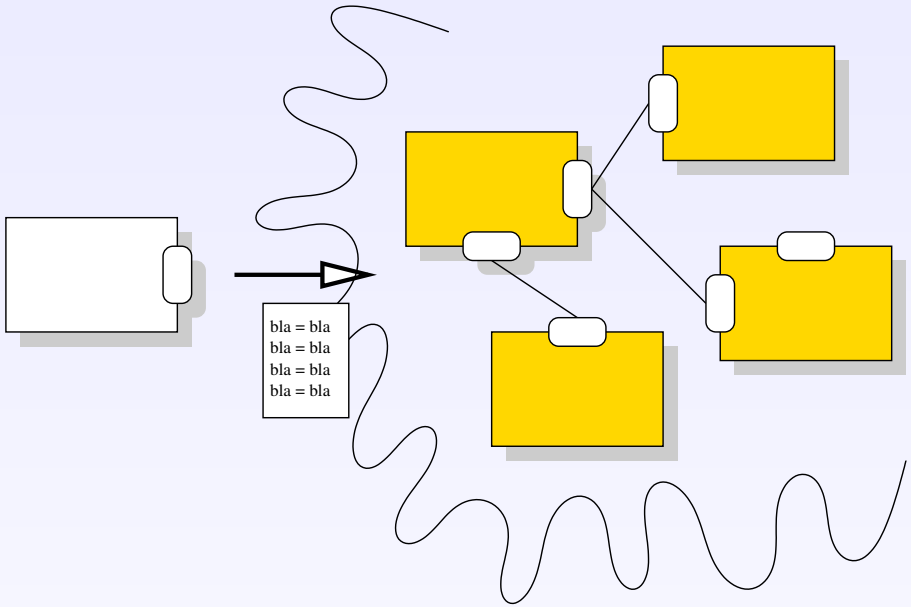
1. le composant demande l'interface du contexte

Retour (le dernier) sur notre scénario



1. le composant demande l'interface du contexte
2. le contexte envoie son interface

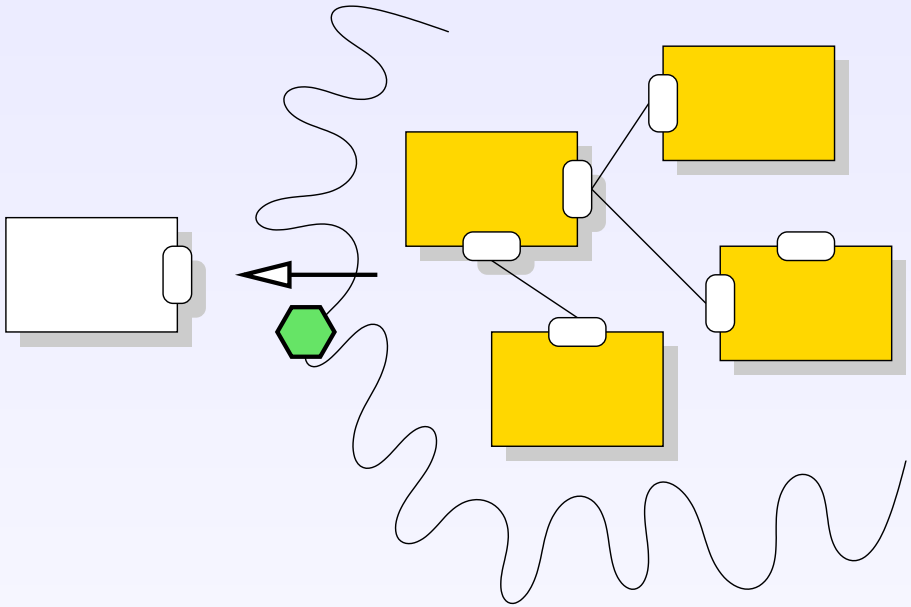
Retour (le dernier) sur notre scénario



1. le composant demande l'interface du contexte
2. le contexte envoie son interface
3. le composant envoie une requète de connexion éventuellement accompagnée d'une information pour l'adaptation :

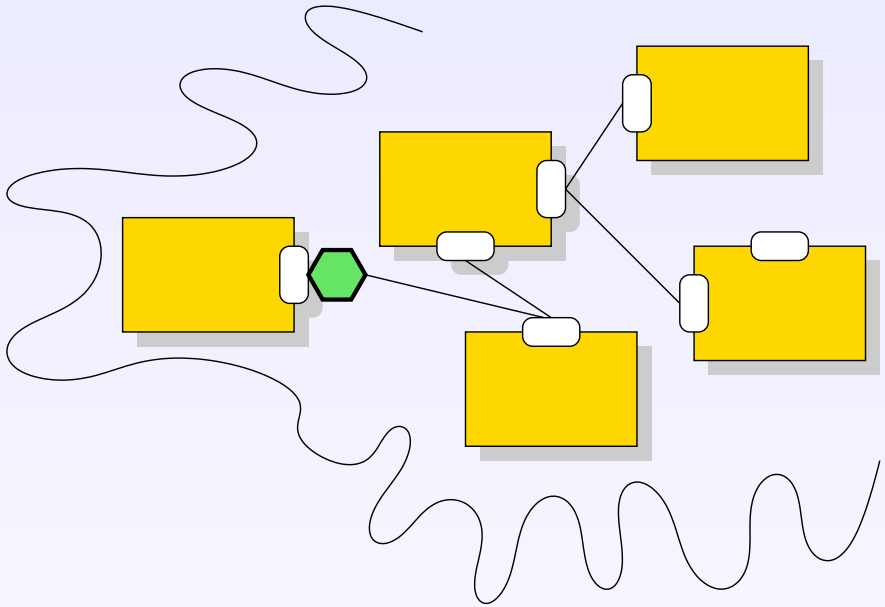
   un mapping

Retour (le dernier) sur notre scénario



1. le composant demande l'interface du contexte
2. le contexte envoie son interface
3. le composant envoie une requète de connexion [...]
4. le contexte construit un adapteur et le retourne au composant

Retour (le dernier) sur notre scénario



1. le composant demande l'interface du contexte
2. le contexte envoie son interface
3. le composant envoie une requète de connexion [...]
4. le contexte construit un adapteur [...]
5. le composant adapté se joint au contexte
   $P|A|C$ ok

Biblio

Introduction

C. Canal, J.M. Murillo, P. Poizat
Software Adaptation
Introduction au numéro spécial de l'Objet, vol. 12, n˚1, pp 9–31, 2006

Atelier WCAT

Third Int. Workshop on Coordination and Adaptation Techniques for Software Entities à ECOOP
Nantes, 4 juillet 2006

État de l'art – travaux annexes

- YS : Yellin & Strom, ACM TOPLAS 19(2) 1997
  - invention concept adaptateur, algorithme
  - machines à états finis
  - 2 composants, interaction par grammaire, compatibilité simple
- IT : Inverardi & Tivoli, JSS 65(3) 2003, FMSA LNCS 2804 2003
  - MSC
  - expression de propriétés
  - outil (SYNTHESIS), application COM
- BBCP : Brogi *et al*, JSS 74(1) 2005, SCP 2006
  - π-calcul
  - mappings expressifs, abstraction
  - algorithme (dumb adaptors), application WS

État de l'art – travaux annexes

- YS : Yellin & Strom, ACM TOPLAS 19(2) 1997
  - invention concept adaptateur, algorithme
  - machines à états finis
  - 2 composants, interaction par grammaire, compatibilité simple
- IT : Inverardi & Tivoli, JSS 65(3) 2003, FMSA LNCS 2804 2003
  - MSC
  - expression de propriétés
  - outil (SYNTHESIS), application COM
- BBCP : Brogi *et al*, JSS 74(1) 2005, SCP 2006
  - π-calcul
  - mappings expressifs, abstraction
  - algorithme (dumb adaptors), application WS

État de l'art – travaux annexes

- YS : Yellin & Strom, ACM TOPLAS 19(2) 1997
  - invention concept adaptateur, algorithme
  - machines à états finis
  - 2 composants, interaction par grammaire, compatibilité simple
- IT : Inverardi & Tivoli, JSS 65(3) 2003, FMSA LNCS 2804 2003
  - MSC
  - expression de propriétés
  - outil (SYNTHESIS), application COM
- BBCP : Brogi *et al*, JSS 74(1) 2005, SCP 2006
  - π-calcul
  - mappings expressifs, abstraction
  - algorithme (dumb adaptors), application WS

État de l'art – travaux annexes

- YS : Yellin & Strom, ACM TOPLAS 19(2) 1997
  - invention concept adaptateur, algorithme
  - machines à états finis
  - 2 composants, interaction par grammaire, compatibilité simple
- IT : Inverardi & Tivoli, JSS 65(3) 2003, FMSA LNCS 2804 2003
  - MSC
  - expression de propriétés
  - outil (SYNTHESIS), application COM
- BBCP : Brogi *et al*, JSS 74(1) 2005, SCP 2006
  - π-calcul
  - mappings expressifs, abstraction
  - algorithme (dumb adaptors), application WS

État de l'art – comparaison

- decription du comportement
  - alg. proc. (BBCP) *vs* Automata/MSC (IT)
- spécification de l'adaptateur
  - simple (IT) *vs* expressive (BBCP)
- adaptation de protocole
  - approche *restrictive* (IT) *vs generative* (BBCP)
  - simple (IT) *vs* reordonnancement (BBCP)
- satisfaction de propriétés
  - limité au non blocage (BBCP) *vs* LTL (IT)
- adaptation binaire *vs* n-aire
  - 1-1 (BBCP) *vs* système + incrémental (IT)

État de l'art – comparaison

- decription du comportement
  - alg. proc. (BBCP) *vs* Automata/MSC (IT)
- spécification de l'adaptateur
  - simple (IT) *vs* expressive (BBCP)
- adaptation de protocole
  - approche *restrictive* (IT) *vs generative* (BBCP)
  - simple (IT) *vs* reordonnancement (BBCP)
- satisfaction de propriétés
  - limité au non blocage (BBCP) *vs* LTL (IT)
- adaptation binaire *vs* n-aire
  - 1-1 (BBCP) *vs* système + incrémental (IT)

État de l'art – comparaison

- decription du comportement
  - alg. proc. (BBCP) *vs* Automata/MSC (IT)
- spécification de l'adaptateur
  - simple (IT) *vs* expressive (BBCP)
- adaptation de protocole
  - approche *restrictive* (IT) *vs generative* (BBCP)
  - simple (IT) *vs* reordonnancement (BBCP)
- satisfaction de propriétés
  - limité au non blocage (BBCP) *vs* LTL (IT)
- adaptation binaire *vs* n-aire
  - 1-1 (BBCP) *vs* système + incrémental (IT)

État de l'art – comparaison

- decription du comportement
  - alg. proc. (BBCP) *vs* Automata/MSC (IT)
- spécification de l'adaptateur
  - simple (IT) *vs* expressive (BBCP)
- adaptation de protocole
  - approche *restrictive* (IT) *vs generative* (BBCP)
  - simple (IT) *vs* reordonnancement (BBCP)
- satisfaction de propriétés
  - limité au non blocage (BBCP) *vs* LTL (IT)
- adaptation binaire *vs* n-aire
  - 1-1 (BBCP) *vs* système + incrémental (IT)

État de l'art – comparaison

- decription du comportement
  - alg. proc. (BBCP) *vs* Automata/MSC (IT)
- spécification de l'adaptateur
  - simple (IT) *vs* expressive (BBCP)
- adaptation de protocole
  - approche *restrictive* (IT) *vs generative* (BBCP)
  - simple (IT) *vs* reordonnancement (BBCP)
- satisfaction de propriétés
  - limité au non blocage (BBCP) *vs* LTL (IT)
- adaptation binaire *vs* n-aire
  - 1-1 (BBCP) *vs* système + incrémental (IT)

État de l'art – comparaison

- decription du comportement
  - alg. proc. (BBCP) *vs* Automata/MSC (IT)
- spécification de l'adaptateur
  - simple (IT) *vs* expressive (BBCP)
- adaptation de protocole
  - approche *restrictive* (IT) *vs generative* (BBCP)
  - simple (IT) *vs* reordonnancement (BBCP)
- satisfaction de propriétés
  - limité au non blocage (BBCP) *vs* LTL (IT)
- adaptation binaire *vs* n-aire
  - 1-1 (BBCP) *vs* système + incrémental (IT)

État de l'art – synthèse

critère	IT	BBCP
BIDL	automates	alg. proc.
propriétés	non blocage, LTL	non blocage —
abstraction	non	oui
adapt. noms	non	oui
données	non	oui
réordo.	non	oui
système	oui	non

BIDL

- LTS : (A, S, I, F, T)
- Algèbre de processus simple :
  `P : := 0 | a?.P | a!.P | P1+P2 | A`
  avec `[i]` and `[f]`
- correspondance simple

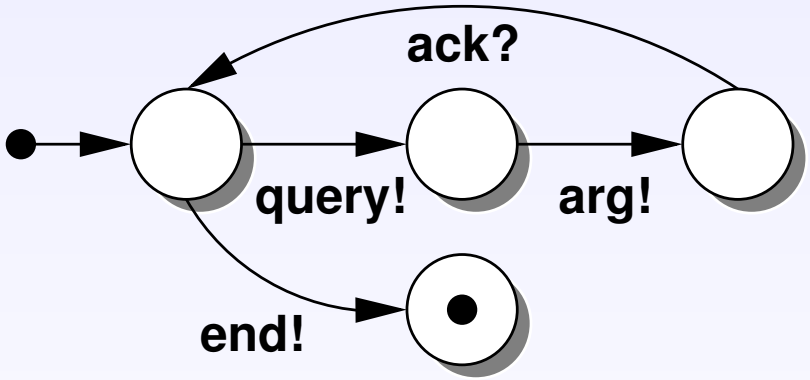
Exemple

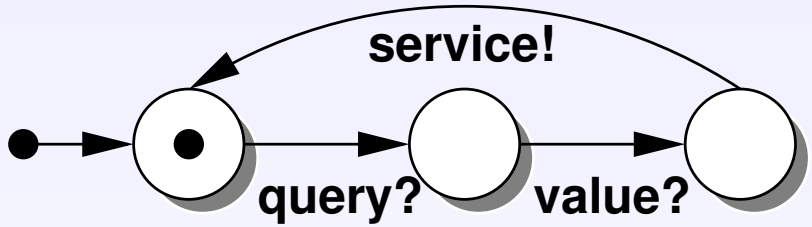
... un client qui envoie de façon répétitive une requête et son argument puis attend un acquitement et peut quitter, un serveur qui attend de façon répétitive une requête et une valeur puis envoie un certain service ...

Exemple

```
Client[i] = query!.arg!.ack?.Client + end![f]

Server[i,f] = query?.value?.service!.Server
```

Exemple

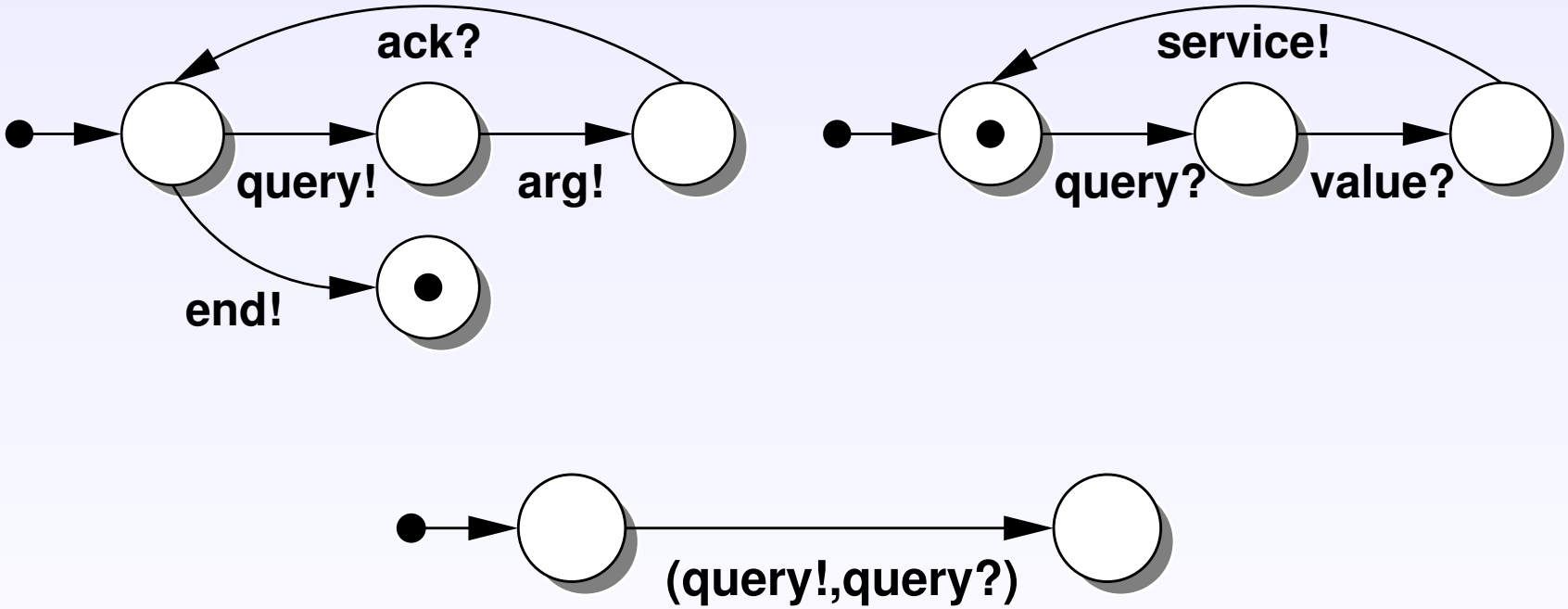




(In)Compatibilité

- Π produit synchronisé de LTS [Arnold 1994]
- $\texttt{dead}(s) : s \in S \wedge s \notin F \wedge \nexists l \in A, s' \in S \,.\, (s, l, s') \in T$
- incompatibilité : $\exists s \in S$ such that $\texttt{dead}(s)$.

Exemple



Outillage

- utilisation de EXP.OPEN (CADP)
- système de base :
  1. traduction C_i en `C_i.aut`,
  2. boucles `accept` sur états finaux,
  3. fichier de synchro, patron `Global.exp` :
     `C_1 || C_2 || ... || C_n`
  4. EXP.OPEN calcule le LTS global (`.aut`).
- système adapté :
  1. traduction adaptateur en `A.aut`
  2. boucles `accept` sur états finaux,
  3. patron `Global.exp` :
     `(C_1 ||| C_2 ||| ... ||| C_n) || A`
  4. EXP.OPEN calcule le LTS global (`.aut`).

Outillage

- utilisation de EXP.OPEN (CADP)
- système de base :
  1. traduction C_i en `C_i.aut`,
  2. boucles `accept` sur états finaux,
  3. fichier de synchro, patron `Global.exp` :

     ```
     C_1 || C_2 || ... || C_n
     ```

  4. EXP.OPEN calcule le LTS global (`.aut`).
- système adapté :
  1. traduction adaptateur en `A.aut`
  2. boucles `accept` sur états finaux,
  3. patron `Global.exp` :

     ```
     (C_1 ||| C_2 ||| ... ||| C_n) || A
     ```

  4. EXP.OPEN calcule le LTS global (`.aut`).

Outillage

- utilisation de EXP.OPEN (CADP)
- système de base :
  1. traduction C_i en `C_i.aut`,
  2. boucles `accept` sur états finaux,
  3. fichier de synchro, patron `Global.exp` :

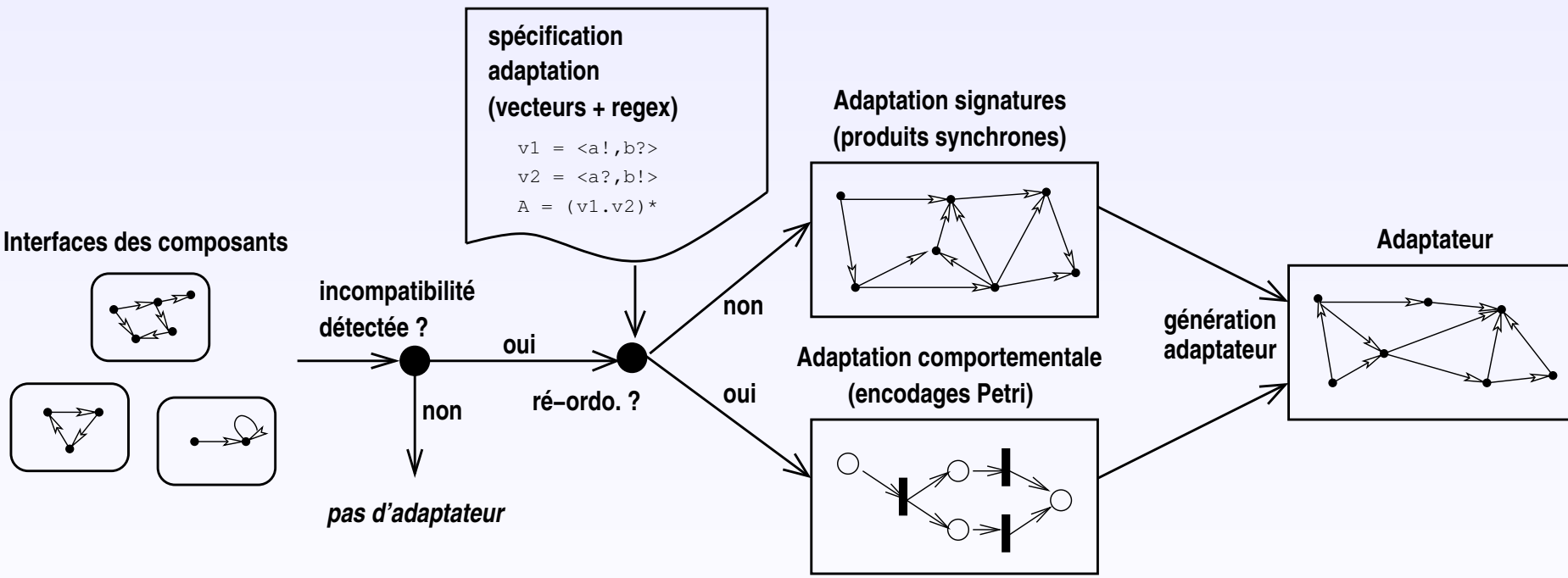
     ```
     C_1 || C_2 || ... || C_n
     ```

  4. EXP.OPEN calcule le LTS global (`.aut`).
- système adapté :
  1. traduction adaptateur en `A.aut`
  2. boucles `accept` sur états finaux,
  3. patron `Global.exp` :

     ```
     (C_1 ||| C_2 ||| ... ||| C_n) || A
     ```

  4. EXP.OPEN calcule le LTS global (`.aut`).

Schéma général



Mappings

Vecteurs de synchronisation

Etant donné un ensemble *Id* indexé de LTS
$L_i = (A_i, S_i, I_i, F_i, T_i)$, $i \in Id$
un *vecteur* est un tuple (e_i) avec $e_i \in A_i \cup \{\varepsilon\}$.

Example

- <send!,rec?,ε>
- <send!,ε,rec?>
- <send!,rec?,rec?>

Mappings

Vecteurs de synchronisation

Etant donné un ensemble *Id* indexé de LTS
$L_i = (A_i, S_i, I_i, F_i, T_i)$, $i \in Id$
un *vecteur* est un tuple (e_i) avec $e_i \in A_i \cup \{\varepsilon\}$.

Example

- <send!,rec?,ε>
- <send!,ε,rec?>
- <send!,rec?,rec?>

Adaptation simple

1. compute $P = (A_P, S_P, I_P, F_P, T_P) = \Pi(L_i, V)$
2. remove paths-to-deadlock (from IT)
3. compute transitions permutations in transactions and reverse actions

Example

```
Client[i] =
c :query!.c :arg!.c :ack?.Client + c :end![f]

Server[i,f] =
s :query?.s :value?.s :service!.Server
```

Vecteurs :

<c :query!,s :query?>, <c :arg!,s :value?>,
<c :ack?,s :service!> et <c :end!,s :ε>.

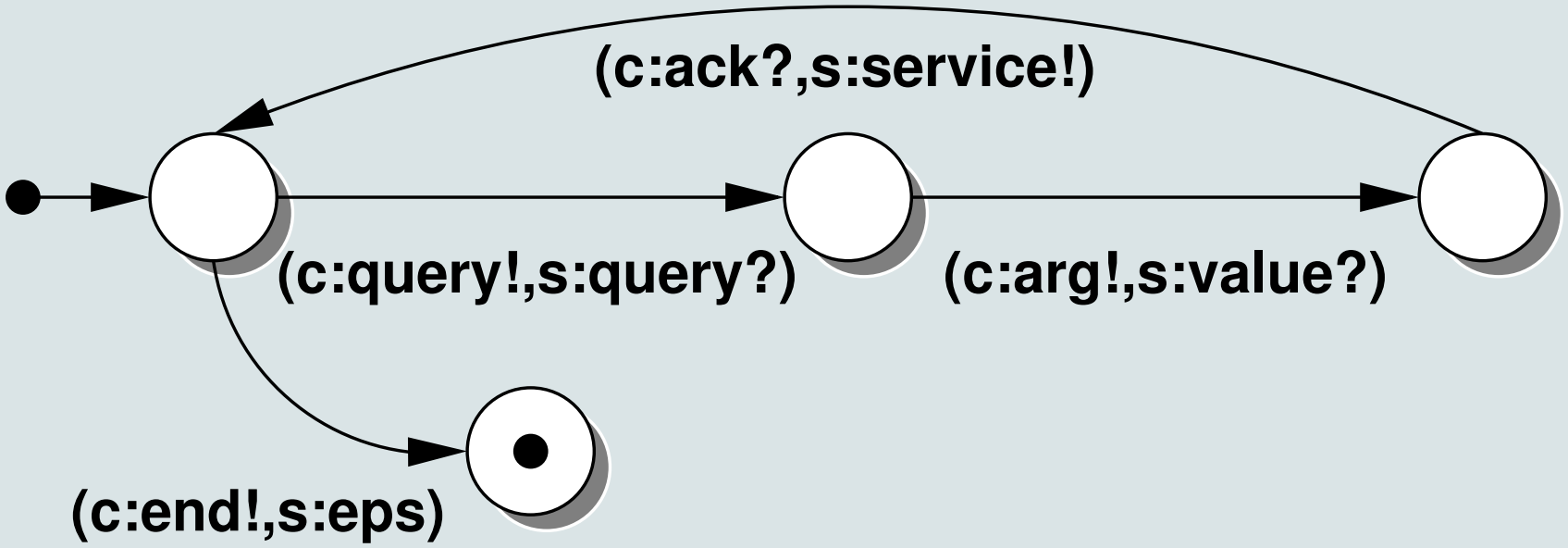
Example

```
Client[i] =
c :query!.c :arg!.c :ack?.Client + c :end![f]

Server[i,f] =
s :query?.s :value?.s :service!.Server
```

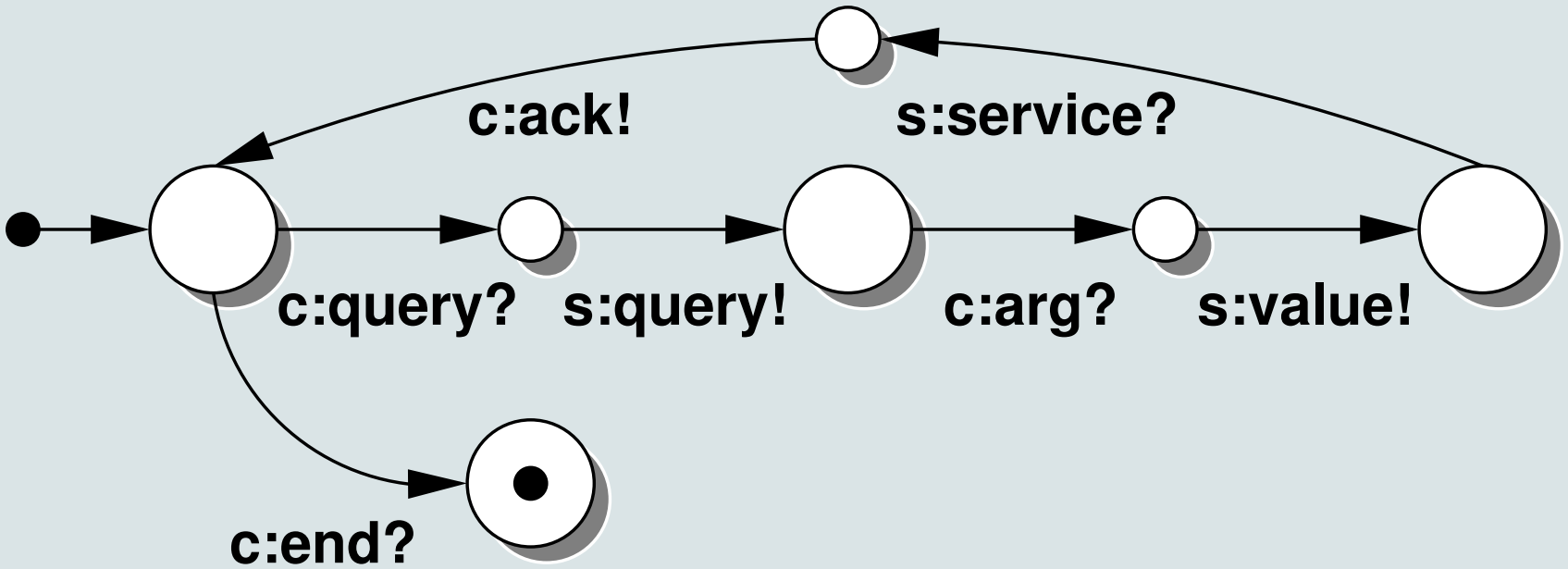
Vecteurs :

<c :query!,s :query?>, <c :arg!,s :value?>,
<c :ack?,s :service!> et <c :end!,s :ε>.

Example



Example



Adaptation réordonnée

1. for each component i with LTS L_i, for each $s_j \in S_i$, add a place `Control-i-s_j`
2. for each component i with initial state I_i, put a token in `Control-i-I_i`
3. for each $a!$ in $\bigcup_i A_i$, add a place `Rec-a`
4. for each $a?$ in $\bigcup_i A_i$, add a place `Em-a`
5. for each component i with LTS L_i, for each $(s, l, s') \in T_i$:
   - add a transition with label $\overline{l}$, one arc from place `Control-i-s` to the transition and one arc from the transition to place `Control-i-s'`
   - if `l` has the form $a!$ then add one arc from the transition to place `Rec-a`
   - if `l` has the form $a?$ then add one arc from place `Em-a` to the transition

Adaptation réordonnée (suite)

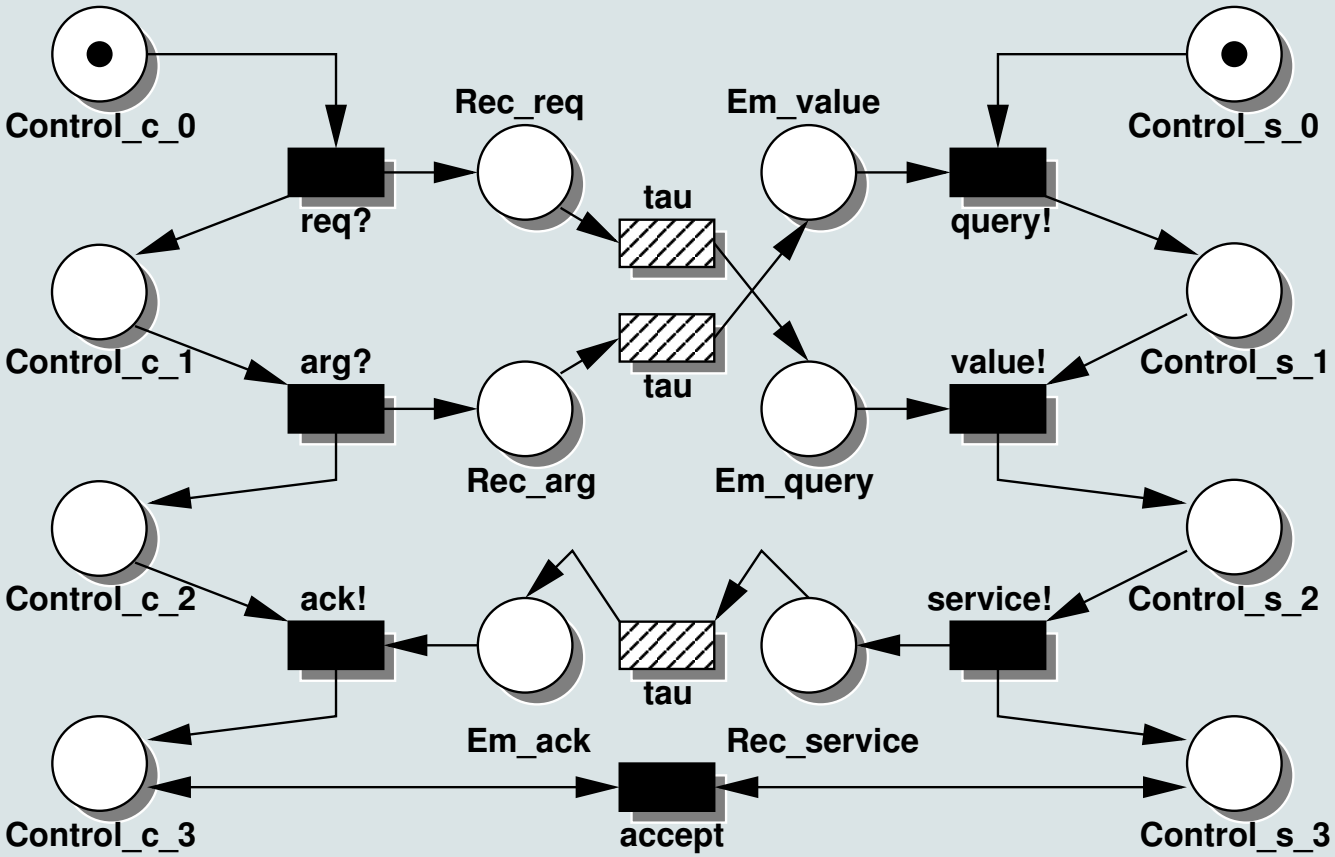
6. for each vector $v = (l_1, \ldots, l_n)$ in V :
   - add a transition with label `tau`
   - for each l_i with form $a!$, add one arc from place `Rec-a` to the transition
   - for each l_i with form $a?$, add one arc from the transition to place `Em-a`
7. for each tuple $(f_1, \ldots, f_n)$, $f_i \in F_i$, of final states, add a (loop) `accept` transition with arcs from and to each of the tuple f_i
8. compute marking / cover graph (TINA)
9. remove paths-to-deadlock
10. apply reduction on adapter (CADP)

Example

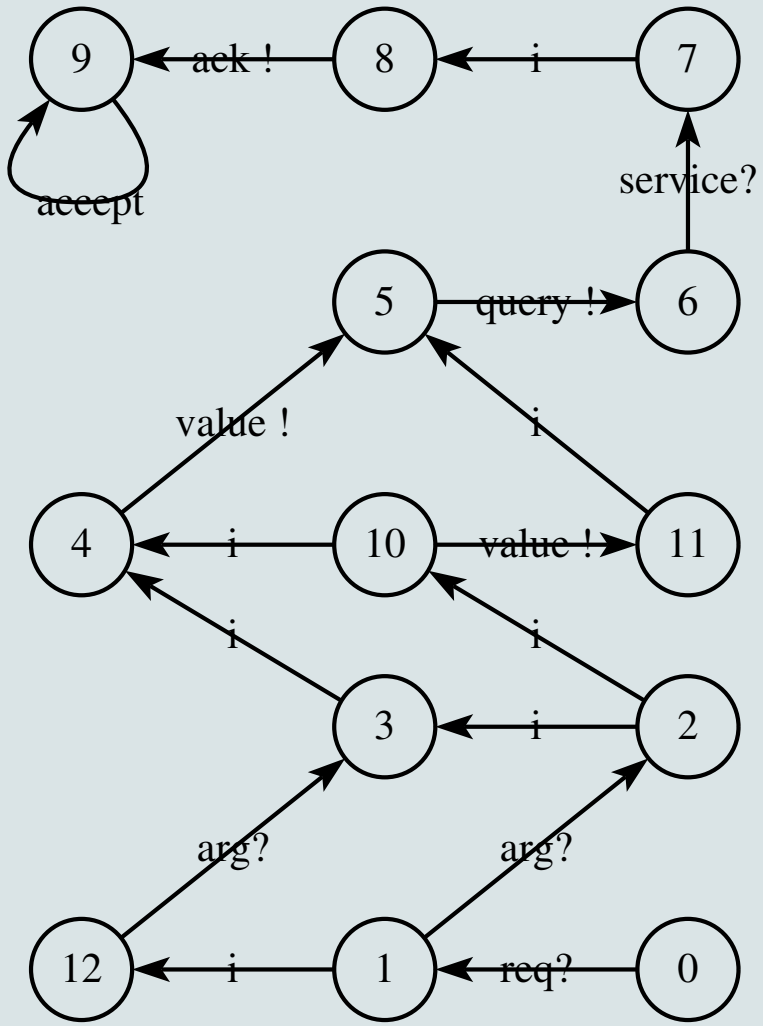
```
Client[i]=req!.arg!.ack?[f]

Server[i]=value?.query?.service![f]
```

Vecteurs :
<req!,query?>,
<arg!,value?> et
<ack?,service!>.

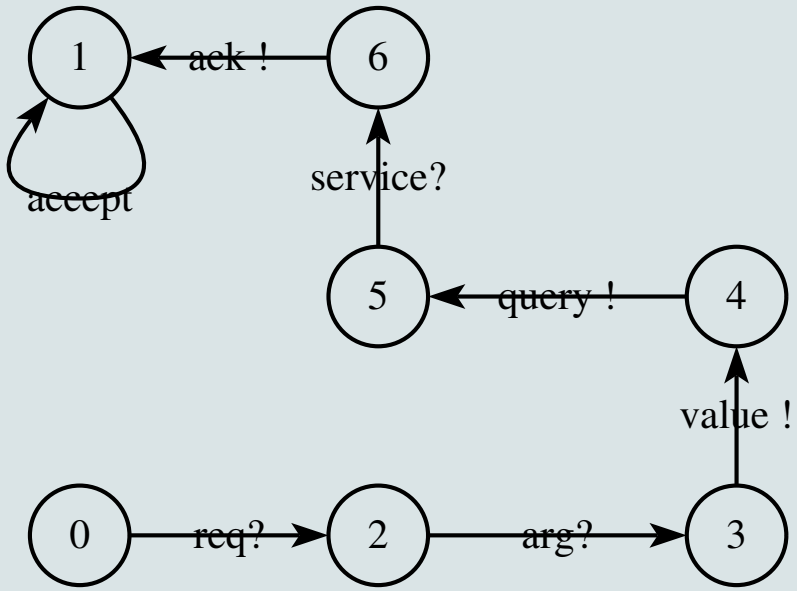
Example


Control_c_0
Rec_req
Em_value
Control_s_0
req?
tau
query!
Control_c_1
arg?
tau
value!
Control_s_1
Rec_arg
Em_query
Control_c_2
ack!
service!
Control_s_2
tau
Em_ack
Rec_service
Control_c_3
accept
Control_s_3

Example


9
ack !
8
i
7
accept
service?
5
query !
6
value !
i
4
i
10
value !
11
i
i
3
i
2
arg?
arg?
12
i
1
req?
0

Example



Limites

- impossible d'avoir des mappings 1-N
  - `<imprimer!,setcopies?.print?>`
- impossible d'exprimer des propriétés sur mappings
  - *... le client accède successivement aux deux serveurs ...*
  - *... le client ne s'identifie qu'au début, le serveur a besoin d'un id à chaque requête ...*
- ambiguité des vecteurs au sein d'un mapping
  - `<c :req!,s1 :query?,s2 :ε>,`
    `<c :req!,s1 :ε,s2 :query?>`
    ET ? OU ?
  - `<c :connect!,s :log?>,<c :ε,s :log?>`
    ET ? OU ? PUIS ?

Limites

- impossible d'avoir des mappings 1-N
  - <imprimer!,setcopies?.print?>
- impossible d'exprimer des propriétés sur mappings
  - *... le client accède successivement aux deux serveurs ...*
  - *... le client ne s'identifie qu'au début, le serveur a besoin d'un id à chaque requête ...*
- ambiguité des vecteurs au sein d'un mapping
  - <c :req!,s1 :query?,s2 :ε>,
    <c :req!,s1 :ε,s2 :query?>
    ET ? OU ?
  - <c :connect!,s :log?>,<c :ε,s :log?>
    ET ? OU ? PUIS ?

Limites

- impossible d'avoir des mappings 1-N
  - <imprimer!,setcopies?.print?>
- impossible d'exprimer des propriétés sur mappings
  - *... le client accède successivement aux deux serveurs ...*
  - *... le client ne s'identifie qu'au début, le serveur a besoin d'un id à chaque requête ...*
- ambiguité des vecteurs au sein d'un mapping
  - <c :req!,s1 :query?,s2 :ε>,
    <c :req!,s1 :ε,s2 :query?>
    ET ? OU ?
  - <c :connect!,s :log?>,<c :ε,s :log?>
    ET ? OU ? PUIS ?

Regex

```
R    ::=  v          (VECTOR)
     |    R1.R2      (SEQUENCE)
     |    R1+R2      (CHOICE)
     |    R*         (ITERATION)
```

where `R`, `R1`, `R2` are regex, and `v` is a vector in V

Example (Usage alterné)

C[i] = c :end![f] + c :req!.c :arg!.c :ack?.C,

S[i,f] = s :value?.s :query?.s :service!.S, et

A[i,f] = a :value?.a :query?.a :service!.A.

Regex :

$(v_{s1} \cdot v_{s2} \cdot v_{s3} \cdot v_{a1} \cdot v_{a2} \cdot v_{a3})* \cdot v_{end}$

Vecteurs :

v_{s1} =<c :req!,s :query?,a :ε>, v_{a1} =<...>,
v_{s2} =<c :arg!,s :value?,a :ε>, v_{a2} =<...>,
v_{s3} =<c :ack?,s :service!,a :ε>, v_{a3} =<...>,
v_{end} =<c :end!,s :ε,a :ε>.

Example (Usage alterné (moins restrictif))

`C[i] = c :end![f] + c :req!.c :arg!.c :ack?.C,`

`S[i,f] = s :value?.s :query?.s :service!.S,` et

`A[i,f] = a :value?.a :query?.a :service!.A.`

Regex :

$(v_{s1} \cdot v_{s2} \cdot v_{s3} + v_{a1} \cdot v_{a2} \cdot v_{a3})^* \cdot v_{\text{end}}$

Vecteurs :

v_{s1} =<c :req!,s :query?,a :ε>, v_{a1} =<...>,
v_{s2} =<c :arg!,s :value?,a :ε>, v_{a2} =<...>,
v_{s3} =<c :ack?,s :service!,a :ε>, v_{a3} =<...>,
v_{end} =<c :end!,s :ε,a :ε>.

Example (Connecté *vs* non connecté)

`C[i]=c :log!.Logged`, avec

`Logged[f]=c :req!.c :ack?.Logged`, et

`S[i,f]=s :log?.s :req?.s :ack!.S`

Regex :

$v_0.v_2.v_3.(v_1.v_2.v_3)*$

Vecteurs :

v_0 =<c :log!,s :log?>,
v_1 =<c :ε,s :log?>,
v_2 =<c :req!,s :req?>,
v_3 =<c :ack?,s :ack!>.

Adaptation simple + regex

1. compute the LTS L_R for the regex R
2. compute the product
   $P_R = (A_{P_R}, S_{P_R}, I_{P_R}, F_{P_R}, T_{P_R}) = \Pi(L_R, L_i)$
3. compute $P = \texttt{proj}(P_R)$

Adaptation reordonnée + regex

1. compute the LTS $L_R = (A_R, S_R, I_R, F_R, T_R)$ for the regex R.
2. build the Petri net encoding replacing part 6 with :
   - for each state s_R in S_R, add a place `ControlR-s_R`
   - put a token in place `ControlR-I_R`
   - for each transition $t_R = (s_R, (l_1, \ldots, l_n), s'_R)$ in T_R :
     - add a transition with label `tau`, one arc from place `ControlR-s_R` to the transition and one arc from the transition to place `ControlR-s'_R`
     - for each l_i which has the form $a!$, add one arc from place `Rec-a` to the transition
     - for each l_i which has the form $a?$, add one arc from the transition to place `Em-a`
3. in the building of `accept` transitions, add F_R to the F_i taken into account (final states now correspond to acceptance states of the regex LTS).

Example (Usage alterné (moins restrictif))

`C[i] = c :end![f] + c :req!.c :arg!.c :ack?.C,`

`S[i,f] = s :value?.s :query?.s :service!.S,` et

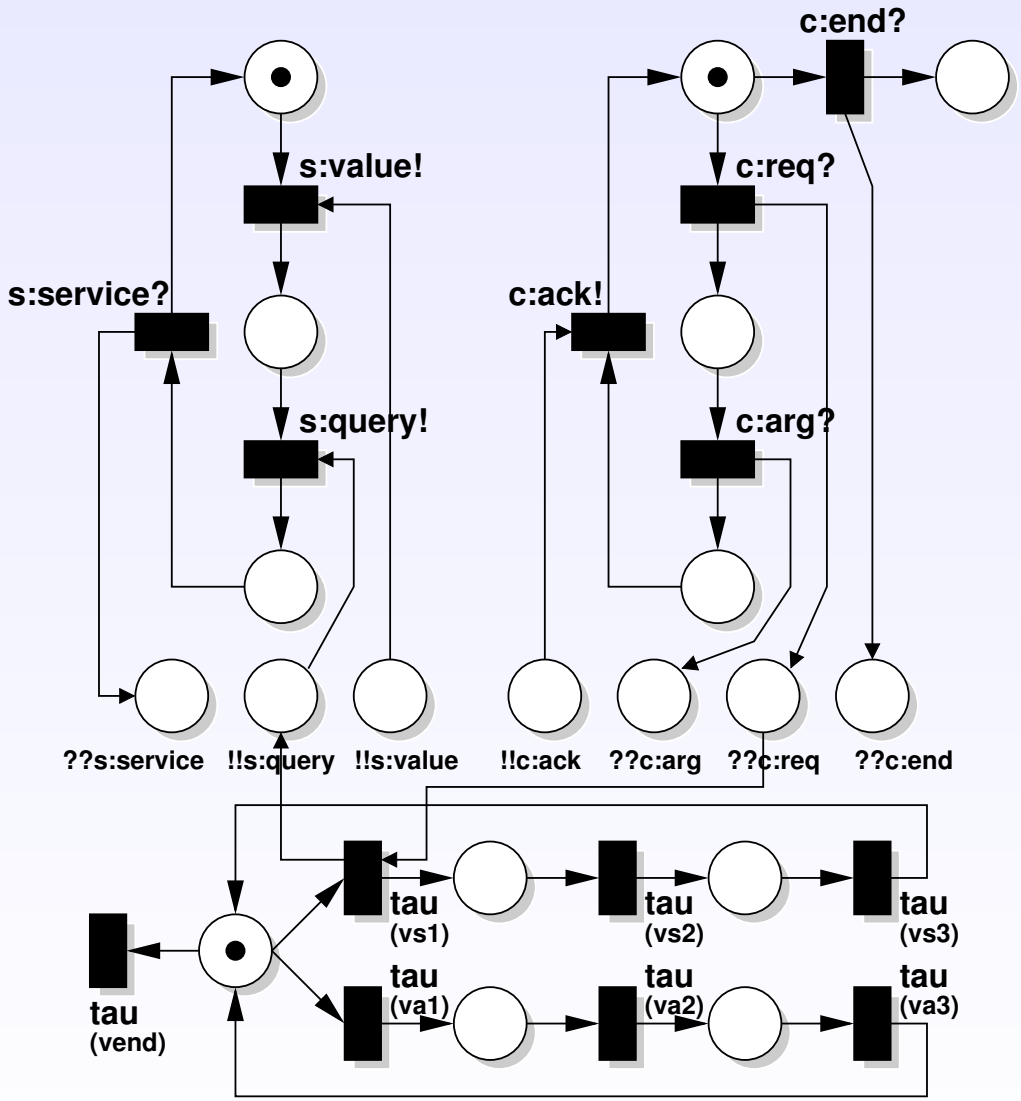
`A[i,f] = a :value?.a :query?.a :service!.A.`

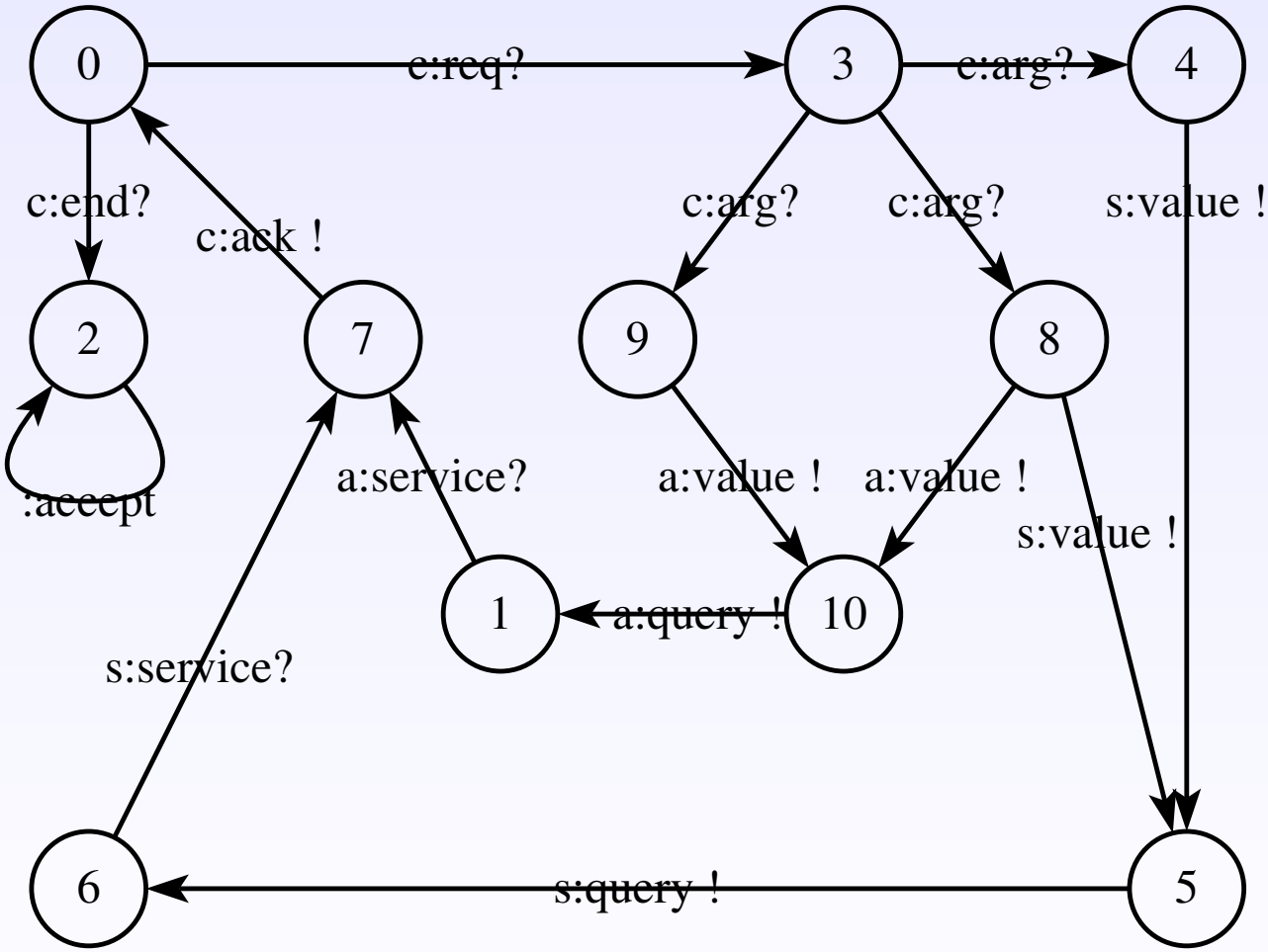
Regex :

$(v_{s1} \cdot v_{s2} \cdot v_{s3} + v_{a1} \cdot v_{a2} \cdot v_{a3})^* \cdot v_{\text{end}}$

Vecteurs :

v_{s1} =<c :req!,s :query?,a :ε>, v_{a1} =<...>,
v_{s2} =<c :arg!,s :value?,a :ε>, v_{a2} =<...>,
v_{s3} =<c :ack?,s :service!,a :ε>, v_{a3} =<...>,
v_{end} =<c :end!,s :ε,a :ε>.


c:end?
s:value!
c:req?
s:service?
c:ack!
s:query!
c:arg?
??s:service
!!s:query
!!s:value
!!c:ack
??c:arg
??c:req
??c:end
tau
(vs1)
tau
(vs2)
tau
(vs3)
tau
(va1)
tau
(va2)
tau
(va3)
tau
(vend)


0
c:req?
3
c:arg?
4
c:end?
c:ack !
c:arg?
c:arg?
s:value !
2
7
9
8
:accept
a:service?
a:value !
a:value !
s:value !
1
a:query !
10
s:service?
6
s:query !
5

Example (Connecté *vs* non connecté)

`C[i]=c :log!.Logged`, avec

`Logged[f]=c :req!.c :ack?.Logged`, et

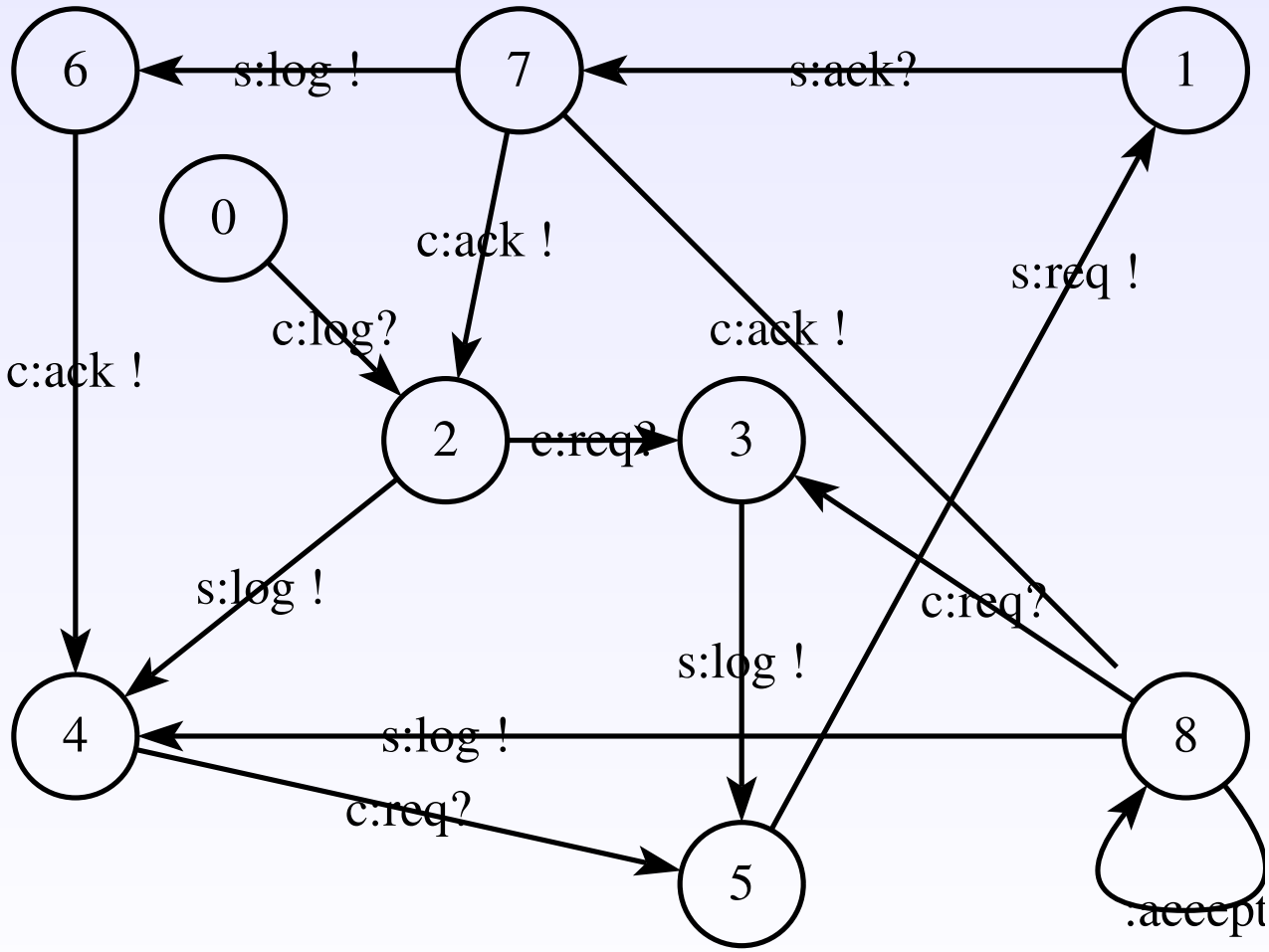
`S[i,f]=s :log?.s :req?.s :ack!.S`

Regex :

$v_0.v_2.v_3.(v_1.v_2.v_3)*$

Vecteurs :

v_0 =<c :log!,s :log?>,
v_1 =<c :ε,s :log?>,
v_2 =<c :req!,s :req?>,
v_3 =<c :ack?,s :ack!>.


6
s:log !
7
s:ack?
1
0
c:ack !
s:req !
c:log?
c:ack !
c:ack !
2
c:req?
3
s:log !
c:req?
s:log !
4
s:log !
8
c:req?
5
.accept

Synthèse

critère	IT	BBCP
BIDL	automates	alg. proc.
propriétés	non blocage, LTL	non blocage —
abstraction	non	oui
adapt. noms	non	oui
données	non	oui
réordo.	non	oui
système	oui	non

Synthèse

critère	IT	BBCP	CPS
BIDL	automates	alg. proc.	LTS ou alg. proc.
propriétés	non blocage, LTL	non blocage —	non blocage, regex
abstraction	non	oui	oui
adapt. noms	non	oui	oui
données	non	oui	non
réordo.	non	oui	oui
système	oui	non	oui

Biblio

Rapport

C. Canal, P. Poizat, G. Salaün
Adaptation of Component Behaviour using Synchronous Vectors
TR ITI–05–10, Universidad de Málaga, décembre 2005
(FMOODS'2006 - à paraître)

Adaptor

TER en cours
BIDL : alg. proc., LTS, XML
Mappings : regex, LTS, XML, (LTL)
(démo)

Conclusions

Adaptation

- dans le cadre de la réutilisation, un problème complémentaire à l'évolution et la customization
- applicable au niveau conceptuel mais aussi dynamique

Conclusions

Adaptation

- dans le cadre de la réutilisation, un problème complémentaire à l'évolution et la customization
- applicable au niveau conceptuel mais aussi dynamique

Résultats

- amélioration des approches existantes en conception
- approche outillée en quasi-totalité (applicabilité dynamique ?)

Perspectives

Général

- obtention des mappings
- adaptation *one-to-many* et avec données
- adaptation de niveau supérieur (sémantique, QoS)

Perspectives

Général

- obtention des mappings
- adaptation *one-to-many* et avec données
- adaptation de niveau supérieur (sémantique, QoS)

Implantation

- choix d'un support
- obtention des interfaces et implantation
- validation dans le cadre d'une approche dynamique

Pascal.Poizat@ibisc.univ-evry.fr