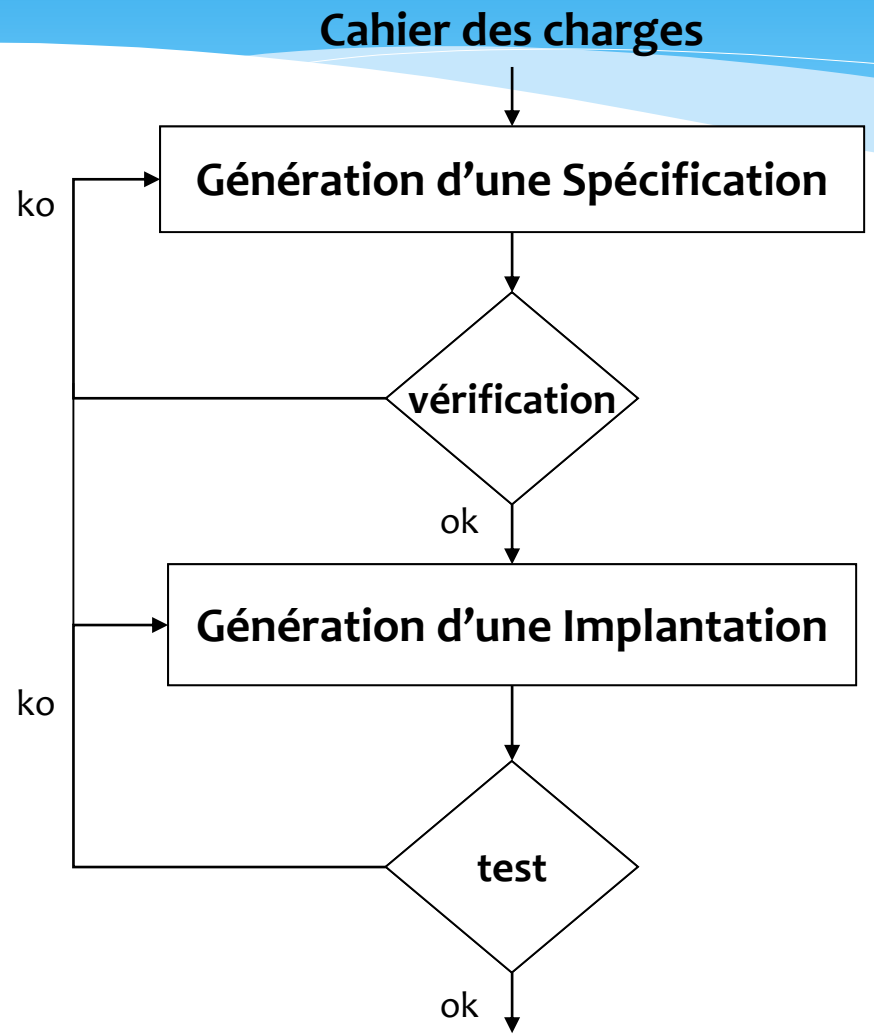
Modélisation/Test de services web

Et compositions dans des PaaS

Cycle de vie du logiciel peut s'écrire



Phase de conception

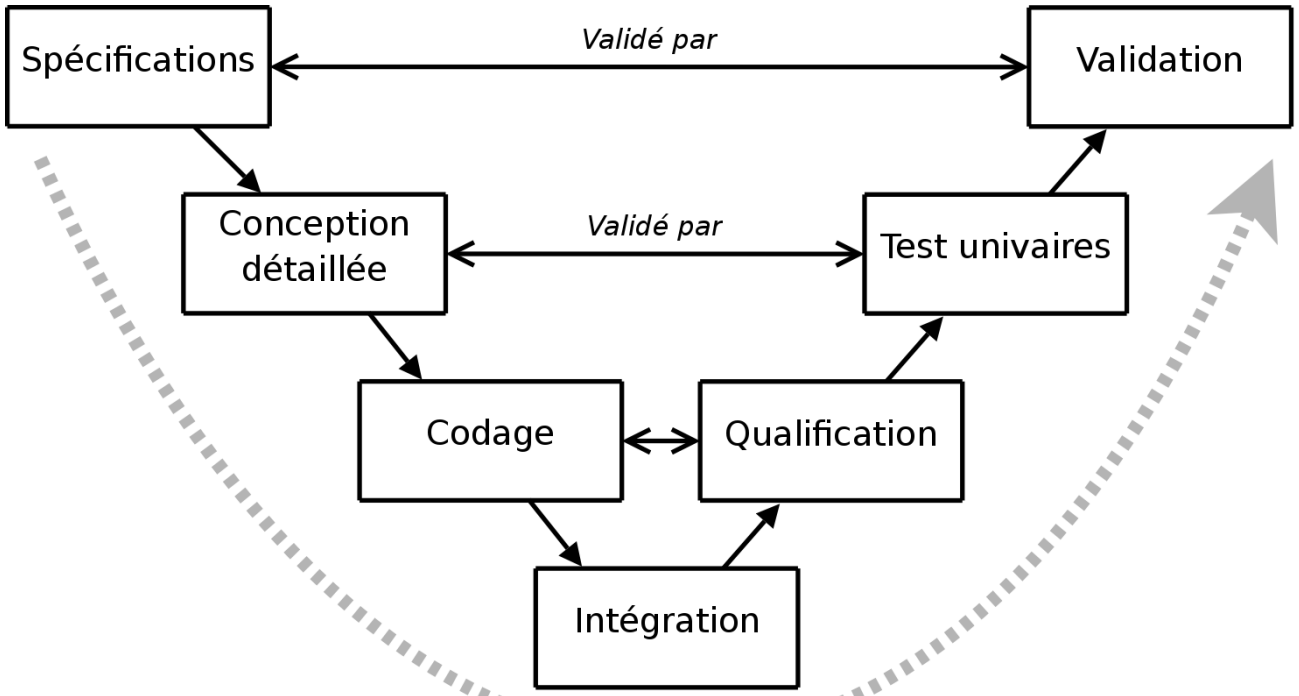
Phase de validation de la spécification

Phase d'implantation

Phase de validation de l'implantation

… Ou encore

Cycle en V



… Ou encore

Méthodes agiles

correspondance continue entre les dev. et le client

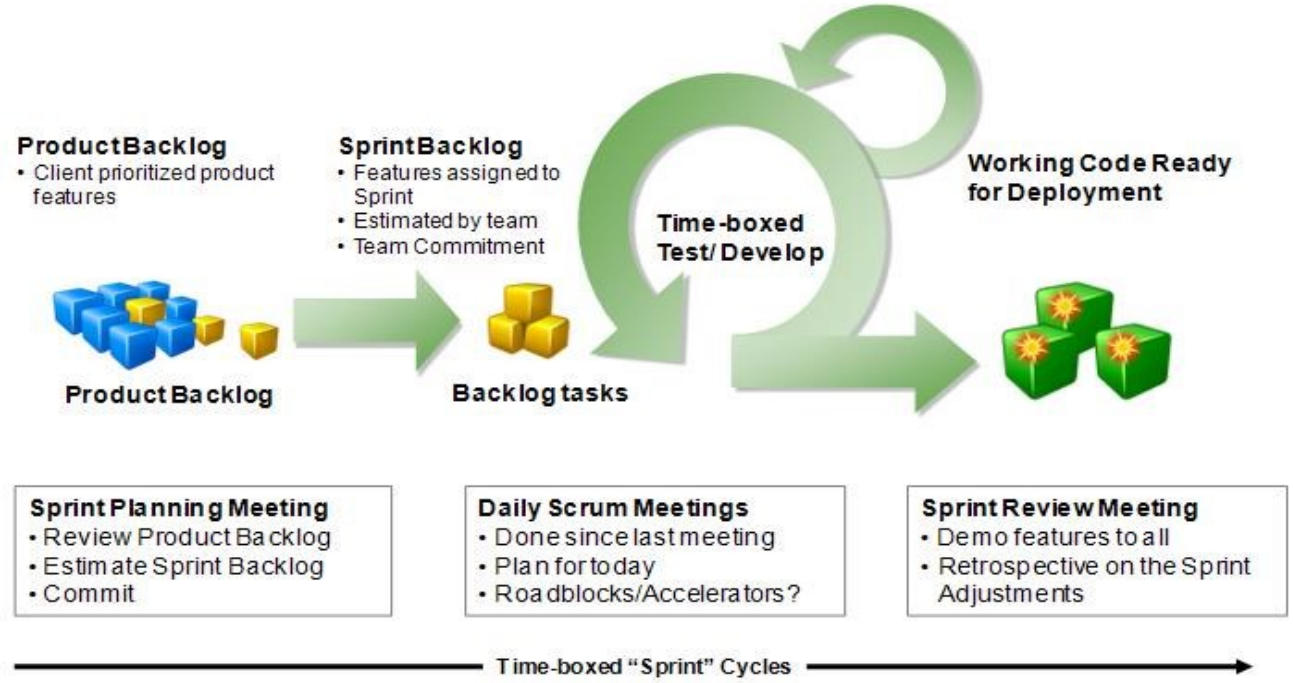
Ex de méthodes:

- XP (extreme programming) : cycles rapides de développement , recherche de scenarios, découpage en taches, attribution des taches, phases de tests et on recommence tant que le client a des scénarios à demander

… Ou encore

Méthodes agiles

SCRUM (transparence, inspection, adaptation):



Description des spécifications

* Du cahier des charges => on en déduit une spécification détaillée (avec pl. étapes)

* Plusieurs modèles de spécifications:

  * Automates, automates temporisés
  * Réseau de petri
  * Lds, lotos et les dérivés stockastiques et temporisés
  * UML, SYSML et les dérivés, merise,…
  * WSBPEL, (services)

Description des spécifications

* Modèles orientés comportement
  * automates finis
  * réseaux de Petri
  * UML statecharts, UML diagrammes d'activité
* Modèles de type oracles (pre/post conditions)
  * méthode B
  * OCL pour UML
  * JML
* Propriétés qui posent pb (pour construction de cas de test ou autre)
  * non déterminisme,
  * variables continues (temps, ou autre)
  * structures de données dynamiques
  * concurrence

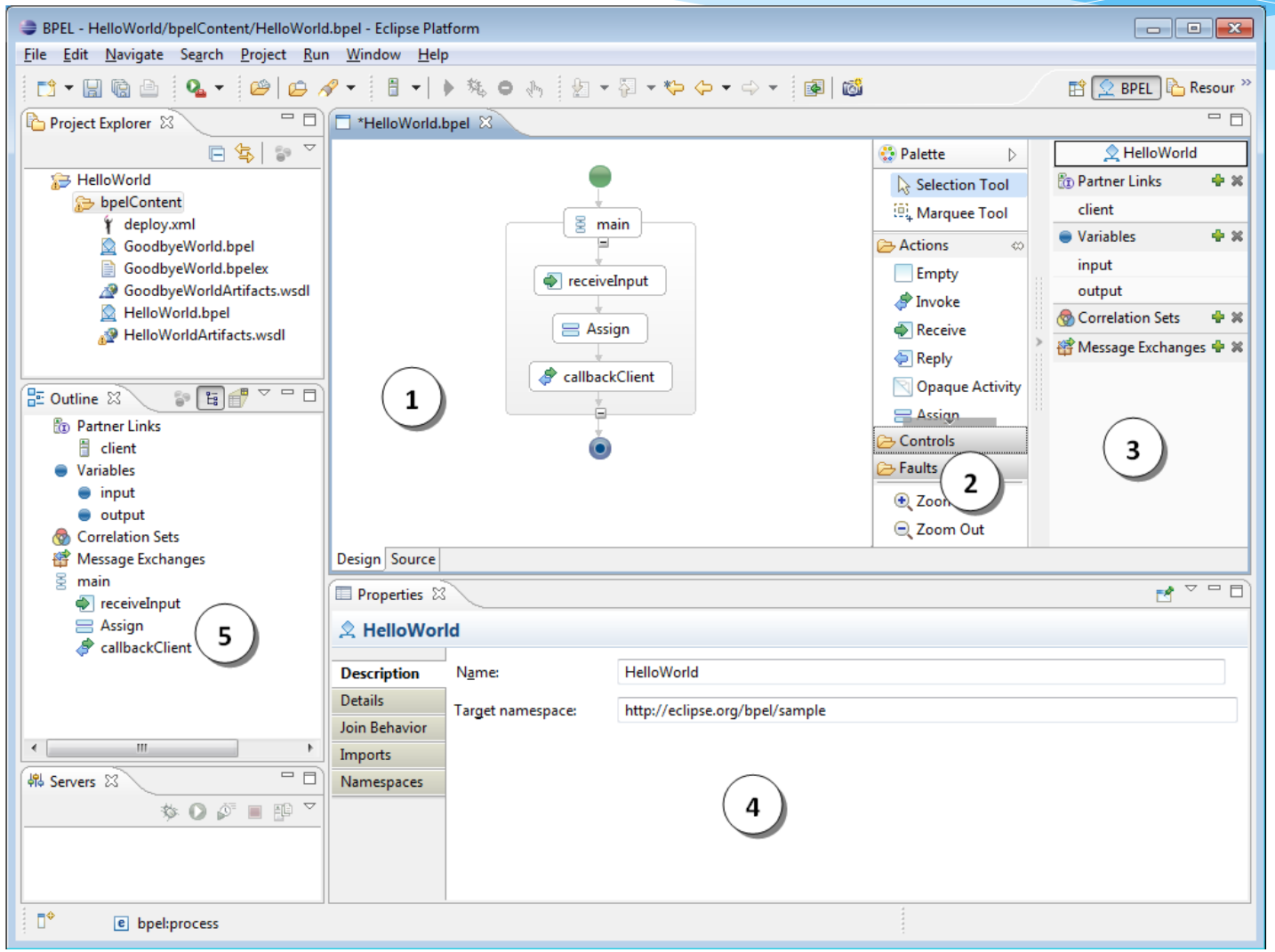
Modélisation de Services

WS-BPEL

- Definition des partenaires
- Utilisation de variables, assignation de valeurs (assign)
- Activités basiques (invoque, receive, reply, wait, throw)
- Activités structurés (while, switch, sequence,pick(temporisation)
- Correlation = session
- Scope découpage d'un processus en plusieurs parties
    - Pl. handler possibles par scope (conpensation, fault, event)

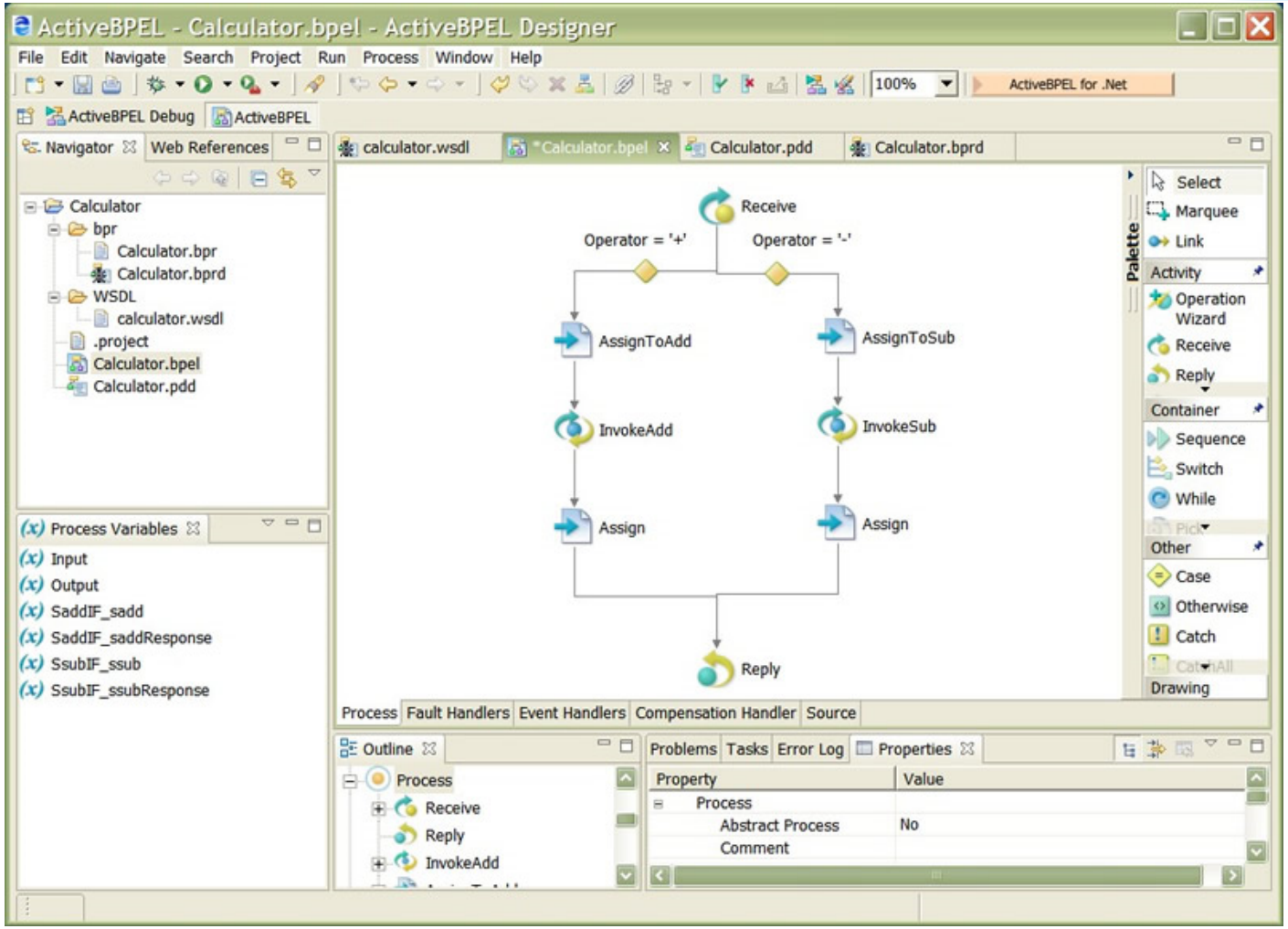
WS-BPEL

Avec Eclipse:



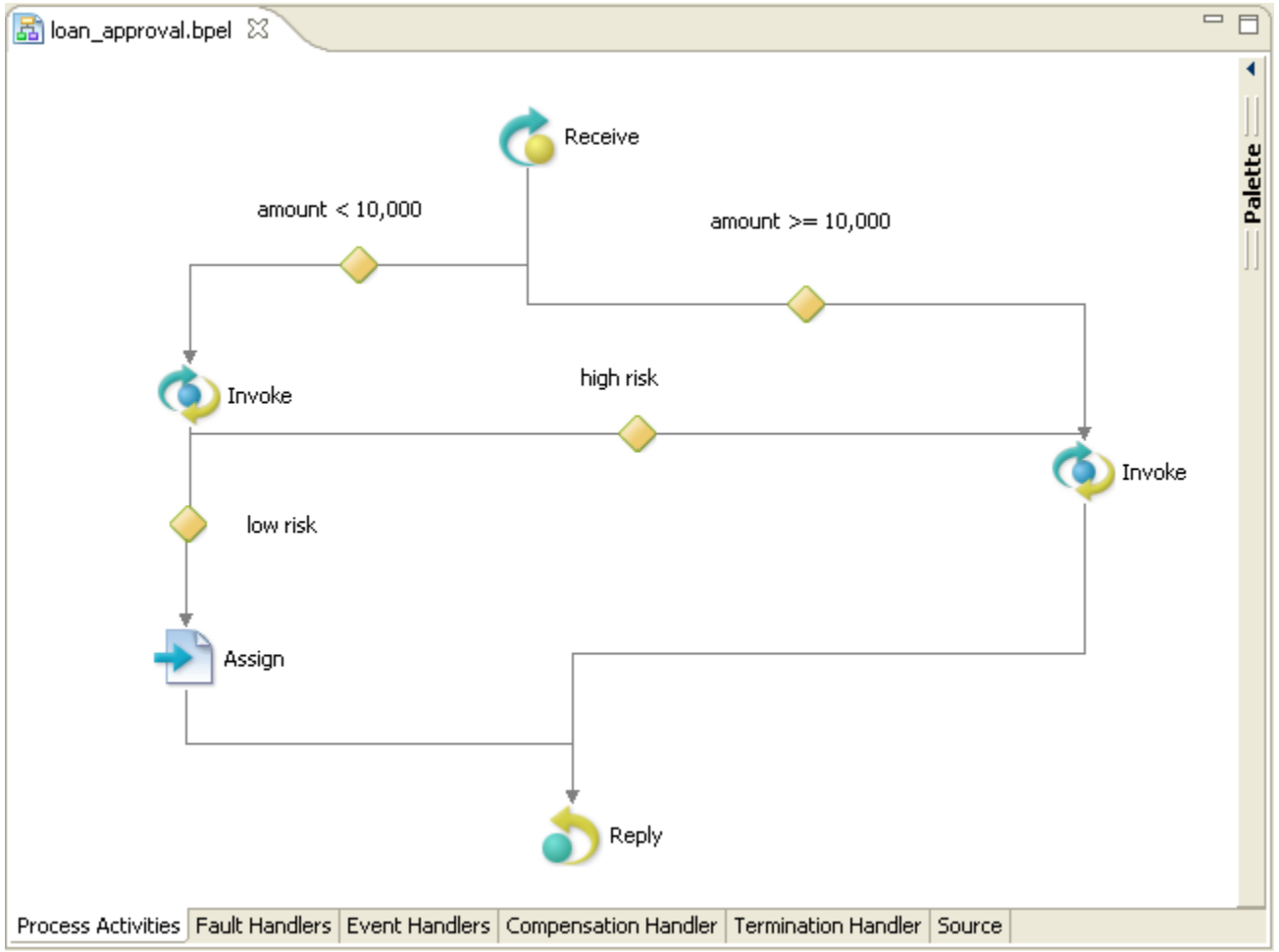
WS-BPEL

Avec ActiveBPEL



WS-BPEL

Avec ActiveBPEL



WS-BPEL

En XML :

```
<bpel:receive createInstance="yes" operation="request"
                        partnerLink="customer" portType="lns:loanServicePT"
                        variable="request">
                        <bpel:sources>
                                <bpel:source linkName="receive-to-assess">
                                        <bpel:transitionCondition>
                                                ($request.amount &lt; 10000)
                                        </bpel:transitionCondition>
                                </bpel:source>
                                <bpel:source linkName="receive-to-approval">
                                        <bpel:transitionCondition>
                                                ($request.amount &gt;= 10000)
                                        </bpel:transitionCondition>
                                </bpel:source>
                        </bpel:sources>
                </bpel:receive>
```

WS-BPEL

En XML :

```
<bpel:invoke inputVariable="request" operation="check"
        outputVariable="risk" partnerLink="assessor"
        portType="lns:riskAssessmentPT">
        <bpel:targets>
                <bpel:target linkName="receive-to-assess" />
        </bpel:targets>
        <bpel:sources>
                <bpel:source linkName="assess-to-setMessage">
                        <bpel:transitionCondition>
                                ($risk.level = 'low')
                        </bpel:transitionCondition>
                </bpel:source>
                <bpel:source linkName="assess-to-approval">
                        <bpel:transitionCondition>
                                ($risk.level != 'low')
                        </bpel:transitionCondition>
                </bpel:source>
        </bpel:sources>
</bpel:invoke>
```

WS-BPEL

pl. moteurs

- Websphere,bpel process manager,biztalk, bpelmaestro
- Activebpel,pxe, twister

BPMN

* BPM (Business Process Management) est la discipline qui consiste à considérer la gestion des processus comme un moyen d'améliorer la performance opérationnelle. BPMN (Business Process Model and Notation)
* BPMN pour informatique
* BPMN v2 = BPMN 1 + WS-BPEL

* **outil de collaboration entre le métier et l'IT**
* modèle graphique pourra être transformé facilement en une application permettant d'« exécuter » le processus.

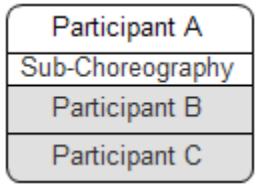
BPMN

* version 2.0: adoption du format XML et la capacité à rendre le modèle exécutable.

Éléments de workflow	Éléments d'organisation	Éléments de lisibilité	Comportements spécifiques
Activities (activités) Events (événements) Gateways (porte logique) Sequence flow (flux séquentiel)	Pools (piscines) Swimlanes (lignes d'eau) Groups (groupes)	Annotation (annotations) Links (liens)	Messages / message flow (messages / flux de messages) Signals (signaux) Timers (minuterie) Errors (erreurs) Repeating (boucles) Correlation (corrélation)

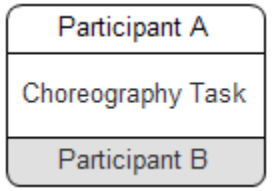
BPMN

Activité de démarrage



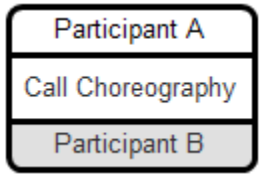
Sub-choreography contains a refined choreography with several interactions.

Activité de fin



Choreography task represents an Interaction (message exchange) between two participants.

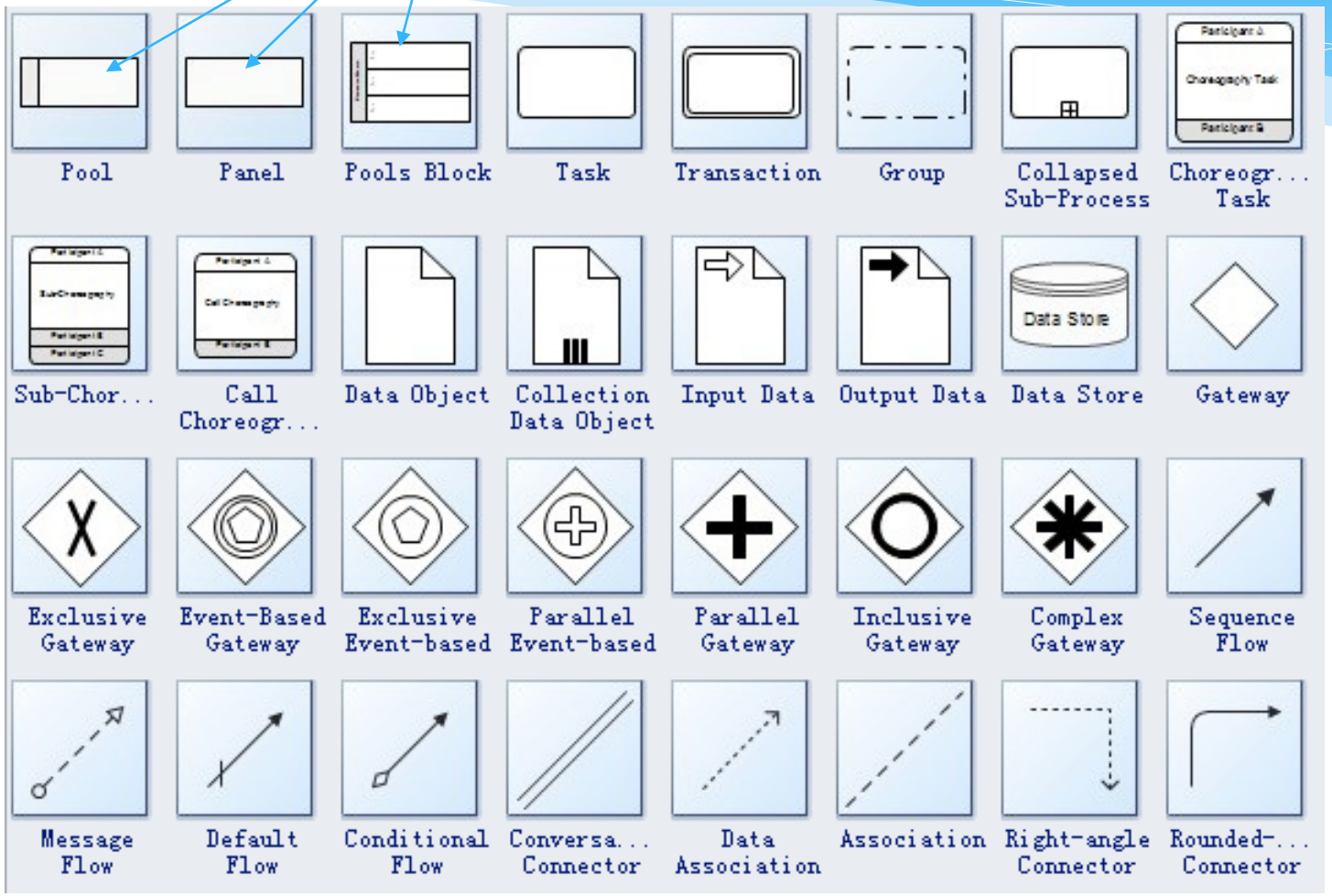
Activité



Call choreography is a wrapper for a globally defined choreography task or sub-choreography..

BPMN

Blocs d'activités



BPMN

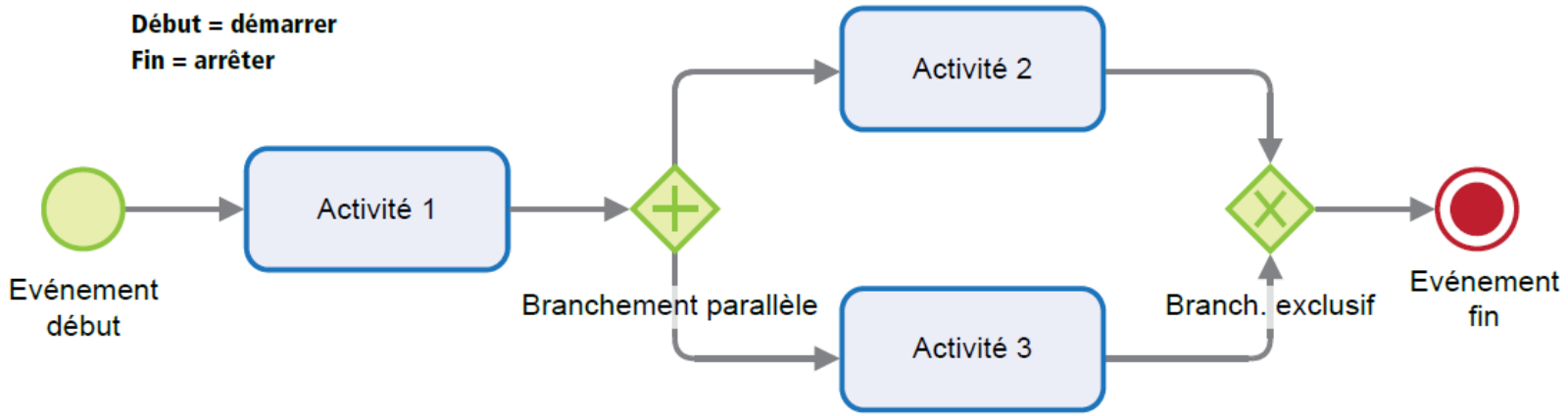
AND (aussi appelés parallèle).
Tous les flux entrants doivent avoir été reçus (quel que soit l'ordre) pour que le processus continue.
Tous les flux sortants sont actifs et le processus s'exécute en parallèle

XOR (aussi appelés exclusif).
Un seul flux entrant est nécessaire.
Un seul flux sortant peut être suivi et une condition est nécessaire pour déterminer quel flux doit être suivi.

Séquence entre activité, séquence peuvent avoir des conditions,
Séquence avec / -> séquence par défaut

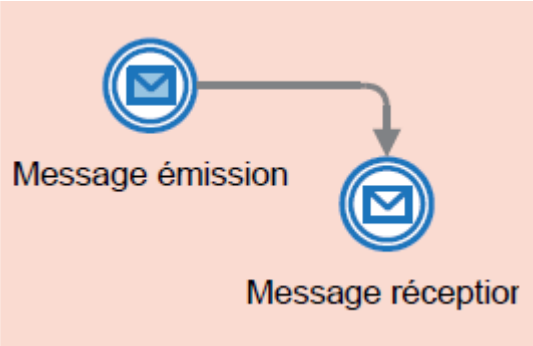
BPMN

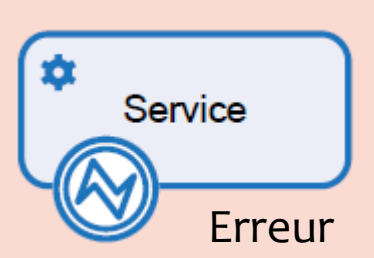
* Exemple : Que modélise cette spécification ?



BPMN

Exemples d’éléments intermédiaires :





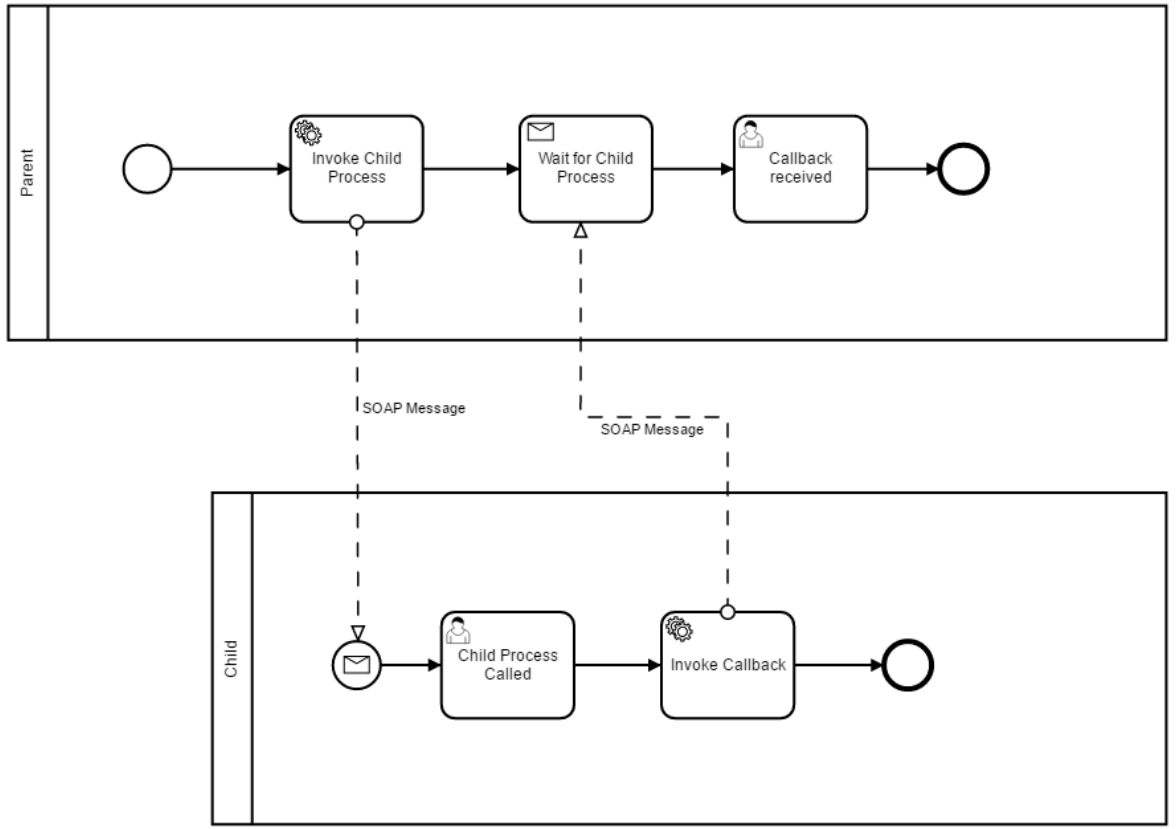
Une tâche humaine doit être réalisée par une personne humaine.

Une activité de service est une activité automatisée.

Une activité appelante représente l'appel à un sous-processus.

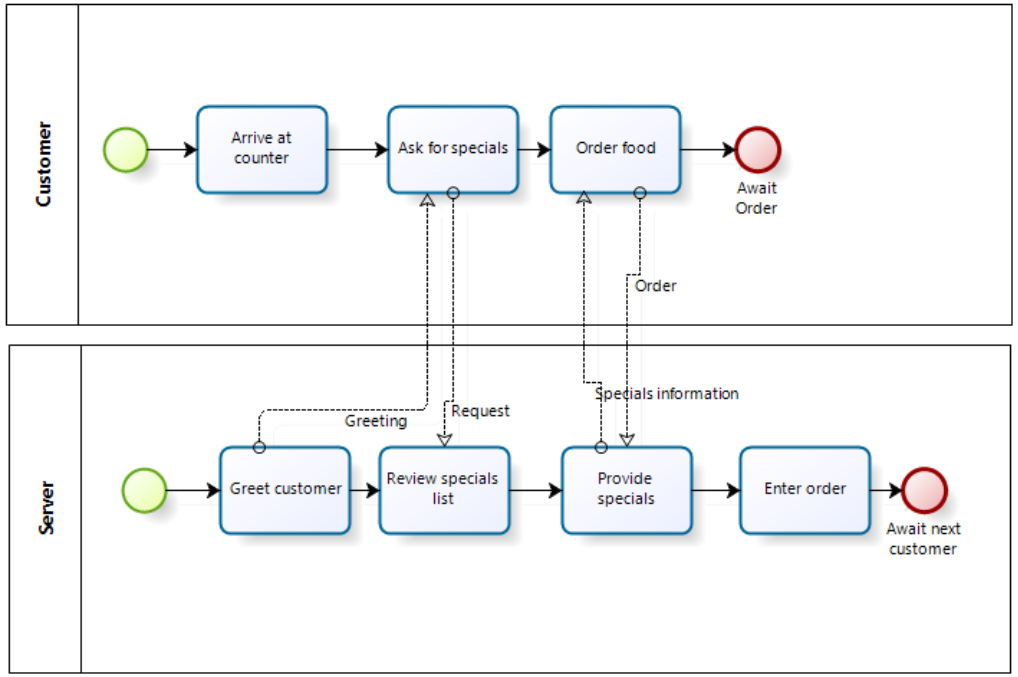
BPMN

Exemple : Que modélise cette spécification ?



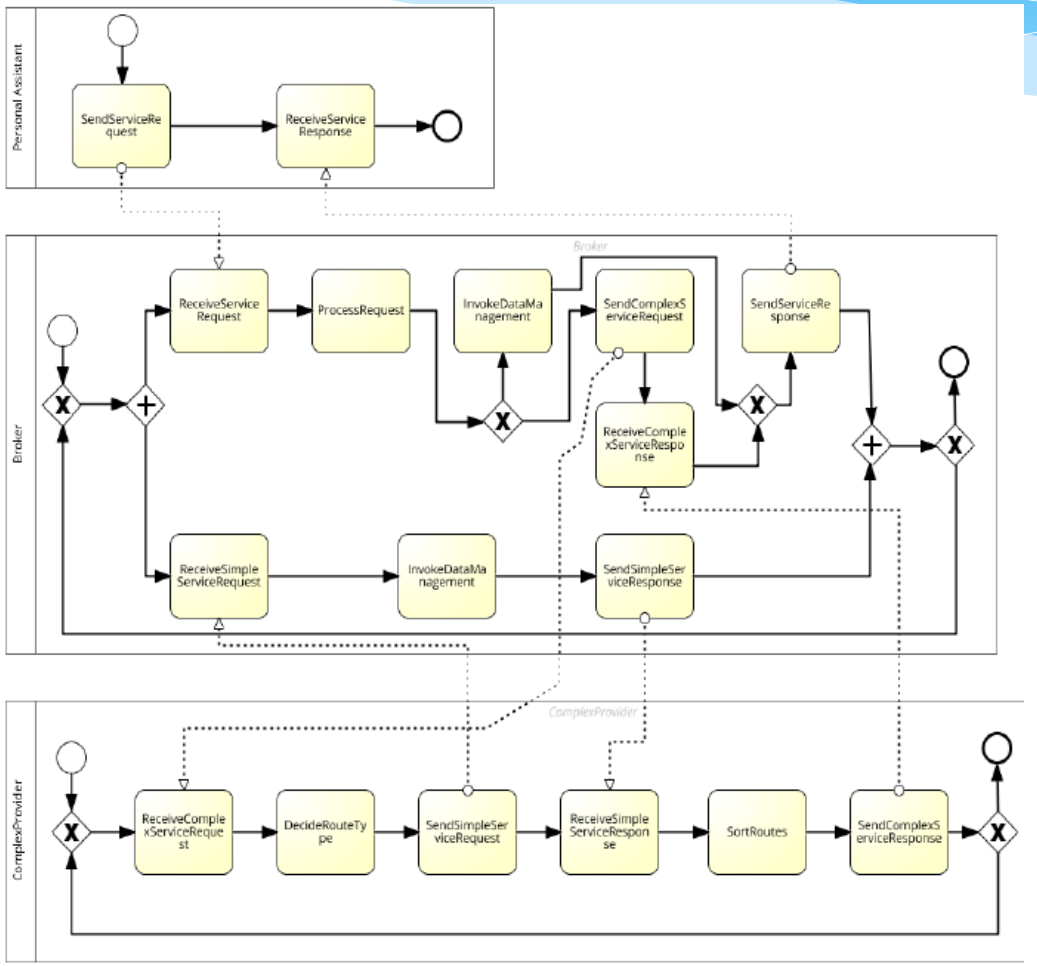
BPMN

Exemple : Que modélise cette spécification ?



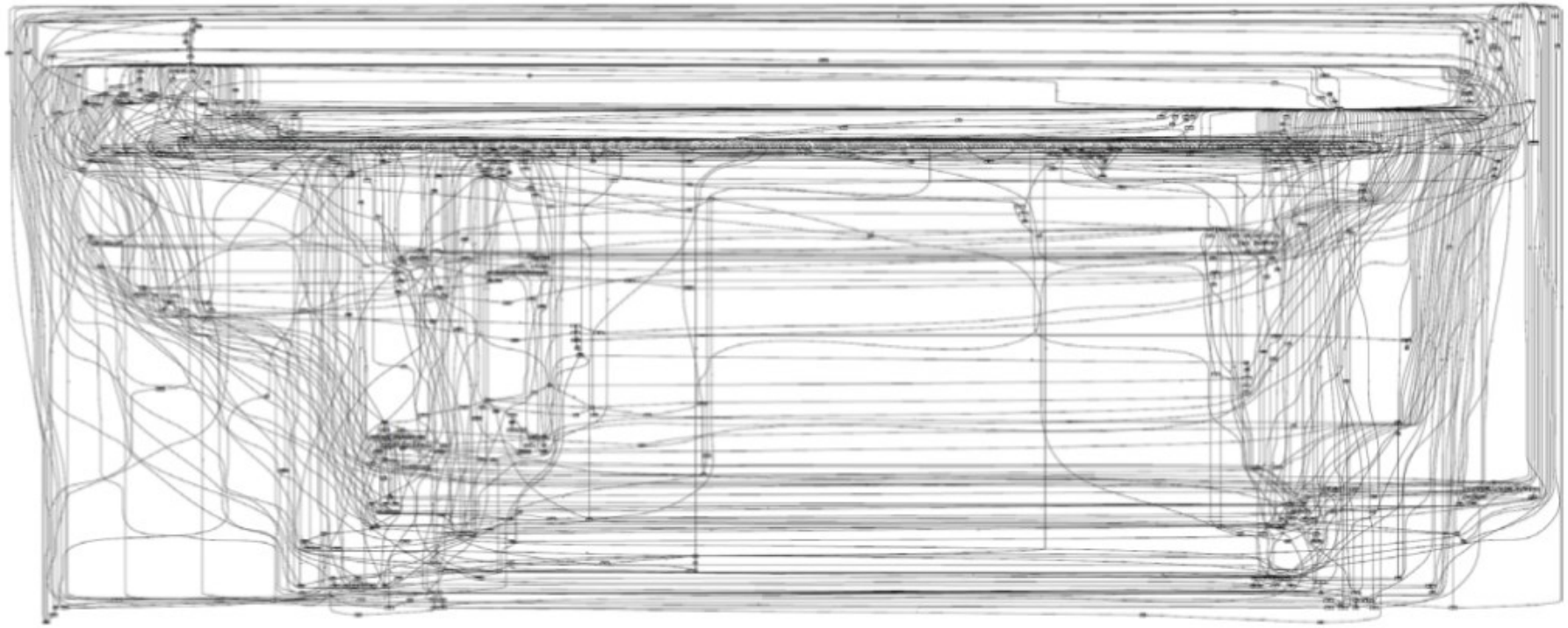
BPMN

Autre exemple plus complexe :



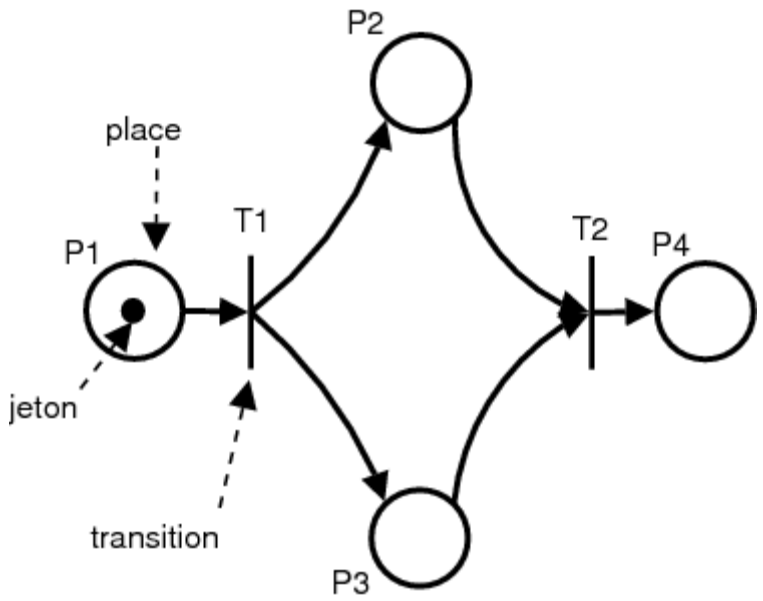
BPMN

Spaghetti problem : (inférence de modèles BPMN depuis des traces)



Les RdP (réseaux de Petri)

* Un réseau de Pétri est un graphe représentant les relations entre trois ensembles d'éléments :
  * places
  * transitions
  * arcs



Modèle LTS

Définition 1.3.2 *Système de Transitions Étiquetées (LTS)*

Un système de transi tiquetées (LTS pour Labeled Transition System) A est un quadruplet $< Q, \Sigma, q_0, \rightarrow >$ *où :*

Q *est un ensemble fini non vide d'états,*

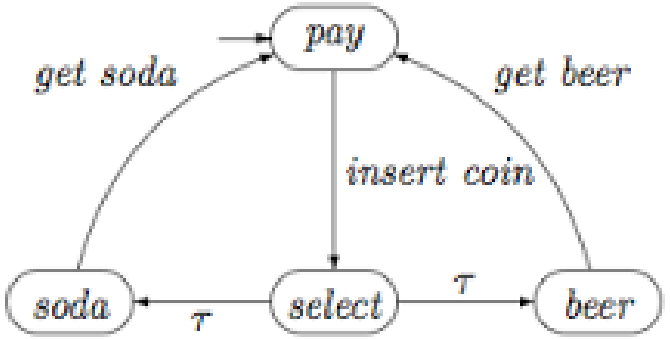
Σ *est un ensemble fini non vide d'interactions (ou d'actions),* τ *est une action particulière, non visible représentant une action interne,*

q_0 *est l'état initial du système de transitions étiquetées.*

$\rightarrow \subseteq Q \times \Sigma \times Q$ *est la relation de transition.* $< q, a, q' >$ *représente une transition qui part de l'état* q *et va vers l'état* q'*. Une telle transition est communément notée par* $q \xrightarrow{a} q'$*.*

Soit le LTS $A =< Q, \Sigma, q_0, \rightarrow >$. A est déterministe ssi :
$\forall q \in Q, \forall t = q \xrightarrow{a} q' \in \rightarrow$ avec $a \neq \tau \in \Sigma, q' \in Q, \forall t' = q \xrightarrow{b} q'' \in \Sigma,$ avec $b \neq \tau \in \Sigma, q'' \in Q$, $a \neq b$ ou $q' = q''$.

Modèle LTS



Modèle IOSTS

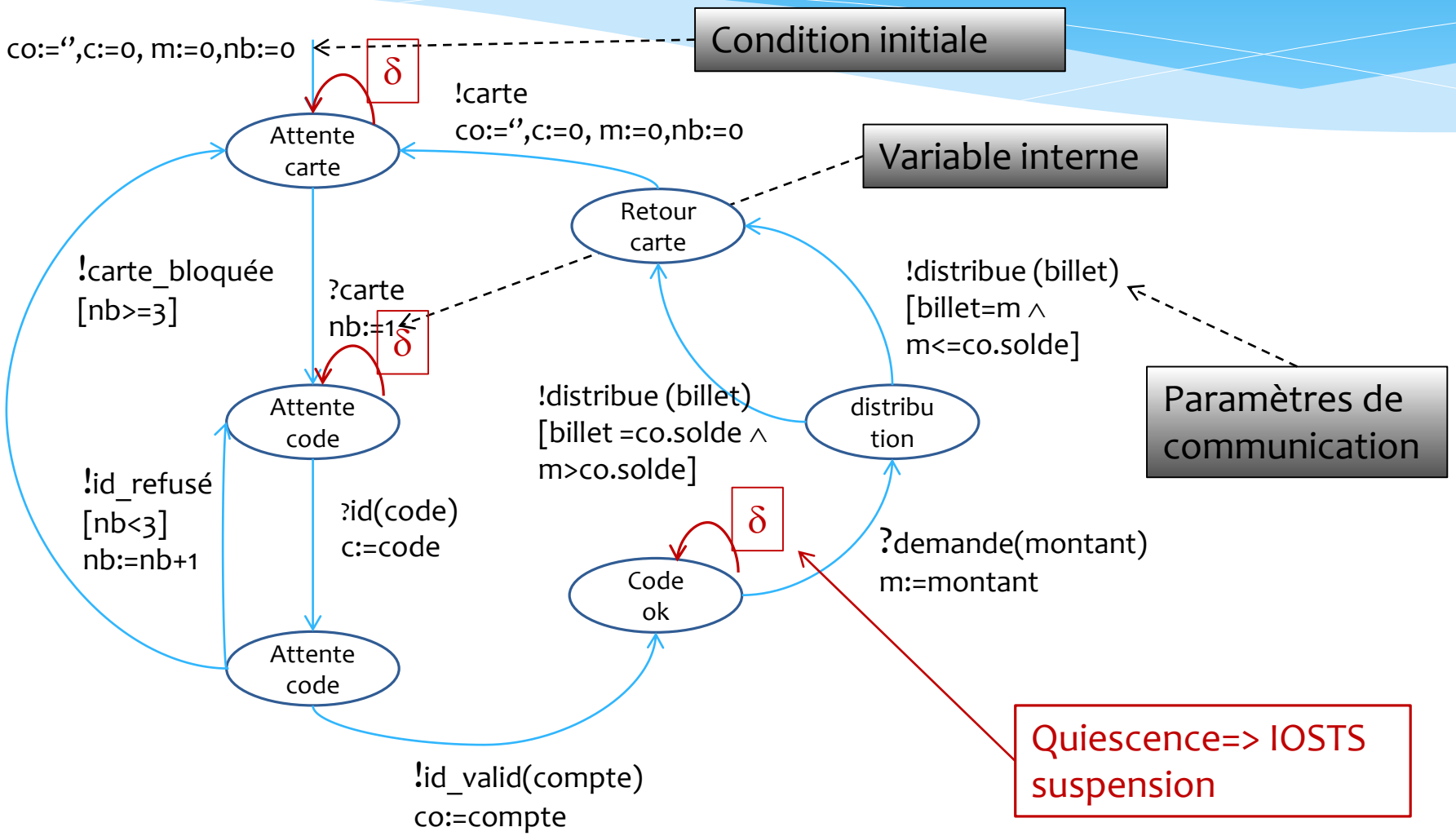
Définition 1.3.3 *Un ne symbolique à transitions STS est un tuple* $< L, l_0, V, V_0, I, \Lambda, \rightarrow >$, *où :*

- L *est l'ensemble fini des états symboliques, avec* l_0 *l'état symbolique initial,*
- V *est l'ensemble fini des variables internes, tandis que* I *est l'ensemble fini des variables externes ou d'interactions. Nous notons* D_v *le domaine dans lequel une variable* v *prend des valeurs. Les variables internes sont initialisées par l'affectation* V_0, *qui est supposée prendre une valeur unique dans* D_V,
- Λ *est l'ensemble fini des actions (symboles), partitionné par* $\Lambda = \Lambda^I U \Lambda^O$ *: les entrées, commençant par ?, sont fournies au système, tandis que les sorties (commençant par !) sont observées depuis ce dernier,*
- $\rightarrow$ *est l'ensemble fini des transitions. Une transition* $(l_i, l_j, a(p), \varphi, \varrho)$, *depuis l'état symbolique* $l_i \in L$ *vers* $l_j \in L$, *aussi noté* $l_i \xrightarrow{a(p),\varphi,\varrho} l_j$ *est étiquetée par* $a(p) \in \Lambda \times \mathcal{P}(I)$, *avec* $a \in \Lambda$ *une action et* $p \subseteq I$ *un ensemble fini de variables externes* $p = (p_1, ..., p_k)$. *Nous notons* $type(p) = (t_1, ..., t_k)$ *le type de l'ensemble des variables* p. $\varphi \subseteq D_V \times D_p$ *est une garde qui restreint l'exécution d'une transition. Des variables internes sont mises à jour avec l'affectation* $\varrho : D_V \times D_p \rightarrow D_V$ *une fois que la transition est exécutée.*

IOSTS suspension

Une extension immédiate du modèle STS est le STS *suspension*, qui décrit en plus la quiescence des états i.e. l'absence d'observation à partir d'état symbolique. La quiescence est dénotée par un nouveau symbole $!\delta$ et un STS complété $\Delta(STS)$. Pour un STS $\mathcal{S}$, $\Delta(\mathcal{S})$ s'obtient en ajoutant une transition étiquetée par $!\delta$ bouclant pour chaque état symbolique à partir duquel la quiescence peut être observée. Les gardes de ces nouvelles transitions doivent retourner vrai pour chaque valeur de $D_{V \cup I}$ qui ne permet pas l'exécution d'une transition étiquetée par une sortie.

Modèle IOSTS

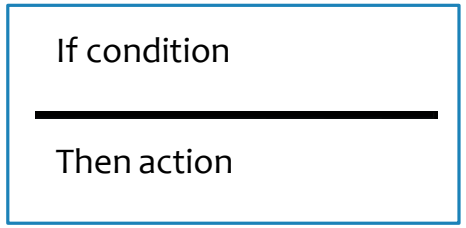


Modèle IOSTS

* LTS et ioSTS peuvent aussi être représentés par un algèbre de processus => possibilité de décrire des algorithmes et transformations par règles d'inférences [Tre96]
  * ?req1;!resp1 | ?req2; !resp2
* Avantage: transformations, modifications peuvent être données par des règles d'inférence

Modèle IOSTS

* Format générique règle



* IOSTS bien adaptés pour représenter des échanges entre SW par messages prenant des paramètres dans l'ensemble I

IOLTS sémantique d'un IOSTS

La sémantique d'un STS, décrivant chaque action valuée et chaque état, est définie par un LTS, nommé le LTS sémantique. Celui-ci correspond à un automate valué sans variable : les états du LTS sémantique sont étiquetés par les valeurs des variables internes, les transitions sont valuées par des valeurs de paramètres.

Définition 1.3.4 *La sémantique d'un STS* $\mathcal{S} =< L, l_0, V, V_0, I, \Lambda, \rightarrow >$ *est le LTS* $||\mathcal{S}|| =< Q, q_0, \sum, \rightarrow >$ *où :*

- $L = Q \times D_V$ *est l'ensemble fini des états,*
- $q_0 = (l_0, V_0)$ *est l'état initial,*
- $\sum = \{(a(p), \theta) \mid a(p) \in \Lambda, \theta \in D_p\}$ *est l'ensemble des symboles valués,*
- $\rightarrow$ *est la relation de transition* $Q \times \Sigma \times Q$ *obtenue par la règle suivante :*

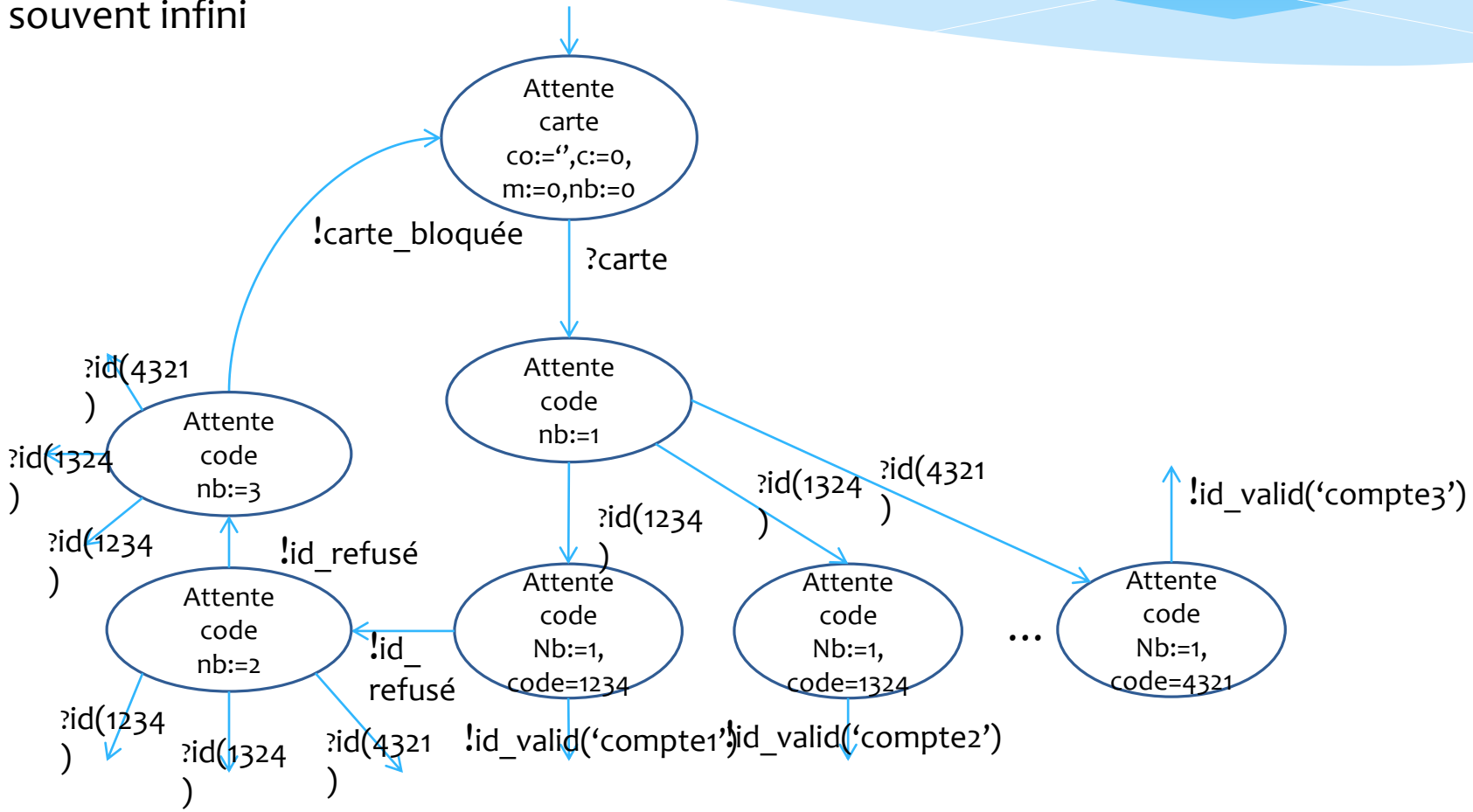
$$\frac{l_i \xrightarrow{a(p),\varphi,\varrho} l_j, \theta \in D_p, v \in D_V, v' \in D_V, \varphi(v,\theta)\ true, v' = \rho(v,\theta)}{(l_i, v) \xrightarrow{a(p),\theta} (l_j, v')}$$

IOLTS sémantique d'un IOSTS

Intuitivement, pour une transition de STS $l_i \xrightarrow{a(p),\varphi,\varrho} l_j$, nous obtenons une transition de LTS $(l_i, v) \xrightarrow{a(p),\theta} (l_j, v')$ avec v un ensemble de valeurs de variables internes, si il existe un tuple de valeurs de paramètre θ tel que la garde $\varphi(v, \theta)$ retourne vrai. Un fois que la transition est exécutée, les variables internes prennent la valeur v' déduite de l'affectation $\varrho(v, \theta)$. Un STS suspension $\Delta(\mathcal{S})$ est associé à son LTS suspension sémantique par $||\Delta(\mathcal{S})|| = \Delta(||\mathcal{S}||)$.

Modèle IOLTS

IOLTS sémantique souvent infini

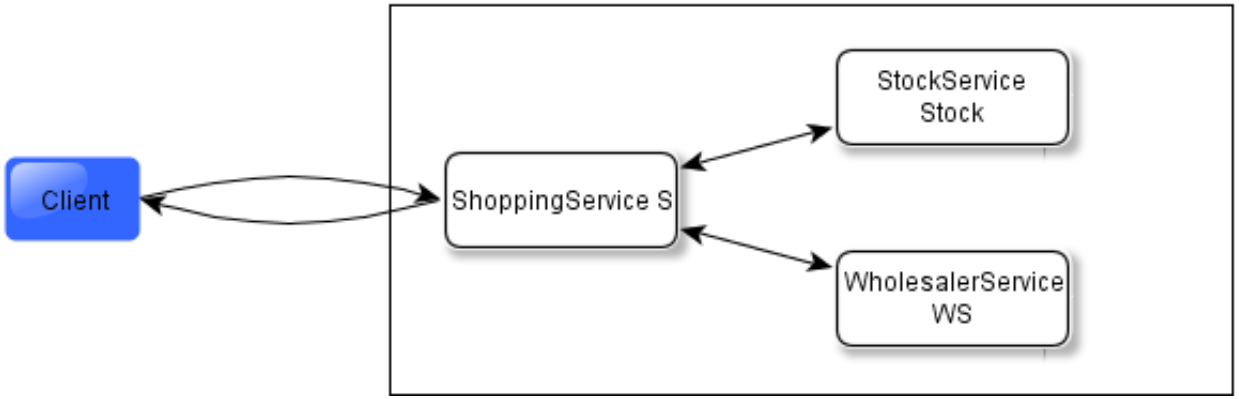


Modélisation de SW

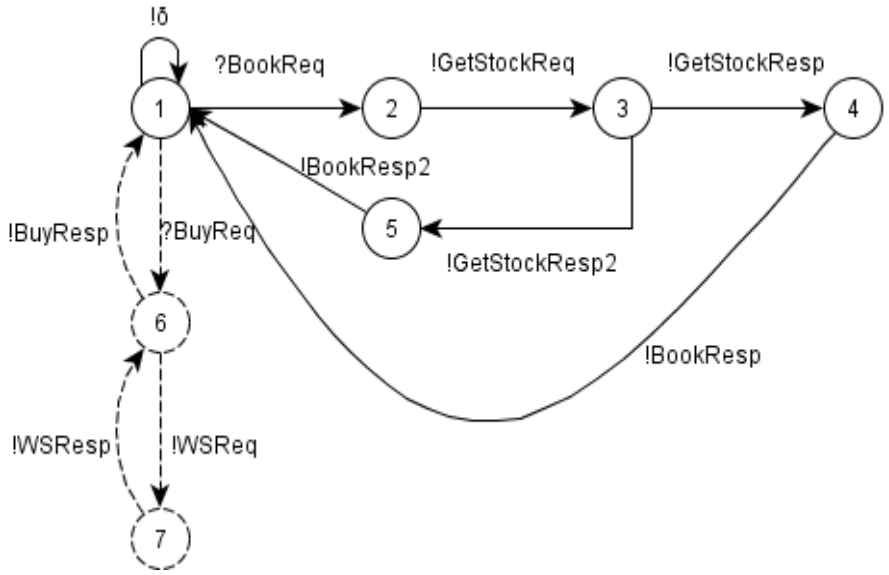
* Description d'un portType par un IOSTS
  * Opérations modélisées par actions
  * Paramètres d'opérations = paramètres de IOSTS
* Opération Oneway => action d'entrée
* Opération Request-Response => action d'entrée suivie par une action de sortie
* Opération de Notification => action de sortie
* Opération Solicit-Response = action de sortie suivie par une action d'entrée

Modélisation de composition de SW

* Description des partenaires, des corrélations via des variables spécifiques
* Exemple:

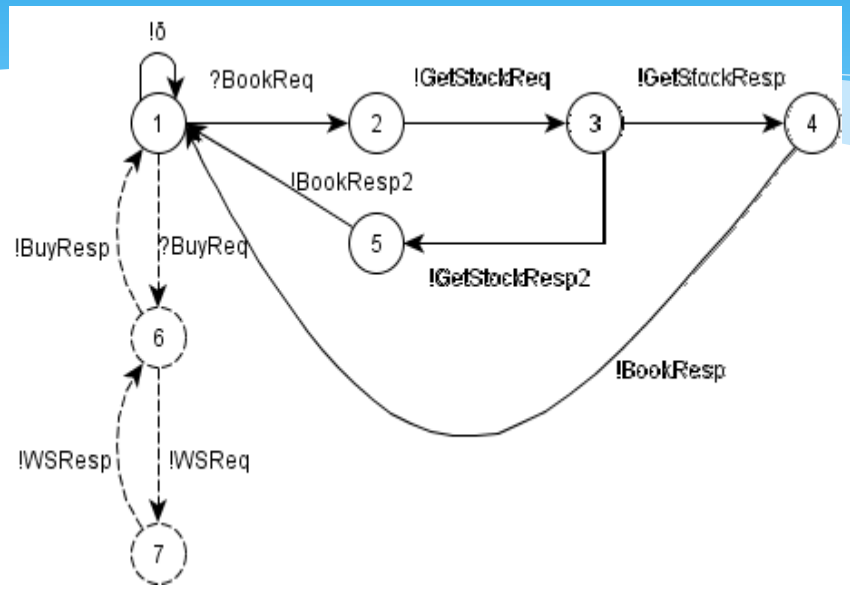


Modélisation de composition de SW



Symbol	Message	Guard	Update
?BookReq	?BookReq(account, isbn, from, to, corr)	[from="Client"∧ to="S" ∧ $corr = \{account\}$]	a:=account, c1:=corr
!GetStockReq	!GetStockReq(isbn, from, to, corr)	$G1$ =[from="S"∧to="Stock"i:=isbn, ∧ $corr = \{a, isbn\}$]	c2=corr
!GetStockResp	!GetStockResp(stock, from, to, corr)	G2=[from="Stock"∧to="S" ∧ valid(i) ∧corr=c2]	st:=stock
!GetStockResp2	!GetStockResp(resp, from, to, corr)	G3=[from="Stock"∧to= "S"∧¬ valid(i)∧resp= "invalid"∧corr=c2]	st:=null
!BookResp	!BookResp(resp, from, to, corr)	G4=[from="S"∧ to="Client"∧ resp="Book available"∧corr=c1]	
!BookResp2	!BookResp(resp, from, to, corr)	G5=[from="S"∧to= "Client"∧resp= "Book unavailable"∧corr=c1]	
?BuyReq	?BuyReq(isbn, quantity, from, to, corr)	[from="Client"∧ to="S"∧isbn=i∧ st≠ null ∧corr=c1]	q:=quantity
!BuyResp	!BuyResp(resp, from, to, corr)	G6=[from="S"∧ to="Client"∧resp="order ready"∧ q ≤ st ∧corr=c1]	st:=st-q
!WSReq	!WholesalerReq(key, isbn, quantity, from, to, corr)	G7=[from="S"∧to="WS" ∧ key="1234"∧ isbn=i∧ q>st ∧ quantity=5∧ corr={key,a,isbn}]	c3:=corr
!WSResp	!WholesalerResp(resp, from, to, corr)	G8=[from="WS"∧to="S" ∧ resp="wholesale order sent" ∧corr=c3]	st:=st+5

Modélisation de composition de SW



Symbol	Message	Guard	Update
?BookReq	?BookReq(account, isbn, from, to, corr)	[from="Client"∧ to="S" ∧ $corr = \{account\}$]	a:=account, c1:=corr
!GetStockReq	!GetStockReq(isbn, from, to, corr)	$G1$ =[from="S"∧to="Stock"i:=isbn, ∧ $corr = \{a, isbn\}$]	c2=corr
!GetStockResp	!GetStockResp(stock, from, to, corr)	G2=[from="Stock"∧to="S" ∧ valid(i) ∧corr=c2]	st:=stock
!GetStockResp2	!GetStockResp(resp, from, to, corr)	G3=[from="Stock"∧to= "S"∧¬ valid(i)∧resp= "invalid"∧corr=c2]	st:=null
!BookResp	!BookResp(resp, from, to, corr)	G4=[from="S"∧ to="Client"∧ resp="Book available"∧corr=c1]	
!BookResp2	!BookResp(resp, from, to, corr)	G5=[from="S"∧to= "Client"∧resp= "Book unavailable"∧corr=c1]	

Appel concurrent à la composition par plusieurs clients : comment un service peux faire appel à la bonne instance d'un service?

Besoin d 'éléments de corrélations = ensemble de variables qui identifient de façon unique une instance de services par un ensemble d'instance

Corrélation la plus simple = identification de session

Test de services

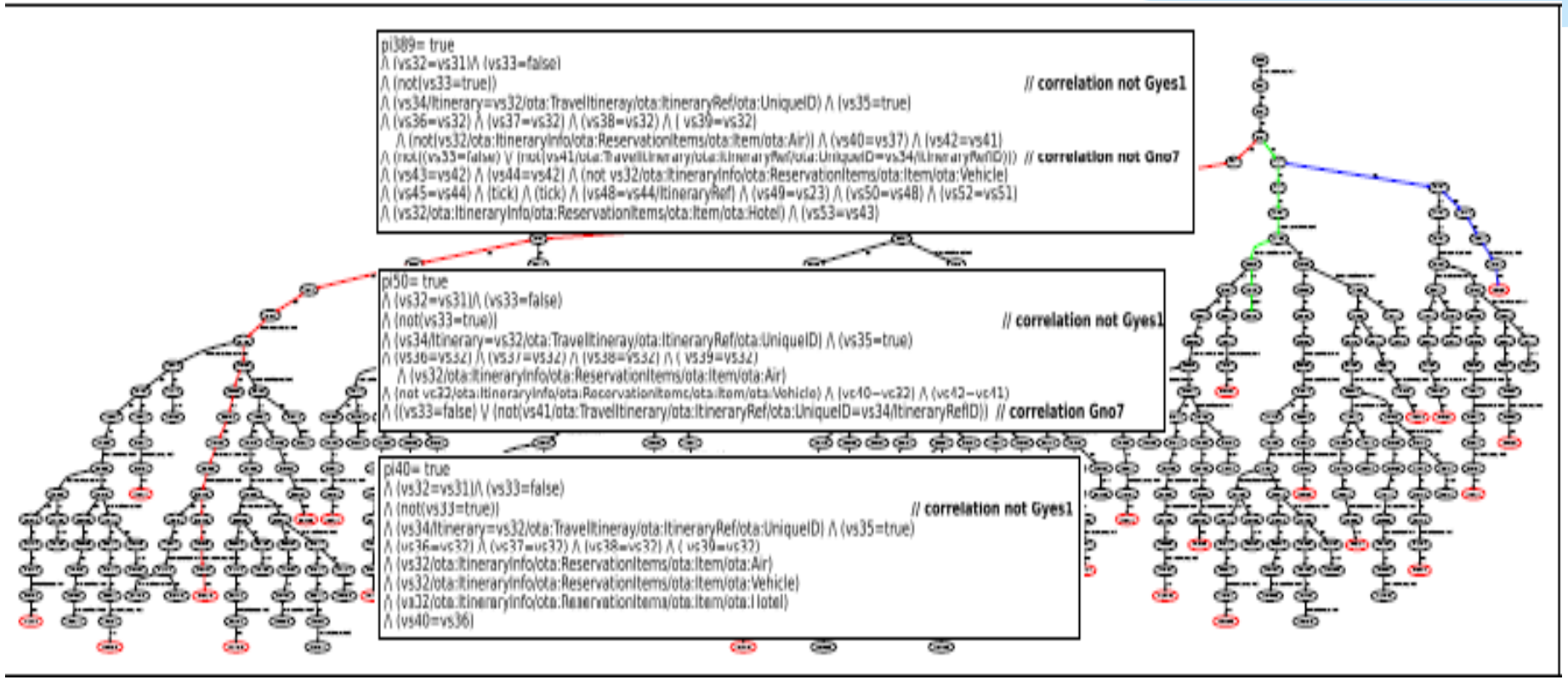
Définition du test

* According to the classic definition of Myers "*Software Testing is the process of executing a program or system with the intent of finding errors.*"
*
* According to the definition given by Hetzel, "*Testing involves any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results*"

Les types de test

* **Phase de Validation de l'implantation :**
  * aussi appelée la phase de Test,
  * permet de vérifier que l'implantation satisfait un certain nombre de propriétés de la spécification. Elle permet donc la détection de défectuosités du fonctionnement de l'implantation du système.
* Différents types de test : conformité, robustesse, performance, interopérabilité, …

Cas de test


pi389= true
/\ (vs32=vs31)/\ (vs33=false)
/\ (not(vs33=true))
// correlation not Gyes1
/\ (vs34/Itinerary=vs32/ota:TravelItineray/ota:ItineraryRef/ota:UniqueID) /\ (vs35=true)
/\ (vs36=vs32) /\ (vs37=vs32) /\ (vs38=vs32) /\ (vs39=vs32)
/\ (not(vs32/ota:ItineraryInfo/ota:ReservationItems/ota:Item/ota:Air)) /\ (vs40=vs37) /\ (vs42=vs41)
// correlation not Gno7
/\ (vs43=vs42) /\ (vs44=vs42) /\ (not vs32/ota:itineraryInfo/ota:ReservationItems/ota:Item/ota:Vehicle)
/\ (vs45=vs44) /\ (tick) /\ (tick) /\ (vs48=vs44/ItineraryRef) /\ (vs49=vs23) /\ (vs50=vs48) /\ (vs52=vs51)
/\ (vs32/ota:ItineraryInfo/ota:ReservationItems/ota:Item/ota:Hotel) /\ (vs53=vs43)
pi50= true
/\ (vs32=vs31)/\ (vs33=false)
/\ (not(vs33=true))
// correlation not Gyes1
/\ (vs34/Itinerary=vs32/ota:TravelItineray/ota:ItineraryRef/ota:UniqueID) /\ (vs35=true)
/\ (vs32/ota:ItineraryInfo/ota:ReservationItems/ota:Item/ota:Air)
/\ (not vc22/ota:itineraryInfo/ota:ReservationItems/ota:Item/ota:Vehicle) /\ (vc40=vc22) /\ (vc42=vc41)
/\ ((vs33=false) \/ (not(vs41/ota:TravelItinerary/ota:ItineraryRef/ota:UniqueID=vs34/ItineraryRefID)) // correlation Gno7
pi40= true
/\ (vs32=vs31)/\ (vs33=false)
/\ (not(vs33=true))
// correlation not Gyes1
/\ (vs34/Itinerary=vs32/ota:TravelItineray/ota:ItineraryRef/ota:UniqueID) /\ (vs35=true)
/\ (vs32/ota:ItineraryInfo/ota:ReservationItems/ota:Item/ota:Air)
/\ (vs32/ota:itineraryInfo/ota:ReservationItems/ota:Item/ota:Vehicle)
/\ (vs32/ota:itineraryInfo/ota:ReservationItems/ota:Item/ota:Hotel)
/\ (vs40=vs36)

Types de test

* Tests dits de granularité
* Ceux qui sont emplyés en entreprise

* Test unitaire
* Test d'intégration
* Test du système
* Test d'acceptation du client
* Test de non-régression
* Test de sécurité

Exemples de types de test

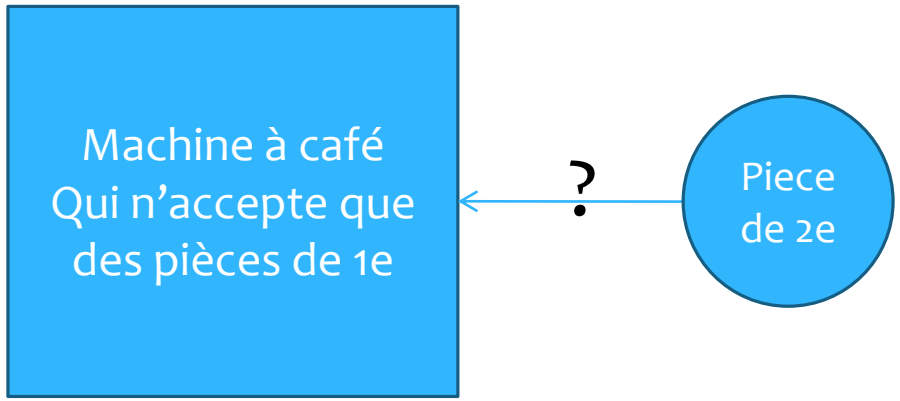
* Tests de l'usager
  * =beta-tests
* Tests d'Interopérabilité
  * Ces tests vérifient si le système développé interagit d'une façon correcte avec d'autres systèmes extérieurs en observant les fonctionnements des différents systèmes et des communications engendrées. Ces tests permettent de vérifier un service plus global fourni aux utilisateurs.

Exemples de types de test

* Tests de Robustesse
  * Ces tests consistent à vérifier la réaction d'un système dans des conditions d'utilisations extrêmes ou bien son exécution dans un environnement dit hostile.
  * Ces conditions ne sont généralement pas prévues dans la spécification, cette dernière référençant des conditions de fonctionnement normales.
  * Ces tests permettent ainsi de vérifier si d'autres erreurs existent telles que des fraudes ou des erreurs d'utilisation du système.
  * En java: Jcrasher

Exemples de types de test

* Tests de Robustesse
* Exemple:



Exemples de types de test

* Test de conformité
  * Cas de test générés à partir de la spécification pour vérifier si le comportement de l'implantation est conforme à celui de la spécification
  * Test en boite noire ou grise
  * Pas d'équivalence: "on peut détecter la présence de fautes pas leur absences"
  * Relations d'implantatations: définition de la relation de conformité
  * Plusieurs relations plus ou moins fortes

Test basé modèle

* Motivation: automatisation de la synthèse des tests depuis une spécification formelle, formalisation de relations de test, etc.

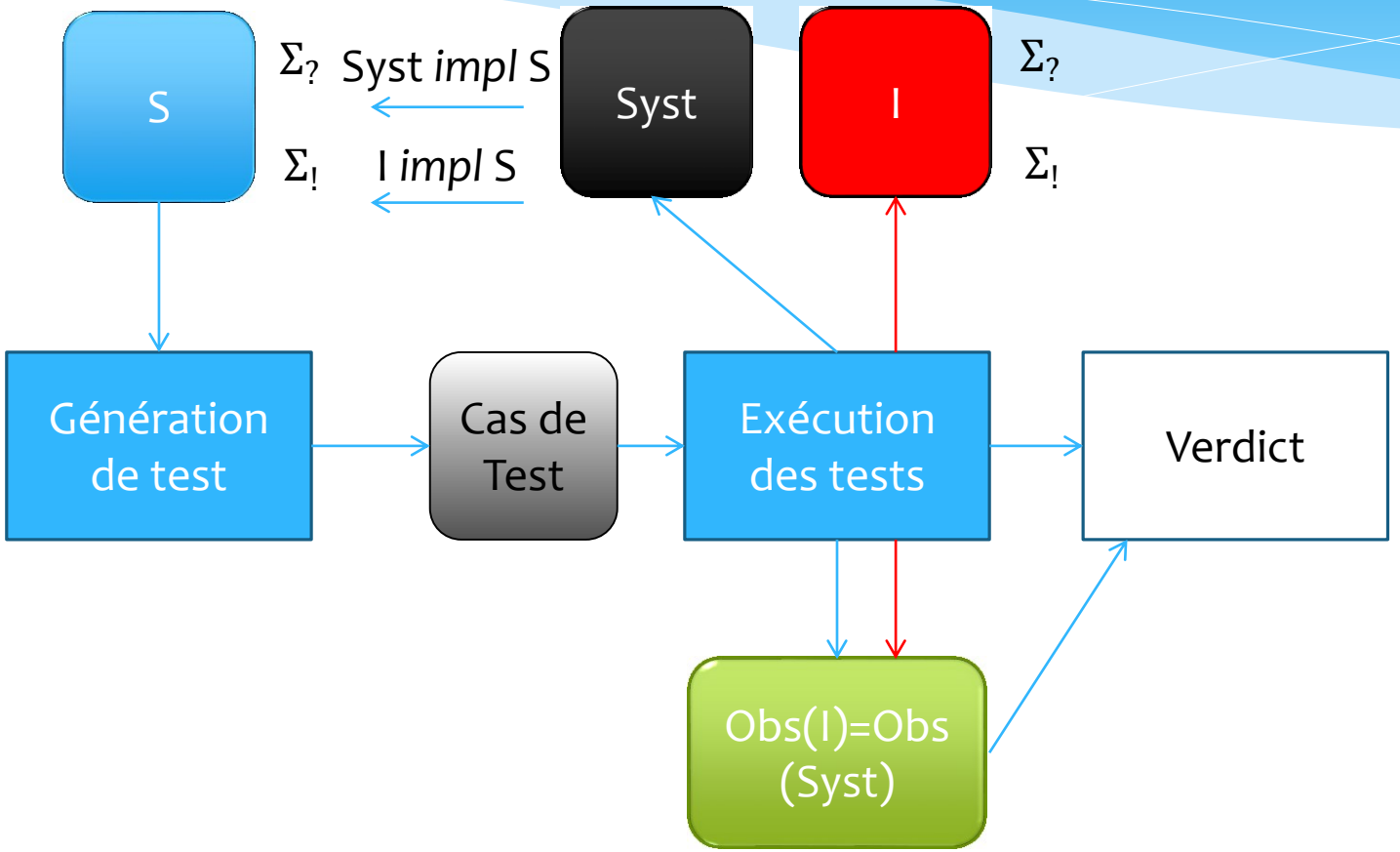
* Ingrédients:
  * spécification, implantation sous test en boîte noire (IUT),
  * Relation de test *impl* (formalise IUT implante S),
  * Cas de test,
  * testeurs

* Recette ?

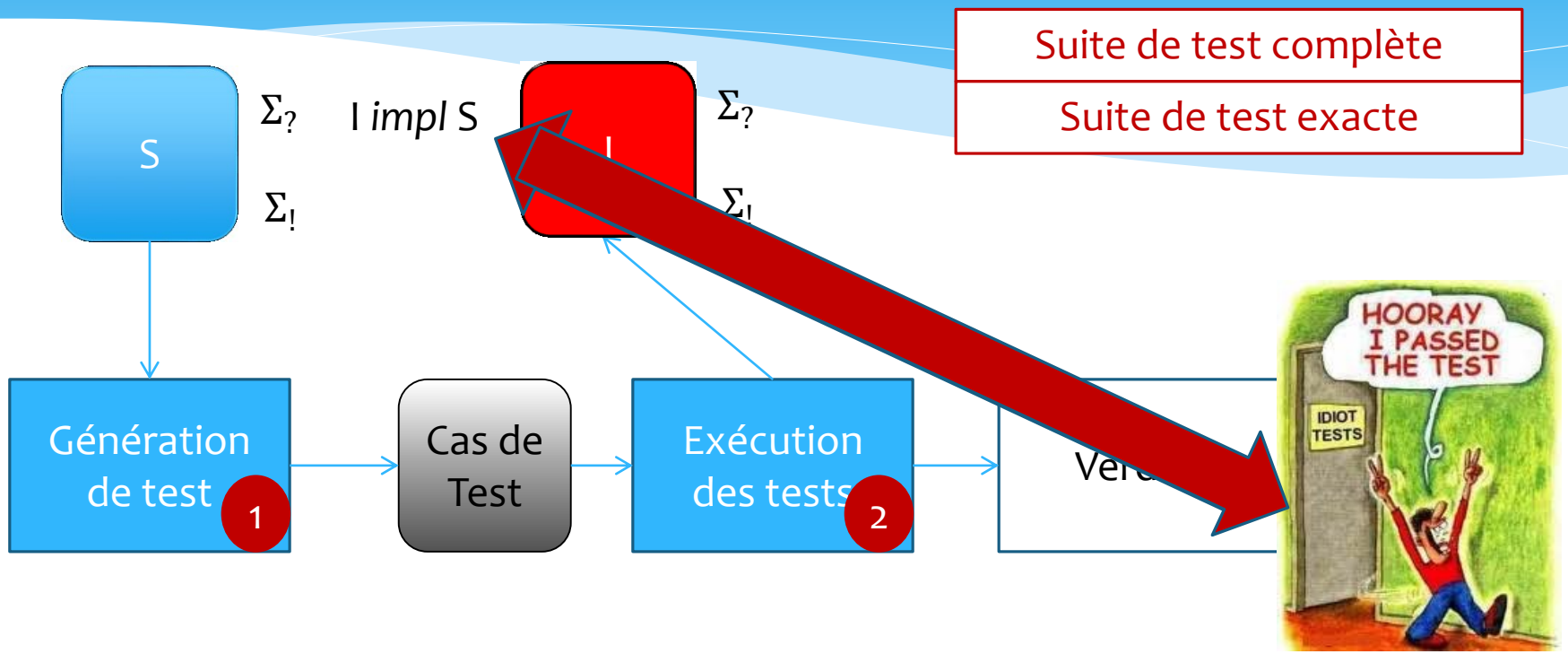
Test basé modèle

* L’implantation est souvent en boite noire
* L’implantation est inconnue mais besoin de la décrire par un modèle pour pouvoir définir des relations entre la spécification (modèle) et des implantations
* => on suppose que l’implantation peut être modélisée par un modèle I qui n’est pas connu et qui retourne les mêmes observations que l’implantation.

Test basé modèle



Test basé modèle

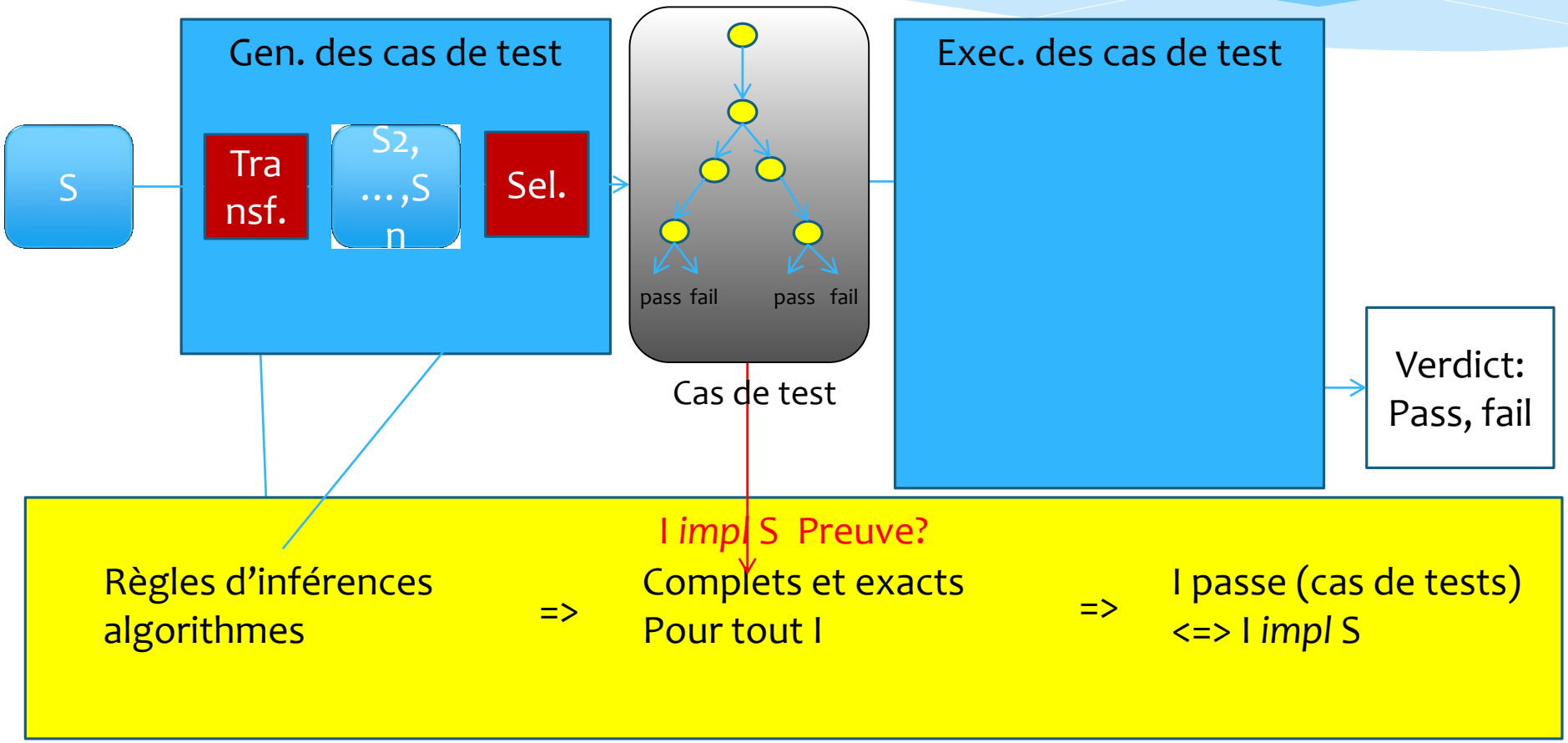


Suite de test complète

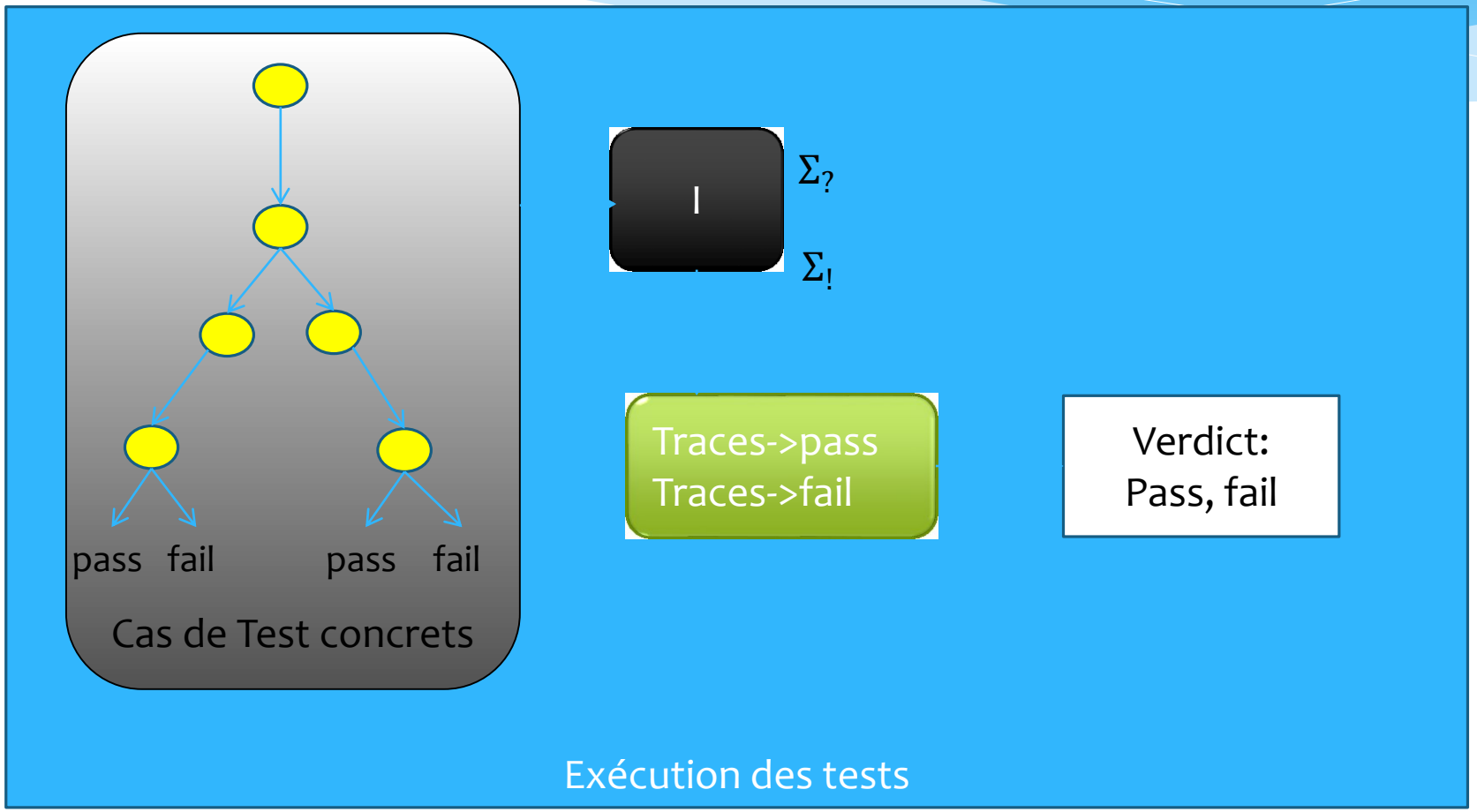
Suite de test exacte

I passe test ⇔ I (Syst) *impl* S ?

Test basé Modèle



Test basé Modèle



Problématique du test

* Besoin d'une suite de test complète et exacte pour que la relation de test soit vraie

  * Obtenir une suite de test complète et exacte est souvent impossible (suite très souvent infinie)
  * Besoin de définir des hypothèses sur les systèmes
    * S Déterminisme, Minimal, fortement connexe
    * I doit avoir les mêmes entrées / sorties que S
    * I et S peuvent être réinitialisés, ont un état initial, etc.

Problématique du test

* Test: processus coûteux
* Formaliser tout en évitant des explosions combinatoires
* Gen. d'une suite de test complète pas toujours possible
  * Choix des hypothèses pour limiter ?
  * Sélection des tests?
* Qualité du test
  * Couverture ?

Travaux connexes

* Autres modèles
  * FSM, EFSM, automates temporisés, UML, etc. (voir e.g. [Dssouli et al.], [De Saqui Sannes et al.], [Hierons et al.], [Lee et al.])
* Test composants
  * Services, composants abstraits (voir e.g. [Gallagher et al.09], [Bartolini et al.09], [Kanso et al.10])
  → Conformité
* Test compositions
  * Autres modèles (BPEL,etc.), composition de composants testés (voir e.g. [Garcia-fanjul06], [Frantzen et al.06], [Bertolino et al.06], [VanderBijl et al03], [Cavalli et al.03-09], [Castanet et al.09-11])
  → Limité dans la prise en compte des env.
* Code-based testing
  * Constraint-based testing, exec. Concolique, couverture de chemins de contraintes (voir e.g. [Offut et al.], [Cadar et al.06], [Godefroid et al.05-08])

Test de services (cloud)

3 2 1 Fight

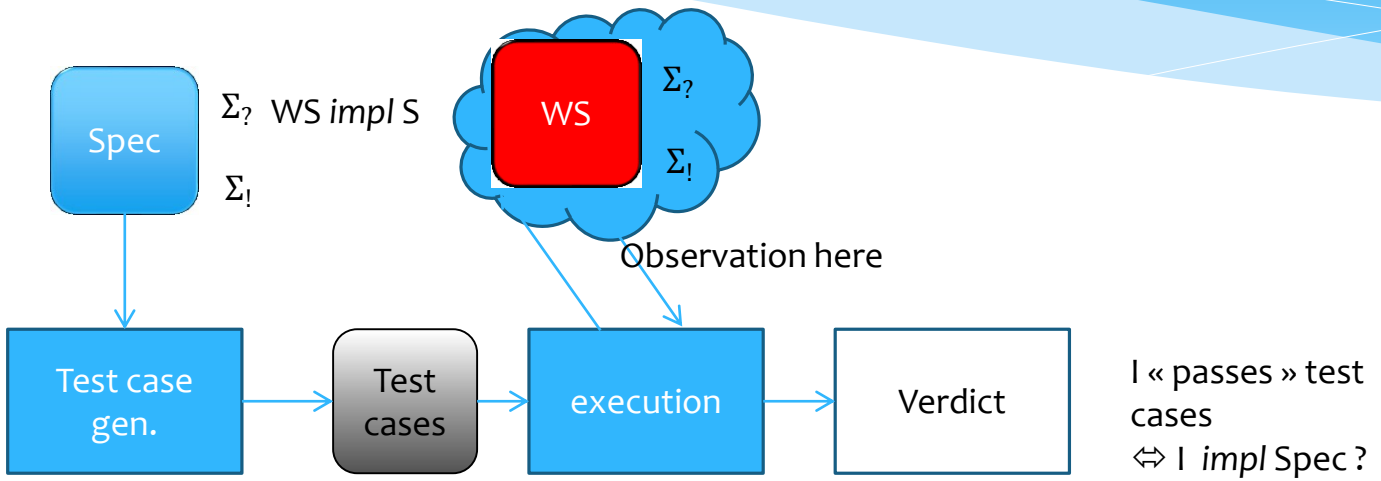
[CHN15]

* **Testing Clouds vs.**
  * Testing cloud architectures (VM, network, load, etc.) => perf, cloud properties [D-Cloud]
  * Cloud simulators (Cloudsim, Greencloud, etc.)
* **Testing with Clouds vs.**
  * Use of clouds for testing
  * Testing as a service (a lot of commercial solutions available: Xamarin Test Cloud, pCloudy)
* **Testing in Clouds**
  * Testing Apps, web services, deployed in clouds

Testing in Clouds

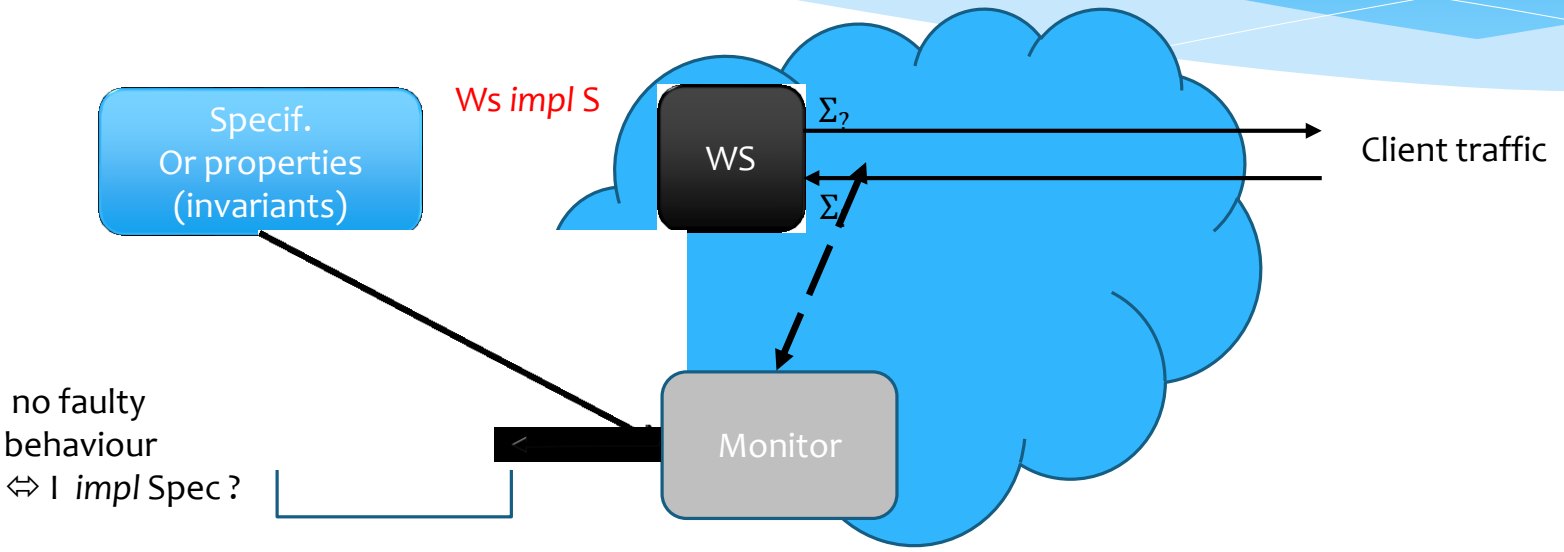
* Conformance testing of Apps
  * Regression testing
* Security testing
  * Availability
  * Checking privacy, secret, authorization, integrity
* Interoperability testing (betwen 2 services in different clouds, etc.)
* Third-party dependencies

Test actif



From web service composition model -> test case gen. -> test case exec. -> verdict

Test passif



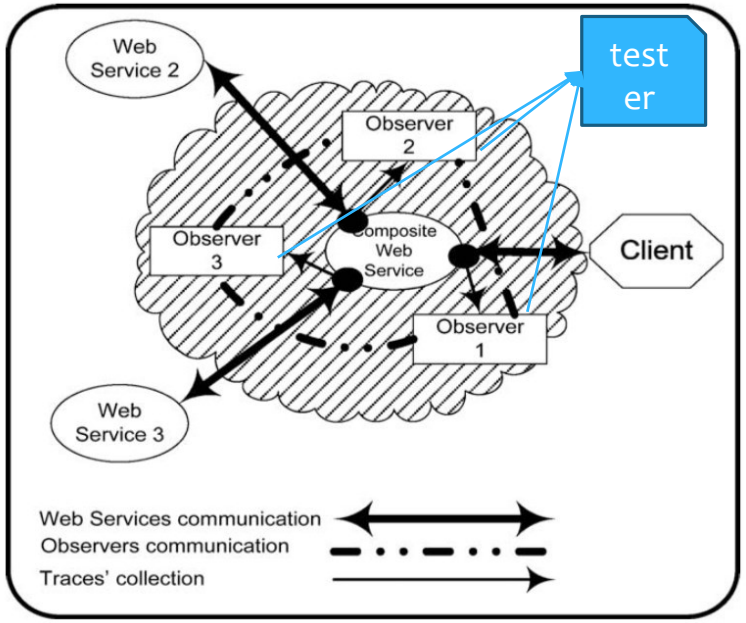
Monitoring de la composition de SW

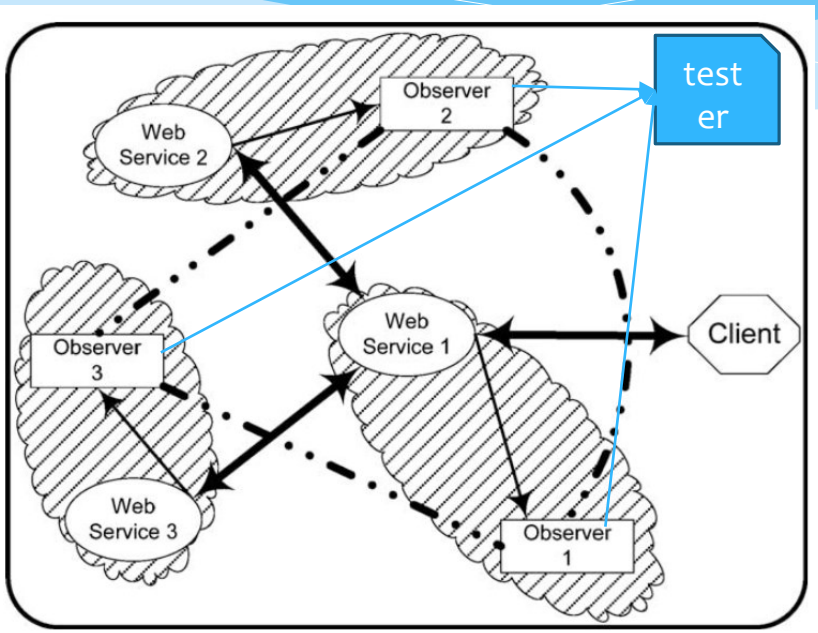
[ACN10][BDANG7][BP09], etc.

Observations, testing architectures

* Collect of the WS requests, responses in Clouds
  * With network sniffers? (when VM are available)
  * By modifying cloud engines ?
    ⇒Difficult
  * By instrumentation of the WS codes
  * With Agents: SNMP agent, mobile agents

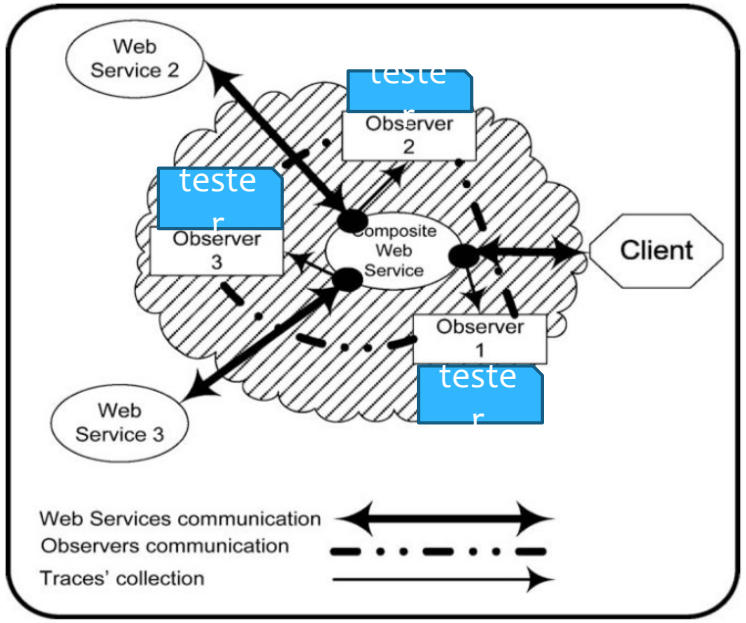
Observations, testing architectures

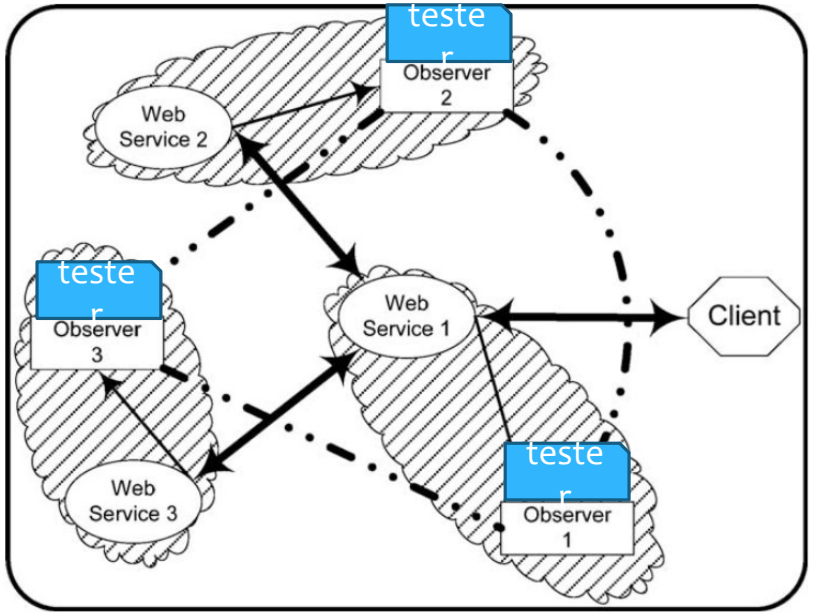




* [BDSG09] [SP15]

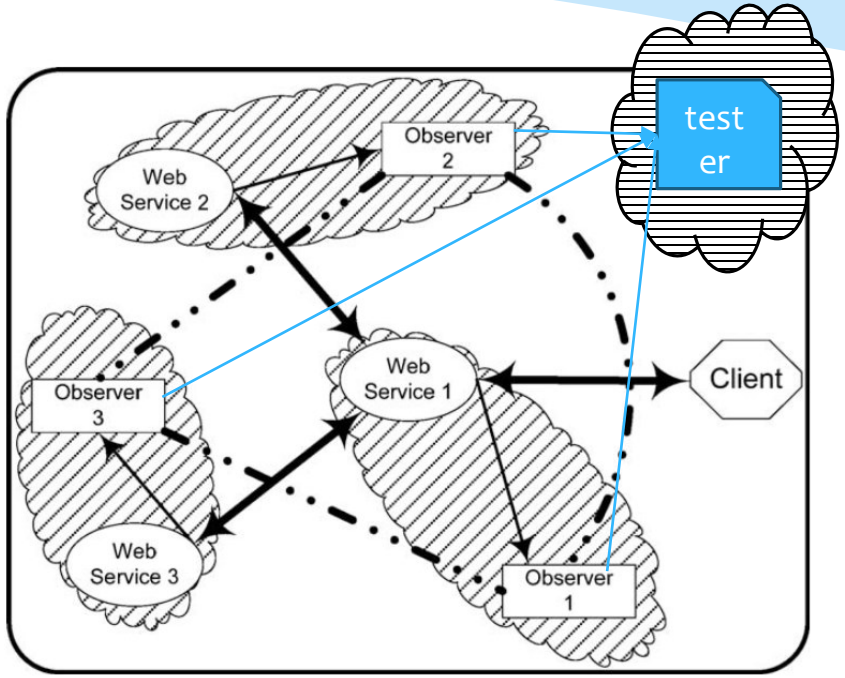
Observations, testing architectures

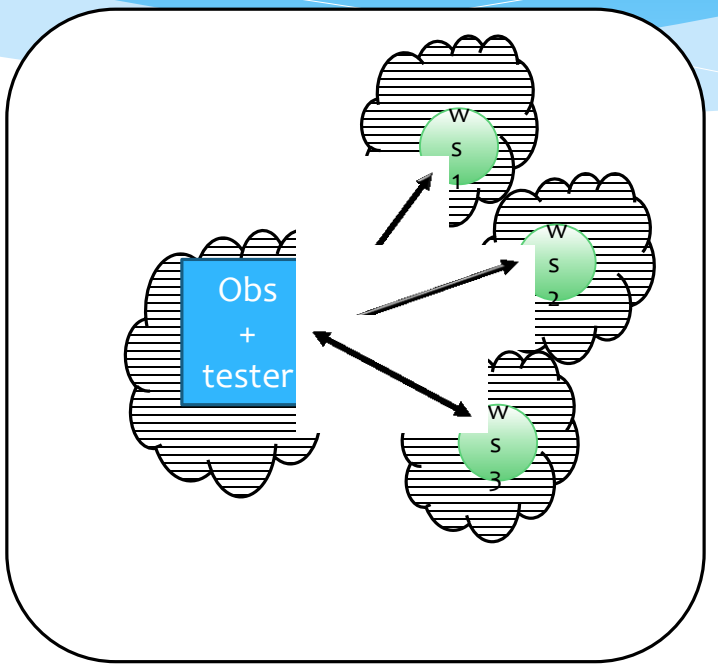




* [BDSG09] [SP15]

Observations, testing architectures





* [BDSG09] [SP15]

Testing in Clouds issues

1. Web service composition level of abstraction ?
   * Test the composite Service
   * Test of all the components?

2. Controllability
   * Can all the service be requested ? (workers: no)

3. Observability of the messages in Clouds ?
   * -> need of specific observers
   * Sniffers cannot be added to PaaS
   * -> code instrumentation, Cloud instrumentation, agents, etc.

Test dans env. partiellement ouverts - Test passif

[S11d]

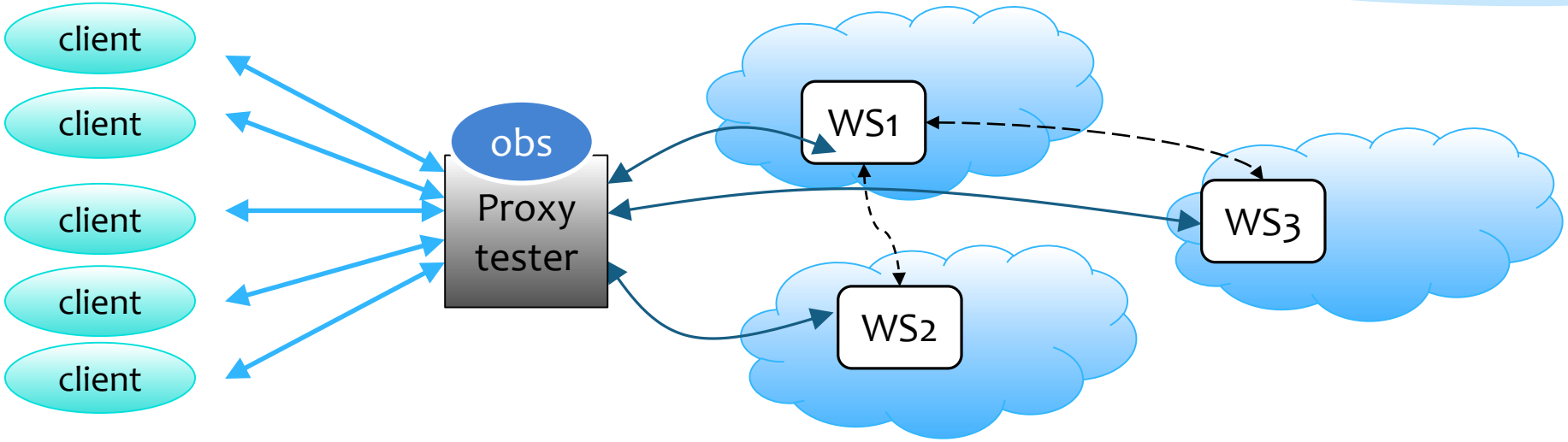
* Problématiques
  * Test actif exhaustif généralement impossible avec modèles symboliques => combiner avec test passif (monitoring)?
  * Test passif basé sur des modules d'observations (sniffeurs)=> très difficile à installer dans certains env. modernes (Clouds, smartphones, etc.)
* Test passif basé sur la notion de proxy transparent

Testing in clouds exemples

Passive testing with proxy-testers

[S11d] [SP15]

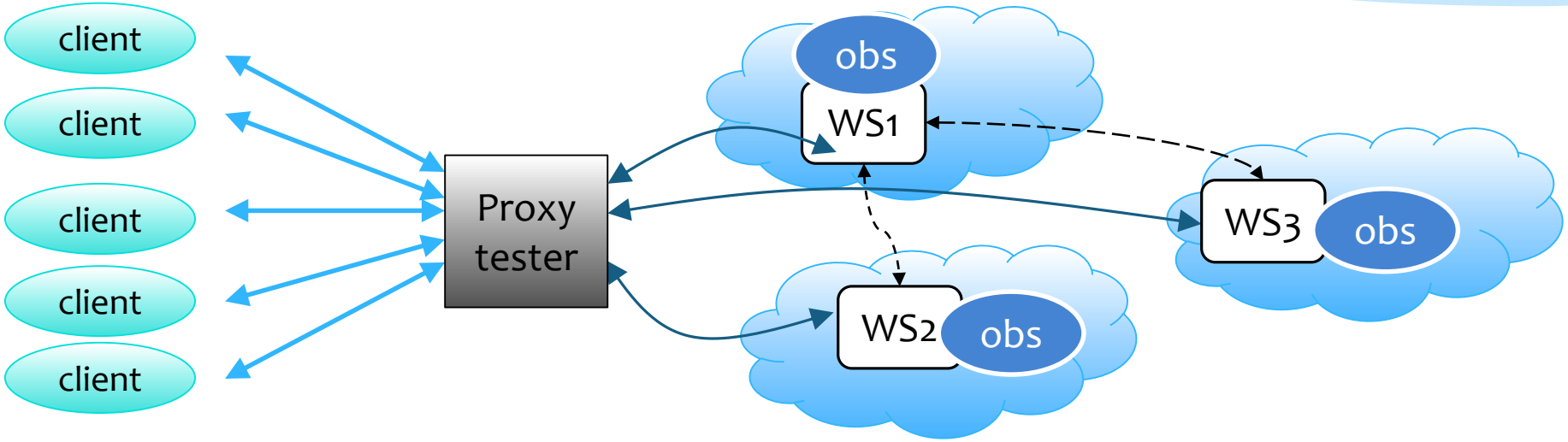
* Proxy-testing principle



- Assumptions: message redirection to proxy (possible in practice), message synchronisation (light protocol to order messages, network latency << quiescence obs.)

Passive testing with proxy-testers
[S11d] [SP15]

* Proxy-testing principle



- Assumptions: message redirection to proxy (possible in practice), message synchronisation (light protocol to order messages, network latency << quiescence obs.)

Passive testing with proxy-testers
[S11d] [SP15]

* Test passif avec concept de proxy formel ? =>
  1. algorithme de testeur passif
  2. + automatic gen. De modèles de proxy-testeurs pour vérifier si relation de test IOCO est satisfaite

* Modèle Proxy-testeur exprime messages échangés
* entre clients <-> Web services
* parmi Web services

* Modèle Proxy-testeur généré depuis spécification

Relation de test ioco

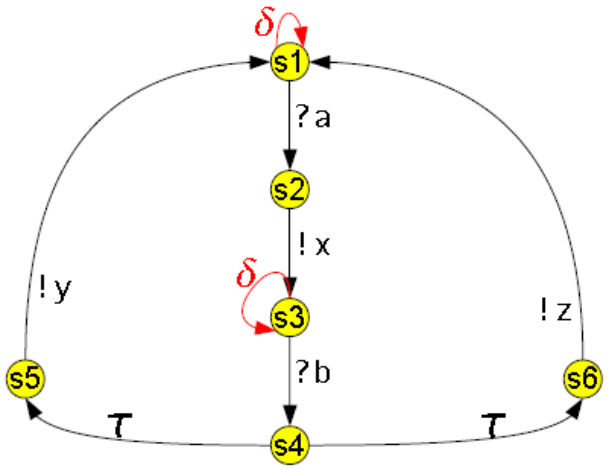
De façon intuitive, une trace d'un LTS représente une séquence d'actions extraites d'un chemin de ce LTS. Cette notion de trace et d'observation est reprise par Tretmans dans [Tre96c] en introduisant la notion d'observateur : un LTS p est équivalent à un LTS q si tout observateur peut faire les mêmes observations avec p et q. Il définit notamment les relations $ioco$, $ioconf$ et $ioco_{\mathcal{F}}$ qui est une généralisation des deux précédentes. Ainsi, la relation $ioco$ entre un LTS S et une implantation I modélisée également par un LTS est définie par :

$$I \text{ ioco } S =_{def} \forall \sigma \in Straces(S) : out(I \text{ after } \sigma) \subseteq out(S \text{ after} \sigma)$$

Relation de test ioco

δ

Definition s after σ = {s' in S \ $s \overset{\sigma}{\Rightarrow} s'$}



s1 after δ = {s1}

s1 after ?a = {s2}

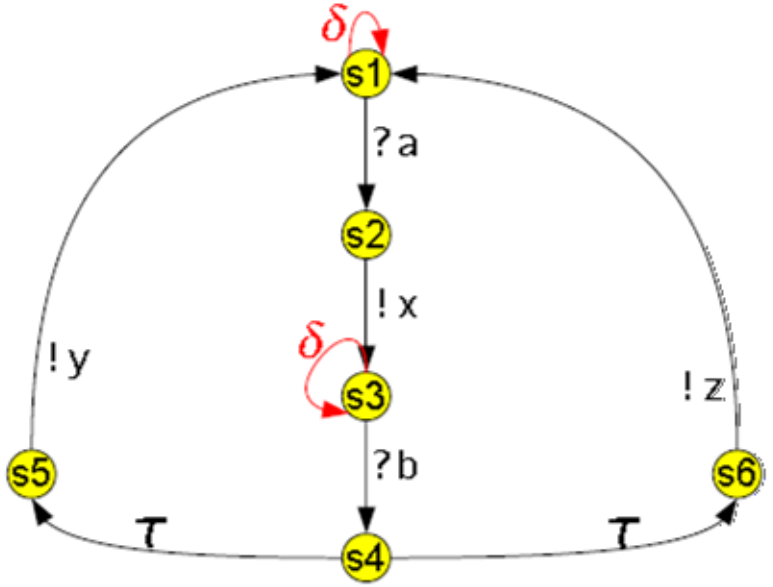
s2 after !x?b = {s4,s5,s6}

s2 after !xδ?b = {s4,s5,s6}

s1 after ?a!x$\delta\delta$?b!zδ = {s1}

Relation de test ioco

- $out(s) =_{def} \begin{cases} \{\delta\} & if\ \delta(s) \\ \{a \in \Sigma_U \mid s \xrightarrow{a}\} & otherwise \end{cases}$
- $out(C) =_{def} \bigcup_{s \in C} out(s)$



out(s1) = {δ}
out(s2) = {!x}
out(s3) = {δ}
out(s4) = Ø
out(s5) = {!y}
out({s1,s2}) = {δ,!x}

out(s1 after δ?a) = {!x}
out(s1 after ?a!x?b) = {!y,!z}

Test dans env. partiellement ouverts - Test passif

[S11d]

⇒combinaison de S (IOSTS) avec son testeur canonique

Testeur canonique pour un IOSTS

Définition 3.2.2 (Testeur canonique) *Soit* $\mathcal{S} =< L_{\mathcal{S}}, l0_{\mathcal{S}}, V_{\mathcal{S}}, V0_{\mathcal{S}}, I_{\mathcal{S}}, \Lambda_{\mathcal{S}}, \rightarrow_{\mathcal{S}}>$ *un STS et* $\Delta(\mathcal{S})$ *son suspension. Le testeur canonique de* $\mathcal{S}$ *est le STS* $Can(\mathcal{S}) =< L_{Can}, l0_{Can}, V_{\mathcal{S}}, V0_{\mathcal{S}}, I_{\mathcal{S}}, \Lambda_{Can}, \rightarrow_{Can}>$ *tel que :*

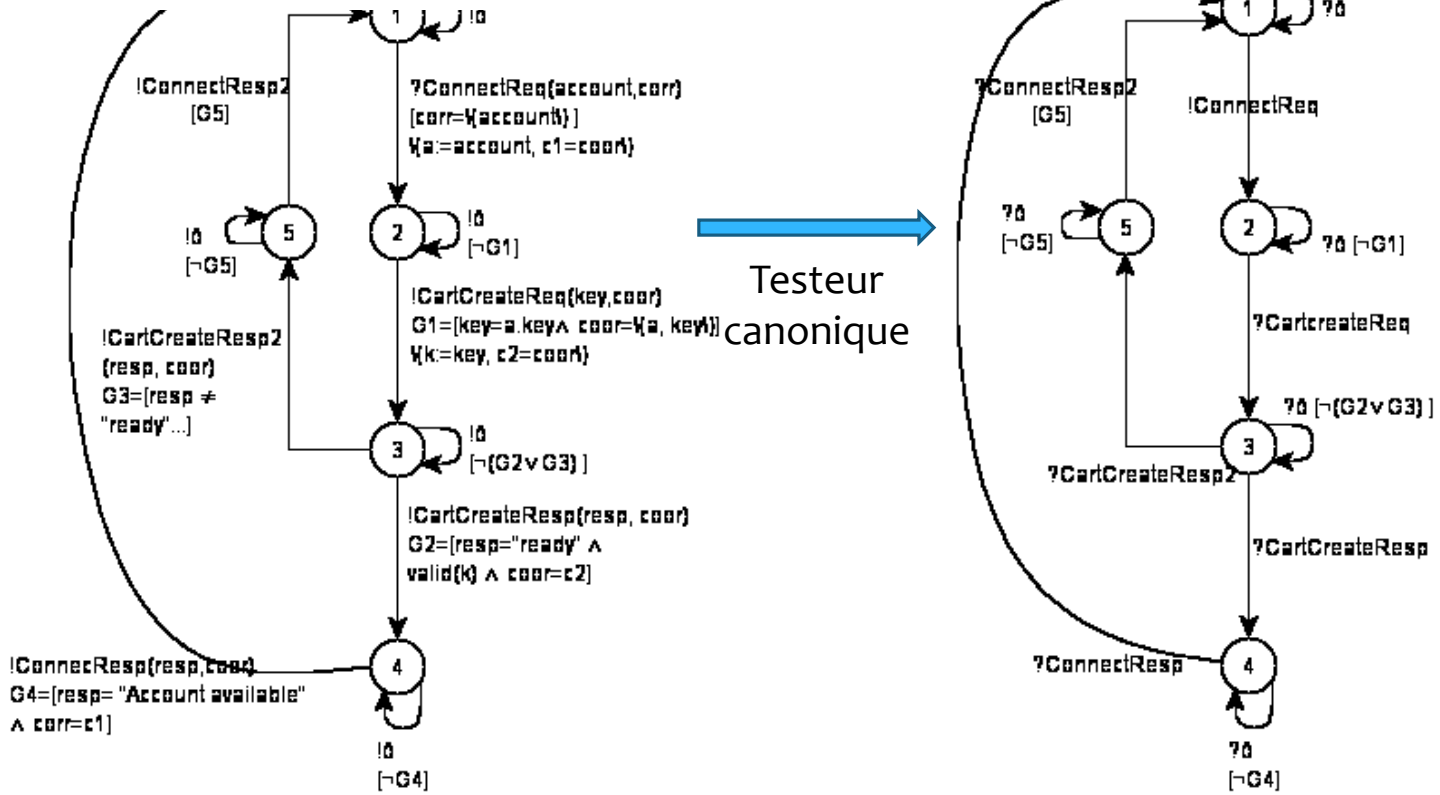
- $\Lambda^I_{Can} = \Lambda^O_{\mathcal{S}} \cup \{?\delta\}$ *et* $\Lambda^O_{Can} = \Lambda^I_{\mathcal{S}}$,
- $L_{Can}, l0_{Can}, \rightarrow_{Can}$ *sont définis par les règles suivantes :*

(keep $\mathcal{S}$ *transitions) :* $\dfrac{t \in \rightarrow_{\Delta(\mathcal{S})}}{t \in \rightarrow_{Can}}$

(incorrect behaviour completion) : $\dfrac{a \in \Lambda^O_{\mathcal{S}} \cup \{!\delta\}, l_1 \in L_{\mathcal{S}}, \varphi_a = \bigwedge_{l_1 \xrightarrow{a(p),\varphi_n,\varrho_n}_{\Delta(\mathcal{S})} l_n} \neg\varphi_n}{l_1 \xrightarrow{?a(p),\varphi_a,\emptyset}_{Can} Fail}$

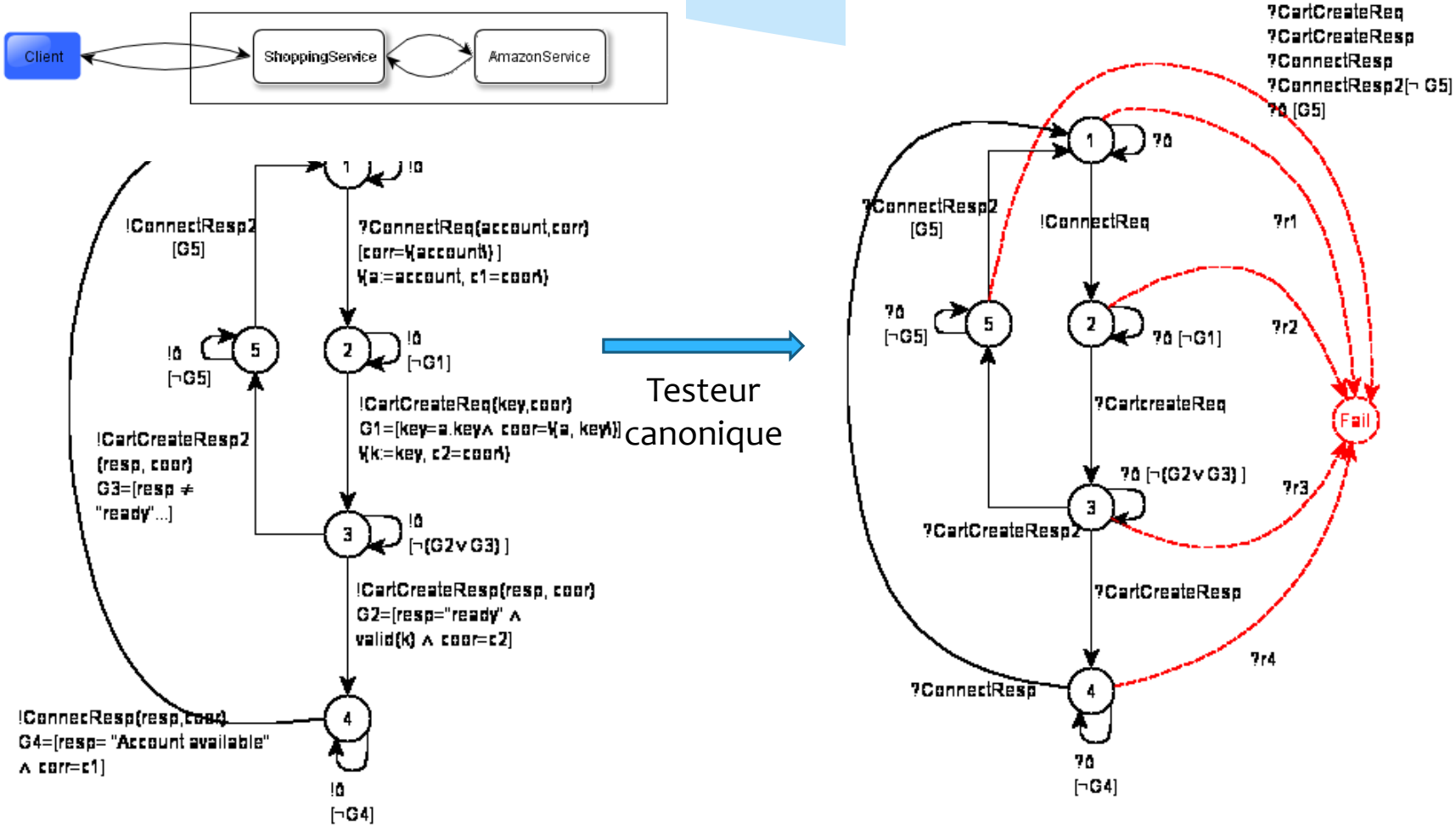
Test dans env. partiellement ouverts - Test passif

[S11d]



Test dans env. partiellement ouverts - Test passif

[S11d]



Test dans env. partiellement ouverts - Test passif

[S11d]

Définition 3.2.3 (proxy-testeur) *Le proxy-testeur $\mathcal{P}(\mathcal{S})$ d'une spécification $\mathcal{S} =< L_{\mathcal{S}}, l0_{\mathcal{S}}, V_{\mathcal{S}}, V0_{\mathcal{S}}, I_{\mathcal{S}}, \Lambda_{\mathcal{S}}, \rightarrow_{\mathcal{S}}>$ est la combinaison de $\Delta(\mathcal{S})$ avec son STS réflexion $\phi(REF(\mathcal{S}))$. $\mathcal{P}(\mathcal{S})$ est défini par un STS $< L_{\mathcal{P}}, l0_{\mathcal{P}}, V_{\mathcal{S}} \cup V_{\phi(\mathcal{S})} \cup \{side\}, V0_{\mathcal{S}} \cup V0_{\phi(\mathcal{S})} \cup \{side := ""\}, I_{\mathcal{S}} \cup I_{\phi(\mathcal{S})}, \Lambda_{\Delta(\mathcal{S})} \cup \Lambda_{REF}, \rightarrow_{\mathcal{P}}>$ tel que $L_{\mathcal{P}}, l0_{\mathcal{P}}$ et $\rightarrow_{\mathcal{P}}$ sont construits par les règles d'inférence suivantes :*

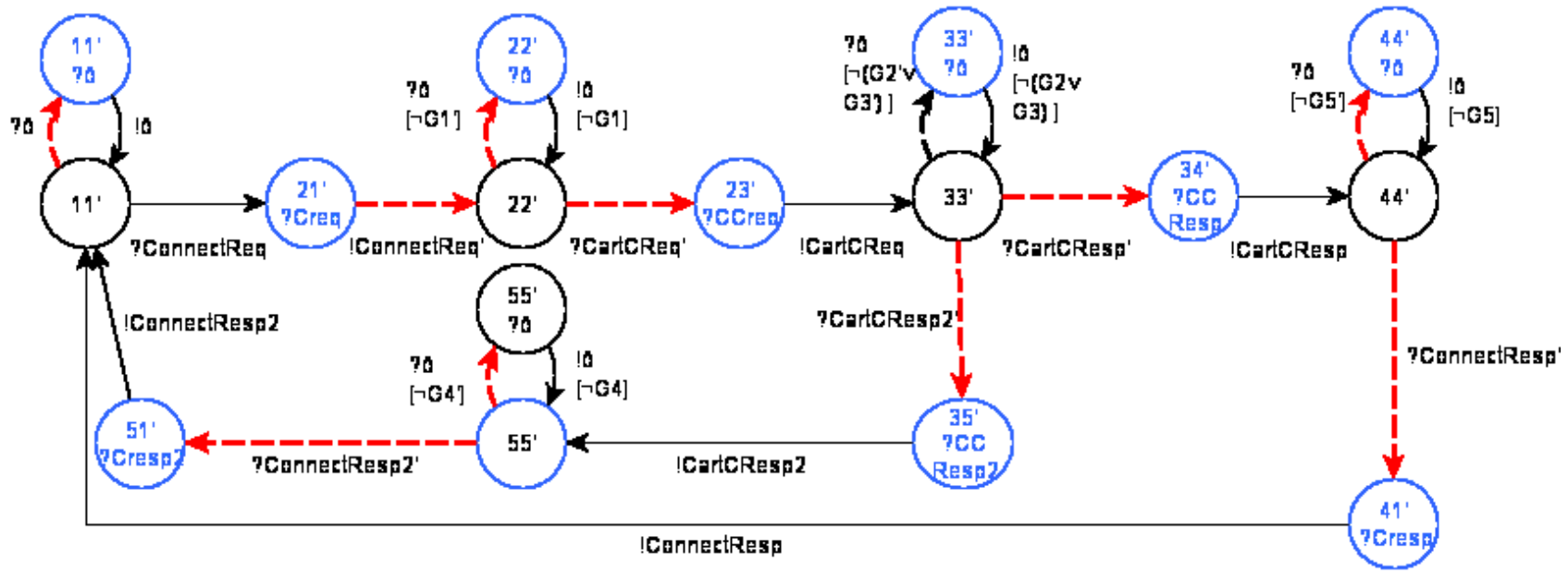
(Env: to IUT)
$$l_1 \xrightarrow{?a(p),G,A}_{\Delta(\mathcal{S})} l_2, l_{1'} \xrightarrow{!a(p'),G',A'}_{REF} l_{2'},$$
$$l'_1 = \phi(l_1), G' = \phi(G), A' = \phi(A)$$
$$\vdash$$
$$(l_1 l_{1'}) \xrightarrow{?a(p),G,A(\{side:=Env\})}_{\mathcal{P}} (l_2 l_{1'} ?aGA)$$
$$\xrightarrow{!a(p'),[p'=p \wedge G'],A'(\{side:=IUT\})}_{\mathcal{P}} (l_2 l_{2'}),$$

(to: Fail)
$$l_{1'} \xrightarrow{b(p),G,A}_{REF} Fail, l_1 \in L_{\mathcal{S}}, l'_1 = \phi(l_1)$$
$$\vdash$$
$$(l_1 l_{1'}) \xrightarrow{b(p),G,A(\{side:=IUT\})}_{\mathcal{P}} Fail$$

(IUT: to Env)
$$l_1 \xrightarrow{!a(p),G,A}_{\Delta(\mathcal{S})} l_2, l_{1'} \xrightarrow{?a(p'),G',A'}_{REF} l_{2'},$$
$$l'_1 = \phi(l_1), G' = \phi(G), A' = \phi(A)$$
$$\vdash$$
$$(l_1 l_{1'}) \xrightarrow{?a(p'),G',A'(\{side:=IUT\})}_{\mathcal{P}} (l_1 l_{2'} ?aGA)$$
$$\xrightarrow{!a(p),[p=p' \wedge G],A(\{side:=Env\})}_{\mathcal{P}} (l_2 l_{2'}),$$

Test dans env. partiellement ouverts - Test passif
[S11d]

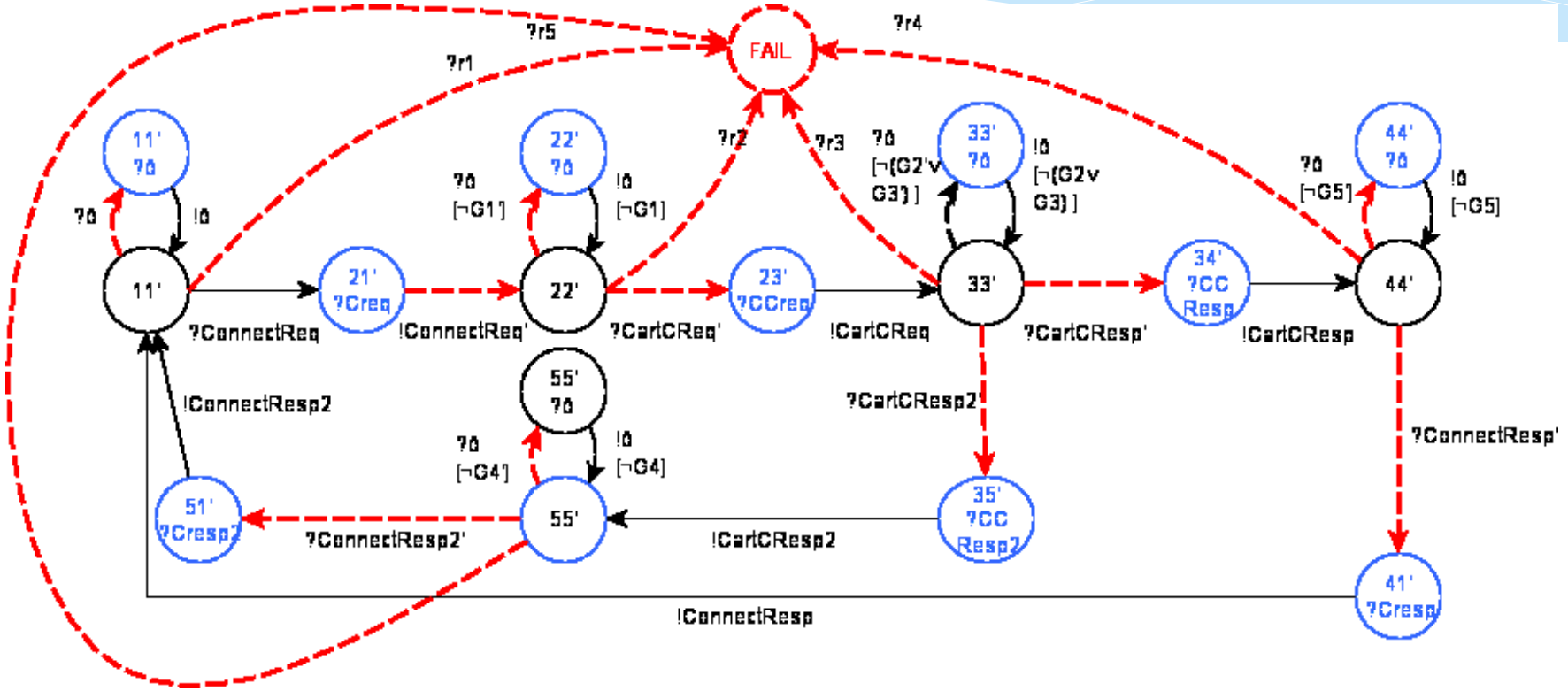
* Proxy-testeur:



Test dans env. partiellement ouverts - Test passif

[S11d]

* Proxy-testeur:



Propriété sur traces: $Traces_{Fail}^{CAN}(P\ (S)) = Traces_{Fail}(CAN(S))$

Test dans env. partiellement ouverts - Test passif

[S11d]

* Ioco proxy-testeur ?

$I\ ioco\ S \Leftrightarrow Traces(\Delta(S)).(\Sigma^{O} \cup \{!\delta\}) \cap \square\ aces(\Delta(I)) \subseteq Traces(\Delta(S))$ (RUSU05a)

$I\ ioco\ S \Leftrightarrow Traces(\Delta(I)) \cap NCTraces(\ \ (S)) = \emptyset$

$I\ ioco\ S \Leftrightarrow Traces(\Delta(I)) \cap Traces_{Fail}^{CAN}(P(S)) = \emptyset$

$I\ ioco\ S \Leftrightarrow Traces_{Fail}(||(\mathrm{Env}, \mathrm{P}, \mathrm{I})) = \emptyset$

Prop. sur traces

$NCTraces(\Delta(S)) = Traces_{Fail}(CAN(S))$

Exécution parallèle

||(Env, P, I) = IOLTS

$$\frac{q_1 \xrightarrow{!\alpha}_{\Delta(Env)} q_2, q_2'' \quad \xrightarrow{?\alpha}_{\Delta(I)} q_3'', q_1' \quad \xrightarrow{?\alpha}_{"\,"\,P} q_2' \quad \xrightarrow{!\alpha}_{"NCO"\,P} q_3'}{q_1 q_1' q_2'' \xrightarrow{?\alpha}_{||(Env,P,I)} q_2 q_2' q_2'' \xrightarrow{!\alpha}_{||(Env,P,I)} q_2 q_3' q_3''}$$

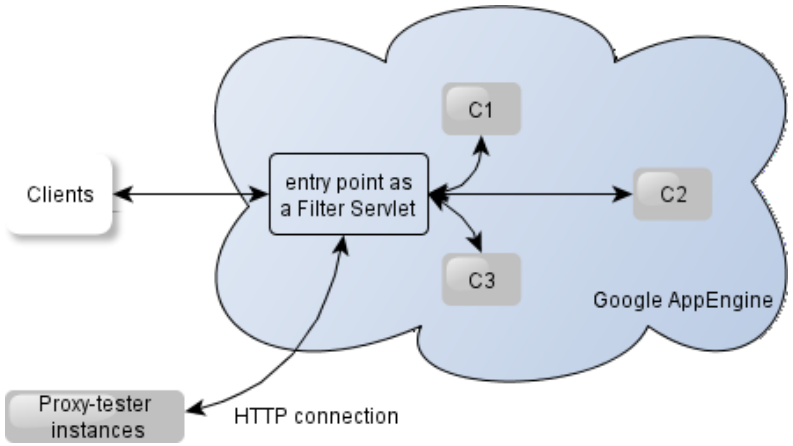
$$\frac{q_2 \xrightarrow{?\alpha}_{\Delta(Env)} q_3, q_1'' \quad \xrightarrow{!\alpha}_{\Delta(I)} q_2'', q_1' \quad \xrightarrow{?\alpha}_{"NCO"\,P} q_2' \quad \xrightarrow{!\alpha}_{"\,"\,P} q_3', q_3' \neq Fail}{q_2 q_1' q_1'' \xrightarrow{?\alpha}_{||(Env,P,I)} q_2 q_2' q_2'' \xrightarrow{!\alpha}_{||(Env,P,I)} q_3 q_3' q_2''}$$

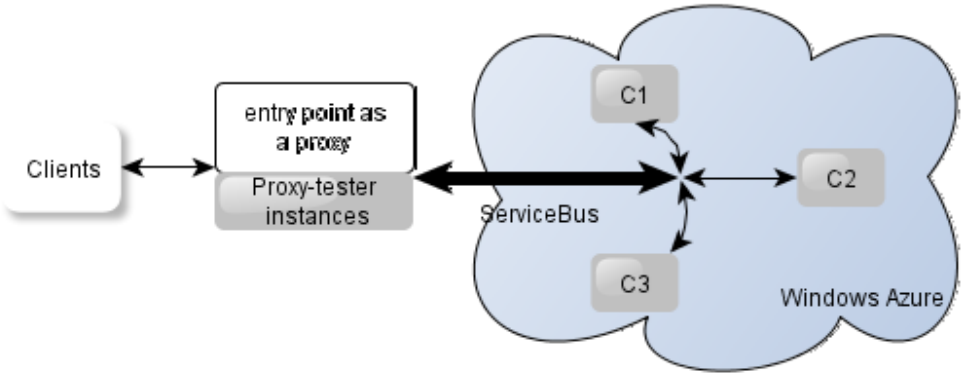
$$\frac{q_2 \xrightarrow{?\delta}_{\Delta(Env)} q_3, q_1'' \quad \xrightarrow{!\alpha}_{\Delta(I)} q_2'', q_1' \quad \xrightarrow{?\alpha}_{"NCO"\,P} Fail}{q_2 q_1' q_1'' \xrightarrow{?\alpha}_{||(Env,P,I)} Fail}$$

=> Algo: const. de traces. Si une trace mène à Fail => erreur

Test dans env. partiellement ouverts - Outillage

* Outils en cours de développement (LIMOS/ Tan Tao University, Vietnam)
  * sur 2 Clouds, Windows Azure et Google AppEngine





Runtime verification

* Méthode tirée de la Vérification
  * Permet de tester passivement des propriétés sur les traces d'exécutions d'une implantation

* Modélisation de propriétés
  * Par logiques (temporelles),
  * automates à états
  * Etc.

Runtime verification avec proxy-testers

* Completion du modèle Proxy-tester avec
  * Safety properties ‘’nothing bad ever happens’’
* Safety property modélisées avec ioSTSs ☺

  Exprime les comportements qui violent la propriété

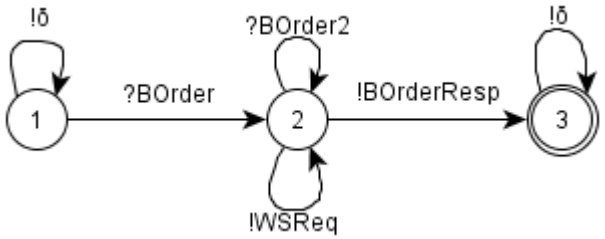
Runtime verification

* Modélisation de propriétés safety
* par ioSTSs)

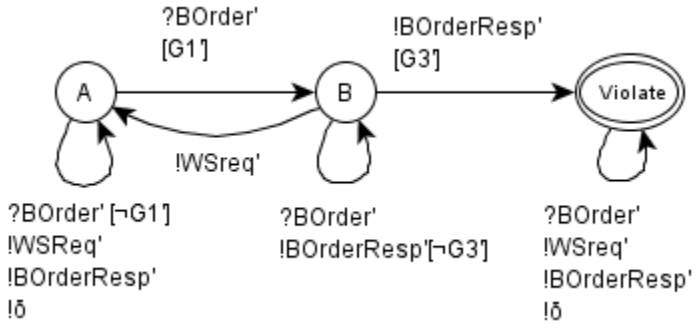
  * Représentation du comportement qui viole la propriété par ajout d'un état violate

Runtime verification

* Exemple de propriétés safety sur composition de services



Symbol	Message	Guard	Update
?BOrder	?BookOrder(List-Books, quantity, account)		q:=quantity, b:=ListBooks
?BOrder2	?BookOrder(List-Books, quantity, account)		
!WSReq	!WholeSaler(isbn, from, to, corr)	$G2$=[isbn=b[q]$\wedge q \geq 1 \wedge$ from="BR"$\wedge$to= "WS"$\wedge$ $corr = \{a, isbn\}$]	$q := q - 1$
!BOrder Resp	!BookOrderResp(resp)	$G3$=[resp="Order done"]	
?R1	?BookOrderResp ?WholeSaler		
?R2	?BookOrderResp ?WholeSaler?δ	[≠G3] [≠G2]	



Symbol	Message	Guard
?BOrderReq'	?BookOrderReq(ListBooks, quantity, account)	G1'=[quantity≥ 1]
!WSReq'	!WholeSalerReq(isbn)	
!BOrderResp'	!BookOrderResp(resp)	G3'=[start(resp)="done"]

"la réception d'une confirmation de commande commençant par "done", sans effectuer de requète à WholeSaler, ne doit pas se produire"

Runtime verification

* Génération de moniteurs à partir des propriétés seules ou combinées avec une spécification
  * Moniteur de plus bas niveau
  * Possède un algorithme de type checker state
  * Si un état (valué en ioSTS) Violate est atteint, alors la safety propriété a été violée.

Runtime verification

* Algorithme simplifié (avec ioSTS) :

On part de l'état initial (lo,Vo).
Etat courant (lc,vc)=(lo,Vo)
Pour tout évènement reçu (quiescence comprise), (a(p), θ),

on recherche la transition $li \xrightarrow{a(p),G,A} lj$ de la propriété,
commençant par lo, telle que vc$\cup$ θ satisfait G,

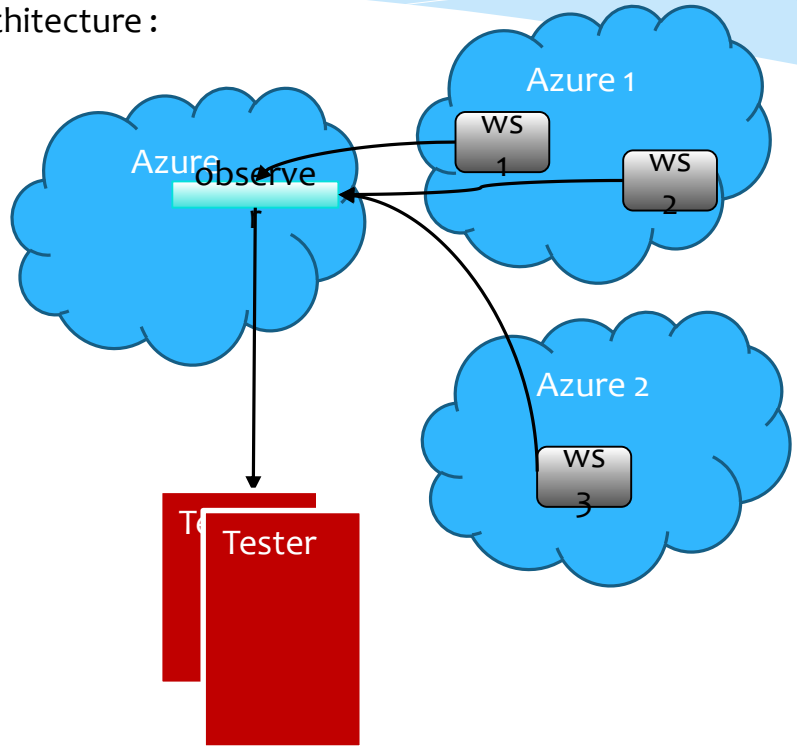
Si lj = VIOLATE Alors une violation est détectée FIN
Sinon Etat courant (lc,vc)=(lj,A(vc $\cup$ θ))
SI quiescence a été détectée plusieurs fois, FIN
FinPour

Runtime verification

* 2 types principaux d'analyse:
  * Synchrone : lecture d'un évènement, calcul du prochain état puis décision (violation ou pas)
  * Asynchrone: lecture des évènements avec analyse différée
* L'analyse synchrone est plus lente (surtout si des solveurs de contraintes sont appelés) mais permet d'effectuer une reprise en cas de détection d'erreur (runtime enforcement) => permet de revenir sur un état ''stable'', de réinitialiser, de compenser, etc.

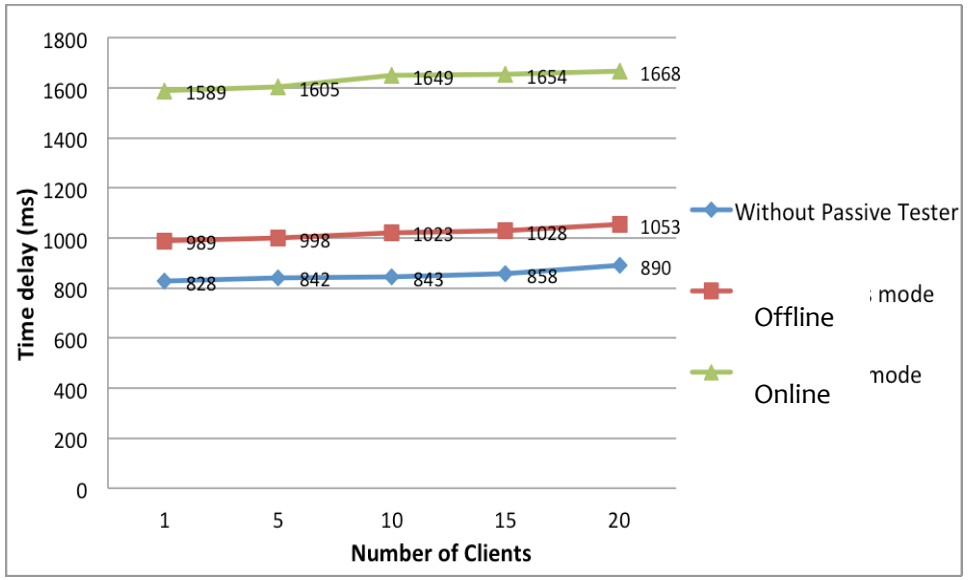
Evaluation

Architecture :



In our laboratory

Cloud = Azure
3 Web services
1-20 mocked clients in the same time doing 20 requests



Conclusion

* L'intérêt du MbT par les sociétés IT grandit de plus en plus
  * Programmes de plus en plus complexes, utilisent de plus en plus de concepts (SOA en autre) => besoin de tester
  * Utilisation de plus en plus grande de la modélisation (UML)
  * Besoin de plus en plus grand d'effectuer des tests
  * MbT permet d'automatiser des étapes faites à la main actuellement

Conclusion

* Quelques Challenges:
  * MbT bien adapté au cycle en V. Besoin d'adapter le MbT aux méthodes agiles
  * Spécification symboliques, temporelles avec probabilités, etc. nécessitent un cout de test trop important
  * Modèles symboliques => besoin de solver de contraintes adaptés (types String, combinaison de types, etc.)
  * Affranchissement du modèle
    * utilisation d'un modèle partiel, construction d'un modèle partiel
    * Tests aléatoires

* [BDSG09]A. . Benharref, R. Dssouli, M. Serhani and R. Glitho, Efficient Traces Collection Mechanisms for Passive Testing of Web Services, *Elsevier Information and Software Technology* 51 (2009), 362 – 374
* [VRT03] Bijl, Machiel van der and Rensink, Arend and Tretmans, Jan (2004) Compositional Testing with ioco. In: Third International Workshop on Formal Approaches to Testing of Software, FATES 2003, October 6, 2003, Montreal, Quebec, Canada (pp. pp. 86-100).
* [NKRW11] Neda Noroozi , Ramtin Khosravi , Mohammad Reza Mousavi , Tim A. C. Willemse , Synchronizing Asynchronous Conformance Testing, In Proc. of SEFM 2011, volume 7041 of LNCS
* [SC14] Sébastien Salva and Tien-Dung Cao, Proxy-Monitor: An integration of runtime verification with passive conformance testing., In International Journal of Software Innovation (IJSI), vol. 2, nb. 3, p. 20--42, IGI Global, 2014
* [SP15] Sébastien Salva and Patrice Laurençot, Conformance Testing with ioco Proxy-Testers: Application to Web service compositions deployed in Clouds, In International Journal of Computer Aided Engineering and Technology (IJCAET), vol. 7, nb. 3, p. 321--347, Inderscience, 2015
* [CHN15] Ana R. Cavalli, Teruo Higashino, Manuel Núñez, A survey on formal active and passive testing with applications to the cloud. Annales des Télécommunications 70(3-4): 85-93 (2015)
* [PYL03]Testing Transition Systems with Input and Output Testers (2003), Alexandre Petrenko , Nina Yevtushenko , Jia Le Huo , PROC TESTCOM 2003, SOPHIA ANTIPOLIS
* [ACN10] Passive Testing of Web Services César Andrés, M. Emilia Cambronero, Manuel Núñez ProceedingWS-FM'10 Proceedings of the 7th international conference on Web services and formal methods
* [BBANG07] New Approach for EFSM-Based Passive Testing of Web Services Abdelghani Benharref, Rachida Dssouli, Mohamed Adel Serhani, Abdeslam En-Nouaary, Roch Glitho, roceedingTestCom'07/FATES'07 Proceedings of the 19th IFIP TC6/WG6.1 international conference, and 7th international conference on Testing of Software and Communicating Systems
* [BPZ09] A Formal Framework for Service Orchestration Testing Based on Symbolic Transition Systems Lina Bentakouk, Pascal Poizat, Fatiha Zaïdi, TESTCOM '09/FATES '09 Proceedings of the 21st IFIP WG 6.1 International Conference on Testing of Software and Communication Systems and 9th International FATES Workshop
* [RPG06] Retracted: Towards Formal Verification of Web Service Composition Mohsen Rouached, Olivier Perrin, Claude Godart, Business Process ManagementVolume 4102 of the series Lecture Notes in Computer Science pp 257-273
* [CPFC10] Automated Runtime Verification for Web Services, Tien-Dung Cao 1 Trung-Tien Phan-Quang 1 Patrick Félix 1 Richard Castanet, IEEE international Conference on Web Services, Jul 2010, Miami, United States. pp.76-8