Testing in Clouds

Sébastien Salva, LIMOS, UDA

12th TAROT Summer School 2016

Who am I?

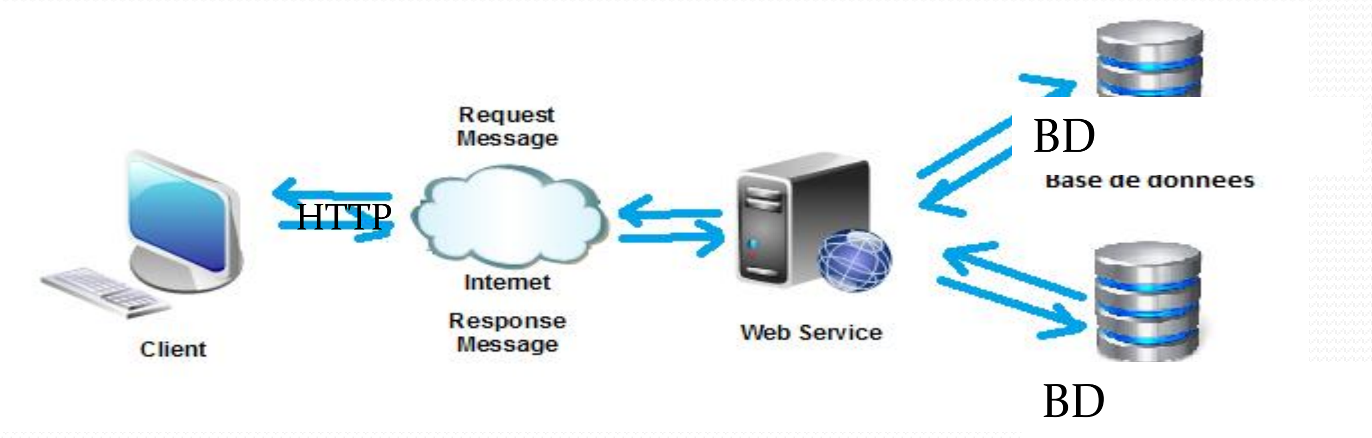
```
Public void setUp(){
Identity id=new Identity(“salva”);}

Public void testid (){
assertEquals(id.surname, ”sébastien”);
assertEquals(id.name, ”salva”);
assertEquals(id.labo, ”LIMOS”);
assertEquals(id.city ”Clermont-Ferrand”);

assertArrayEquals(id.recherche, new String[] {‘’Model-based Testing”, ‘’model inference‘’, ‘’passive testing‘’, ‘’security‘’});
}
```

Outline

- Cloud computing ?
- Testing in clouds
- Model-based testing example

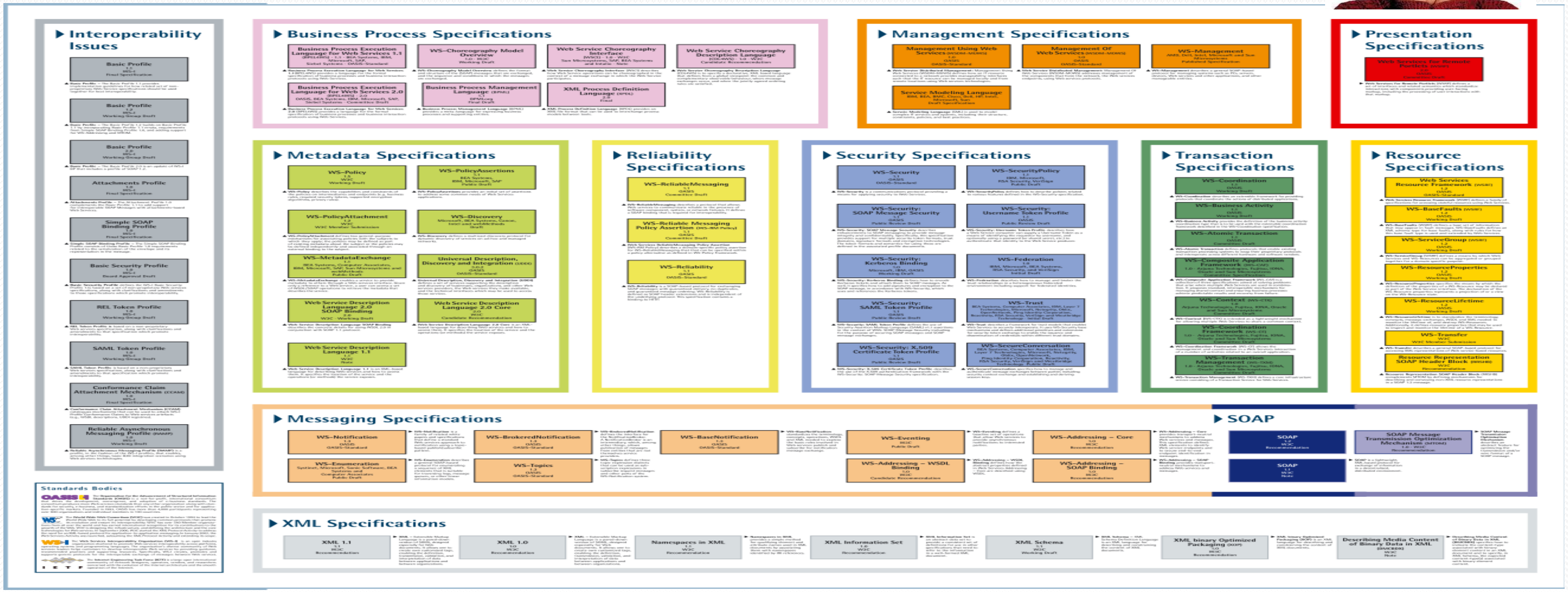
A Short comment on Apps

- In this talk, apps deployed in clouds are Web services
  - Why? most of the Apps deployed in Clouds (PaaS) are Web services
    - A lot of works about Web service testing, Web service composition, etc.
  - SOAP, REST ?
  - Composite Web service ? Orchestration, choregraphy ?



A Short comment on Apps

- Some WS standards



I Cloud computing

Cloud computing definition ?

"... the market seems to have come to the conclusion that cloud computing has a lot in common with obscenity--you may not be able to define it, but you'll know it when you see it"

James Urquhart – The Wisdom of Clouds

Cloud origin

- Cloud computing, introduced by

- Amazon (2002), suite of cloud-based services including storage, computation and even human intelligence through the Amazon Mechanical Turk.
- 2006, Amazon launched its Elastic Compute cloud (EC2)
- was announced as "Azure" in October 2008 and was released on 1 February 2010 as Windows Azure, before being renamed to Microsoft Azure on 25 March 2014. **Google App Engine** (often referred to as **GAE** or simply **App Engine**)

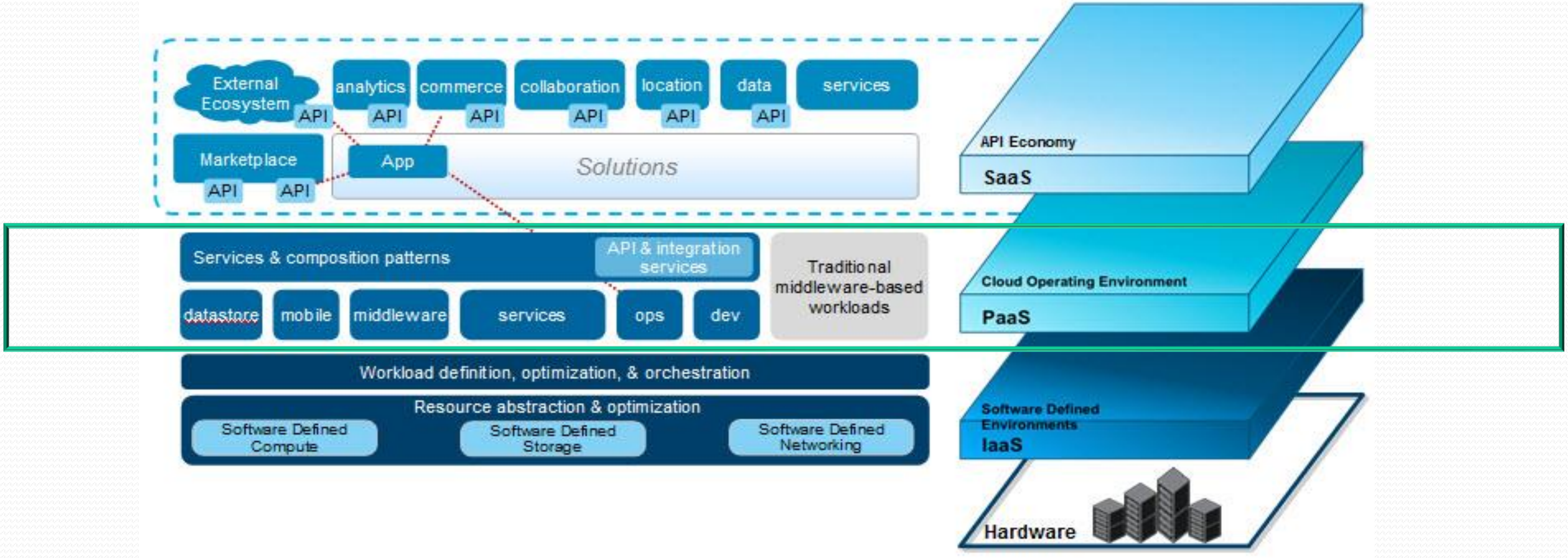
Cloud origin

- Now: GAE, Azure EC2, IBM SmartCloud, Oracle Cloud, Heroku, etc.
  - Dockers, micro-services

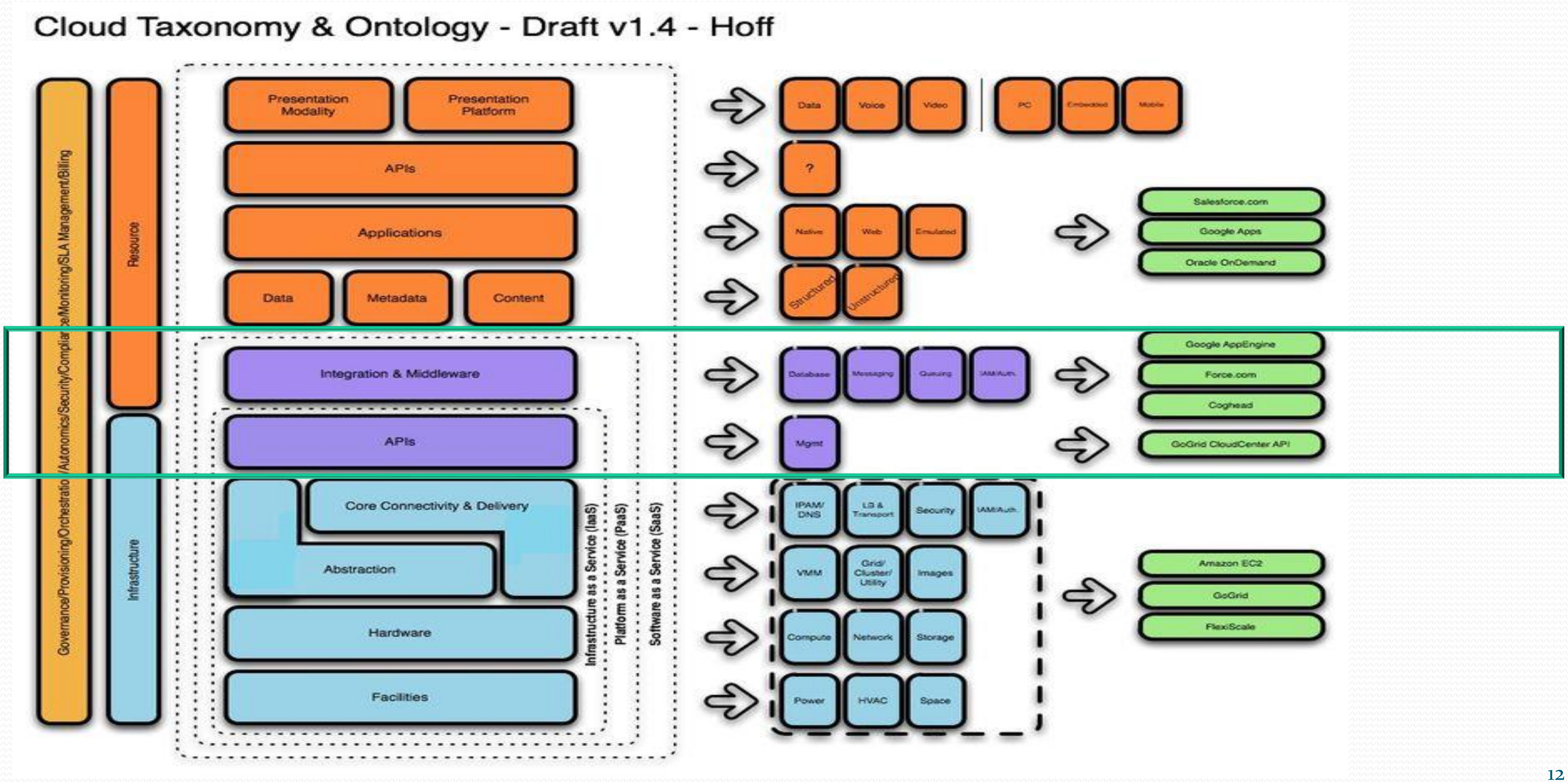
Cloud features :

- new API,
- storage,
- compute,
- Scalability (long term),
- Elasticity (short term),
- etc.

Architecture



Architecture



Architecture

- PaaS : platform as a service
  - Deployment of apps (web services, etc.) in extensible env.
  - OS+ App server (glassfish, jboss, etc.) + persistance layer + API
  - Ex: GAE, Windows Azure, openshift, etc.

- SaaS : software as a service
  - Service proposed to Customers (Dropbox, ?)

Deployment models

- Public Cloud: solutions open for public use with access over a network (Internet)
  - ex: Amazon, Microsoft, Google

- Private Cloud: private infrastructure available to a unique organisation.

  - Hardware, software have to be managed by the organisation.

  - Need of re-evaluating the required resources periodically and the Security issues after every modification

  - Loss of several advantages of Clouds: flexibility, scalability

Deployment models

- Hybrid Cloud :
  - Composed of 2 or more private, public clouds bound together(several providers)
  - Support several deployment models
  - Share the same advantages as public and private clouds (flexibility, scalability)
  - Sensitive data can be stored into the private part

Some Open source PaaS

	Year	sponsors	Languages
CLOUD FOUNDRY	2011	VMware	Spring,Rails, sinatra, node.js
OPENSHIFT PaaS by Red Hat Cloud	2011	Red hat	Express-ruby, PHP, python, flex, jboss, java EE6
WSO2 Stratos	2009	WSo2	Tomcat, jboss, java EE6
stackato	2012	HP	Java, Ruby, Perl, Java, etc

platform: Openstack

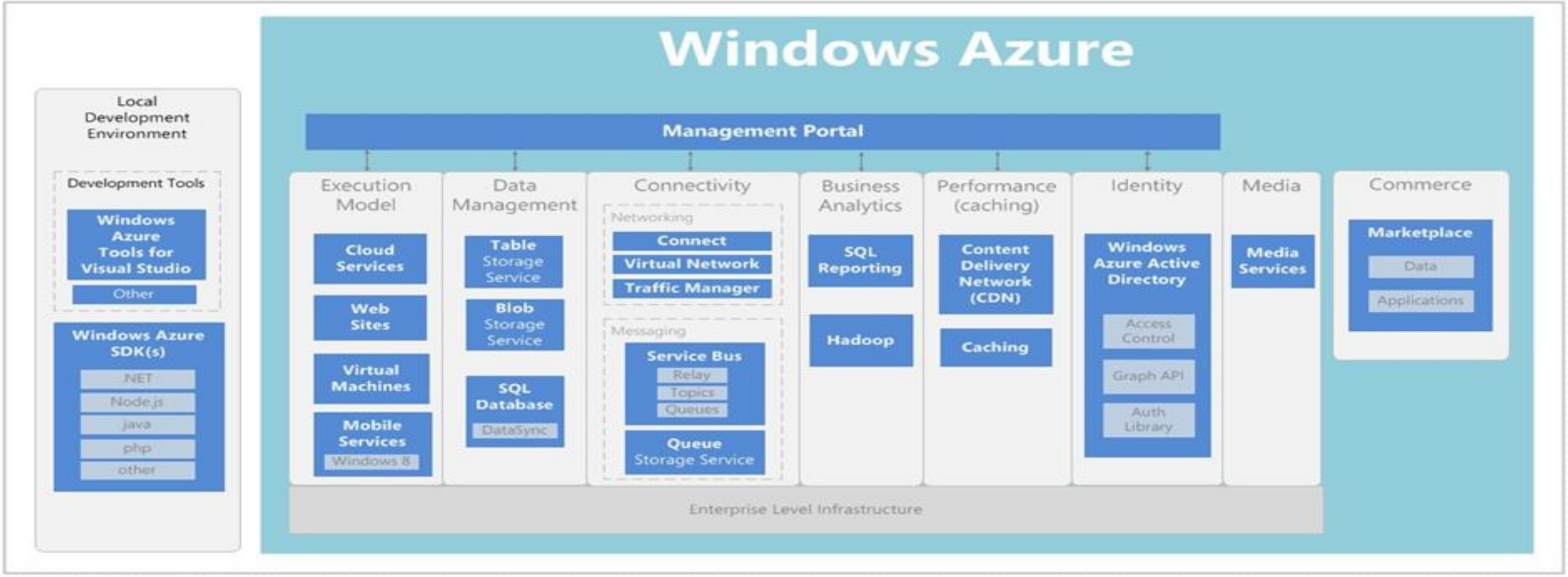
Cloud example:

Windows Azure insight

PaaS Windows Azure

- IaaS and SaaS layers not seen here
- Services of the PaaS layer :
  - Langages:
    - C# VB, Python, Java, PHP, Ruby, etc.
  - Type of Apps :
    - Web Services SOAP, REST, plain/text,
    - Web sites
  - Admin, performance analysis, interfaces, etc.

PaaS Windows Azure

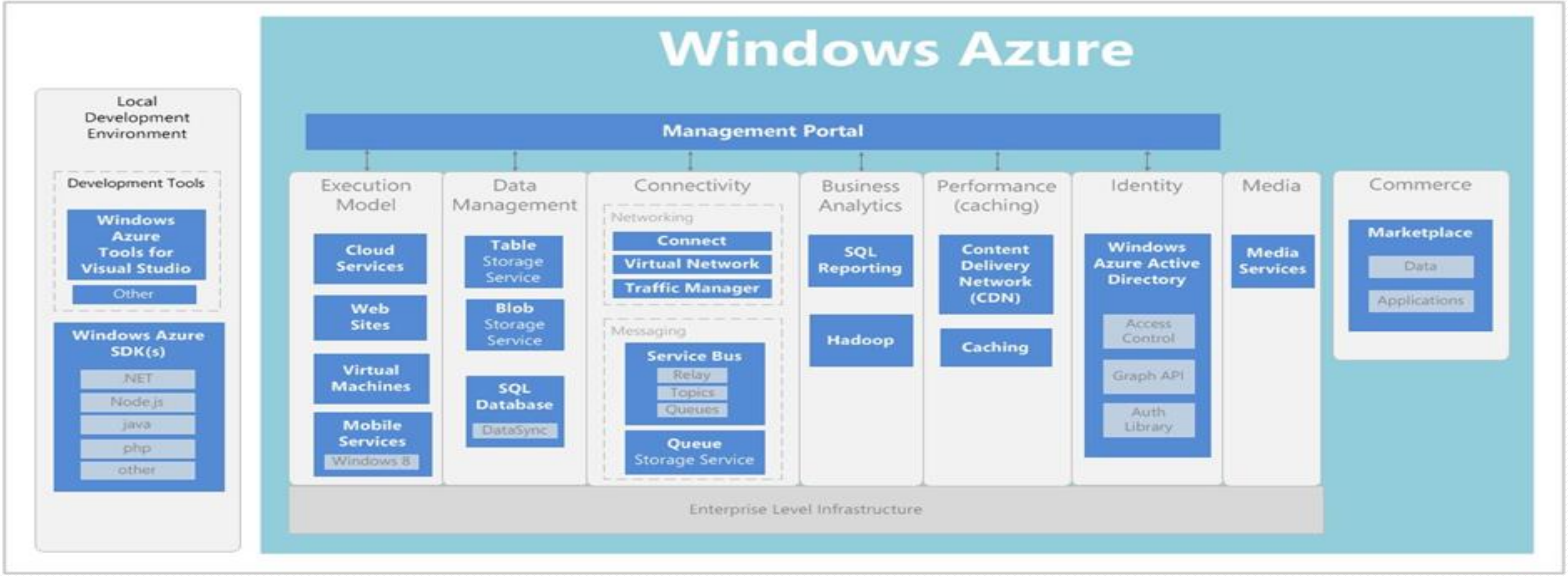


Service Bus: message queuing platform build by Azure that provides Relay and Brokered messaging capabilities

Identity/Acces control: manages access to service bus, supports protocols like OAuth v1 v2, Simple Web Tokens (SWT) for REST services, or SAML, WS-Federation et WS-Trust for SOAP services

Cloud services : SOAP Rest web services, web role, worker roles

PaaS Windows Azure



Blobs: blob files allowing to store files or meta-data
Table: non relational tables, fulfilled with entities,
Queue asynchronous FIFO between apps
Drive manage and configure vituel disks

PaaS Windows Azure

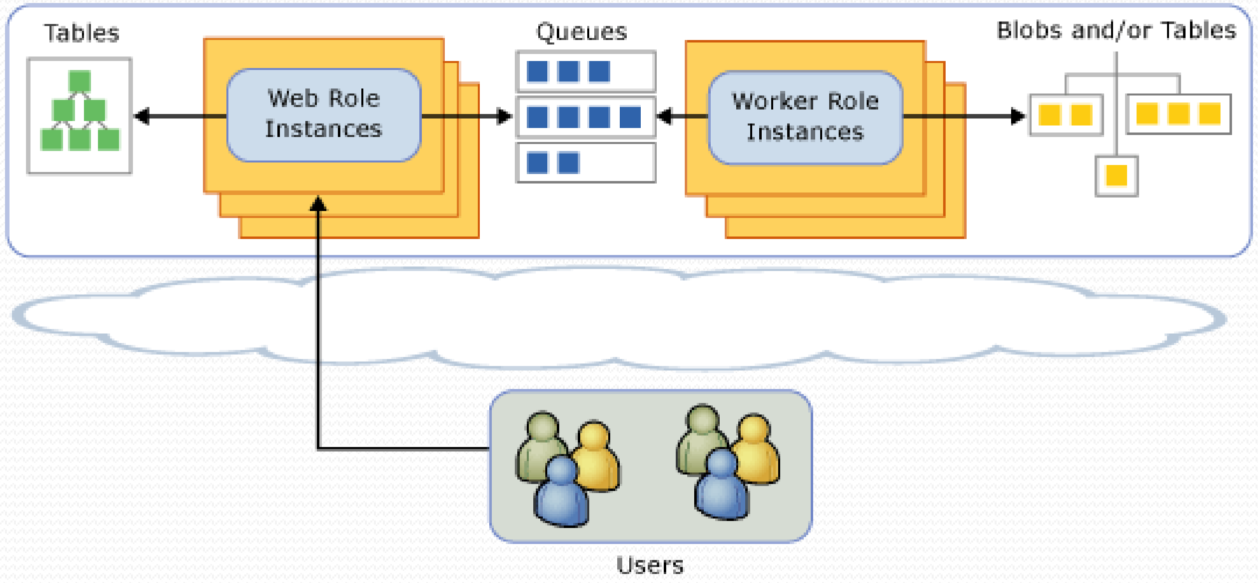
- Web and worker roles:
- Web Role:
  - Apps called with HTTP Requests / responses (Web pages, WCF Web services, etc.)
- Worker role:
  - Service running in the background. Cannot be called via HTTP
- Web services and workers can interact through Queues:
  - workers yield Data, Web services read it and answer

PaaS Windows Azure

Web and worker roles

- Web service can change of state
- Web and worker roles can be put in different VMs (manual distribution)

A massively scalable web app with background processing

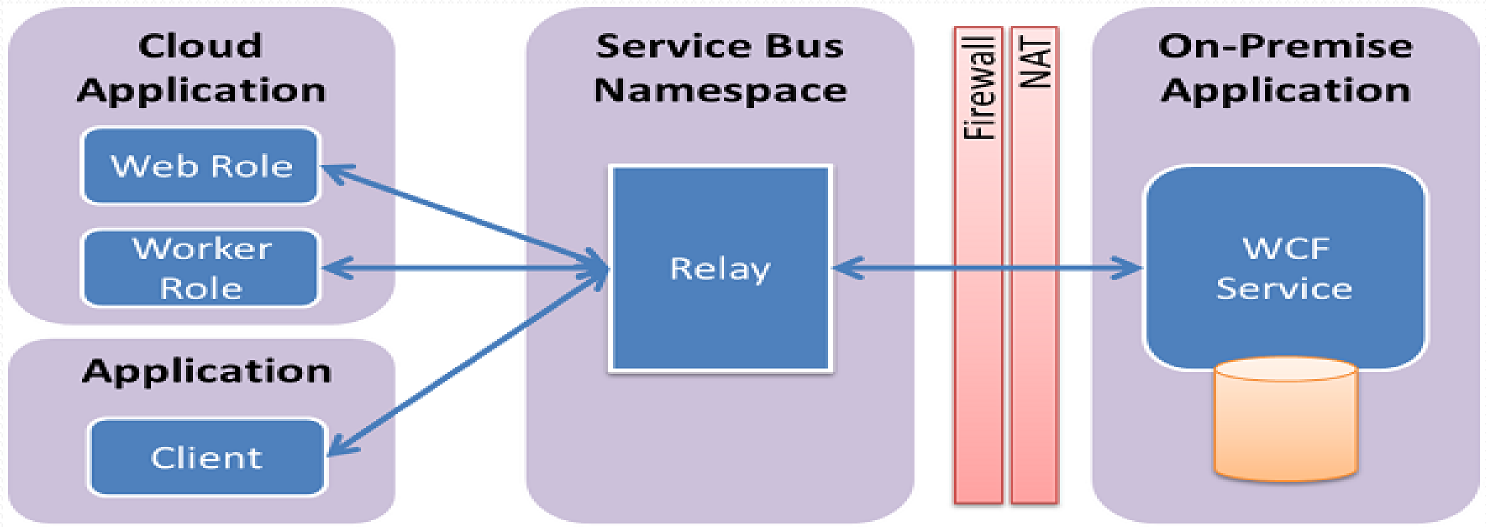


PaaS Windows Azure

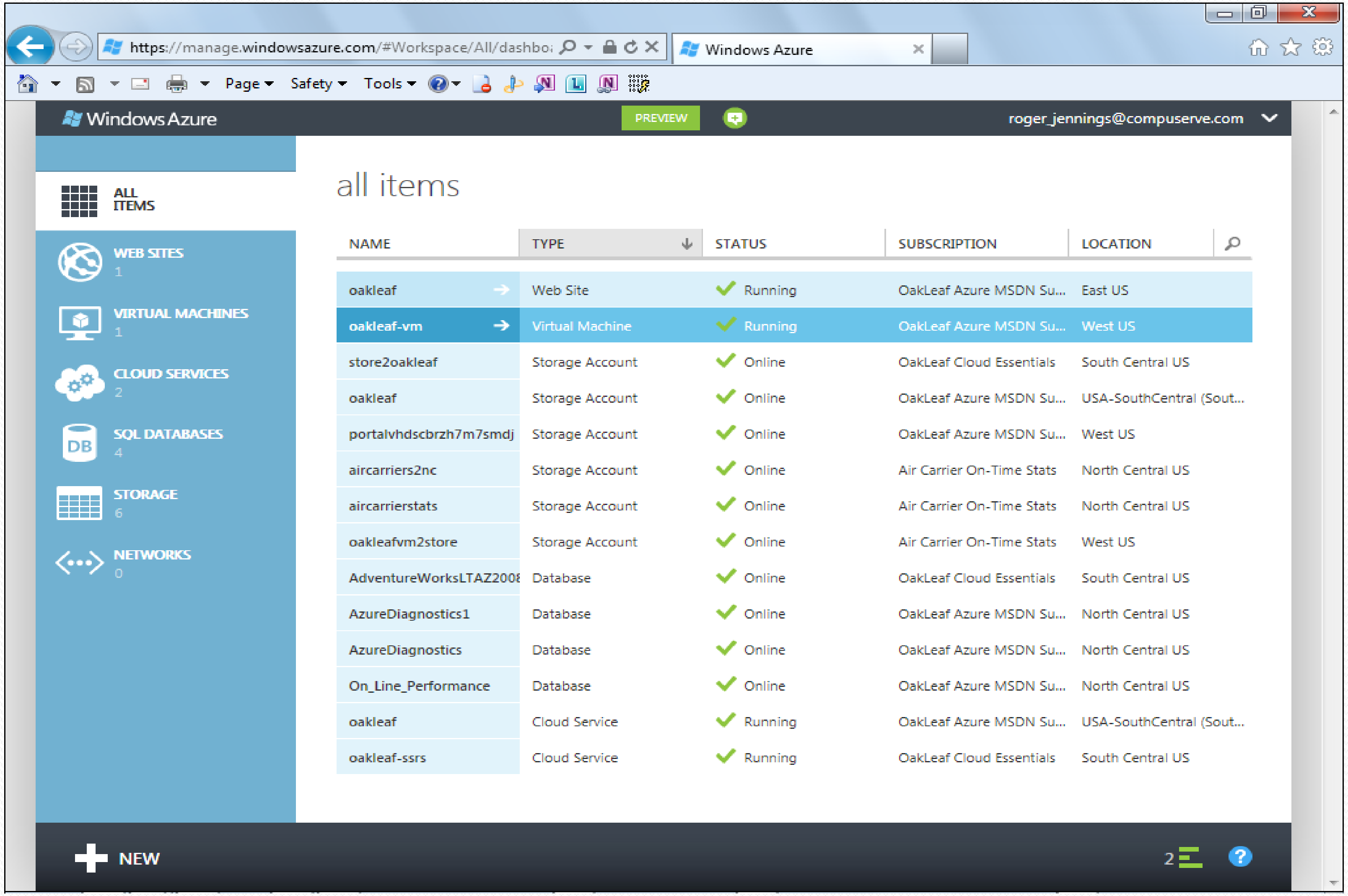
- Example with ServiceBus:

- Relay messaging: Relay between entitites:
- Build hybid apps partly deployed in Azure,
- The whole app is secured by the Relay

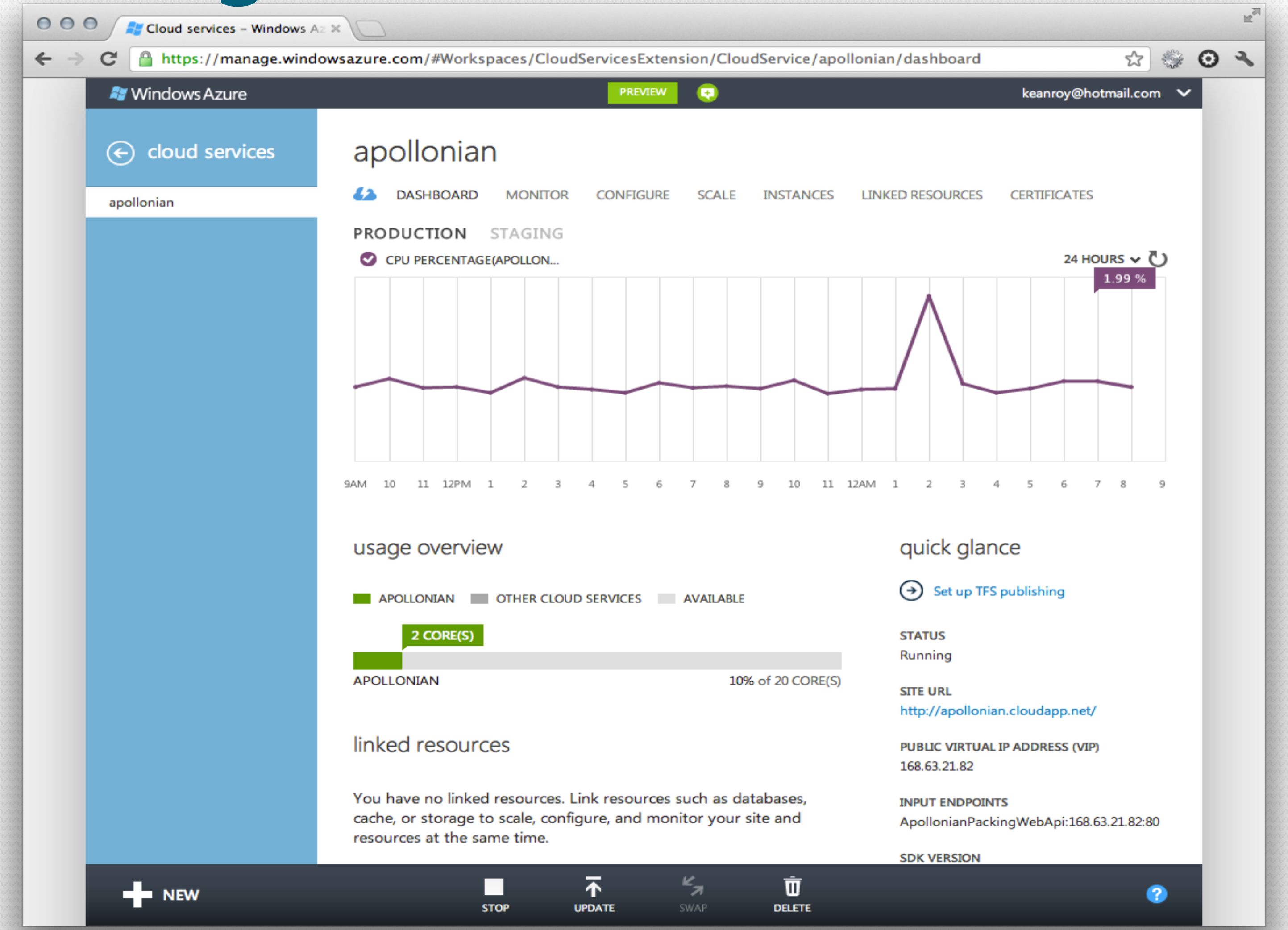
- https://www.windowsazure.com/en-us/develop/net/how-to-guides/service-bus-relay/



Azure Management Console



Azure Management Console



Apps localisation

Where is my M App ?

II Model based testing / clouds

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

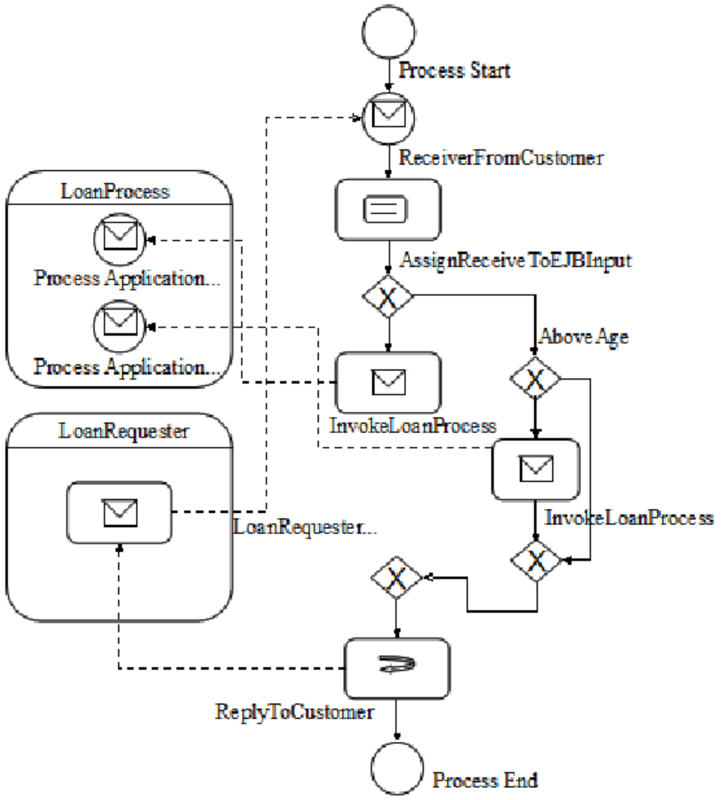
- Interoperability testing (betwen 2 services in different clouds, etc.)

- Third-party dependencies

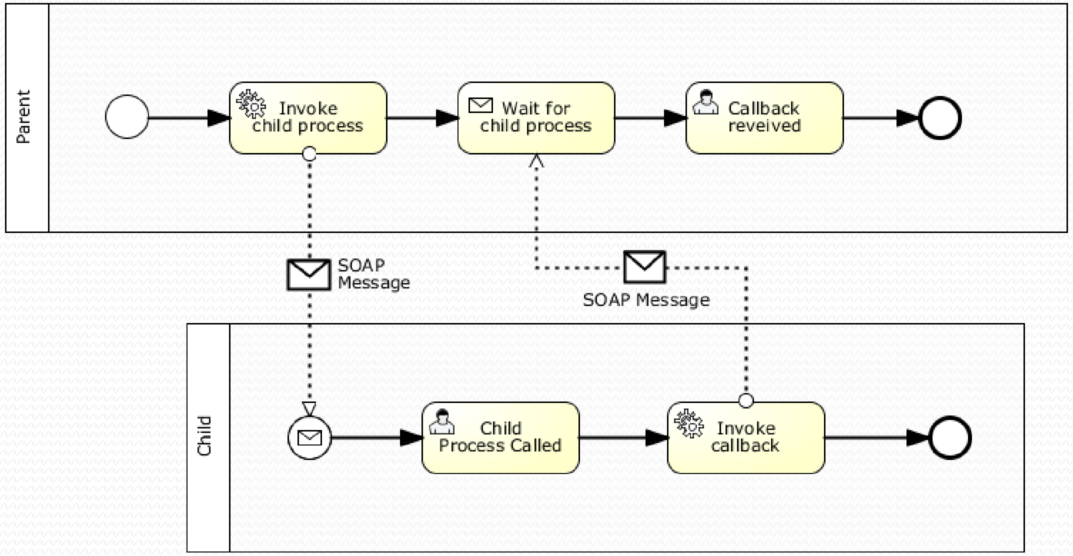
Models

High level languages (ws-BPEL, BPMN, etc.)

WS-BPEL



BPMN



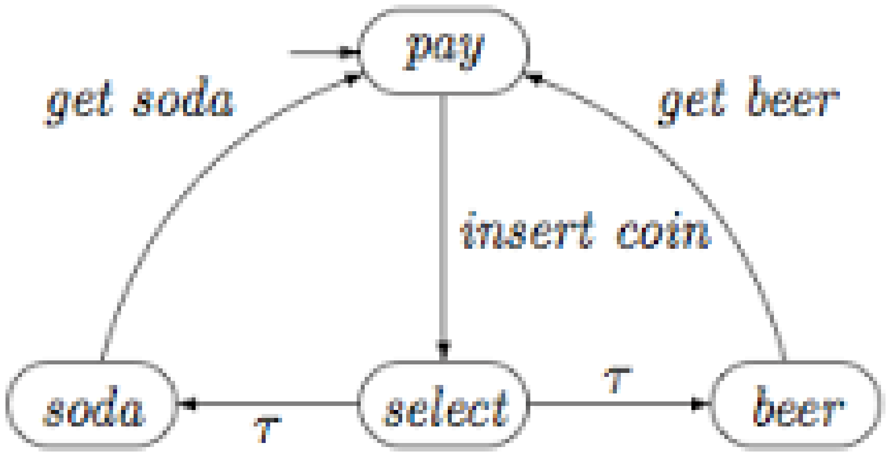
Formal Model based on transition systems

=>Formal models encoding the functionnal behaviours of WS, of composite WS

- Transition systems
- Transition labels: ! stands for emission and ? stands for reception
  - Supported by many tools

Model name ?

LTS



Formal Model based on transition systems

- Symbolic models:
  - Modelisation of parameters, data constraints

STG, IOSTS, EFSM

- Timed models:

Add the modelisation of time constraints (delays between two calls, etc.)

TA, TEFSM

IOSTS

- IOSTS (IOLTS) considered here

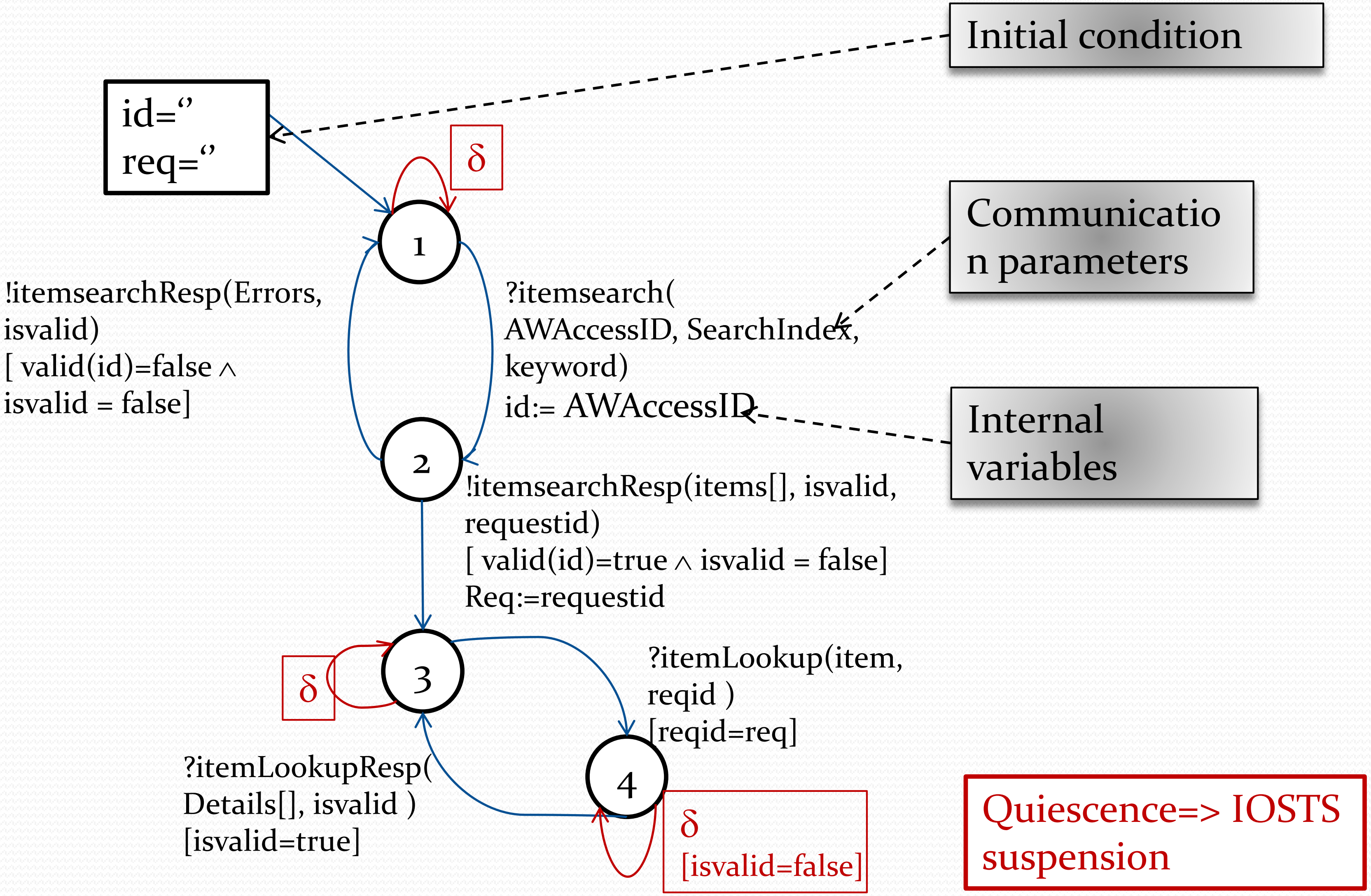
Why ?

  - IOSTS (and IOLTS) can be represented with graphs and with process-algebraic behaviour expression [Tre96]
  - ?req1;!resp1 | ?req2; !resp2
  - Advantage: model transformations, modifications can be given with inference rules

$$\frac{\text{If condition}}{\text{Then action}}$$

IOSTS

Example:
Amazon Web service



IOSTS->IOLTS

- Express behaviours that may be infinite

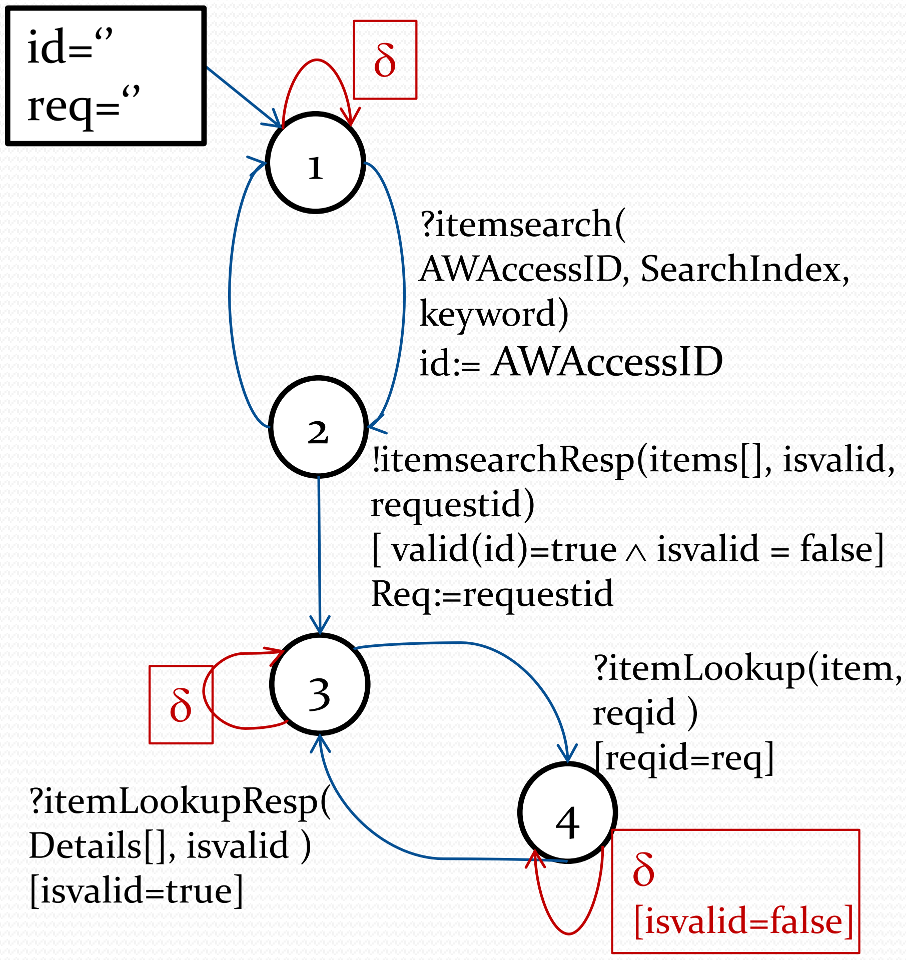
$\rightarrow$ underlying (valued) model : IOLTS semantic

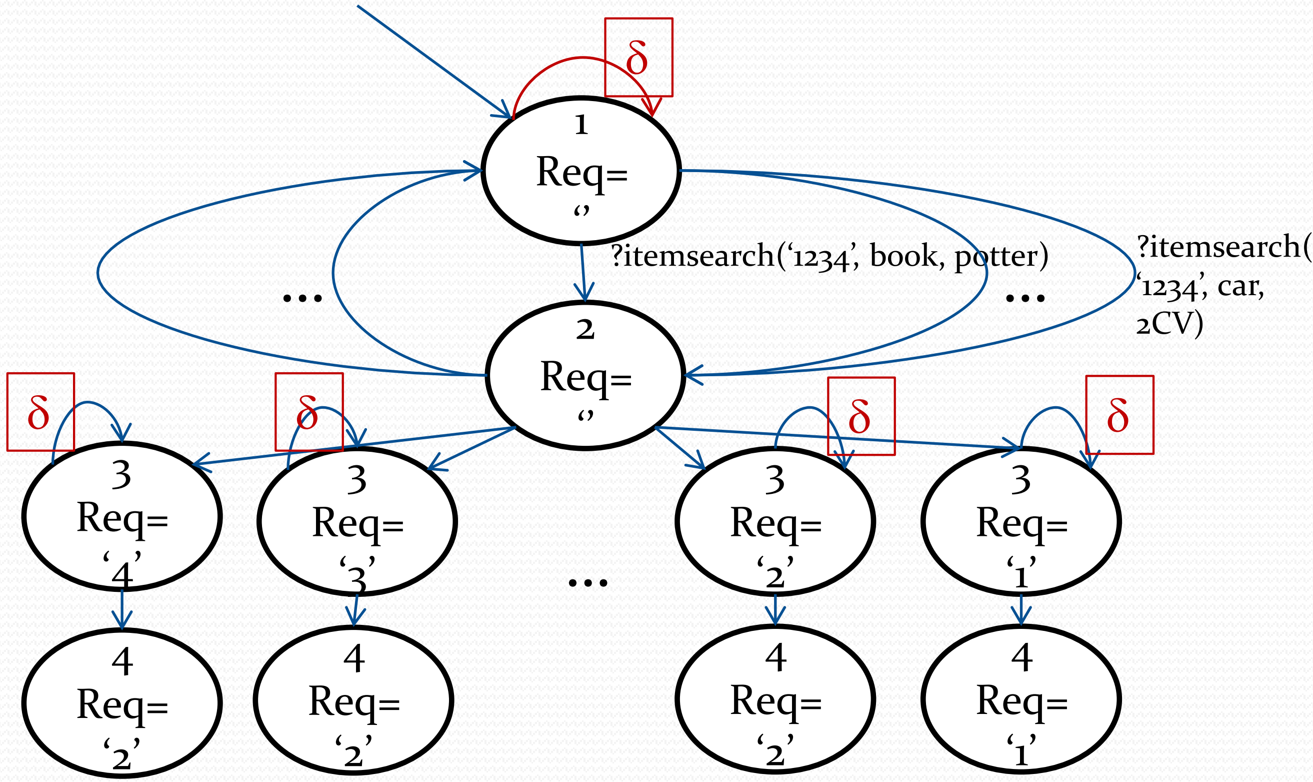
Definition (ioLTS semantics) The semantics of an ioSTS $S = \langle L, l_0, V, V_0, I, \Lambda, \rightarrow \rangle$ is the ioLTS $||S|| = \langle Q, q_0, \Sigma, \rightarrow \rangle$ where:

- $Q = L \times D_V$ is the set of states;
- $q_0 = (l_0, V_0)$ is the initial state;
- $\Sigma = \left\{ a(p), q \mid a(p) \in \Lambda, q \in D_p \right\}$ is the set of valued symbols. Σ^I is the set of input actions and Σ^O is the set of output ones,
- $\rightarrow$ is the transition relation $Q \times \Sigma \times Q$ deduced by the following rule:

$$\frac{l_1 \xrightarrow{a(p),G,A} l_2, q \in D_p, v \in D_V, v' \in D_V, v \cup q \models G, v' = A(v \cup q)}{(l_1, v) \xrightarrow{a(p),q} (l_2, v')}$$

IOSTS->IOLTS


id=‘’
req=‘’
δ
1
?itemsearch(
AWAccessID, SearchIndex,
keyword)
id:= AWAccessID
2
!itemsearchResp(items[], isvalid,
requestid)
[valid(id)=true ∧ isvalid = false]
Req:=requestid
δ
3
?itemLookup(item,
reqid)
[reqid=req]
?itemLookupResp(
Details[], isvalid)
[isvalid=true]
4
δ
[isvalid=false]

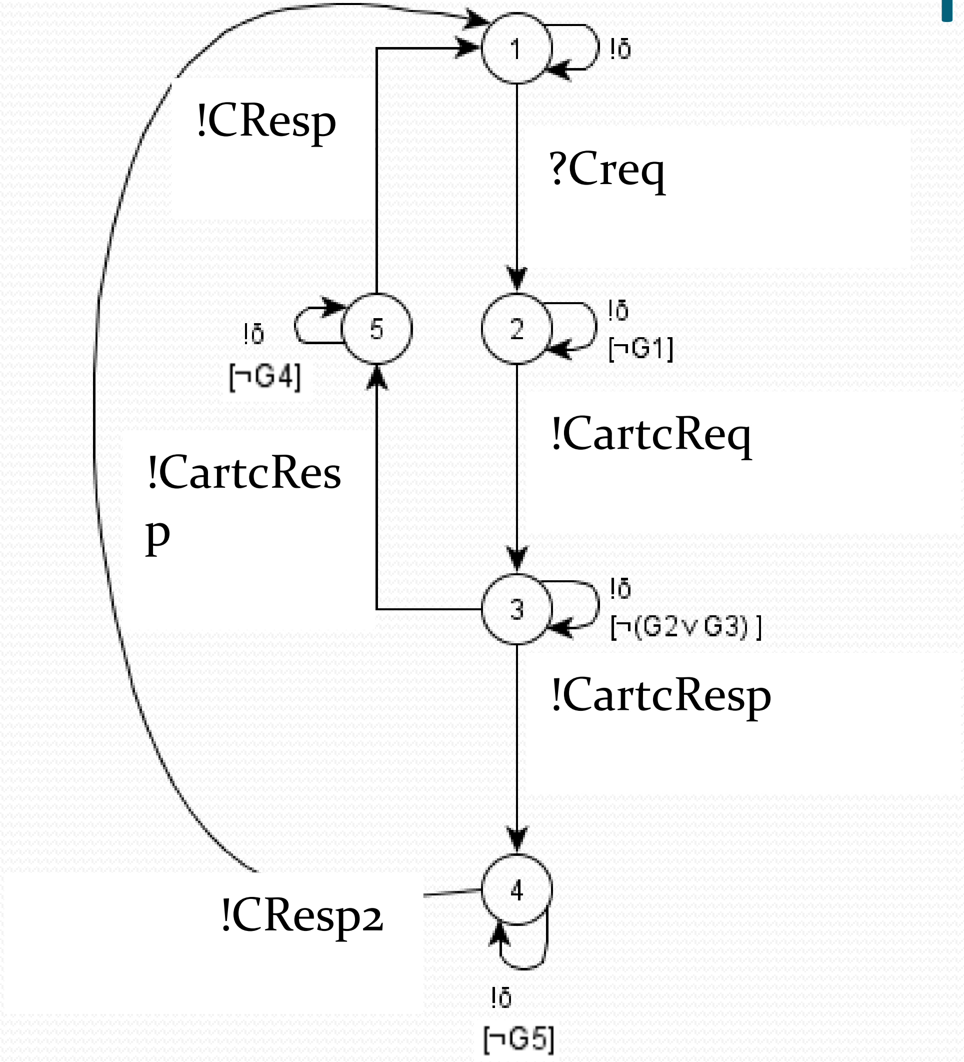

δ
1
Req=
‘’
?itemsearch(‘1234’, book, potter)
?itemsearch(
‘1234’, car,
2CV)
…
…
2
Req=
‘’
δ
δ
δ
δ
3
Req=
‘4’
3
Req=
‘3’
…
3
Req=
‘2’
3
Req=
‘1’
4
Req=
‘2’
4
Req=
‘2’
4
Req=
‘2’
4
Req=
‘1’

Web service composition modelisation

- Desc. of the services, parameters, correlations, etc.

- Example:

Web service composition modelisation



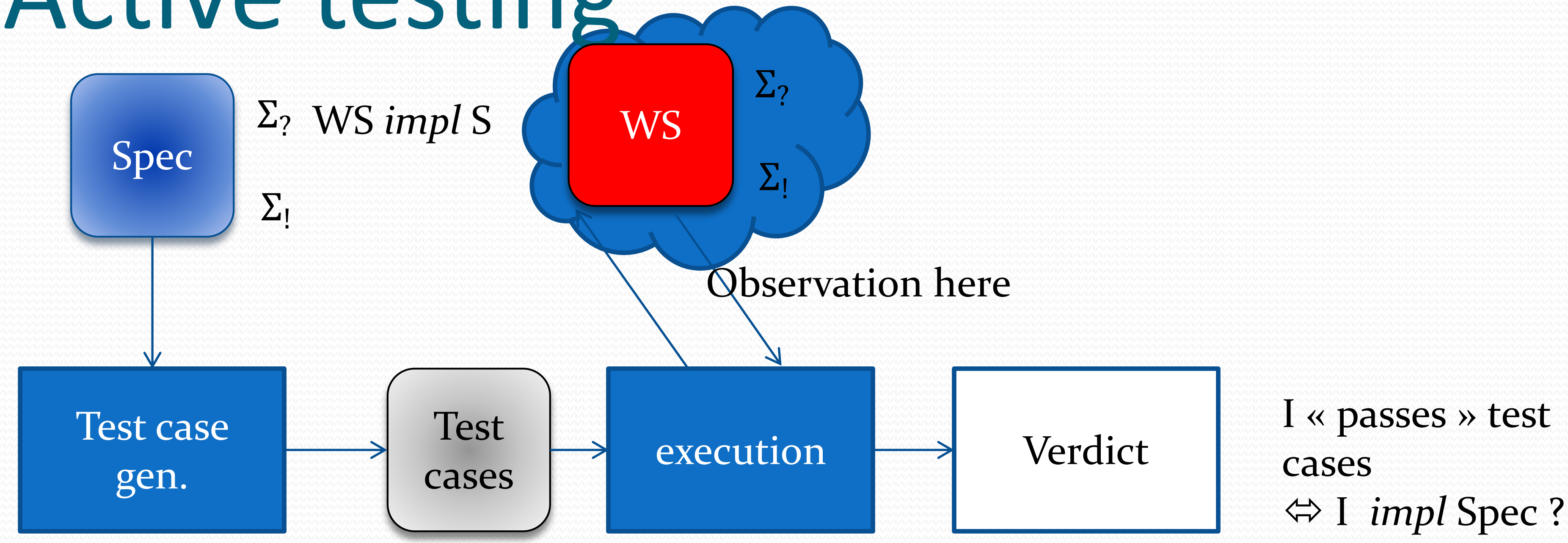
Symbol	Message	Guard	Update
?Creq	?ConnectReq(account,from, to,coor)	from="Env"∧ to="S"∧ $corr = \{account\}$]	{a:=account,c1=coor}
!CartcReq	!CartCreateReq(key, from, to, coor)	$G1$ =[from="S"∧to="A"∧ $coor = \{a, key\}$ ∧key=valid(a)]	c2=coor
!CartcResp	!CartCreateResp(resp, idc, from, to, coor)	[from="A"∧to="S" ∧resp ≠"invalid"∧coor=c2]	cartid:=idc
!CartcResp2	!CartCreateResp(resp, idc, from, to, coor)	G2=[from="A"∧to="S" ∧resp="invalid"∧coor=c2]	cartid:=""
!CResp	!ConnectResp(resp, from, to, coor)	G4=[from="A"∧ to="S"∧ resp="error"∧corr=c1]	
!CResp2	!ConnectResp(resp, from, to, coor)	G5=[from="A"∧ to="S"∧ r="connected"∧coor=c1]	

Model-based testing in clouds

- Type of testing
  - active
  - passive, Runtime Verification

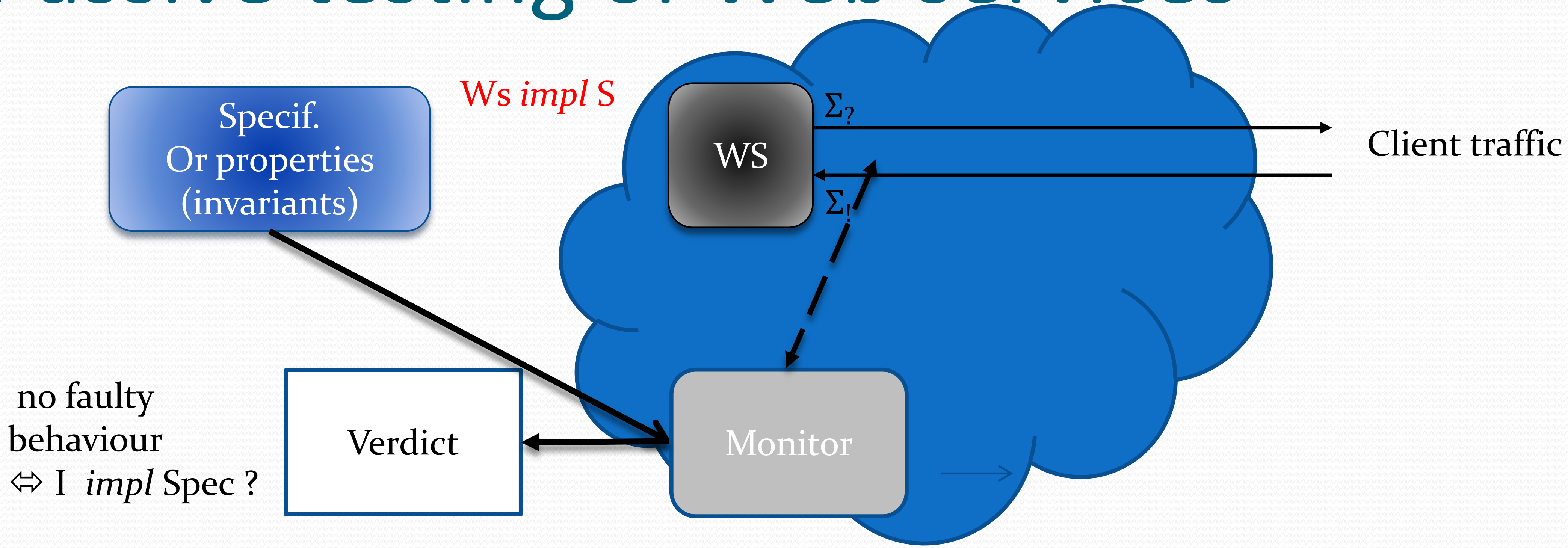
- Security, robustness, conformance etc.

Active testing



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

Passive testing of Web services



Monitoring of web service compositions
No direct interaction with WS

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

Passive testers

- Offline modes
- Trace collection
- Trace of WS belongs to traces of S?
- Or property traces ?

- Online mode

Online Tester based on a « checker state algorithm »

- Simplified algo:
  - Stores the specification states reached in L
- Message observed m =>
  - Covers specification (or derived model) from states of L with m -> set of states S'
  - Check whether the states of S' are « bad » states => fail
  - Check whether the states of S' are « good » states => invariant holds
    - L = L'
    - And so forth

Runtime verification of Web services

- Comes from verification
- Verification of prop. at runtime (during execution)

- Prop. in logics (LTL, CTL, nomad, etc.), automata, etc.

- Check whether prop. hold at runtime (passively)

  - Generation of a Monitor model from properties
  - Monitor + passive tester -> verdicts: violation of prop, etc.

- [CPFC10][RPG06] [SC14], etc.

Observations, testing architectures

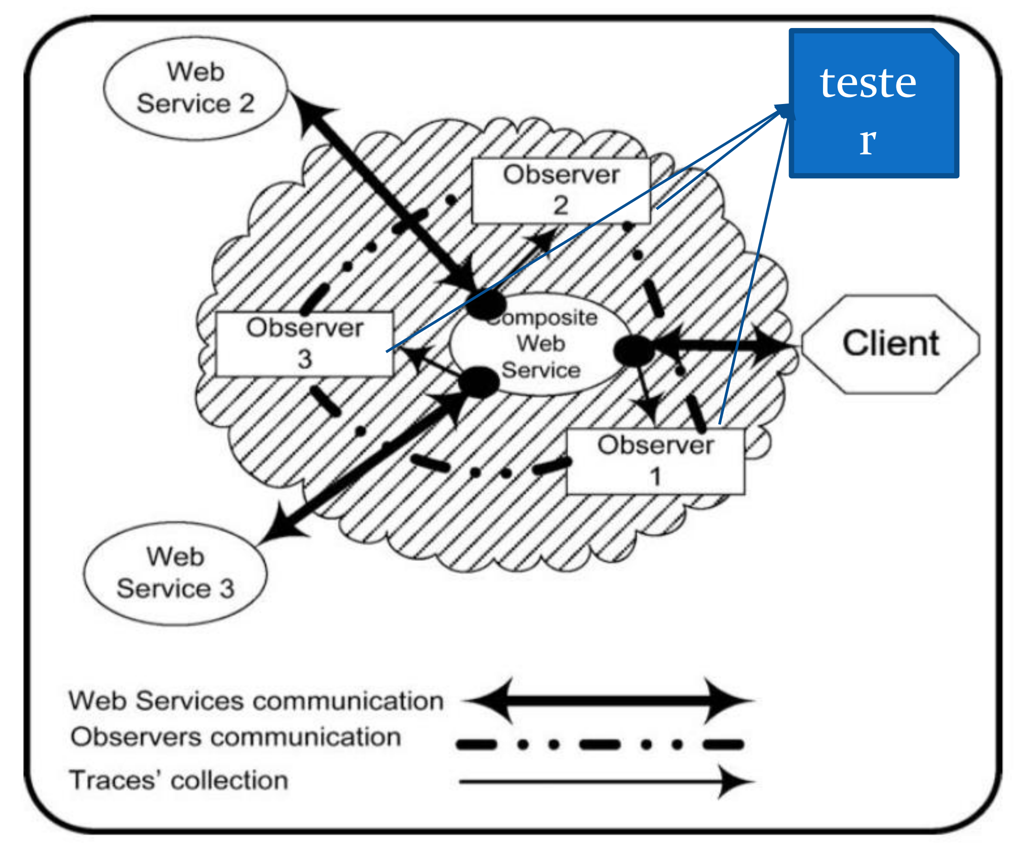
- Collect of the WS requests, responses in Clouds

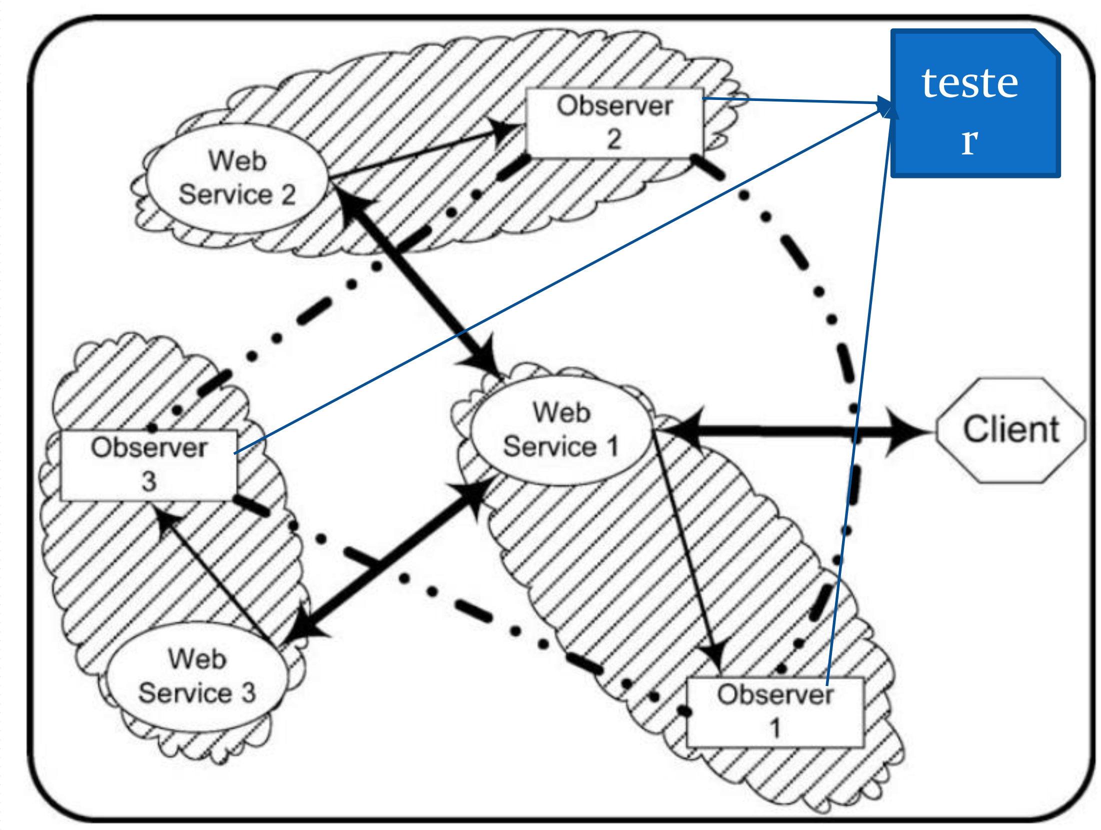
  - With network sniffers? (when VM are available)
  - By modifying cloud engines ?

    ⇒ Difficult

  - By instrumentation of the WS codes
  - With Agents: SNMP agent, mobile agents

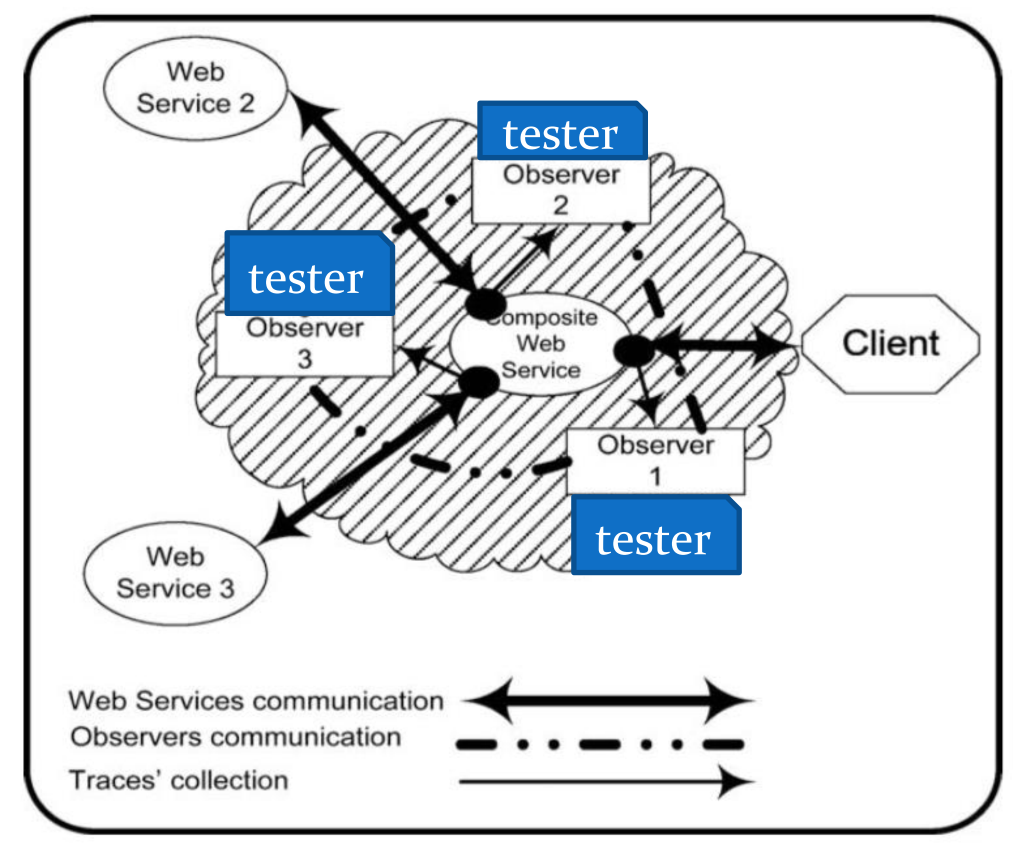
Observations, testing architectures

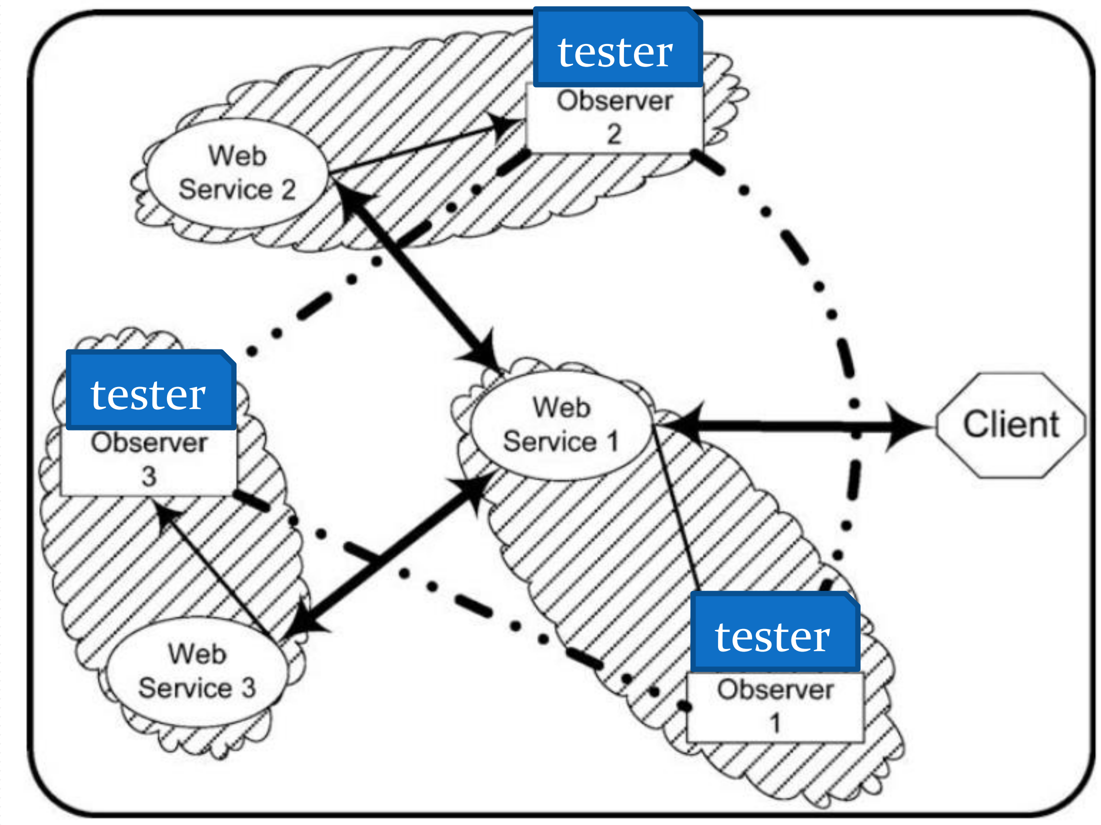




- [BDSG09] [SP15]

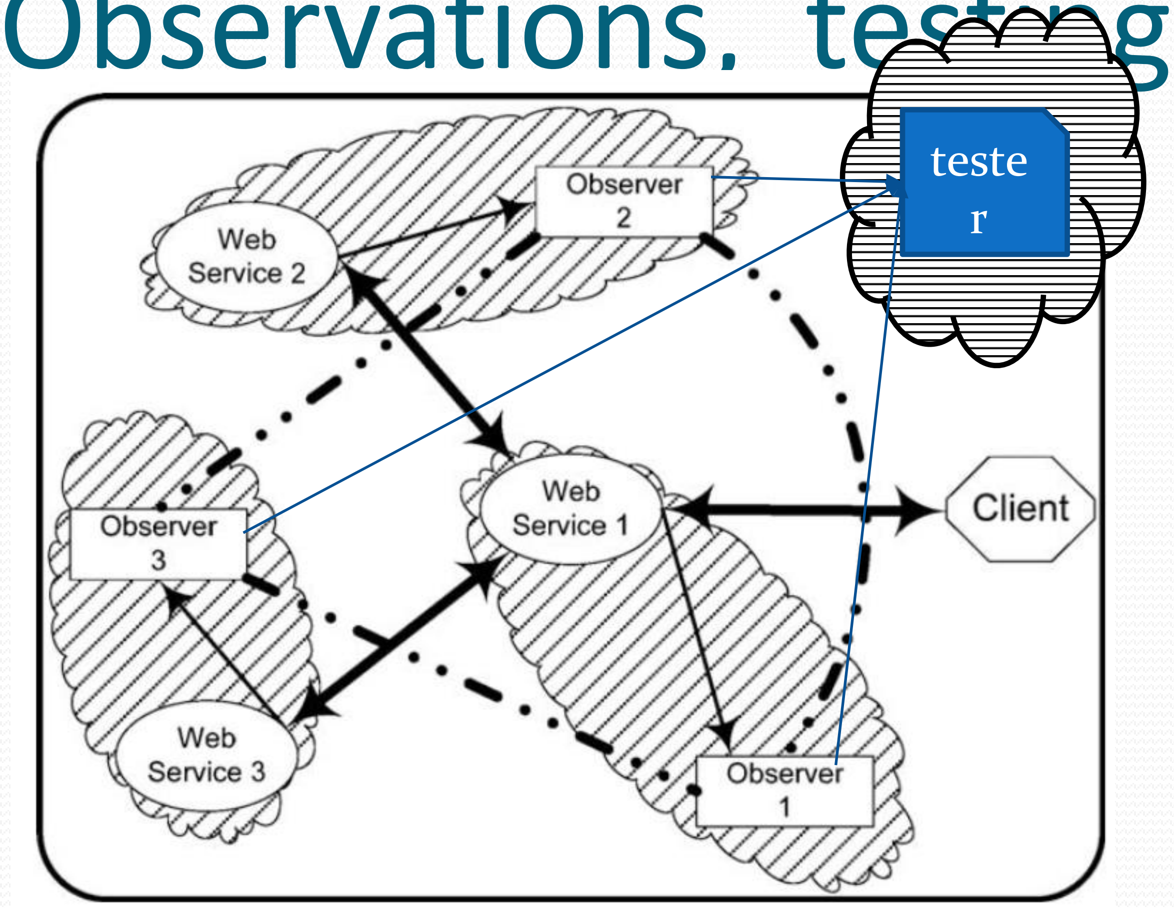
Observations, testing architectures

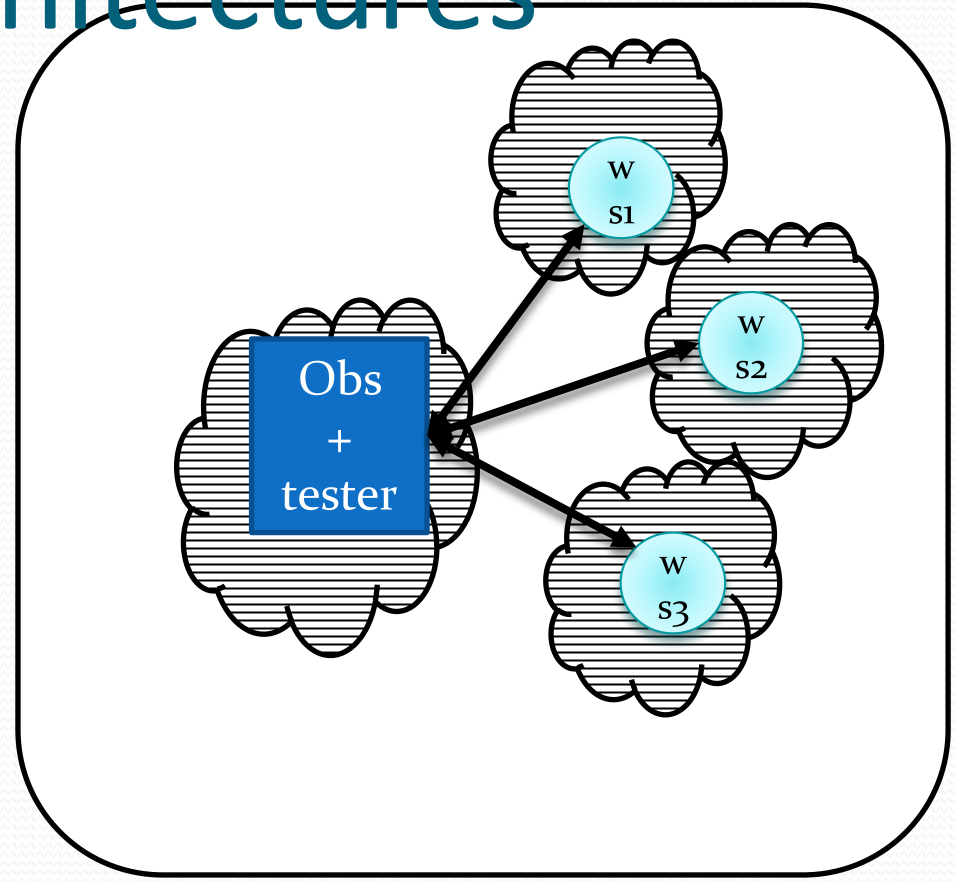




- [BDSG09] [SP15]

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.

Testing in Clouds issues

4. Message receipt modes
   - Synchronous mode ? No
   - Clouds => delays => asynchronous mode is closer to reality [NKRW11]
   - "Asynchronous communication delays obscure the observation of the tester"
   - Loss of messages, interleaving, delays (HTTP timeouts, etc.)-> see [PYL03] [NKRW11] , etc.

=> Different implementation relations

- Preorder
- ioco -> $ioco_U$ (under-specified models) [VRT03], etc.

=> Show that you have Finite test case number / sound test algorithms

- WS methods composed of parameters -> difficult to build exhaustive test suite
- -> need of test assumptions

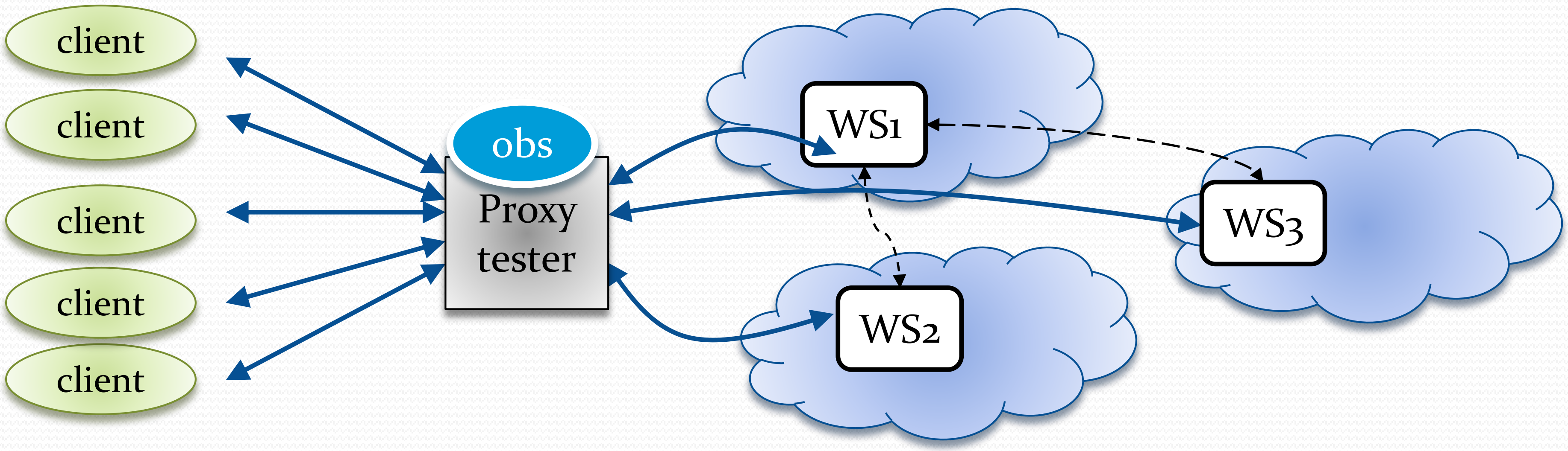
III Testing in clouds example

Passive testing with proxy-tester

Passive testing with proxy-testers

[S11d] [SP15]

- Proxy-testing principle

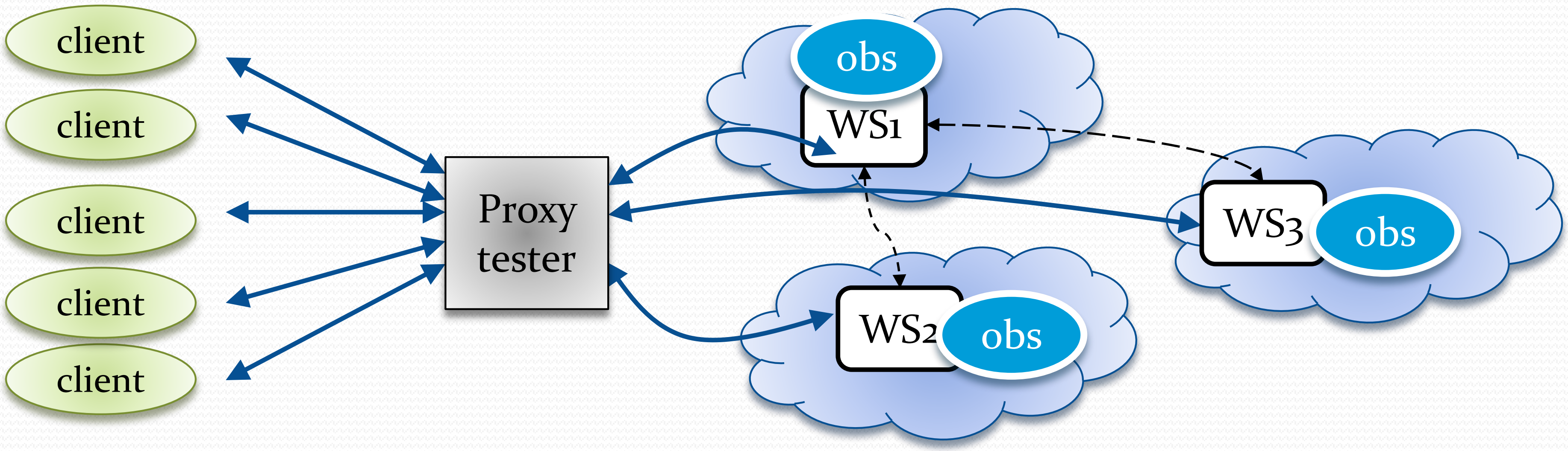


- Assumptions: message redirection to proxy (possible in practice), message synchronisation (light protocol to order messages, network latency << quiescence obs.)

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]

- Passive testing with proxy concept ? =>
  1. passive tester algorithm
  2. + automatic gen. of proxy-tester models for checking whether ioco holds

- Proxy-tester model to express message exchanged
- between client <-> Web services
- among Web service

- Proxy-tester model generated from specification

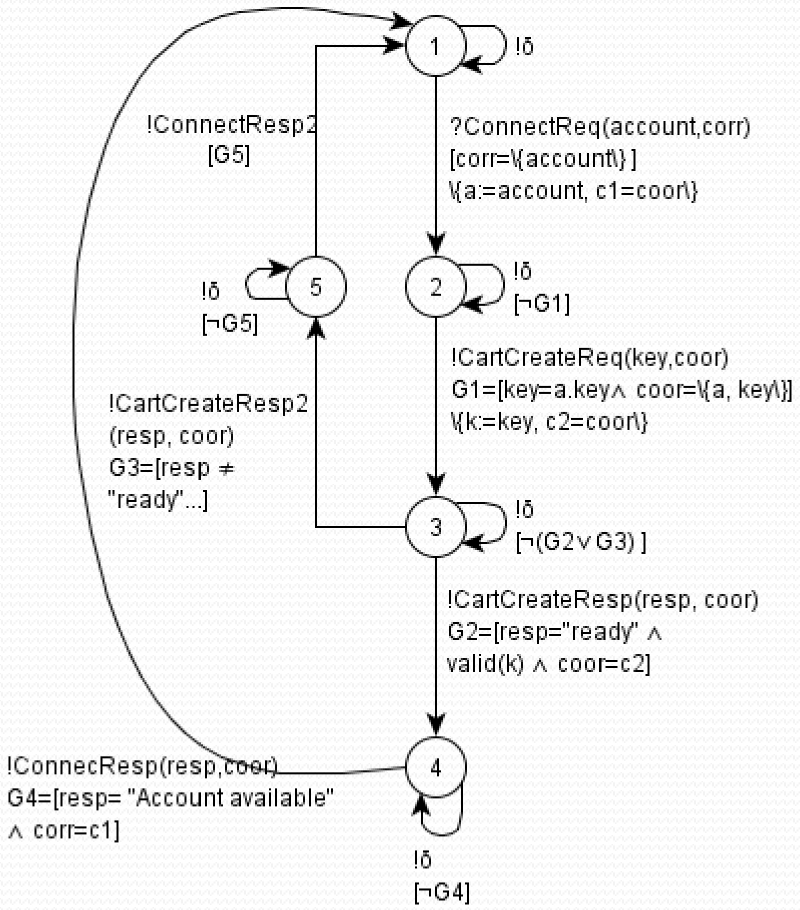
IOSTS canonical tester

Definition | **(ioSTS Canonical Tester).** Let $S = \langle L_S, l_S^0, V_S, V_S^0, I_S, \Lambda_S, \rightarrow_S \rangle$ be an ioSTS and $\Delta(S)$ be its suspension. The Canonical tester of S is the ioSTS $Can(S) = \langle L_S \cup LF_{Can(S)}, l_S^0, V_S, V_S^0, I_S, \Lambda_{refl(S)}, \rightarrow_{Can(S)} \rangle$ such that $LF_{Can(S)} = \{Fail\}$ is the Fail location set composed here of the $Fail$ location. $\rightarrow_{Can(S)}$ is defined by the rules:

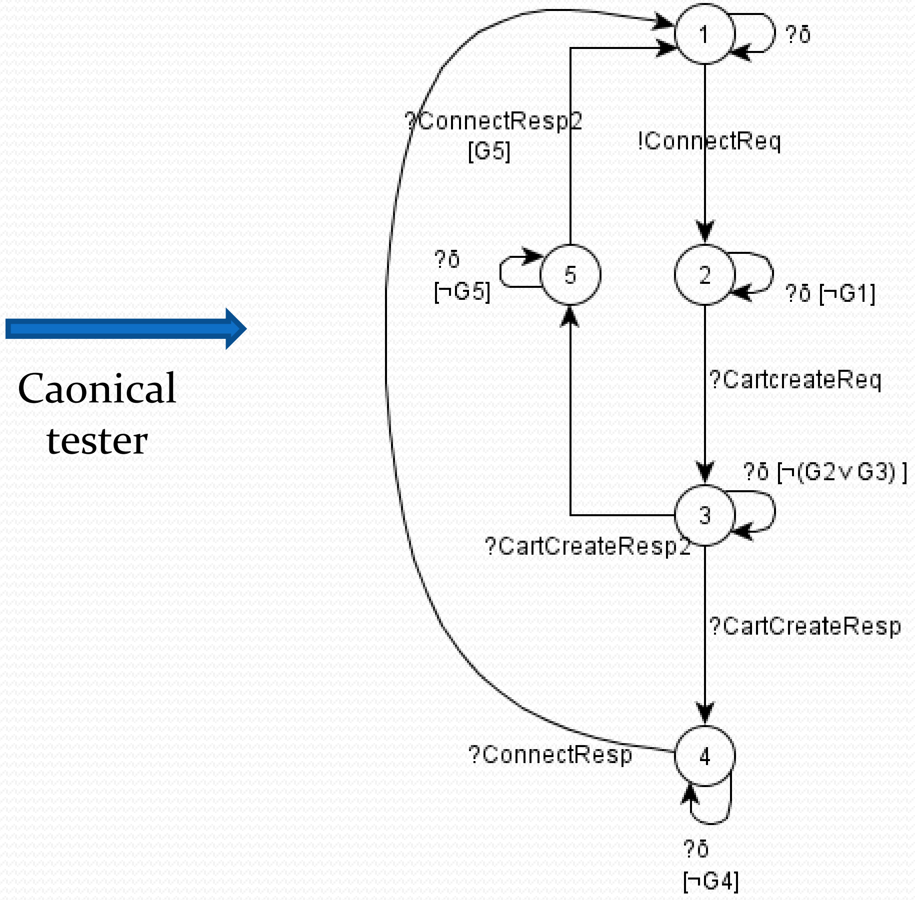
(keep $\mathcal{S}$ transitions) : $\dfrac{t \in \rightarrow_{\Delta(\mathcal{S})}}{t \in \rightarrow_{Can}}$

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

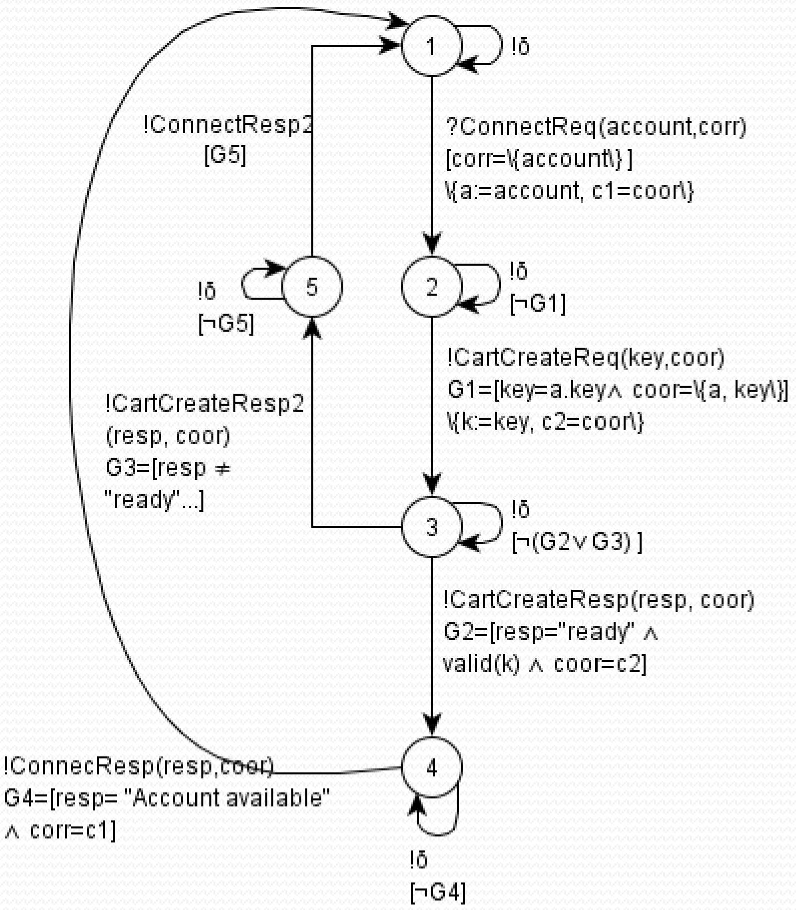
IOSTS canonical tester



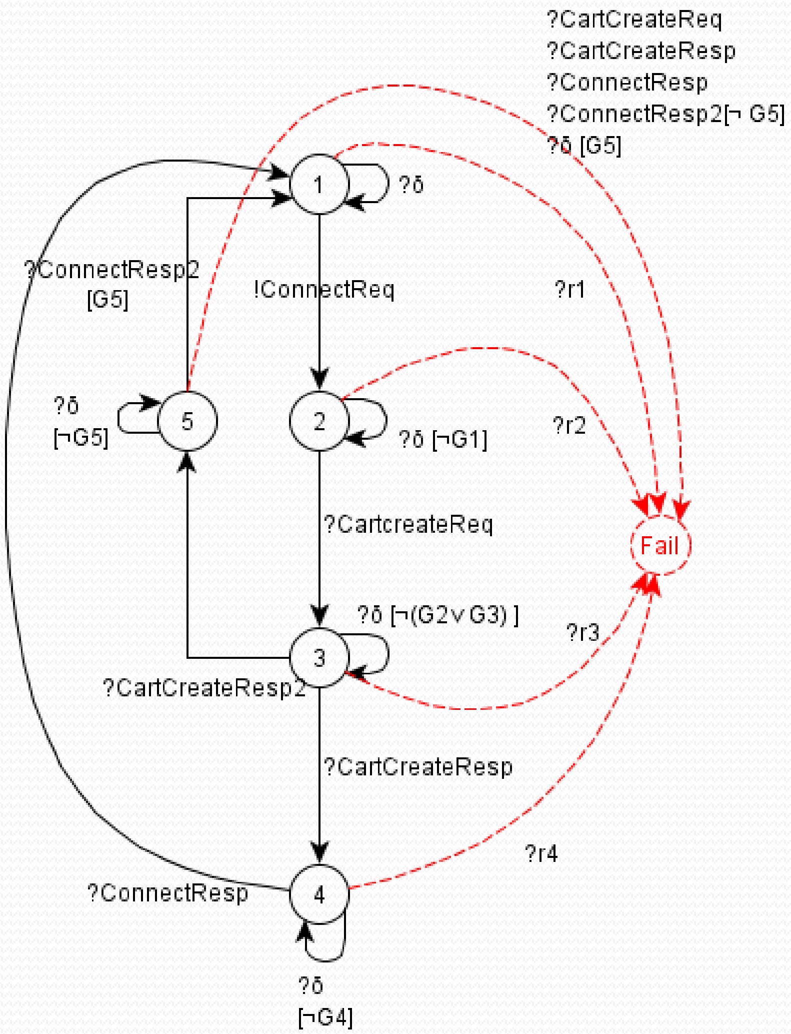
Caonical tester



IOSTS canonical tester



Caonical tester



Proxy-tester model gen.

Definition (Proxy-tester) The Proxy-tester of the ioSTS $S=\langle L_S, l_S^0, V_S, V_S^0, I_S, \Lambda_S, \rightarrow_S \rangle$ is the ioSTS $Pr(Can(S))$ where Pr is an ioSTS operation such that

$Pr(Can(S)) =_{def} \langle L_P \cup LF_P, l^0_{Can(S)}, V_{Can(S)} \cup \{side, pt\}, V^0_{Can(S)} \cup \{side := "", pt := ""\}, I_{Can(S)}, \Lambda_P, \rightarrow_P \rangle$.

$LF_P = LF_{Can(S)} = \{Fail\}$ is the Fail location set. L_P, Λ_P and $\rightarrow_P$ are constructed with the following rules:

CLient to WS

$$\frac{l_1 \xrightarrow{!a(p),G,A}_{Can(S)} l_2, l_2 \notin LF_{Can(S)}}{l_1 \xrightarrow{?a(p),G,A \cup \{p_t := p, side := ""\}}_P (l_1, l_2, a(p), G) \xrightarrow{!a(p),\{(x:=x)_{x \in V_{Can(S)}}, side := "Can"\}}_P l_2}$$

WS to Any

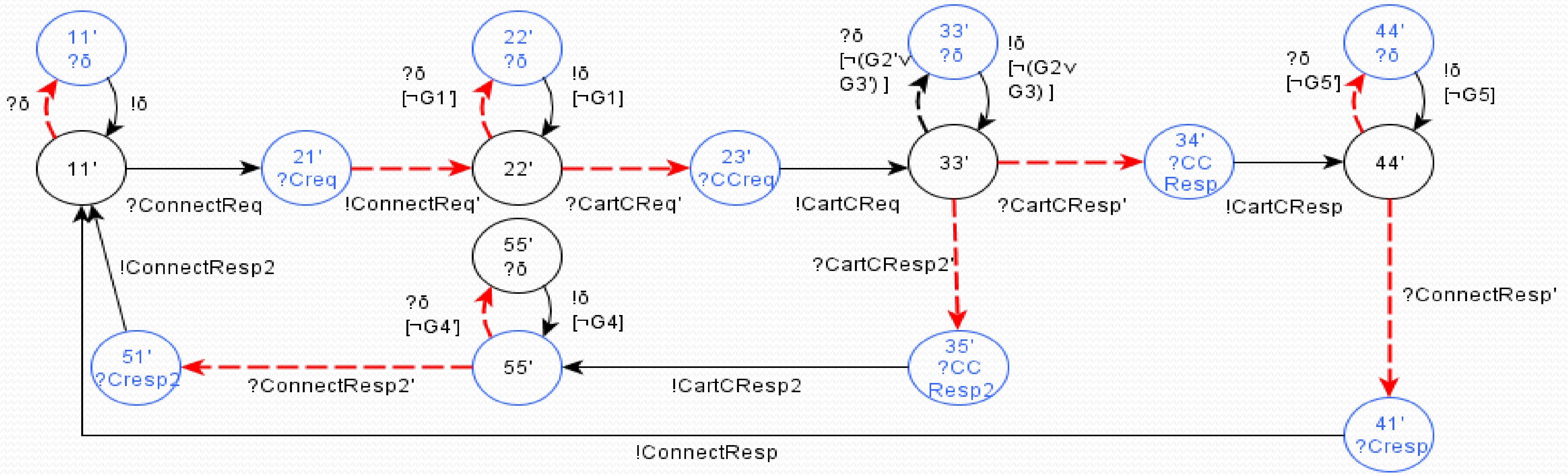
$$\frac{l_1 \xrightarrow{?a(p),G,A}_{Can(S)} l_2, l_2 \notin LF_{Can(S)}}{l_1 \xrightarrow{?a(p),G,A \cup \{p_t := p, side := "Can"\}}_P (l_1, l_2, a(p), G) \xrightarrow{!a(p),\{(x:=x)_{x \in V_{Can(S)}}, side := ""\}}_P l_2}$$

Wrong behaviour

$$\frac{l_1 \xrightarrow{a(p),G,A}_{Can(S)} l_2, l_2 \in LF_{Can(S)}}{l_1 \xrightarrow{a(p),G,A \cup \{p_t := p, side := "Can"\}}_P l_2}$$

Proxy-tester model gen.

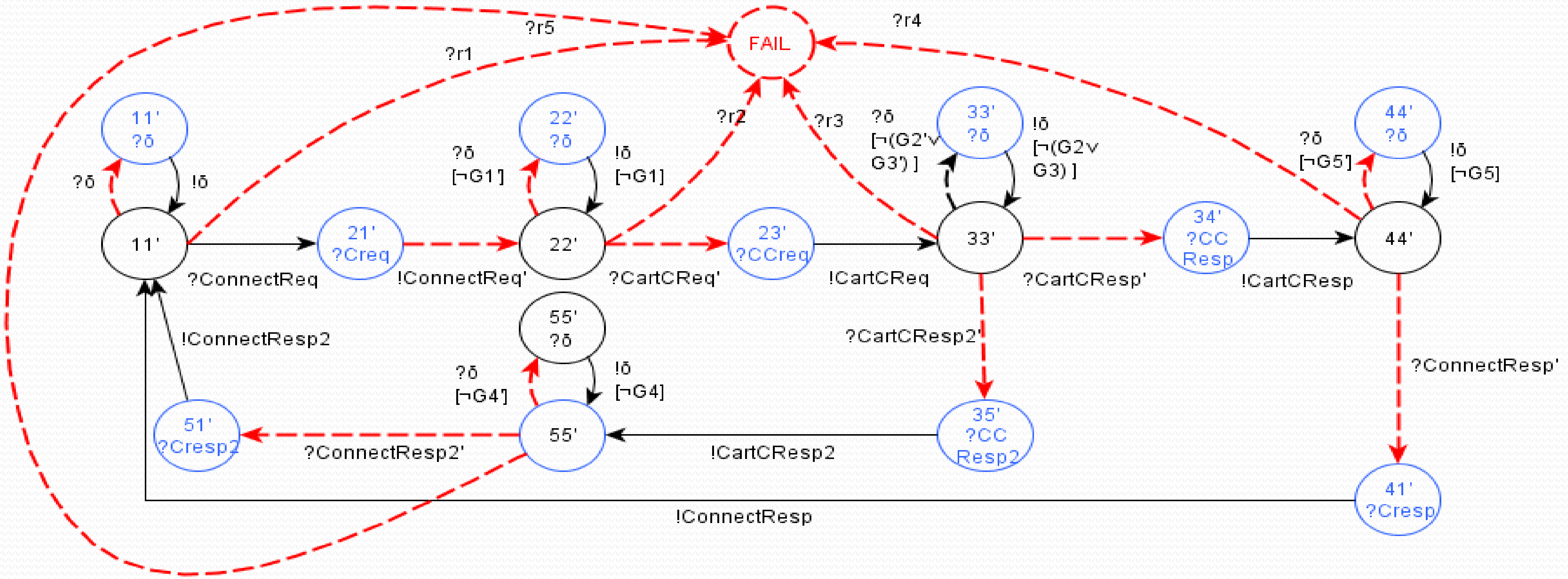
- Illustration:



Property on traces: $Traces_{Fail}^{CAN}(P\ (S)) = Traces_{Fail}(CAN(S))$

Proxy-tester model gen.

- Illustration:



Property on traces: $Traces^{CAN}_{Fail}(P\ (S)) = Traces_{Fail}(CAN(S))$

What to do with proxy-tester model ?

- Ioco implementation relation

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

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

Prop. on traces
$NCTraces(\Delta(S))$
$= Traces^{can}_{Fail}(CAN(S))$

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

Def. Parallel execution
$||(\mathrm{Env}, \mathrm{P}, \mathrm{I}) = \mathrm{IOLTS}$

$$\frac{q_1 \xrightarrow{!a}_{\Delta(Env)} q_2, q_2'' \xrightarrow{?a}_{\Delta(I)} q_3'', q_1' \xrightarrow[\text{""}]{?a}_P q_2' \xrightarrow[\text{"NCO"}]{!a}_P q_3'}{q_1 q_1' q_2'' \xrightarrow{?a}_{||(Env,P,I)} q_2 q_2' q_2'' \xrightarrow{!a}_{||(Env,P,I)} q_2 q_3' q_3''}$$

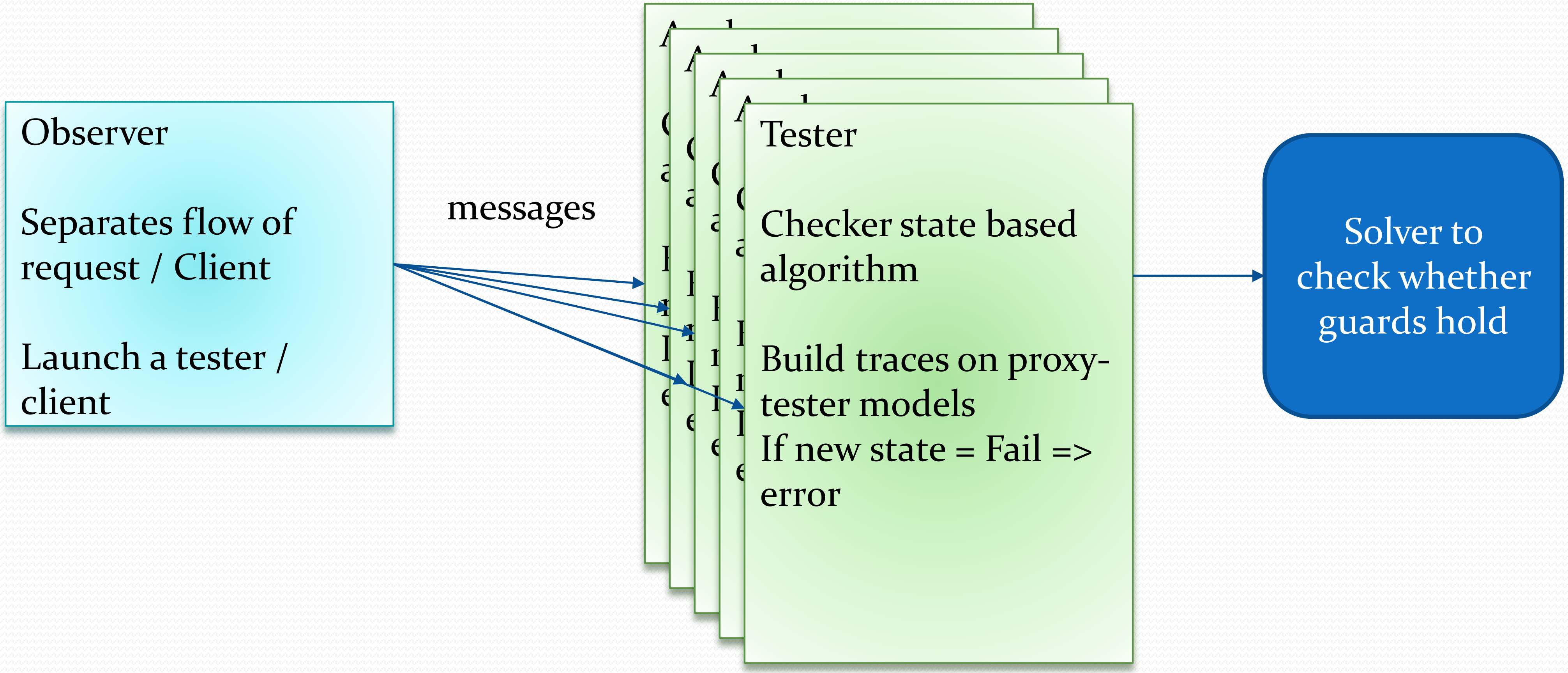
$$\frac{q_2 \xrightarrow{?a}_{\Delta(Env)} q_3, q_1'' \xrightarrow{!a}_{\Delta(I)} q_2'', q_1' \xrightarrow[\text{"NCO"}]{?a}_P q_2' \xrightarrow[\text{""}]{!a}_P q_3', q_3' \neq Fail}{q_2 q_1' q_1'' \xrightarrow{?a}_{||(Env,P,I)} q_2 q_2' q_2'' \xrightarrow{!a}_{||(Env,P,I)} q_3 q_3' q_2''}$$

$$\frac{q_2 \xrightarrow{?\delta}_{\Delta(Env)} q_3, q_1'' \xrightarrow{!a}_{\Delta(I)} q_2'', q_1' \xrightarrow[\text{"NCO"}]{?a}_P Fail}{q_2 q_1' q_1'' \xrightarrow{?a}_{||(Env,P,I)} Fail}$$

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

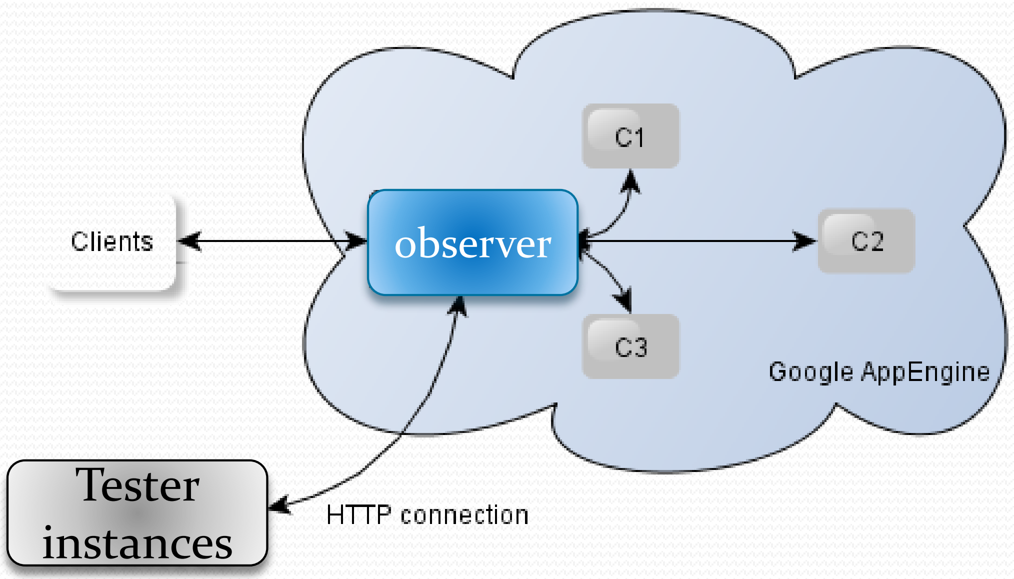
$\Rightarrow$ Proxy tester + passive tester Algo:
Builds traces
If a trace -> Fail => error

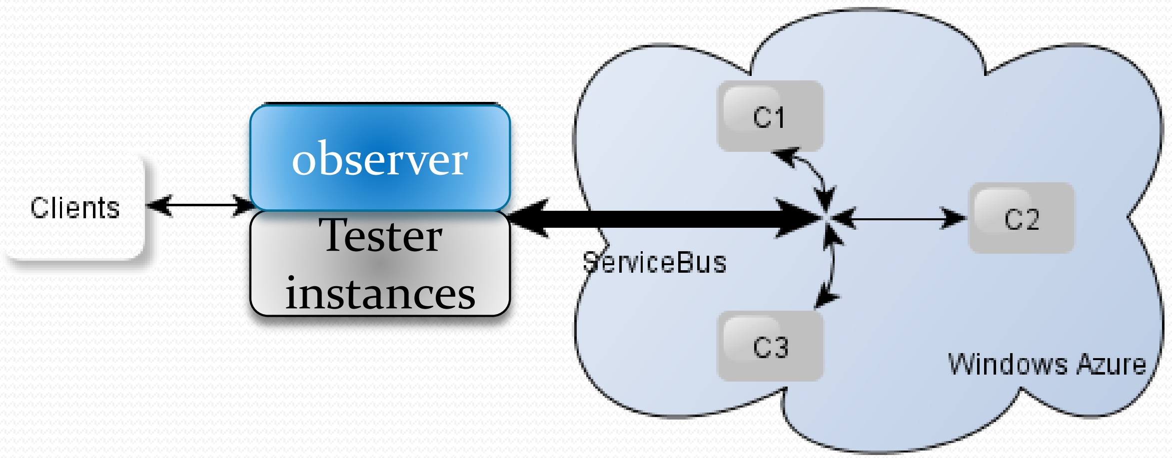
Passive tester algorithm



Passive testing with proxy-tester

- Implementation on 2 Clouds
  - Windows Azure and Google AppEngine





Runtime verification with proxy-testers

- Completion of Proxy-tester models with

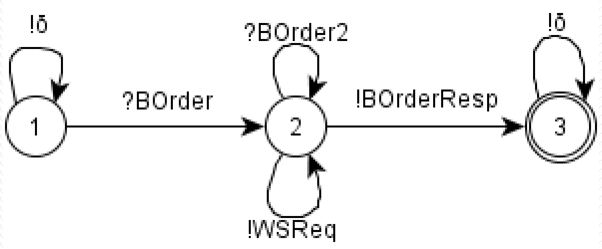
  - Safety properties ''nothing bad ever happens''
  - "A language L is a *safety language* if every word not in L has a finite bad prefix"

- Safety property modeled with ioSTSs ☺

  IOSTS expresses behaviours that violates property with a Violate state

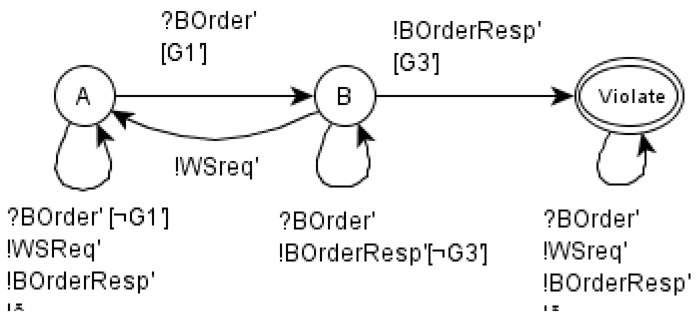
Runtime verification with proxy-testers

safety property example



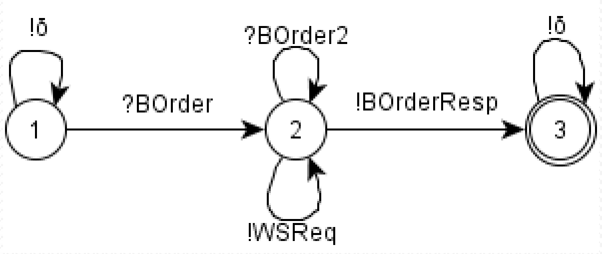
Symbol	Message	Guard	Update
?BOrder	?BookOrder(List-Books, quantity, account)		q:=quantity, b:=ListBooks
?BOrder2	?BookOrder(List-Books, quantity, account)		
!WSReq	!WholeSaler(isbn, from, to, corr)	$G2$=[isbn=b[q]∧$q \geq 1 \wedge$ from="BR"∧to= "WS"∧ $corr$ = $\{a, isbn\}$]	$q := q - 1$
!BOrder Resp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

"the receipt of an order confirmation (labelled by done) without requesting the wholesaler is BAD"

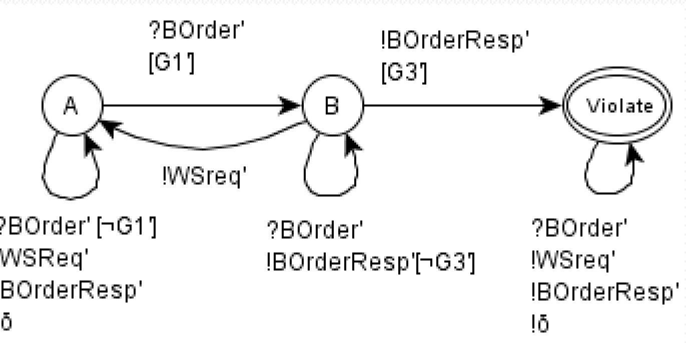


Symbol	Message	Guard
?BOrderReq'	?BookOrderReq(ListBooks, quantity, account)	G1'=[quantity≥ 1]
!WSReq'	!WholeSalerReq(isbn)	
!BOrderResp'	!BookOrderResp(resp)	G3'=[start(resp)="done"]

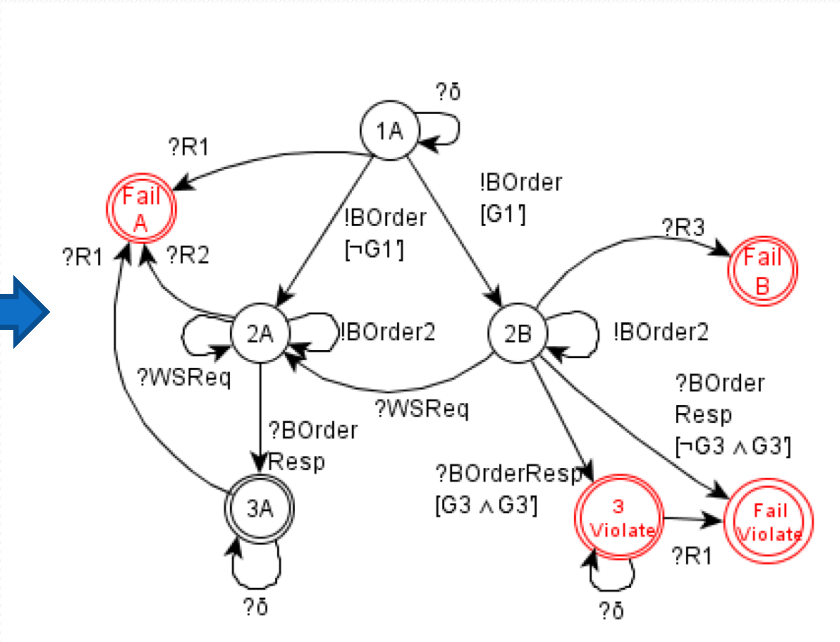
Runtime verification with proxy-testers



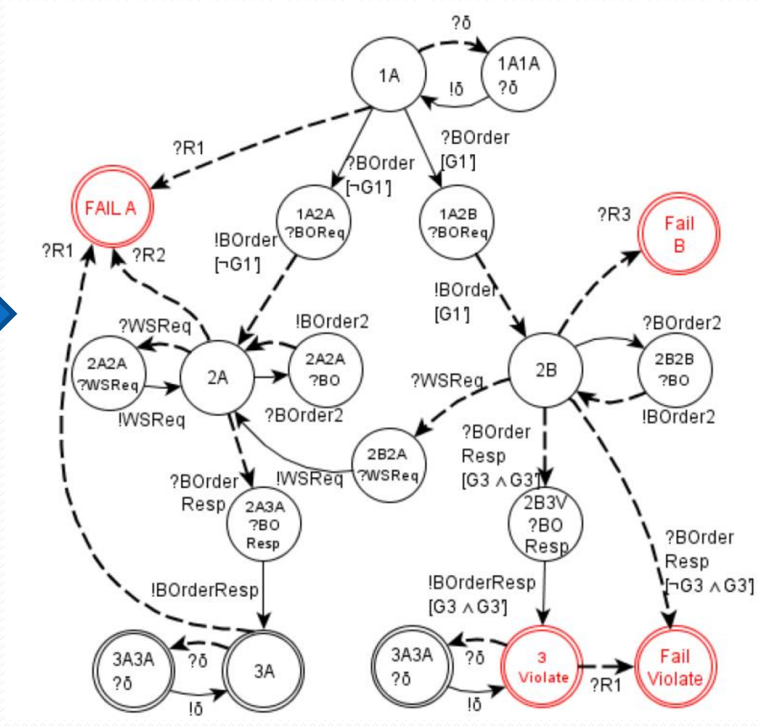
specification



safety property



Monitor (canonical tester // prop)



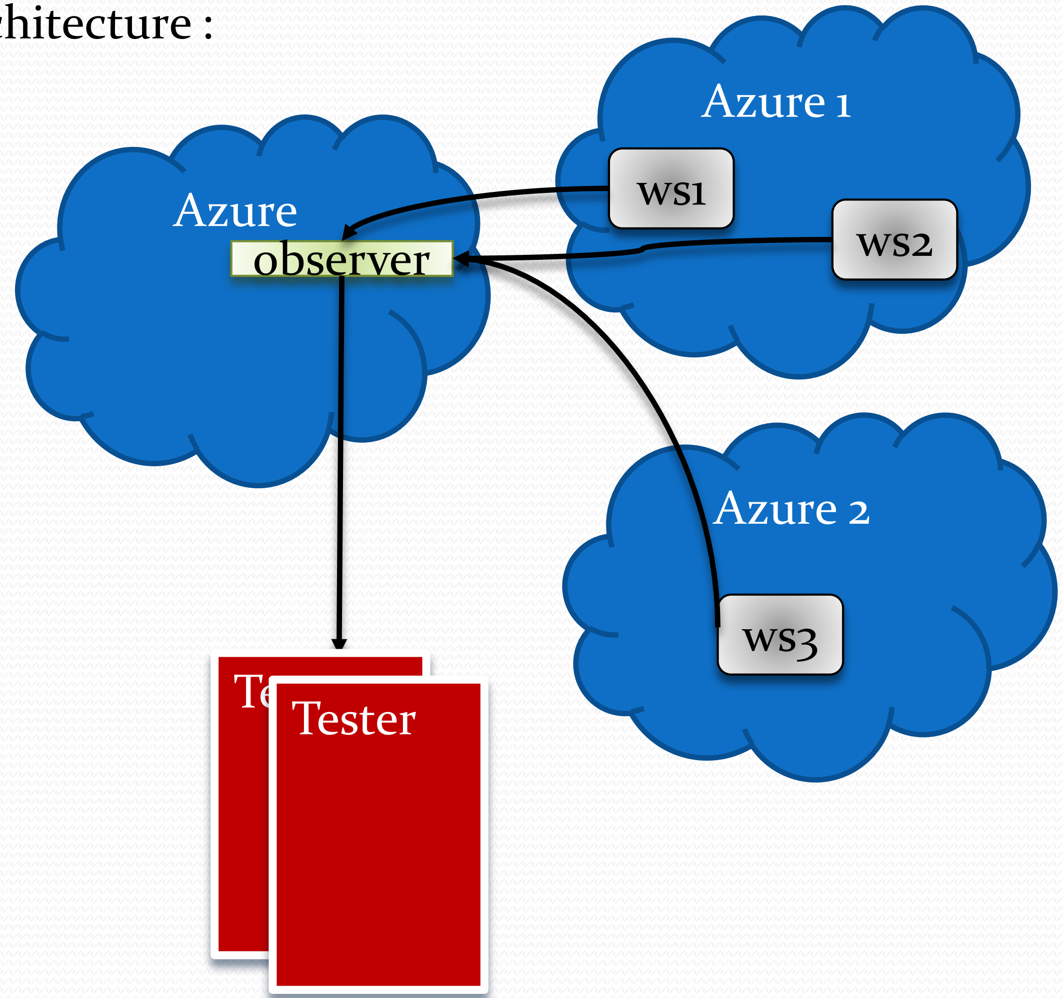
Proxy monitor

Runtime verification with proxy-testers

- Algorithm soundness
- Trace -> Fail => ioco not safisfied
- Trace -> Violate => safety prop. Violated
- Trace -> Fail/Violate => both

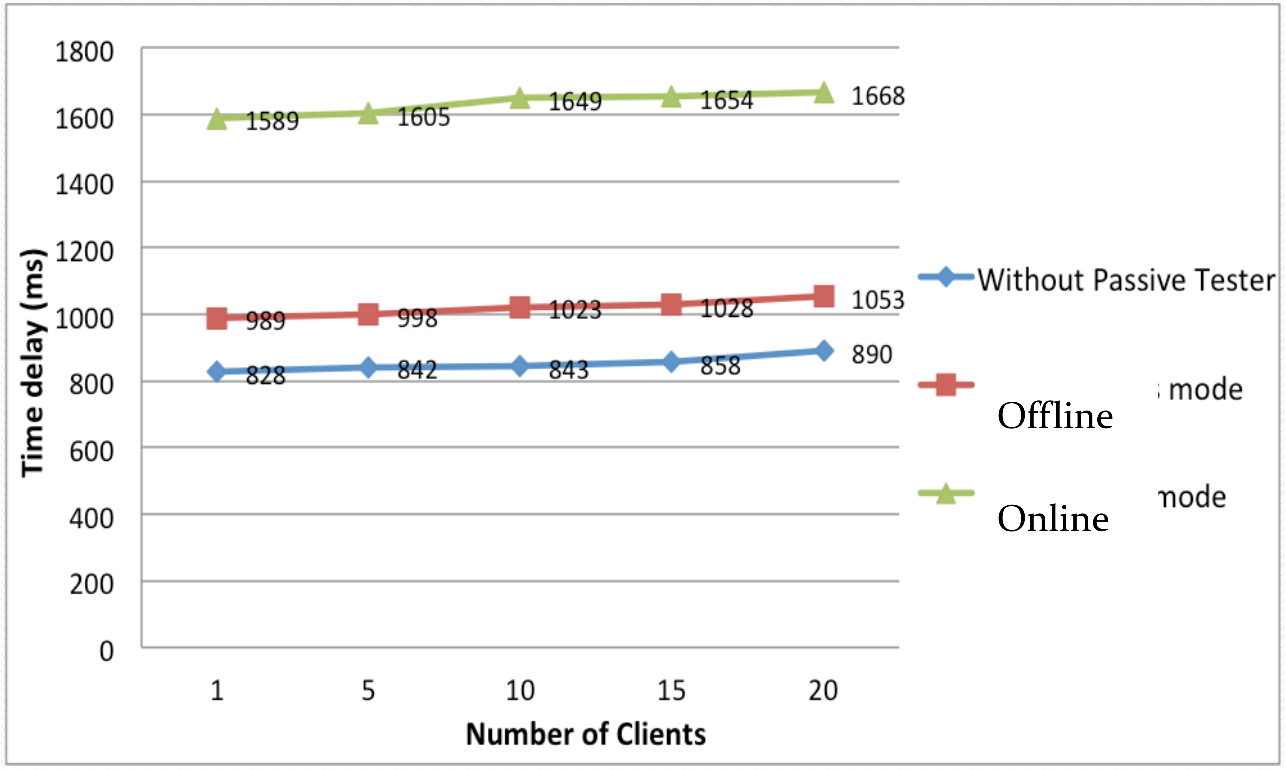
Evaluation

Architecture :



In our laboratory

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



Limitations ?

- Bottlenecks on observer, Solver -> latencies issues

- The more clients, the more testers => requires more resources

>50 clients => online mode ko

But?

- We could benefit from the cloud features !
- Unlimited number of VMs and cpu => parallel observer, unlimited tester instances

Conclusion

Conclusion

- What makes testing apps in clouds more difficult ?
  - Dynamic nature of clouds
  - difficulty to observe outputs (asynchronous communication mode, hidden messages in compositions)
  - Protocols, APIs,

- Need of additional test hypotheses or to revisit Implementation relations

- But, testing in clouds can benefits from clouds
  - Rely on the flexibility of clouds to implement testers

Some Perspectives

- Other kinds of observers for clouds ?
  - Add Monitor services to Web service compositions
  - Complete Web service codes with observers ?
  - Build Docker containers for testing
-
- Model-based testing requires models
  - Writing model is dificult and error-prone
  - -> model inference of composite service ? (active, passive inference, etc.)

- Apps developped for clouds often associated with Big data
  - Testing the «big data » side of these apps (robusteness)?

Thank you

- Questions ?

- [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

Choregraphie, orchestration ?

Orchestration des services

- Lorsqu'un service web coordonne d'autres services

- Par des processus BPEL (processus écrit en XML qui décrit comment interagissent les WS suivant des stimuli extérieurs)

- Besoin d'un serveur qui exécute les processus BPEL
la gestion des erreurs doit être gérée par le processus (mécanisme de replis, re-exécution du processus)

- Langage de programmation de processus mais aussi interface graphique (boites)

Chorégraphie de services

- Chaque service web mêlée dans la chorégraphie connaît exactement quand ses opérations doivent être exécutées et avec qui l'interaction doit avoir lieu.

- Description des interactions de service uniquement de pair à pair

- Pas de processus, chaque service connait les actions à effectuer par rapport aux messages reçus

- Langage en XML WS-CL ou WSCI

Apercu de WSBPEL

•Definition des partenaires

•Utilisation de variables, assignation de valeurs (assign)

•Activités basiques (invoque, receive, reply, wait, throw)

•Activités structurés (while, switch, sequence,pick(temporisation)

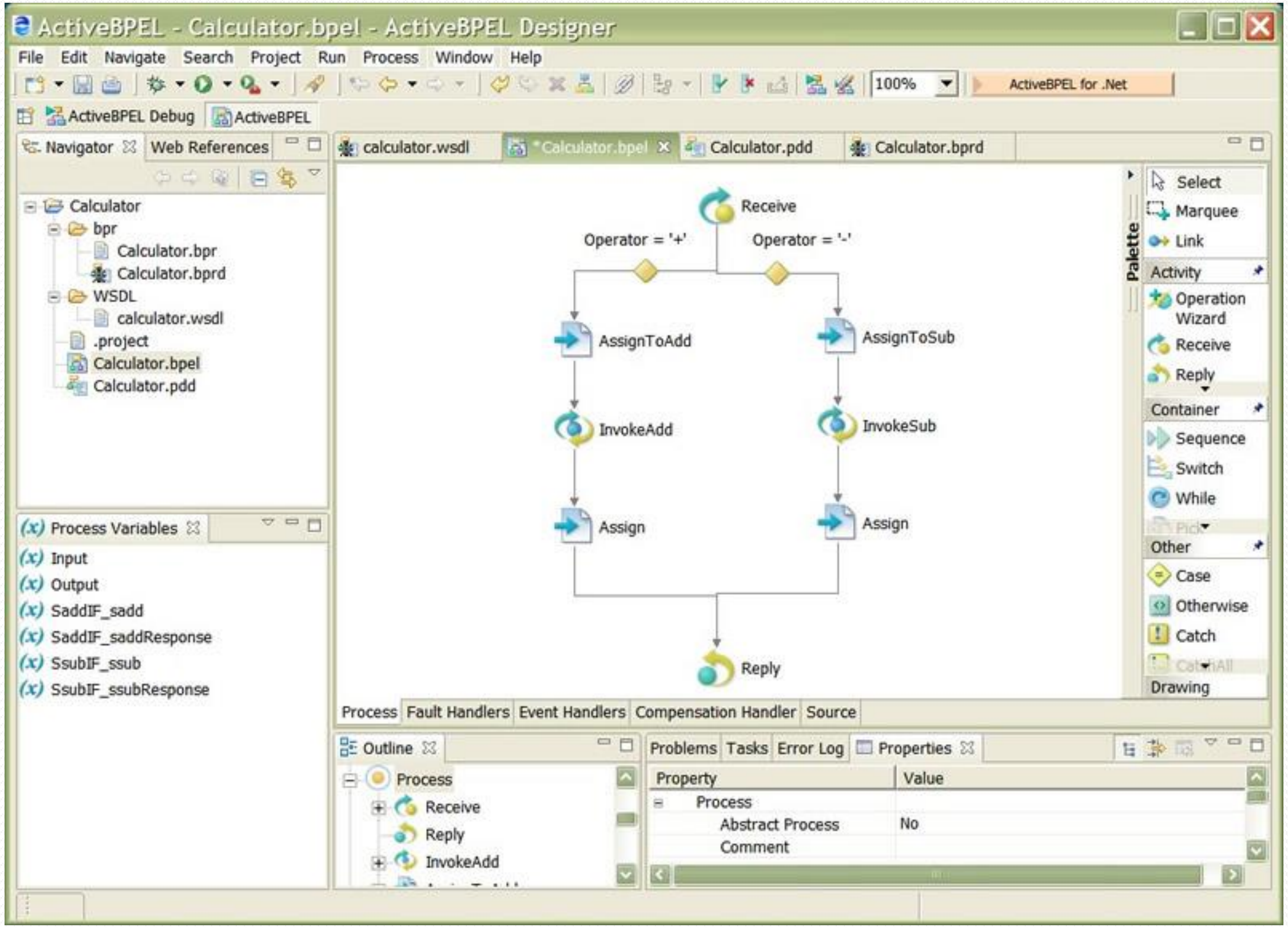
•Correlation = session

•Scope découpage d’un processus en plusieurs parties

  •Pl. handler possibles par scope (conpensation, fault, event)

Apercu de WSBPEL

Avec ActiveBPEL



Avec ActiveBPEL