

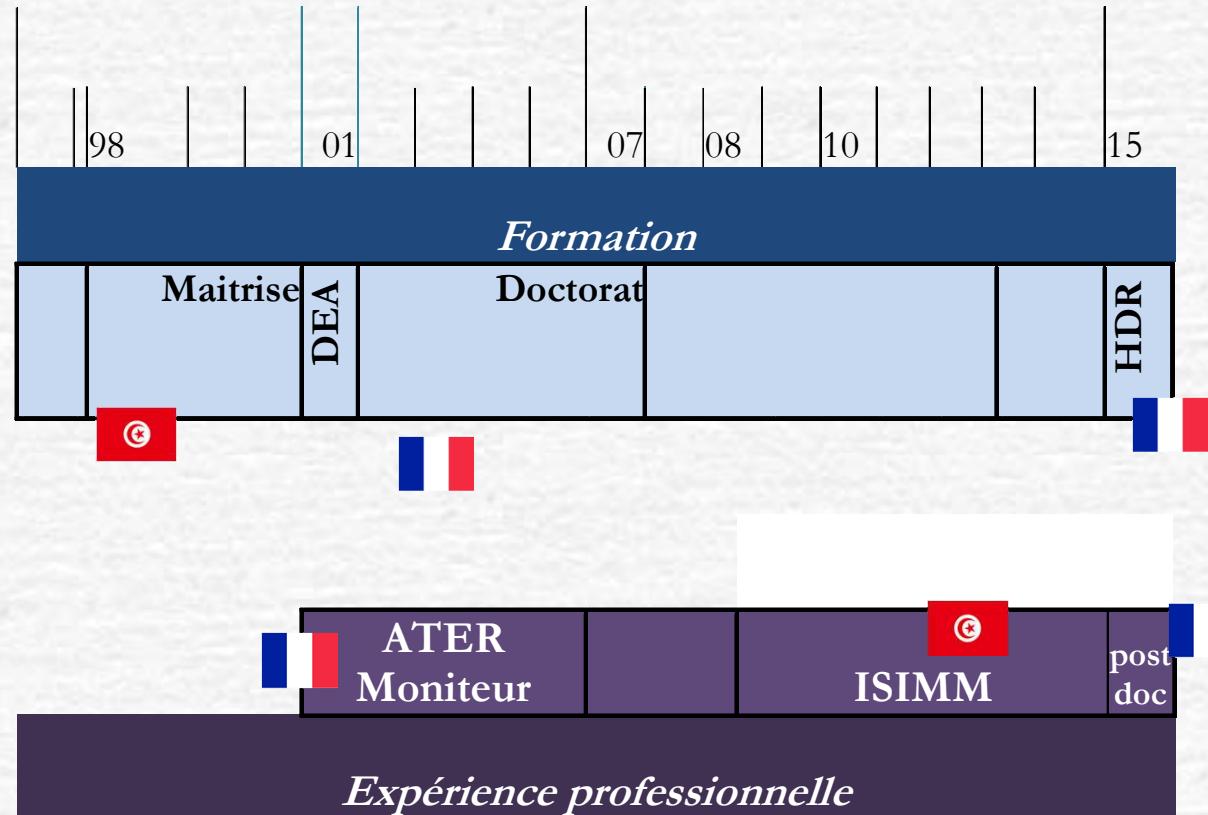
# Approche formelle pour la vérification de la composition et la sécurité de logiciels

Mohamed GRAIET

Docteur/HDR en informatique

Enseignant chercheur à ENSAI Rennes

# CV



CV

Recherche

Rayonnement

# Contexte générale

## ■ Les architectures logicielles:

- Composant
- Service
- Composant service
- Les logiciels FOSS (Free and Open-Source Software)

## ■ Modélisation et vérification formelle

- Méta-modélisation
- Les ADLs (ACME\ARMANI, Wright)
- B\Event-B
- IDM

## ■ Composition et sécurité de logiciels

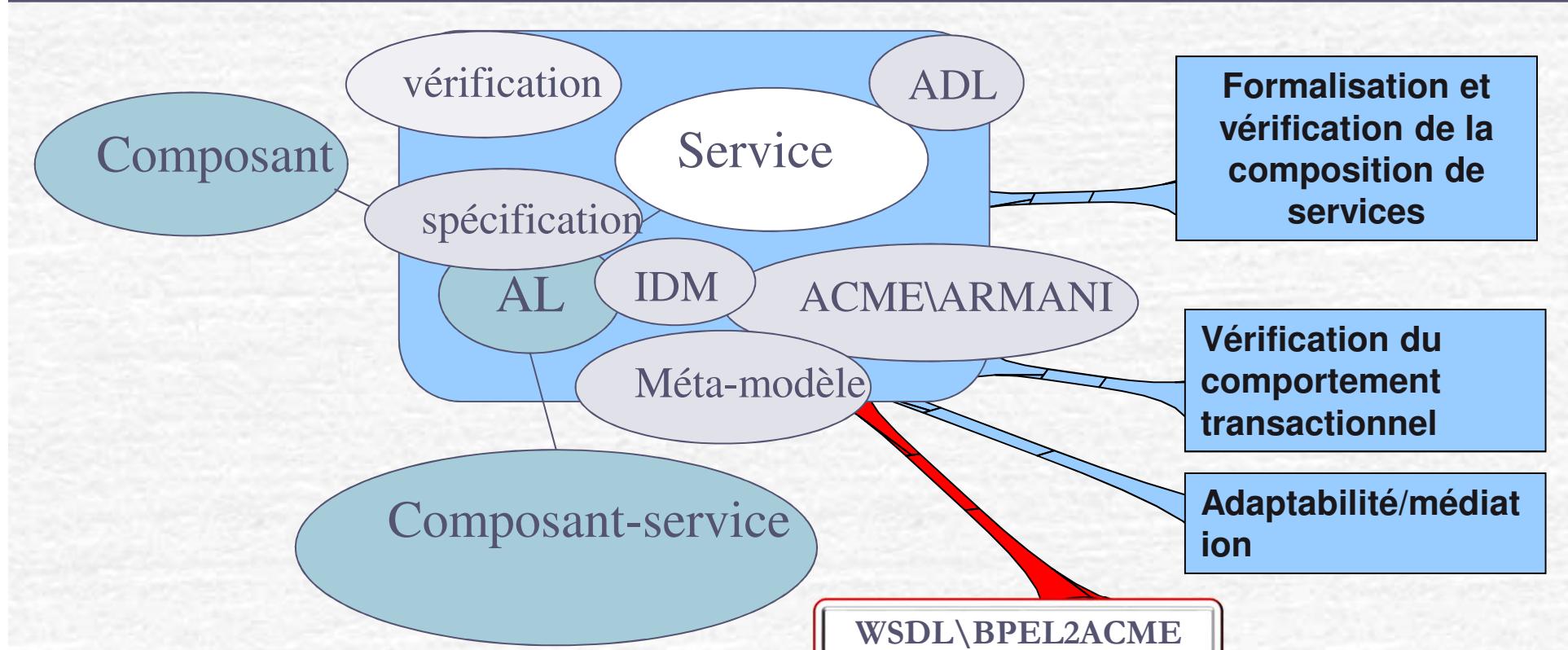
CV

Recherche

Rayonnement

# Contribution 1

## Vérification formelle (Service Web, IDM)



### Publications

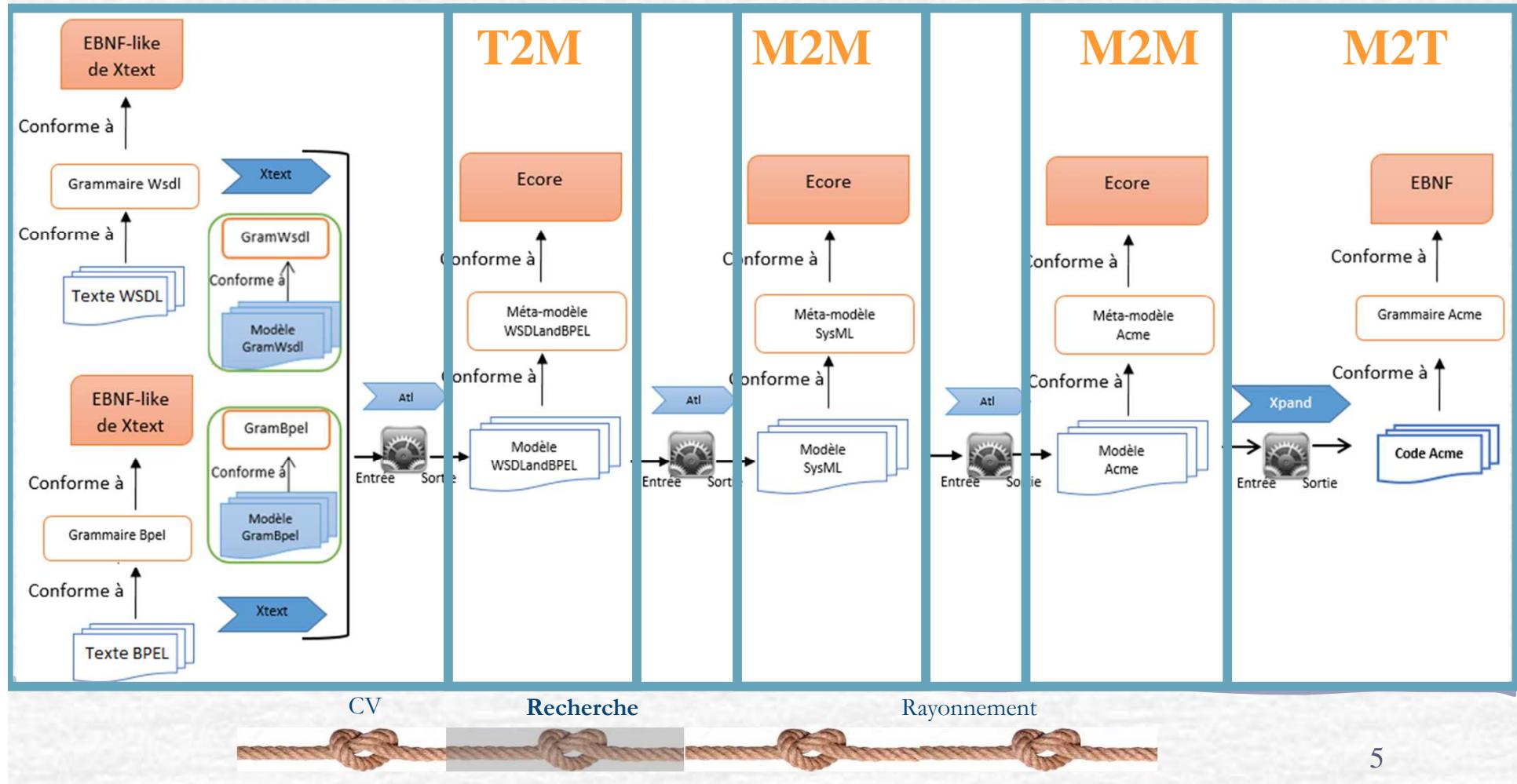
- **Journaux** : International Journal of Web Information Systems [SJR, Q3]
- **Conférences**: SCC [A], ICWS [A], Wetice [B]

CV

Recherche

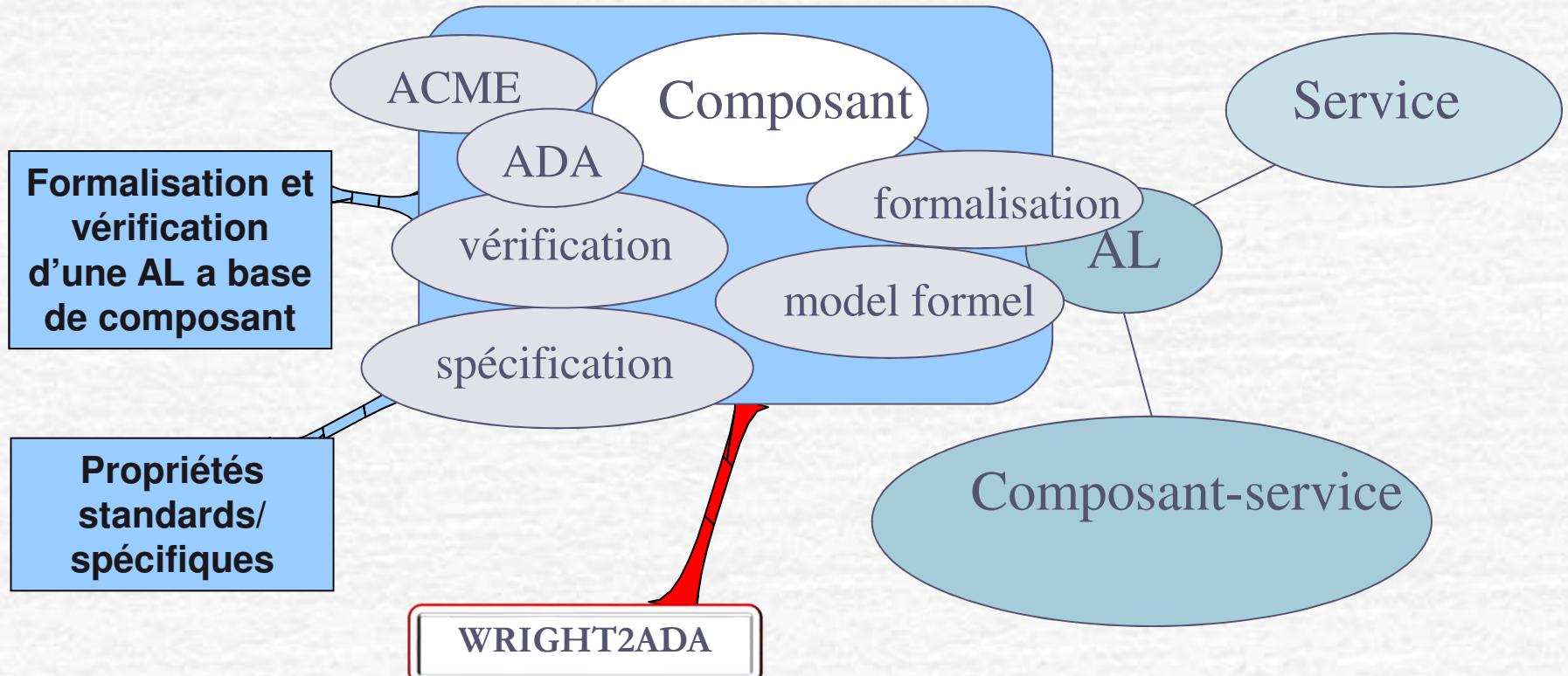
Rayonnement

# Outil : de WSDL\BPEL2ACME



# Contribution 2

## Vérification formelle (Composant, IDM, ACME)



### Publications

- Journaux : TSI, RNTI
- Conférences: ICWS [A], ECBS[B], CAL

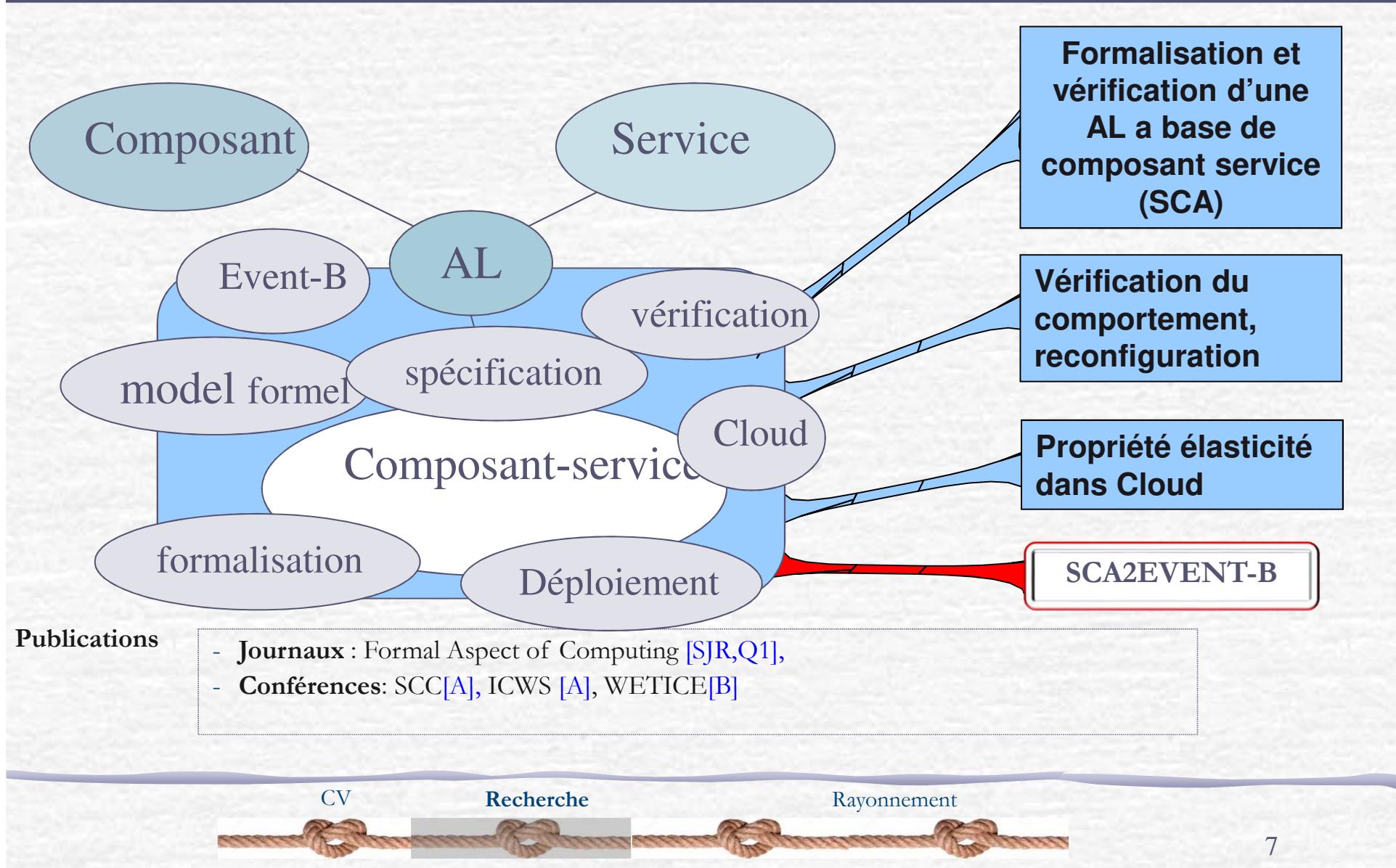
CV

Recherche

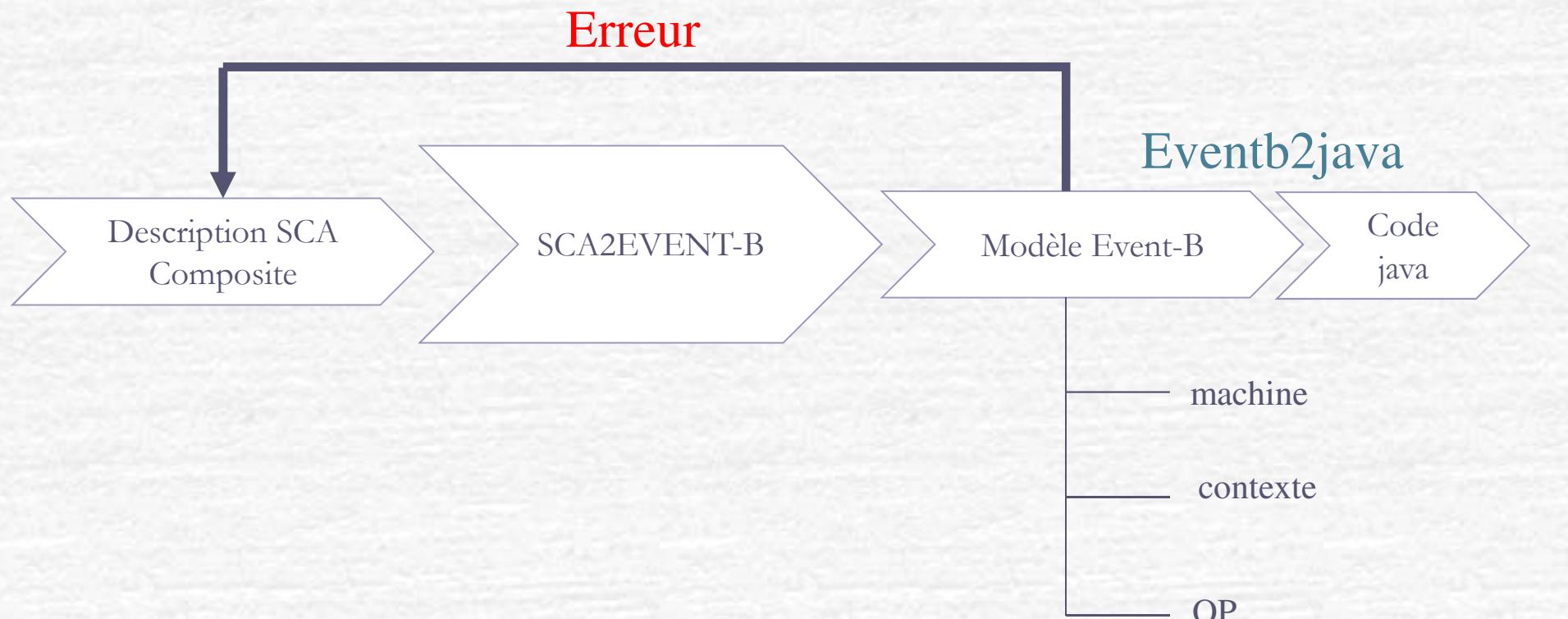
Rayonnement

# Contribution 3

## Vérification formelle(SCA, Event-B)



# Vue d'ensemble SCA2B



CV

Recherche

Rayonnement

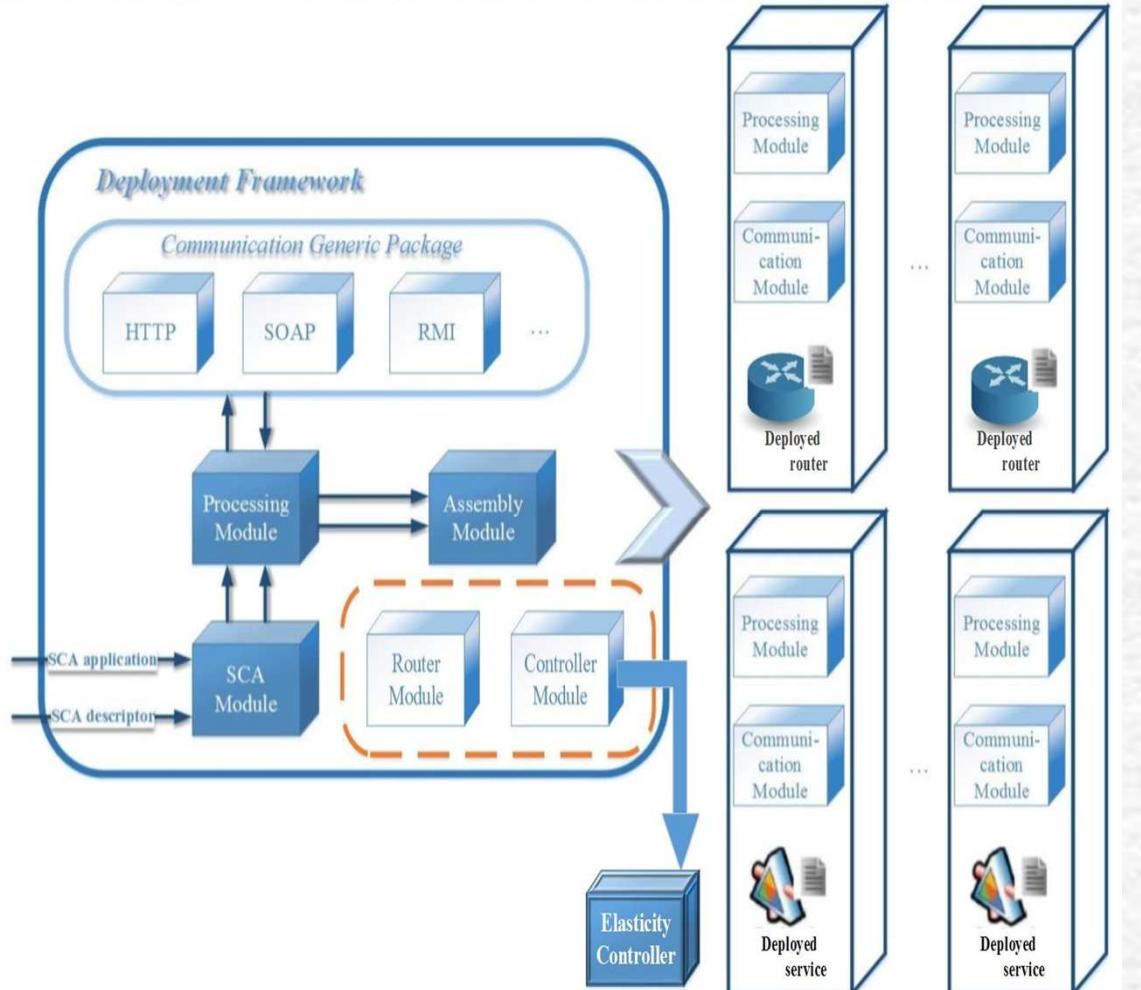
# Vue d'ensemble de la méthode déploiement dans le cloud

L'architecture du Framework :

- la plateforme de déploiement de service
  - Module SCA
  - Module de traitement
  - Module de montage
- Micro-conteneur
- Client SaaS

Notre extension consiste en :

- Module Routeur
- Module Contrôleur
  - Duplication
  - Consolidation



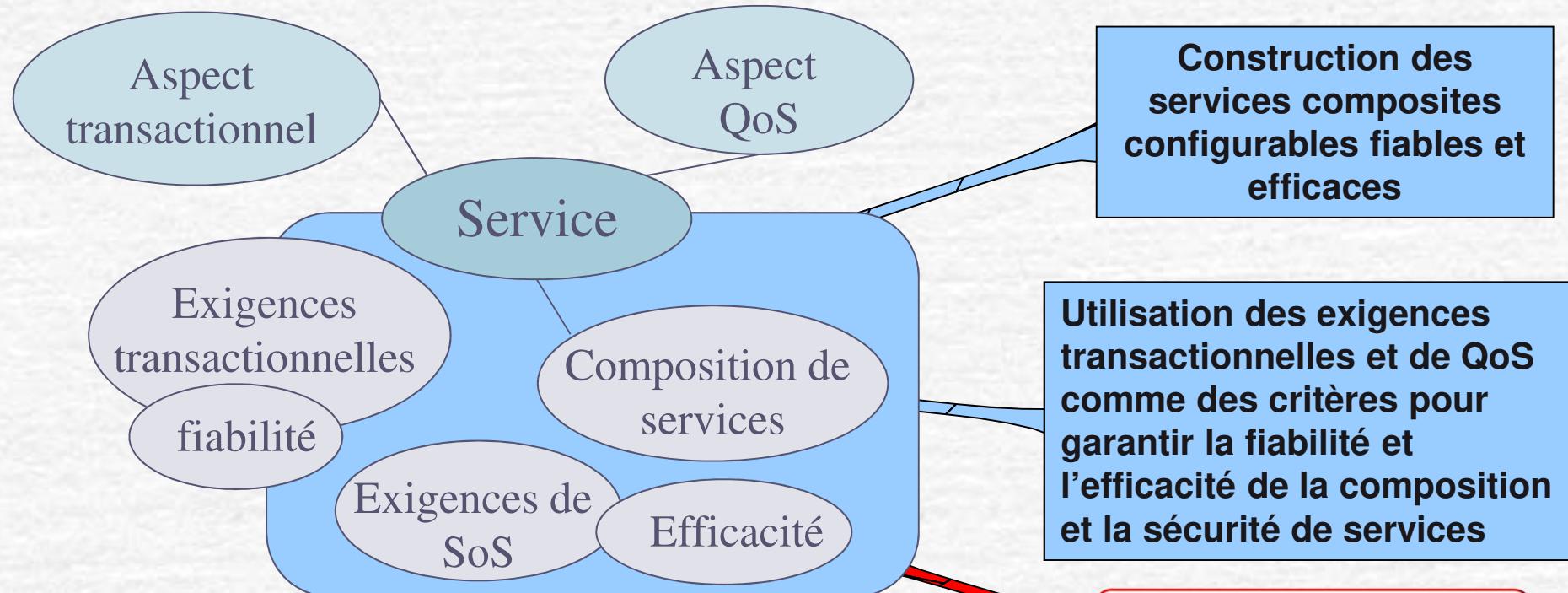
CV

Recherche

Rayonnement

# Contribution 4

## Composition et sécurité de logiciels



### Publications

- Journaux : IEEE Systems Journal [SJR,Q1],
- IEEE Trans. Network and Service Management [SJR,Q1]
- Future Generation Computer Systems [SJR,Q1],
- Conférence: WISE [A]

FOSS2EB

# Contribution

## ■ Deux classes de logiciels:

- Les services Web
- Les logiciels FOSS (Free and Open-Source Software).

## ■ Composition de logiciels: le processus de la réutilisation d'un ensemble de logiciels existants pour construire de logiciels plus complexes et sécurisé.

## ■ Besoins de méthodes formelles:

- La vérification de la correction de la composition et la sécurité de services Web.
- Vérification de la correction de la composition et la sécurité de logiciels FOSS.

CV

Recherche

Rayonnement

# Contribution

## Exigences de la correction de la composition de services Web:

- Exigences de QoS définies sous la forme d'un contrat SLA (service-level agreement) : sécurité métier
- Exigences transactionnelles définies par les concepteurs en utilisant le concept d'Etats de Terminaison Acceptés (ETA): sécurité de l'orchestration des services web

## Exigences de la correction de la composition de logiciels FOSS:

- Exigences de dépendances des logiciels FOSS : Sécurité de transaction
- Exigences de capacité d'utilisation des ports des logiciels FOSS
- Exigences d'allocation de ressources cloud pour les logiciels FOSS

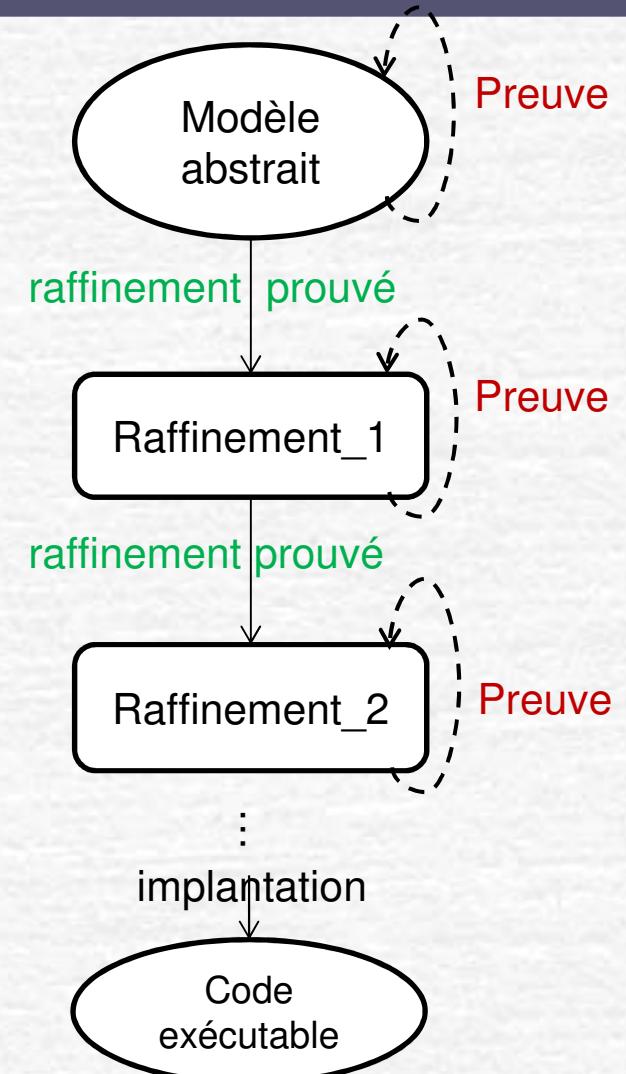
CV

Recherche

Rayonnement

# Contribution

- ✓ Méthode formelle repose sur des fondements mathématiques:
  - La théorie des ensembles
  - La logique de premier ordre.
- ✓ Un modèle Event-B est définie par les concepts clés suivants:
  - Contexte
  - Machine
  - Raffinement
- ✓ La correction d'un modèle Event-B est assurée par des règles d'obligations de prévue (OP).



CV

Recherche

Rayonnement

## Contribution 1

### Vérification de la composition et la sécurité de services Web

La formalisation de la composition de services se repose sur les étapes suivants:

- Etape 1: formalisation des services Web.
- Etape 2: formalisation de la composition statique de services Web.
- Etape 3: formalisation de la composition dynamique de services Web.
- Etape 4: Correction et validation.

CV

Recherche

Rayonnement

# Contribution

## Vérification de la composition et la sécurité de services Web

### Etape 1: formalisation des services Web

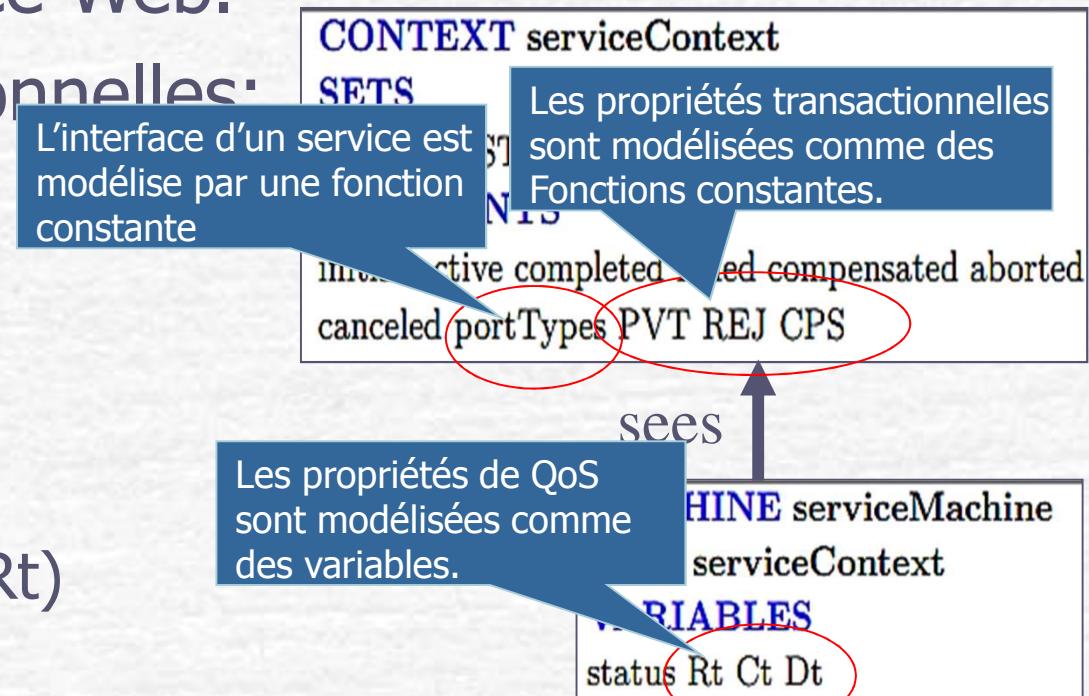
#### Interface d'un service Web.

#### Propriétés transactionnelles:

- Pivot (PVT)
- Rejouable (REJ)
- Compensable (CPS)

#### Propriétés de QoS:

- Temps de réponse (Rt)
- Coût (Ct)
- Disponibilité (Dt)



CV

Recherche

Rayonnement

## Contribution 1

### Vérification de la composition et la sécurité de services Web

#### Etape 2: formalisation de la composition statistique de services Web

La formalisation de la composition statique est effectuée en deux sous-étapes:

- Formalisation des relations de dépendance services Web.
- Formalisation des patrons de workflow.

CV

Recherche

Rayonnement

# Contribution 1

## Vérification de la composition et la sécurité de services Web

### Etape 2: formalisation de la composition statistique de services Web

#### Les types de relations de dépendance:

- Activation
- Abandon
- Annulation
- Compensation

#### Les relations de dépendance sont modélisées comme des relations binaires.

Les relations de dépendance sont modélisées comme des variables.

~~EXTENDS service~~  
~~CONSTANTS~~  
depAct depAbt depCnl depCps  
~~AXIOMS~~  
axmt :  $depAct \in \mathbb{P}(SERVICE \times SERVICE)$   
...

Les relations de dépendance sont modélisées comme des relations binaires.

**MACHINE** compositionMachine  
**REFINES** serviceMachine  
**SEES** compositionContext  
**VARIABLES**  
status Rt Ct Dt  
**INVARIANTS**  
...



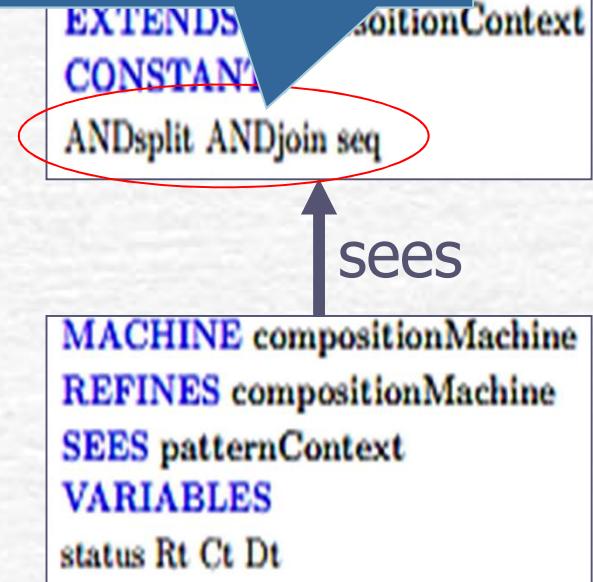
# Contribution 1

## Vérification de la composition et la sécurité de services Web

### Etape 2: formalisation de la composition statistique de services Web

- Patrons de workflow pour définir l'orchestration de services.
- Un patron est modélisé par un couple:
  - AND-split=<ANDsplit, activerANDsplit>:
    - $\text{ANDsplit} \in \text{SERVICE} \rightarrow \text{P(SERVICE)}$
    - activerANDsplit: opération d'activation
  - AND-join=<ANDjoin, activerANDjoin>:
    - $\text{ANDjoin} \in \text{P(SERVICE)} \rightarrow \text{SERVICE}$
    - activerANDjoin: opération d'activation
  - Séquence=<seq, activerSeq>:
    - $\text{seq} \in \text{SERVICE} \rightarrow \text{SERVICE}$
    - activerSeq: opération d'activation

Les fonctions ANDsplit ANDjoin et seq sont définies comme des constantes.



CV

Recherche

Rayonnement

## Contribution 1

# Vérification de la composition et la sécurité de services Web

## Etape 3: formalisation de la composition dynamique de services Web

- Notre approche de composition dynamique de services Web:
  - Etape 3.1: la sélection basée sur les exigences transactionnelles et de QoS.
  - Etape 3.2: la composition des services sélectionnés.
- Critères de classement des services Web:
  - Une fonction **DSC** (Distance de Satisfaction de Contraintes): utilisé pour classer les services selon les exigences transactionnelles: sécurité de l'orchestration des services web
  - Une fonction **score**: Utilisé pour classer les services valides en fonction des exigences de QoS : sécurité QoS

Question: comment peut-on formaliser cette approche de composition dynamique de services Web?

CV

Recherche

Rayonnement

### Etape 3: formalisation de la composition dynamique de services Web

- La formalisation de notre approche de composition dynamique de services Web se fait en trois volets:
  - Étape 3.1: modification de la formalisation de la composition statique de services.
  - Etape 3.2: formalisation de la sélection de services.
  - Etape 3.3: formalisation du processus de composition de services sélectionnés.

CV

Recherche

Rayonnement

## Phase de modification

### Première modification: définition des relations de dépendance

Les relations de dépendances ne peuvent jamais être modifiées car elles sont définies comme des constantes.

```
EXTENDS serviceContext
CONSTANTS
depAct depAbt depCnl depCps
AXIOMS
axm1 : depAct ∈ ℙ(SERVICE × SERVICE)
...
END
```

```
MACHINE compositionMachine
REFINES serviceMachine
SEES compositionContext
VARIABLES
status Rt Ct Dt
INVARIANTS
...
```

modifications

```
CONTEXT compositionContext
EXTENDS serviceContext
CONSTANTS
ET t_max c_max d_min
AXIOMS
axm1 : ET ∈ SERVICE → ℙ(STATE)
...
END
```

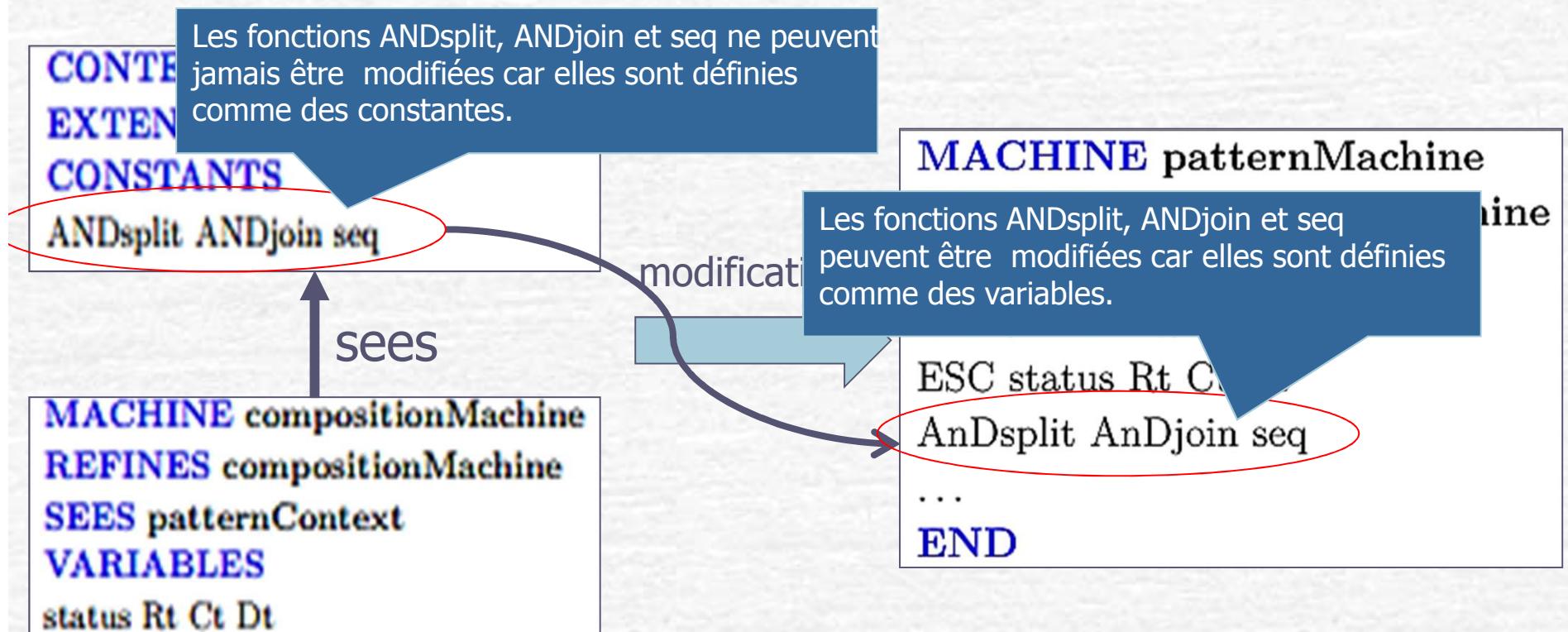
Les relations de dépendances peuvent être modifiées car elles sont définies comme des variables.

```
VARIABLES
ESC states Rt Ct Dt
depAct depAbt depCnl depCps
...
```

# Vérification de la composition et la sécurité de services Web

## Phase de modification

- Deuxième modification: définition des patrons de workflow comme des variables.



## Phase de formalisation de l'approche de sélection

Etape 1: Vérification de la cohérence transactionnelle des services web.

L'événement prend en entrée deux paramètres: un service Web et un élément de ETA

- Calcul de la valeur de satisfiabilité des services Web.

```

verifierSAT = ANY s i
WHERE
  grd1 :  $s \in SERVICE$ 
  grd2 :  $i \in 1 .. card(ETA)$ 
  grd3 :  $s \mapsto i \notin dom(sat)$ 
THEN
  act1 :  $sat : |(ran(portTypes(s) \lhd ETA(i)) \subseteq TS(s) \Rightarrow sat' = sat \cup \{(s \mapsto i) \mapsto 1\}) \wedge (ran(portTypes(s) \lhd ETA(i)) \not\subseteq TS(s) \Rightarrow sat' = sat \cup \{(s \mapsto i) \mapsto 0\})$ 
END

```

L'événement retourne comme sortie la valeur de satisfiabilité du service s.

## Phase de formalisation de l'approche de sélection

Etape 2: classement des services Web : sécurité Qos

- Calcul de la valeur de DSC des services Web.

```

calculerDSC ≡ ANY s
WHERE
  grd1 :  $s \in SERVICE$ 
  grd2 :  $\forall i \cdot i \in 1..card(ETA) \Rightarrow s \mapsto i \in sat$ 
THEN
  act1 :  $DSC(s) := sum(\{i \cdot i \in 1..card(ETA) | i \mapsto (1 - sat(s \mapsto i)) * W(i)\})$ 
END

```

L'événement retourne comme sortie la valeur de DSC du service s.

CV

Recherche

Rayonnement

## Phase de formalisation de l'approche de sélection

Etape 3: sélection des services web valides:

- Calcul de l'Ensemble des Services Valide (ESV) pour une opération donnée.

```
verifierESV = ANY op  
WHERE  
grd1 : op ∈ operations  
THEN  
act1 : ESV(op) = {s|s ∈ SERVICE ∧ op ∈ portTypes(s) ∧ DSC(s)=0}  
END
```

L'événement retourne comme sortie l'Ensemble des Services Valide (ESV) pour une opération

CV

Recherche

Rayonnement

## Phase de formalisation de l'approche de sélection

Etape 4: classement des services Web valides:

- Calcul de la valeur du score des services valides.

```

calculerSCORE = ANY A B C
WHERE
  grd1 : A ∈ N ∧ B ∈ N
  grd2 : A+B+C=1
THEN
  act1 : score := {s : s ∈ union(ran(ESV)) | s → ( A*(Rt(s)-t_max)+B*(Ct(s)-c_max)+C*(d_min-Dt(s)))}
END

```

Les préférences des clients sont exprimées en affectant un poids à chaque propriété de QoS.

La somme des valeurs des poids doit être égale à 1.

L'événement calcul de la valeur du score de services valides.

CV

Recherche

Rayonnement

## Phase de formalisation de l'approche de sélection

Etape 5: sélection des meilleurs services valides.

- La sélection L'ensemble des services web valides qui optimisent la fonction score.

```

selectionner ≡ ANY SWT
WHERE
grd1 : SWT ∈ P(SERVICE)
grd2 : ∀ s · s ∈ SWT ∩ dom(score) ⇒ score(s) ≤
grd3 : portTypes[SWT] ⊆ operations
grd4 : ∀ s · s ∈ SWT ∩ ESV(op) ⇒
    score(s)=min({x·x ∈ union(ran(ESV)) ∧ portTypes(s)=portTypes(x)|score(x) })
THEN
act1 : status := {s· s ∈ SWT |s ↪ initial }
act2 : ES := ES ∪ SWT
END

```

Les services sélectionnés doivent être valides et minimisant la fonction score.

## Contribution 1

# Vérification de la composition et la sécurité de services Web

## Phase de formalisation de la composition et de la vérification de la sécurité des services Web

Génération de transaction(s) entre les services sélectionnés.

L'événement prend comme entrée deux paramètres : SWT et ERD. SWT est un ensemble de services Web valides (SWT) et ERD est un ensemble de relations de dépendances (ERD).

```
composer = ANY SWT ERD
WHERE
grd1 : SWT ∈ P(SWP)
grd2 : ∀ s . s ∈ SWP
grd3 : ∀ s . s ∈ SWP
grd4 : ERD ∈ P(ERD)
THEN
act1 : depAct := ERD
act2 : depAbt := {s1 ↪ s2 | s1 ∈ SWT ∧ s2 ∈ SWT}
act3 : depCnl := {s1 ↪ s2 | s1 ∈ SWT ∧ s2 ∈ SWT}
act4 : depCps := {s1 ↪ s2 | s1 ∈ SWT ∧ s2 ∈ SWT \ ERD[ {s1, s2} ] ≠ ∅ ∨ s2 ↪ s1 ∈ ERD}
}
...
END
```

une dépendance d'abandon est définie si  $s_1$  est non rejouable et il existe une dépendance d'activation de  $s_2$  vers  $s_1$ .

une dépendance de compensation de  $s_1$  vers  $s_2$  est définie si :

- $s_1$  est non rejouable,  $s_2$  est compensable et il existe une dépendance d'activation de  $s_2$  vers  $s_1$ ;
- $s_1$  est non rejouable,  $s_2$  est compensable et  $s_1$  et  $s_2$  s'exécutent en parallèle et en synchronisation.

CV

Recherche

Rayonnement

## Etape 4: Correction et validation.

- Démarche de vérification de la formalisation se fait en deux volets:
    - Etape 1: Déchargement des obligations de preuve
    - Etape 2: Validation par animation
  - Les outils utilisés :
    - La plateforme RODIN
    - L'animateur ProB
  - Mise de l'approche : IDM



## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

La formalisation de la composition de logiciels FOSS se repose sur les étapes suivants:

- ✓ Étape 1: Modélisation de logiciels composites FOSS non élastiques
- ✓ Étape 2: Modélisation de logiciels composites FOSS élastiques
- ✓ Étape 3: Vérification de l'allocation de ressources cloud pour les logiciels FOSS
- ✓ Étape 4: Vérification et validation

CV

Recherche

Rayonnement

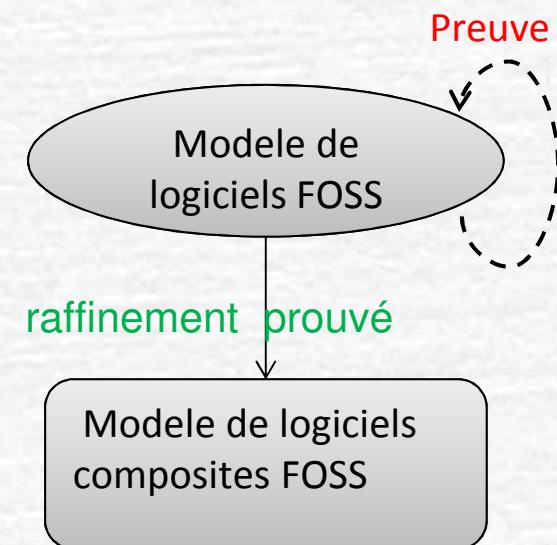
## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

## Etape 1: Modélisation de logiciels composites FOSS non élastiques

La formalisation de la composition statique est effectuée en deux sous-étapes:

- Etape 1.1: Modélisation abstraite des concepts de base de logiciels FOSS.
- Etape 1.2: Raffinement de modèle abstrait de logiciels FOSS en introduisant le concept de dépendance afin de définir formellement le processus de composition : sécurité de transaction



CV

Recherche

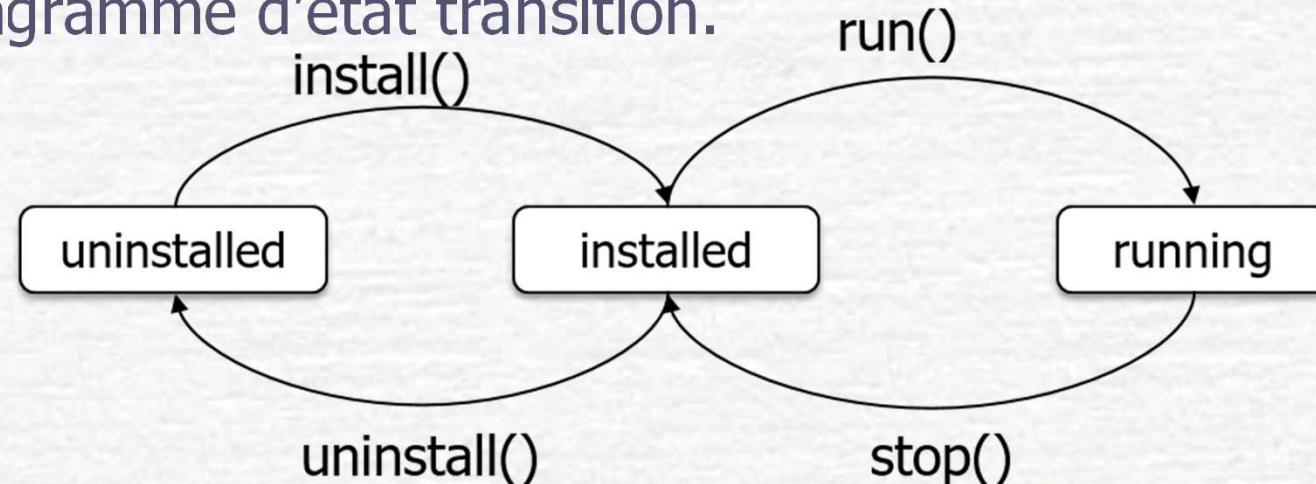
Rayonnement

## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

### Etape 1.1: Modelisation du logiciel FOSS

- Un logiciel FOSS fournit une collection de ports de deux types:
  - Port requis
  - Port fourni
- Le comportement d'un logiciel FOSS peut être modélisé par un diagramme d'état transition.



CV

Recherche

Rayonnement

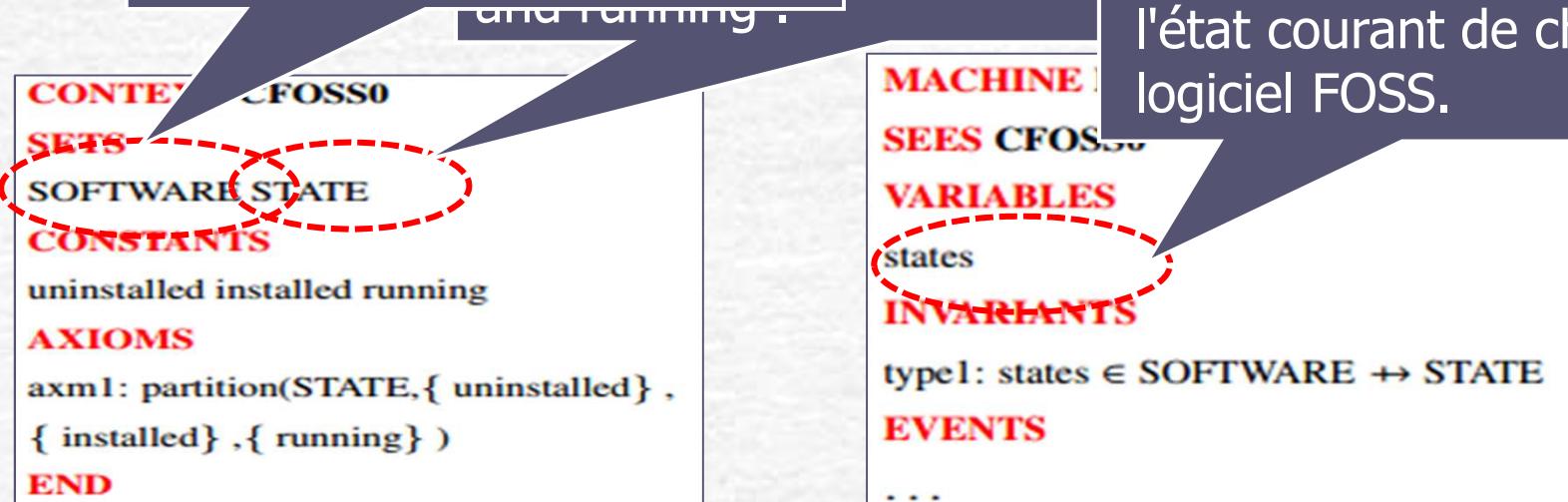
## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

### Etape 1.1: Modélisation du logiciel FOSS

- Pour modéliser le logiciel FOSS, nous définissons un modèle Event-B composé d'une machine et d'un contexte :

- Le contexte **CFOSO** L'ensemble énuméré STATE représente les états possibles des composants FOSS.
- La machine **MACHINE** SOFTWARE dénote l'ensemble de logiciels FOSS. Les variables states spécifient l'état courant de chaque logiciel FOSS.



## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

### Etape 1.1: Modélisation du logiciel FOSS

- Les processus d'installation, de désinstallation, d'exécution et d'arrêt de logiciels FOSS sont modélisés dans **MFOSSO** comme des événements.

```
install ≡ ANY s
WHERE
grd1: s ∈ dom(states)
grd2: states(s)=uninstalled
THEN
act1: states(s):=installed
END
uninstall ≡ ANY s
WHERE
grd1: s ∈ dom(states)
grd2: states(s)=installed
THEN
act1: states(s):=uninstalled
END
```

```
run ≡ ANY s
WHERE
grd1: s ∈ dom(states)
grd2: states(s)=installed
THEN
act1: states(s):=running
END
stop ≡ ANY s
WHERE
grd1: s ∈ dom(states)
grd2: states(s)=running
THEN
act1: states(s):=installed
END
```

## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

### Etape 1.2: Modélisation et sécurité du logiciel composite FOSS

- ☞ Un logiciel composite FOSS est le résultat de la combinaison d'un ensemble de logiciels FOSS existants (Di Cosmo et al., 2014).
- ☞ Le logiciel composite FOSS est défini à l'aide d'un ensemble de préconditions spécifiant trois types de dépendances:
  - Les dépendances d'installation expriment les conditions dans lesquelles un logiciel peut être installé.
  - Les dépendances d'exécution expriment les conditions dans lesquelles un logiciel peut être exécuté.
  - Les dépendances de désinstallation expriment les conditions dans lesquelles un logiciel peut être désinstallé.

CV

Recherche

Rayonnement

## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

### Etape 1.2: Modélisation et sécurité du logiciel composite FOSS

- ☞ Un ensemble d'exigences doivent être considérées lors de la définition d'un logiciel composite FOSS (Di Cosmo et al., 2014):
  - REQ1: l'installation d'un logiciel nécessite l'installation de tous les logiciels requis.
  - REQ2: les logiciels en cours d'exécution ou ayant fourni des ports utilisés par d'autres ne peuvent pas être désinstallés .
  - REQ3: l'exécution d'un logiciel nécessite l'exécution de tous les logiciels requis.
  - REQ4: les ports fournis doivent satisfaire un nombre maximal de liaisons.
  - REQ5: les ports doivent satisfaire à un nombre minimal de liaisons.

CV

Recherche

Rayonnement

## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

### Etape 1.2: Modélisation et sécurité du logiciel composite FOSS



Pour modéliser le logiciel com

Une fonction qui définit l'ensemble des ports dans un composant.

Une fonction qui définit les ports dans un composant.

Une fonction qui définit les contraintes de capacités requises et fournis des composants.

SETS  
PORTS  
CONSTANTS  
requiredPorts providedPorts portsCapacity  
AXIOMS

axm1: providedPorts  $\in$  SOFTWARE  $\times$  STATE  $\leftrightarrow$  P(PORT)  
axm2: requiredPorts  $\in$  SOFTWARE  $\times$  STATE  $\leftrightarrow$  P(PORT)  
axm3: portsCapacity  $\in$  SOFTWARE  $\times$  PORT  $\leftrightarrow$  N  
...  
END

Nous utilisons la variable SB pour définir l'ensemble des liaisons entre les composants logiciels.

VARIABLES  
CS states SB ports  
INVARIANTS  
type1: CS e P  
type2: SB e P  
type3: portsCapacity e N  
END

Un ensemble fini dénote tous les ports qui sont fournis ou requis par un logiciel.

CV

Recherche

Rayonnement

## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

### Etape 1.2: Modélisation et sécurité du logiciel composite FOSS

- Nous avons formalisé les exigences de composition dans la CFOSSMachine comme des invariants :

**REQ1:**  $\forall c1, p \cdot c1 \in CS \wedge states(c1) = installed \wedge p \in requiredPorts(c1 \mapsto installed)$   
 $\Rightarrow (\forall c2 \cdot c2 \in CS \wedge SB(c2 \mapsto c1) = p \Rightarrow states(c2) \in \{installed, running\})$

**REQ2:**  $\forall c1 \cdot c1 \in CS \wedge (\exists p, c2 \cdot p \in ports \wedge c2 \in CS \wedge SB(c1 \mapsto c2) = p)$   
 $\Rightarrow states(c1) \in \{installed, running\}$

**REQ3:**  $\forall c1, p \cdot c1 \in CS \wedge states(c1) = running \wedge p \in requiredPorts(c1 \mapsto running)$   
 $\Rightarrow (\forall c2 \cdot c2 \in CS \wedge SB(c2 \mapsto c1) = p \Rightarrow states(c2) = running)$

**REQ4:**  $\forall c1, p, s \cdot c1 \in CS \wedge s \in STATE \wedge p \in providedPorts(c1 \mapsto s)$   
 $\Rightarrow portsCapacity(c1 \mapsto p) \leq card(\{c2 \cdot c2 \in CS \wedge (c1 \mapsto c2) \mapsto p \in SB|c2\})$

**REQ5:**  $\forall c1, p, s \cdot c1 \in CS \wedge s \in STATE \wedge p \in requiredPorts(c1 \mapsto s)$   
 $\Rightarrow portsCapacity(c1 \mapsto p) \geq card(\{c2 \cdot c2 \in CS \wedge (c2 \mapsto c1) \mapsto p \in SB|c2\})$

CV

Recherche

Rayonnement

## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

## Etape 2: Modélisation de logiciels composites FOSS élastiques

- L'élasticité est la caractéristique principale du paradigme Cloud Computing.
- Nous distinguons deux types de propriétés d'élasticité :
  - **Propriété d'élasticité verticale:** permet d'augmenter ou de réduire la capacité des logiciels déployées dans le Cloud en ajoutant de nouvelles ressources (logiciels) ou en supprimant celles qui ne sont pas en cours d'utilisation
  - **Propriété d'élasticité horizontale:** permet d'augmenter ou de diminuer les ressources logicielles en ajoutant de nouvelles copies de composants existants, tout en supprimant les inutiles



## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

## Les propriétés d'élasticité



L'élasticité verticale est définie par les opérations suivantes:

- create: une opération pour la création d'un nouveau logiciel FOSS.
- destroy: une opération pour la suppression d'un logiciel FOSS existant.
- bind: une opération pour la définition d'une liaison entre deux logiciels FOSS existants.
- unbind: une opération pour la suppression d'une liaison existante.

L'élasticité horizontale définie par les opérations suivantes:

- Duplicate: une opération pour la création de nouvelles instances (copies) de logiciels FOSS existants.
- Consolidate: une opération de suppression d'instances existantes de logiciels FOSS existants.

CV

Recherche

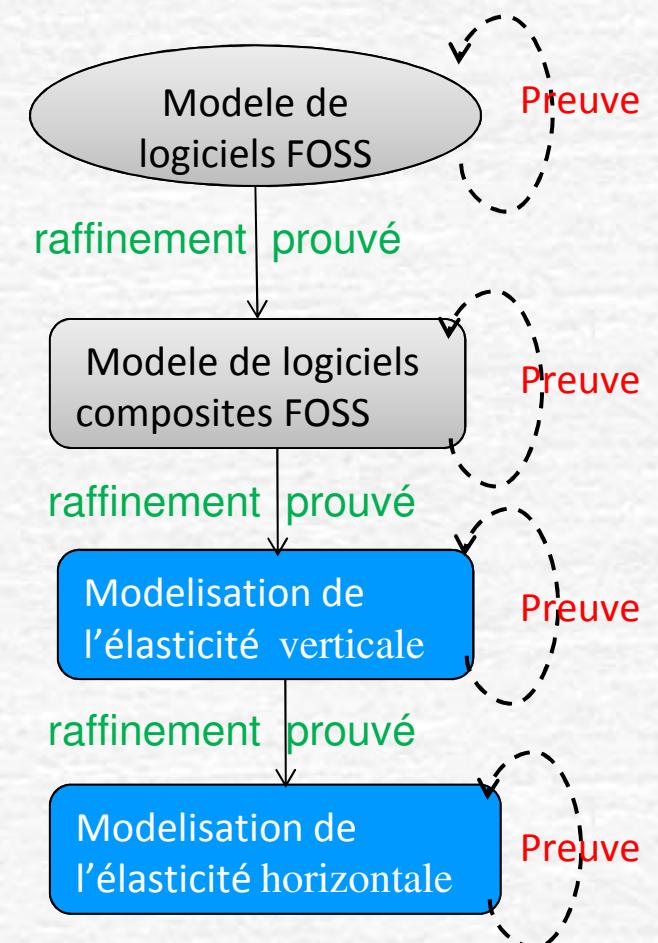
Rayonnement

## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

## Approche de modélisation des propriétés d'élasticité

- Pour modéliser les propriétés d'élasticité du logiciel composite FOSS, nous avons adopté l'approche suivante composée de deux sous-étapes :
  - Étape 2.1: Modélisation de l'élasticité verticale de logiciel composite FOSS.
  - Étape 2.2: Modélisation de l'élasticité horizontale de logiciel composite FOSS.



## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

### Étape 2.1: modélisation de l'élasticité verticale

- Pour modéliser l'élasticité verticale dans les logiciels FOSS, nous avons

Les variables abstraites de la machine MFOSS1, CS, states, SB et ports, sont respectivement remplacées dans la nouvelle machine par les suivantes : CS1, states1, SB1 and ports1.

```
MACHINE  
SEES CFOSS1  
VARIABLES  
CS1 states1 S  
INVARIANT  
type1: CS1 e  
type2: states1  
type3: ports1  
type4: SB1 e  
glu1: CS = CS1  
glu2: states = states1  
glu3: ports = ports1  
glu4: SB = SB1  
...  
END
```

Nous avons défini un ensemble d'invariants de collage pour assurer l'exactitude du raffinement. Ces invariants particuliers définissent une relation d'égalité entre les variables abstraites et concrètes..

CV

Recherche

Rayonnement

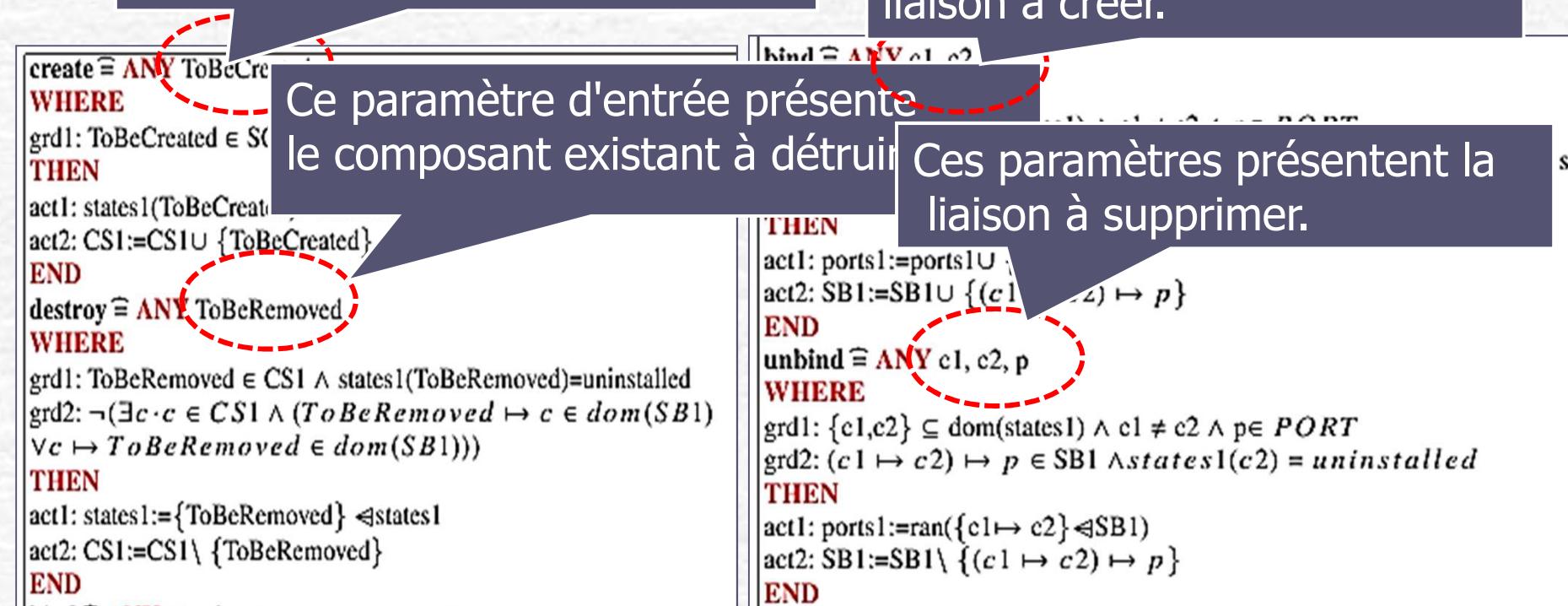
## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

### Étape 2.1: modélisation de l'élasticité verticale

Ch Ce paramètre d'entrée présente dans le nouveau composant à créer.

ticale est formellement définie  
ch Ces paramètres présentent la liaison à créer.



CV

Recherche

Rayonnement

## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

### Étape 2.2: modélisation de l'élasticité horizontale

- Pour modéliser l'élasticité horizontale, nous avons adopté les étapes:

Un nouveau paramètre est ajouté lors de l'extension de l'événement create.

Ce nouveau paramètre présente le composant à dupliquer.

ération de duplication.  
Opération de

Consolidation

**duplicate** = ~~REFINES~~ **create**  
**ANY** ToBeCreated, component  
**WHERE**

...  
grd2: component ∈ CS1

...

**THEN**

...  
act3: copies(component):=copies(component) ∪ {ToBeCreated}  
act4: SB1:=SB1 ∪ {c · c ∈ CS1 ∧ c ↦ component ∈ dom(SB1)  
|(c ↦ ToBeCreated) ↦ SB1(c ↦ component)} ∪ {c · c ∈ CS1 ∧ component ↦ c ∈ dom(SB1) |(ToBeCreated ↦ c) ↦ SB1(component ↦ c)}

**END**

**consolidate** = ~~REFINES~~ **destroy**  
**ANY** ToBeRemoved, component  
**WHERE**

Ce nouveau paramètre présente le composant à consolider.

**THEN**

...

**END**

CV

Recherche

Rayonnement

## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

## Etape 3: Modélisation de l'allocation de ressources cloud pour les logiciels FOSS

- ☞ Trois types de ressources cloud:
  - Computing
  - Networking
  - Storage
- ☞ Propriétés des ressources cloud
  - Les propriétés d'élasticité vertical et horizontale: La duplication et la consolidation de ressources
  - Les propriétés de partage de ressources cloud
- ☞ Relations de substitution entre les ressources cloud
- ☞ Capacité de partage de ressources cloud.



## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

## Etape 3: Modélisation de l'allocation de ressources cloud pour les logiciels FOSS



### Exigences d'allocation de ressources cloud:

- CRA-1: une ressource cloud peut être remplacée par une autre de même type.
- CRA-2: une ressource cloud dont la capacité est inférieure à 1 n'est pas partageable.
- CRA-3: une ressource cloud ayant une capacité est supérieure à 2 est partageable.
- CRA-4: une ressource cloud doit être allouée en fonction de sa capacité de partage.

CV

Recherche

Rayonnement

## Contribution

# Vérification de la composition et sécurité de logiciels FOSS

## Etape 3: Modélisation de l'allocation de ressources cloud pour les logiciels FOSS



### Exigences d'allocation de ressources cloud (suite):

- CRA-5: une ressource cloud est déclarée libre lorsqu'elle n'est utilisée par aucun composant FOSS.
- CRA-6: une ressource cloud allouée doit être compatible avec les types de ressources cloud requis par FOSS.
- CRA-7: l'exécution d'un logiciel FOSS nécessite l'allocation de toutes ses ressources cloud requises.
- CRA-8: la désinstallation d'un logiciel FOSS existant déclenche la libération des ressources cloud qui lui sont alloué

CV

Recherche

Rayonnement

## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

### Etape 3: Modélisation de l'allocation de ressources cloud pour les logiciels FOSS

- ☞ Allocation/libération de ressources cloud:
  - **allocate**: allocation d'une ressource cloud à un FOSS
  - **deallocate**: libération d'une ressource cloud utilisée par un FOSS
- ☞ Elasticité verticale de ressources cloud
  - **create\_resource**: création de nouvelles ressources cloud
  - **destroy\_resource**: destruction des ressources inutiles
- ☞ Elasticité horizontale de ressources cloud
  - **duplicate\_resource**: création de nouvelles copies de ressources cloud
  - **consolidate\_resource**: destruction des copies inutiles de certaines ressources

CV

Recherche

Rayonnement

## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

## Etape 3: Modélisation de l'allocation de ressources cloud pour les logiciels FOSS

- Pour modéliser la gestion de ressources cloud se fait en deux étapes:
  - Étape 3.1: modélisation de l'allocation/libération et la propriété d'élasticité verticale de ressources
  - Étape 3.2: modélisation de la substitution et des propriétés de partage et d'élasticité horizontale de ressources.



## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

### Étape 3.1: modélisation de l'allocation/libération et la propriété d'élasticité verticale

- En premier lieu, nous créons un contexte appelé CFOSS2 qui étend le contexte CFOSS1 suivants:

- **RESOURCES**: un ensemble paramétrable.
- **RTYPES**: un ensemble énumérant les types de ressources.
- **RSTATES**: un ensemble énumérant les états de ressources à savoir free, allocated.
- **hasTypes**: une fonction déterminant les types d'une ressource.
- **requiredResources**: une fonction spécifiant l'ensemble des ressources requises par les FOSS.

```
CONTEXT CFOSS2
EXTENDS CFOSS1
SETS
RESOURCES RTYPES RSTATES
CONSTANTS
compute storage network hasType requiredResources free allocated
AXIOMS
axm1: finite(RESOURCES)
axm2: partition(RTYPES, {compute}, {storage}, {network})
axm3: hasType ∈ RESOURCES → RTYPES
axm4: partition(RSTATES, {free}, {allocated})
axm5: requiredResources ∈ SOFTWARE ↦ P(RESOURCES)
END
```

## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

### Étape 3.1: modélisation de l'allocation/libération et la propriété d'élasticité verticale

- En second lieu et compte tenu de ce nouveau contexte, nous raffinons la machine MFOSS3 en ajoutant les éléments suivants:
  - allocation:** une variable qui stocke les ressources allouées à chaque processus. Elle est définie comme une fonction de type CS1 vers SB1 ports1 copies.
  - resources:** une variable qui stocke les ressources utilisées par le composant. Elle est définie comme une fonction de type CS1 vers P(resources).
  - resourceStatus:** une variable pour spécifier l'état courant de chaque ressource.

```
MACHINE MFOSS4 REFINES MFOSS3
SEES CFOSS2
VARIABLES
CS1 states1 SB1 ports1 copies
allocation resources resourceStatus
INVARIANTS
inv1: resourceStatus ∈ RESOURCES → RSTATES
inv2: resources ∈ P(RESOURCES)
inv3: allocation ∈ CS1 → P(resources)
...
END
```

CV

Recherche

Rayonnement

## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

## Étape 3.1: modélisation de l'allocation/libération et la propriété d'élasticité verticale

- En second lieu et compte tenu de ce nouveau contexte, nous raffinons la machine MFOSS4 par l'introduction des éléments suivants (suite):
  - **allocate**: un évènement pour l'allocation d'une ressource à un FOSS passé en paramètre.
  - **deallocate**: un évènement pour libérer une ressource l'alloué à un FOSS passé en paramètre.
  - **create\_resource**: un évènement pour créer une nouvelle ressource qui sera allouer par la suite aux composants FOSS.
  - **destroy\_resource**: un évènement pour supprimer une ressource n'est pas encours d'utilisation (libre). Une telle ressource doit être à l'état 'free'.

CV

Recherche

Rayonnement

## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

## Étape 3.2: modélisation de la substitution et des propriétés de partage/ élasticité horizontale

En premier lieu l'introduction de

- **elastic:** une fonction dans  $\text{BOOL}$ . Ces ressources sont élastiques.
- **shareable:** une fonction  $\text{RESOURCES} \rightarrow \text{BOOL}$  qui indique si les ressources sont partageables.
- **depALT:** une fonction de remplacement entre les ressources.

**EXTENDS CFOSS2**

**CONSTANTS**

$\text{elastic shareable depALT resourceCapacity}$

**AXIOMS**

$\text{axm1: } \text{elastic} \in \text{RESOURCES} \rightarrow \text{BOOL}$

$\text{axm2: } \text{shareable} \in \text{RESOURCES} \rightarrow \text{BOOL}$

$\text{axm3: } \text{depALT} \in \mathbb{P}(\text{RESOURCES} \times \text{RESOURCES})$

$\text{axm4: } \text{resourceCapacity} \in \text{RESOURCES} \leftrightarrow \mathbb{N}$

CV

Recherche

Rayonnement

## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

## Étape 3.2: modélisation de la substitution et des propriétés de partage/ élasticité horizontale

- En second lieu et compte tenu du contexte CFOSS2, nous raffinons la machine MFOSS4 de la manière suivante:
  - Le raffinement d'une propriété de partage
  - L'ajout d'un évènement de consolidation de ressources en fonction de la duplication de l'opération de raffinement de **destroy\_resource**
  - L'ajout d'un évènement, nommé **consolidate\_resource** pour modéliser la consolidation de ressources. Cet évènement est défini par raffinement de l'évènement **destroy\_resource**.

**MACHINE MFOSS5 REFINES MFOSS4**  
**SEES CFOSS3**  
**VARIABLES**  
CS1 states1 SB1 ports1 copies  
allocation resources resourceStatus resourceCopies  
**INVARIANTS**  
inv1:  $resourceCopies \in resources \leftrightarrow P(resources)$

CV

Recherche

Rayonnement

## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

## Étape 3.2: modélisation de la substitution et des propriétés de partage/ élasticité horizontale

- En troisième lieu, nous modélisons les exigences de la gestion de ressources cloud de la manière suivante:
  - Les contraintes CRA-1, CRA-2, CRA-3 et CRA-4 sont modalisées comme des axiomes dans le contexte CFOSS3.

CRA-1:  $\forall r1, r2 \cdot r1 \in RESOURCES \wedge r2 \in RESOURCES \wedge r1 \mapsto r2 \in depALT$

$$\Rightarrow hasType(r1) = hasType(r2)$$

CRA-2:  $\forall r \cdot r \in \text{dom(resourceCapacity)} \wedge resourceCapacity(r) > 1$

$$\Rightarrow shareable(r) = \text{TRUE}$$

CRA-3:  $\forall r \cdot r \in \text{dom(resourceCapacity)} \wedge resourceCapacity(r) \leq 1$

$$\Rightarrow shareable(r) = \text{FALSE}$$

→ Ces axiomes sont vérifiées statiquement par les outils Rodin

CV

Recherche

Rayonnement

## Contribution 2

# Vérification de la composition et sécurité de logiciels FOSS

## Étape 3.2: modélisation de la substitution et des propriétés de partage/ élasticité horizontale

- En troisième lieu, nous modélisons les exigences de la gestion de ressources cloud de la manière suivante (suite):
  - Les contraintes CRA-5, CRA-6, CRA-7 et CRA-8 sont modalisées comme des invariants dans la machine MFOSS5.

CRA-4:  $\forall r \cdot r \in resources \Rightarrow card(\{c \cdot c \in CS1 \wedge r \in allocation(c)|c\}) < resourceCapacity(r)$

CRA-5:  $\forall r \cdot r \in dom(resourceStatus) \wedge resourceStatus(r) = free \Rightarrow \{c \cdot c \in CS1 \wedge r \in allocation(c)|c\} = \emptyset$

CRA-6:  $\forall c, r \cdot c \in CS1 \wedge r \in allocation(c) \Rightarrow r \in requiredResources(c)$

CRA-7:  $\forall c \cdot c \in dom(states1) \wedge states1(c) = uninstalled \Rightarrow c \notin dom(allocation)$

CRA-8:  $\forall c, r \cdot c \in dom(states1) \wedge states1(c) = running \wedge r \in RESOURCES \wedge r \in requiredResources(c) \Rightarrow c \in dom(allocation) \wedge r \in allocation(c)$

CV

Recherche

Rayonnement

## Etape 4: Correction et validation.

- Démarche de vérification de la formalisation se fait en trois étapes:
  - Étape 1: Écrire un modèle Event-B
  - Étape 2: Décharge des obligations de preuve
  - Étape 3: Validation par animation.
- L'activité de vérification est basée sur:
  - Preuves de théorèmes.
  - Model-checking.
- Les outils utilisés :
  - La plateforme RODIN
  - L'animateur ProB.

# Plan

CV

Rayonnement

Recherche

# Activités de recherche

## Quelques Indicateurs

### ■ Encadrements

- 4 thèses soutenues
- 3 thèses en cours
- 15 stages de master

### ■ Publications

- 10 Journaux 4[Q1], 3[Q2]
- 50 Conférences : 16 [A], 12 [B], 15[C]

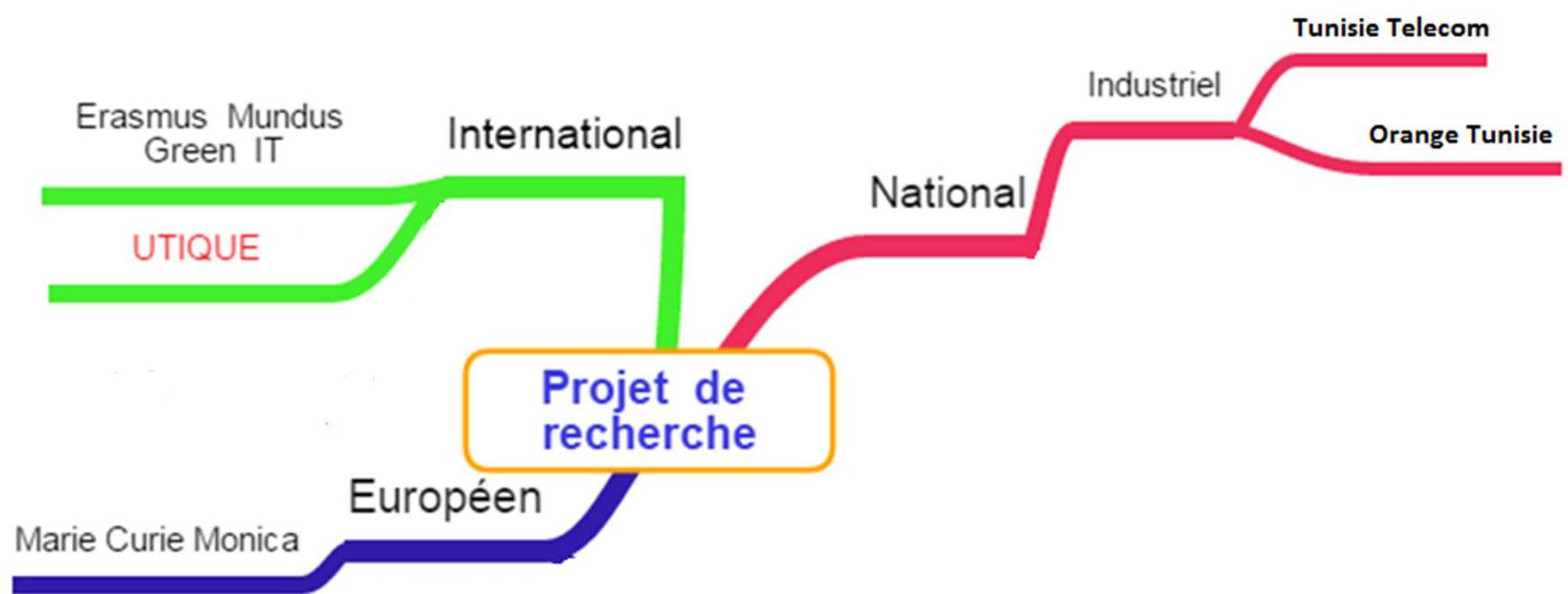
CV

Recherche

Ravonnement

# Activités de recherche

## Projets Récents

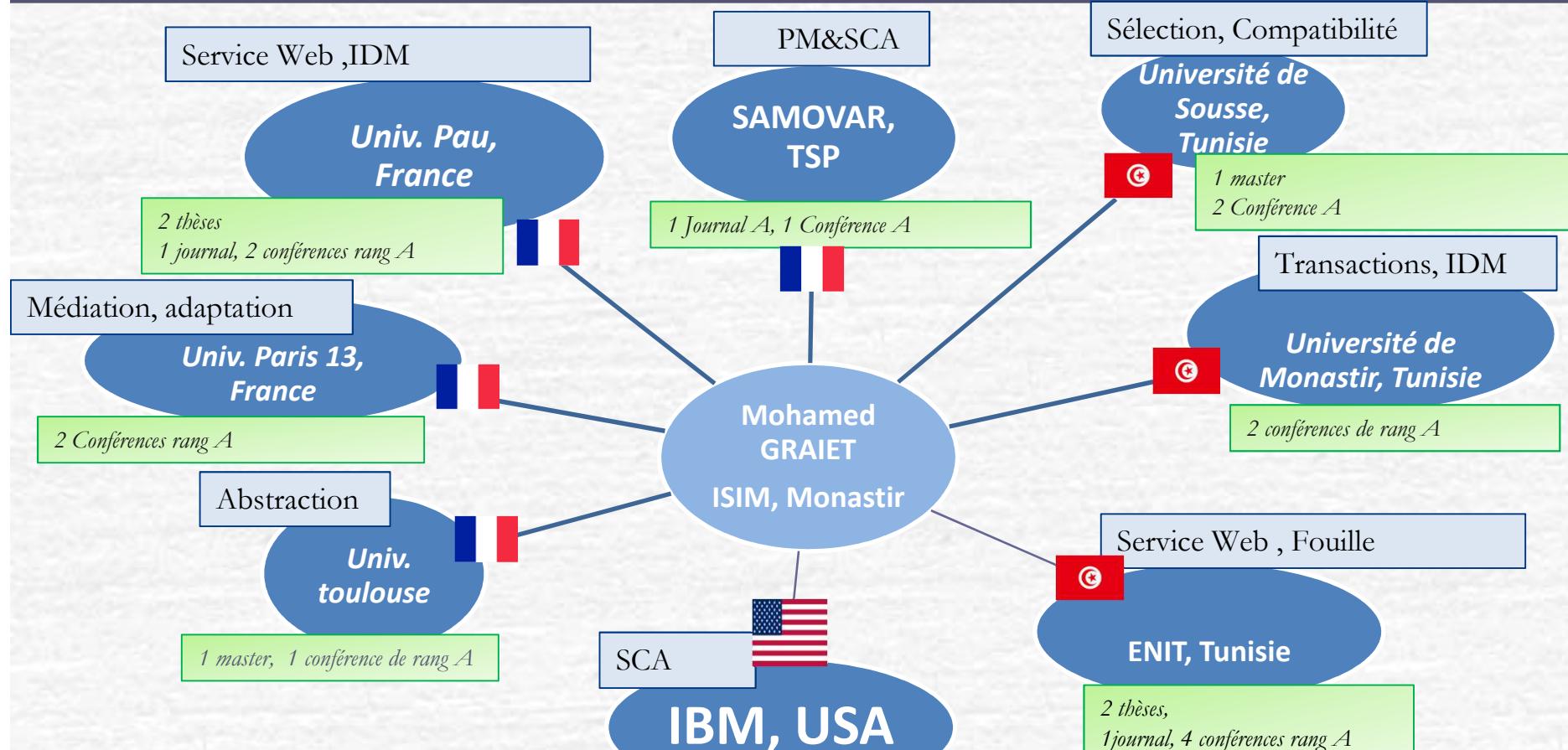


CV

Recherche

Rayonnement

# Rayonnement Collaborations nationales et internationales



61

CV

Recherche

Rayonnement

**Merci de votre  
attention!**