

# ACI Dispo - Aspects de disponibilité

Stéphane HONG TUAN HA, Pascal FRADET

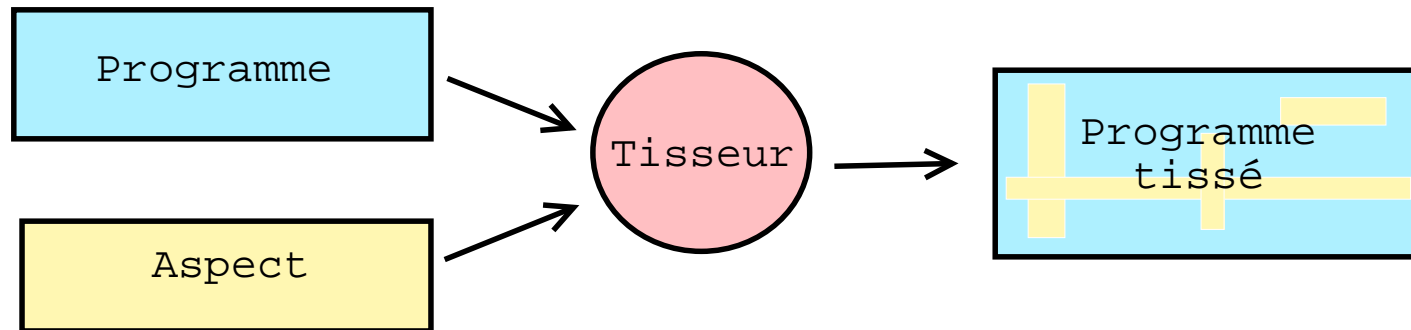
IRISA/INRIA Rennes & INRIA Rhône-Alpes

# Contexte et Motivation (1)

- ▶ Problématique de la disponibilité
  - ▶ "assurer l'accès aux informations"
  - ▶ dénis de services logiciels ou matériels
  - ▶ politiques et propriétés de disponibilité
- ▶ Proposition d'une solution pour les dénis de services logiciels liés à la gestion des ressources

# Contexte et Motivation (1)

- ▶ La programmation par aspects



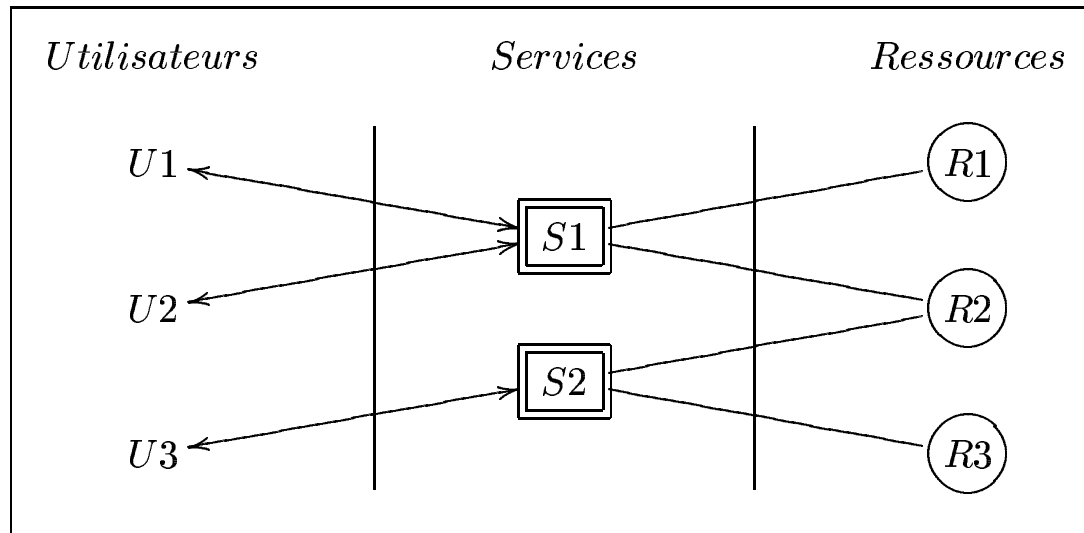
- ▶ Notre approche
  - ▶ les aspects comme des propriétés
  - ▶ représenter les aspects comme des automates
  - ▶ et modéliser le tissage comme un produit d'automates
- ▶ Appliquer la programmation par aspects à la disponibilité

# Contexte et motivation (2)

- ▶ Notre travail
  - ▶ prévention des dénis de service logiciels comme un aspect
  - ▶ aspect pour exprimer la politique de disponibilité
    - ▶ politiques locales à un service
    - ▶ politiques de type “temps borné”
  - ▶ formalisation reposant sur les automates temporisés

# Modèle pour la disponibilité

- ▶ Modèle structuré en 3 couches



- ▶ correspond à un modèle client-serveur
- ▶ services traitent les requêtes des utilisateurs
- ▶ services en concurrence pour l'accès aux ressources
- ▶ possibilité de tisser les services

# Prévention des dénis de service

- ▶ Solution standard : insertions manuelles de timers dans le code du service
  - ▶ préoccupation transverse au traitement des utilisateurs
  - ▶ pas de support pour la configuration et la réutilisation
  - ▶ impossible de connaître la politique de disponibilité
- ▶ Un aspect de disponibilité
  - ▶ pour séparer la politique de disponibilité et le code du service
  - ▶ pour exprimer des politiques de disponibilité plus élaborées
  - ▶ pour vérifier des propriétés de disponibilité

# Les services

## ▶ Syntaxe

- ▶ définition d'un service par un ensemble d'instructions de la forme :

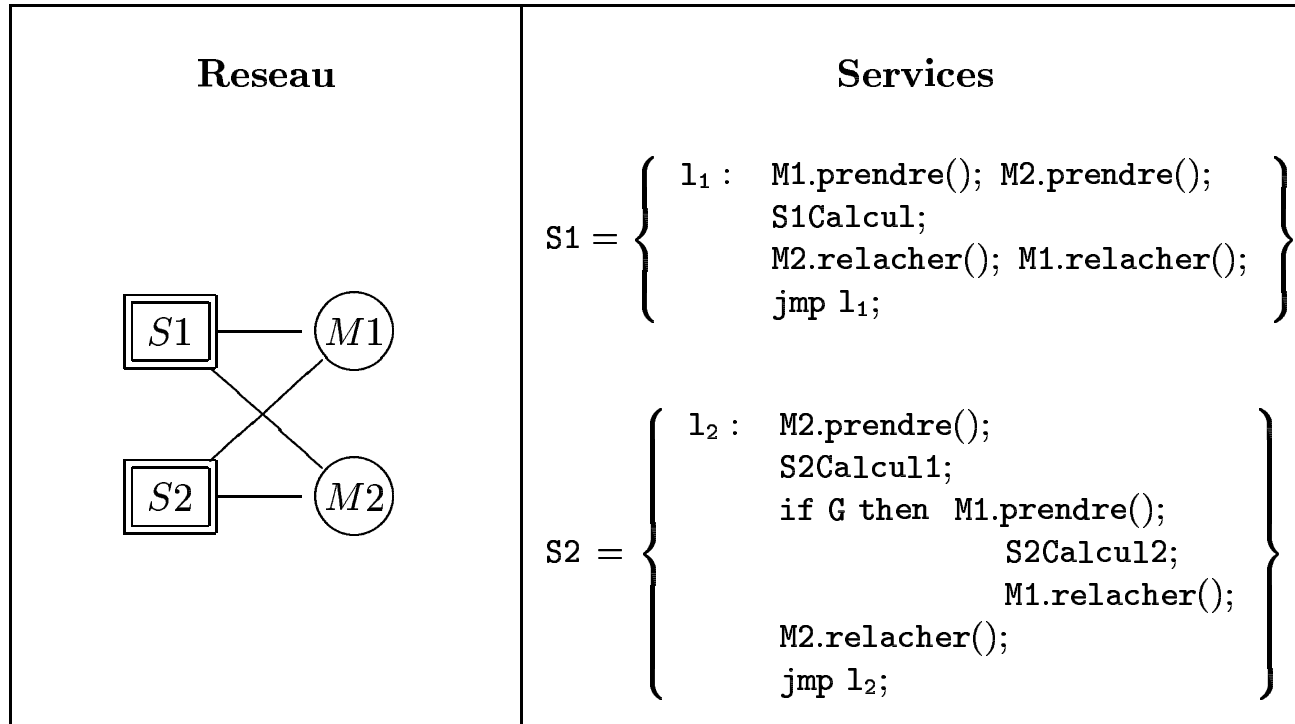
$$I ::= l_1 : c \rightsquigarrow l_2 \mid l_1 : g \rightsquigarrow l_2 ; l_3$$

- ▶  $l_1, l_2$  et  $l_3$  : des *étiquettes*
- ▶  $c$  une *commande* (e.g., une affectation)
- ▶  $g$  un *test*

## ▶ Sémantique

- ▶ donnée par un LTS
- ▶ représente le comportement du service

# Exemple de système



- ▶ 2 possibilités de dénis de service :
  - ▶ dénis de service de type famine si *S2Calcul1* ne termine pas
  - ▶ interblocage pour la prise des 2 ressources

# Le langage d'aspect (1)

- ▶ Inspiré du framework de (Douence et al, 2004)
  - ▶  $(C \triangleright I)$  : filtre la trace d'exécution et quand un événement correspond à  $C$  exécute  $I$
  - ▶ combinaison de la commande  $(C \triangleright I)$  par séquence, répétition ou choix
- ▶ Modifications pour traiter la disponibilité
  - ▶ ajout de gardes temporelles dans le crosscut
  - ▶ ajout d'inserts dédiés à la disponibilité

# Le langage d'aspect (2)

► Début de la grammaire :

$I ::=$  *reset* ( $i, k$ ) ; programme l'exception  $i$  pour un  
; déclenchement dans  $k$  unités de temps  
| *cancel* ( $i$ ) ; annule l'exception  $i$   
| *start* ( $t$ ) ; initialise le minuteur  $t$   
| *wait* ( $t, k$ ) ; attend jusqu'à ce que  $t$  vaille  $k$   
| *nop* ; opération vide

# Le langage d'aspect (3)

► Suite de la grammaire :

$A$	$::=$	$\{a_1 = E_1, \dots, a_n = E_n\}$	; équations mutuellement récursives
$E$	$::=$	$E_1 \square E_2$	; choix
		$((F, G) \triangleright L); a_i$	; (*)
$F$	$::=$	$Motif \mid F_1 \wedge F_2 \mid \neg F$	; filtre d'évènement de base
$G$	$::=$	$\{\dots, t \odot k, \dots\}$	; garde temporelle $\odot \in \{\leq, <, \dots\}$
$L$	$::=$	$\{I; \dots; I\}$	; liste d'inserts

(\*) : ajoute la liste d'inserts  $L$  si l'instruction est filtrée par le filtre d'évènement  $F$  et la garde temporelle  $G$  et continue avec  $a_i$

# Le langage d'aspect (4)

- ▶ Exemples de politiques de disponibilité :
  - ▶ relacher la ressource avant 25 secondes sinon interruption du service

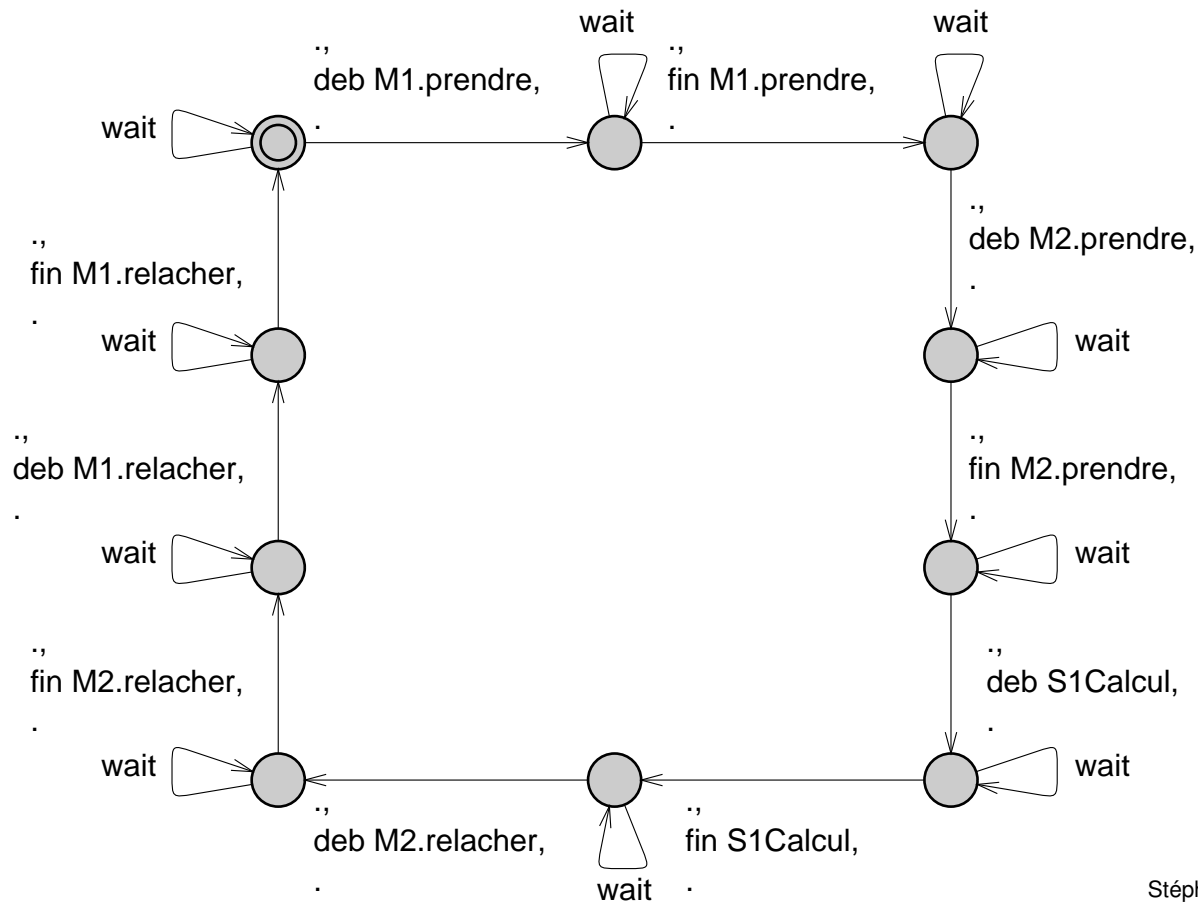
$$A_1 = \left\{ \begin{array}{l} a_1 = \text{M1.prendre} \triangleright \text{reset}(i_1, 25); a_2 \\ a_2 = \text{M1.relacher} \triangleright \text{cancel}(i_1); a_1 \end{array} \right\}$$

- ▶ attendre 20 secondes entre chaque prise de la ressource

$$\left\{ \begin{array}{l} a_1 = \text{M.prendre} \triangleright \text{start}(t); a_2 \\ a_2 = \text{M.prendre} \triangleright \{ \text{wait}(t, 20); \text{start}(t) \}; a_2 \end{array} \right\}$$

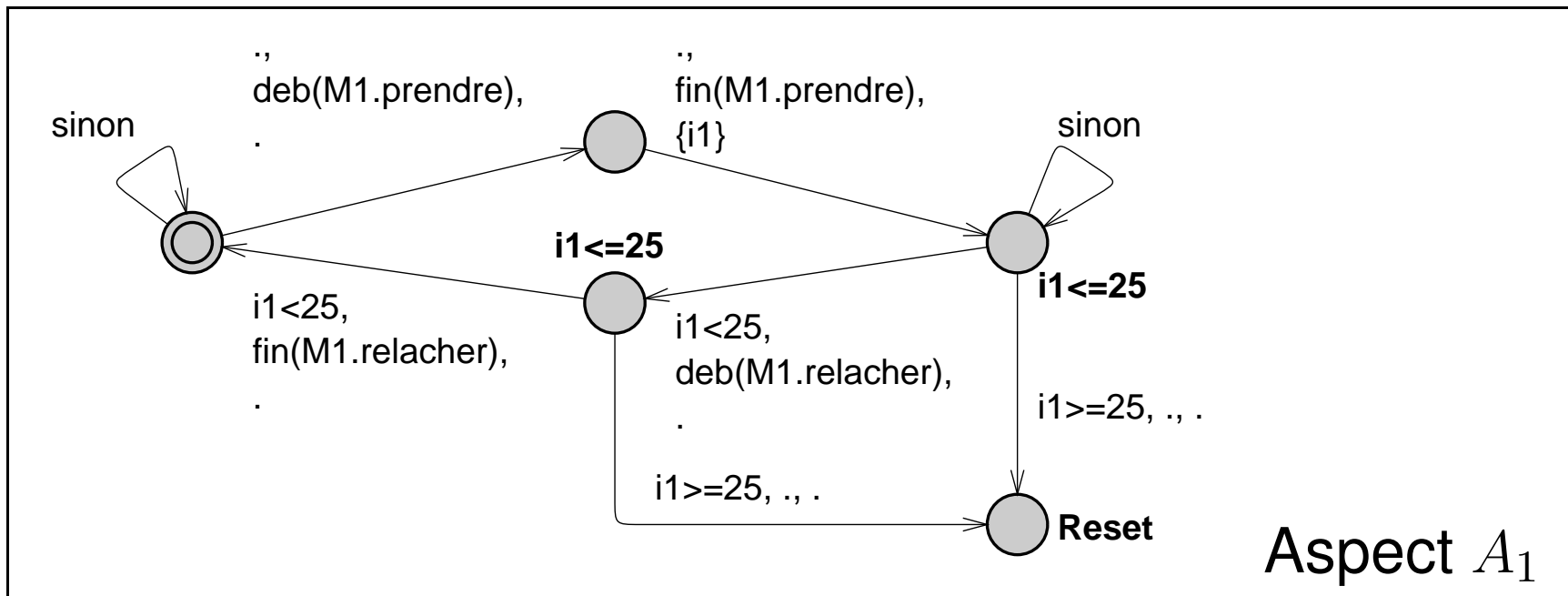
# Abstraction des services

- ▶ Abstraction du service en un automate temporel classique
- ▶ Exemple du service  $S1$



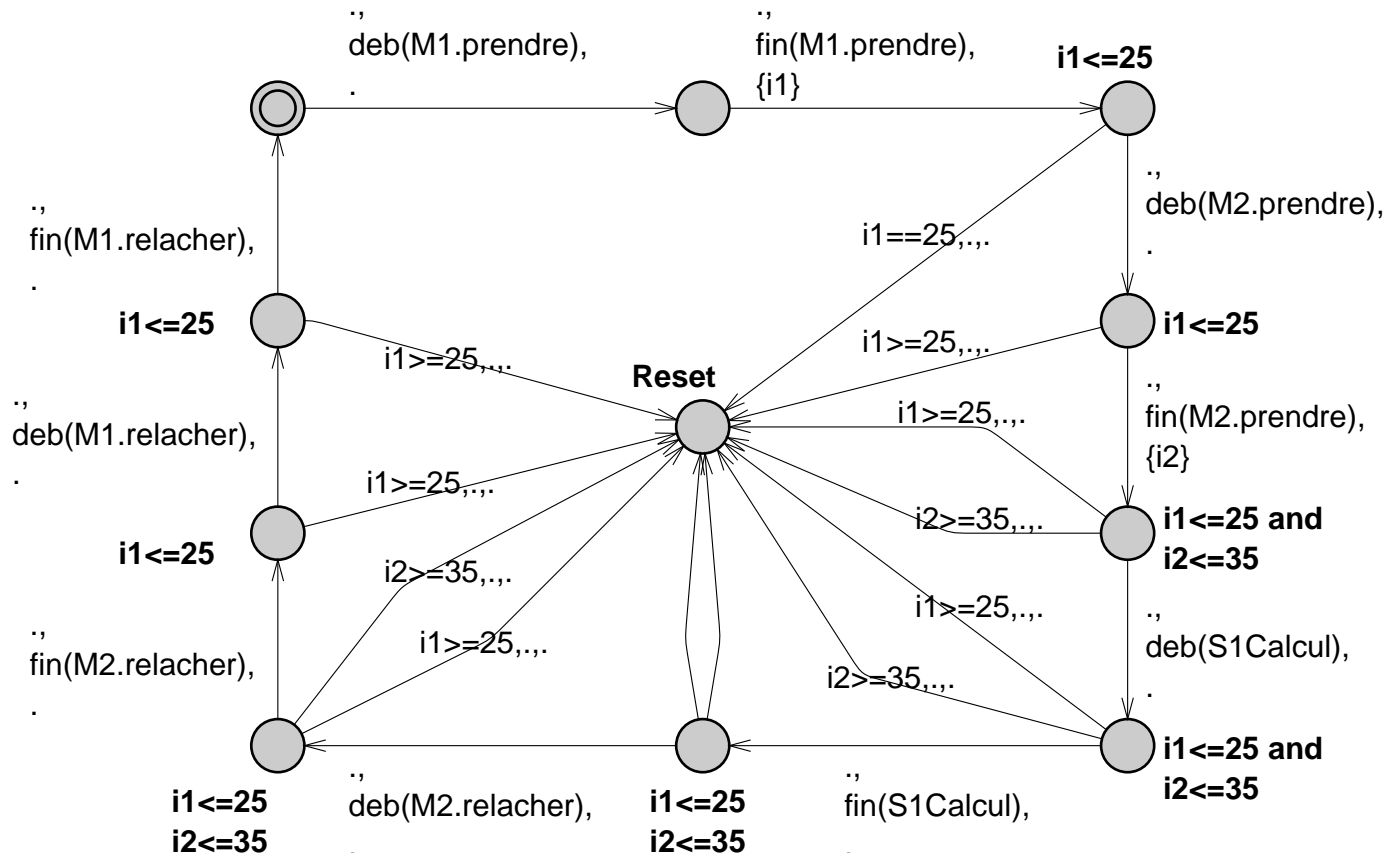
# Transformation des aspects

- ▶ Transformation en automates temporisés
- ▶ Exemple d'aspects pour le service S2 :
  - ▶  $A_1$  : relacher M1 avant 25 secondes sinon Reset
  - ▶  $A_2$  : relacher M2 avant 35 secondes sinon Reset



# Tissage

- ▶ Effectué par un produit d'automates temporisés
  - ▶ (*service* × *aspect*)
  - ▶ calcul de l'intersection des traces



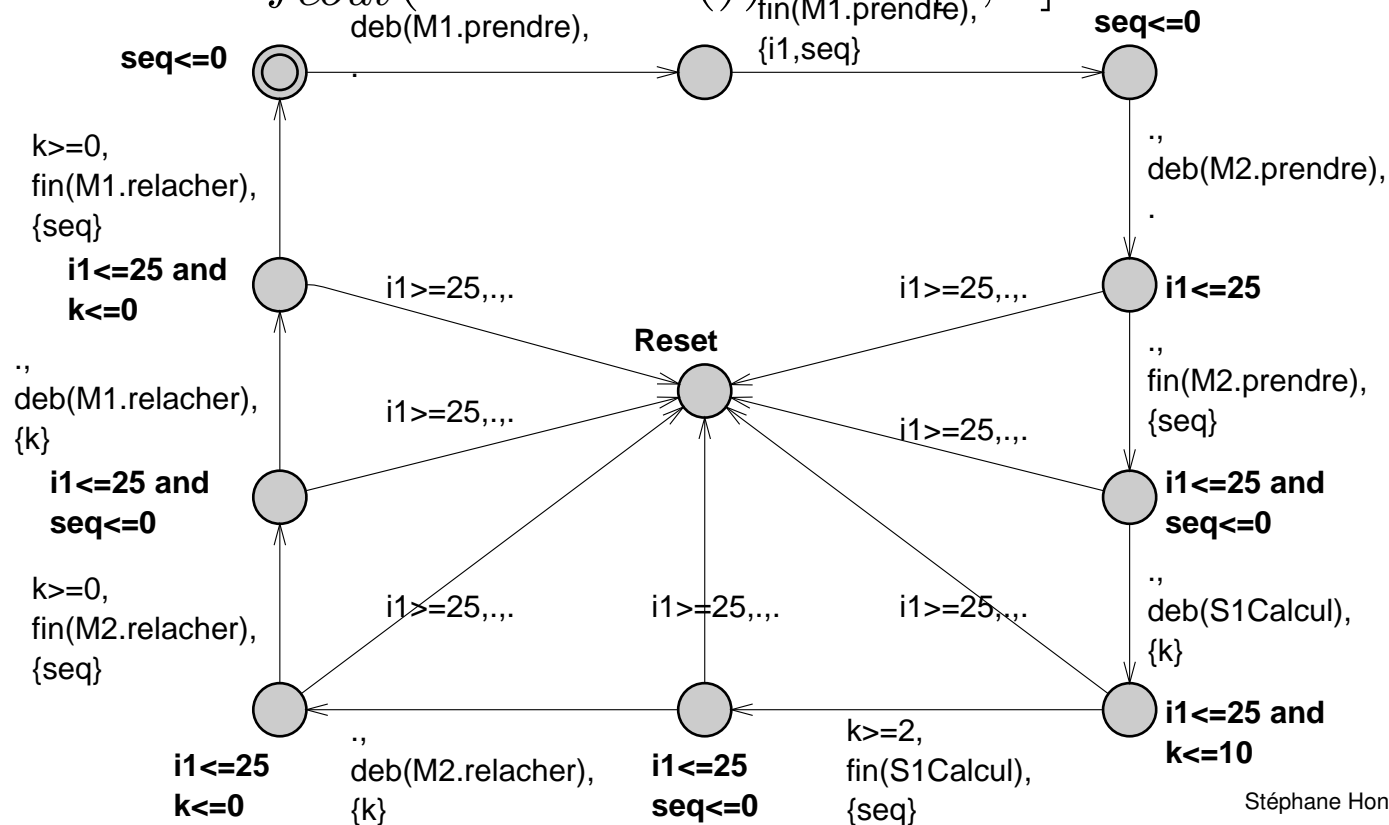
# Optimisation

- Hypothèse d'une fonction de coût qui donne le BCET et le WCET

$$f_{cout}(S1calcul) = [2, 10]$$

$$f_{cout}(prendre()) = [0, +\infty]$$

$$f_{cout}(relacher()), = [0, 0]$$



# Vérification

- ▶ Système constitué de :
  - ▶ ensemble des services tissés
  - ▶ ensemble des spécifications des ressources
- ▶ Utilisation du model-checker "UPPAAL" pour analyser le système (ressources + services tissés)
- ▶ Propriétés de disponibilité sur l'exemple après tissage et optimisation :
  - ▶ "le système est bien temporisé et n'arrive pas à des situations d'interblocages"
  - ▶ "le service S1 accomplit un tour de boucle en moins de 45 secondes"

# Concrétisation

- ▶ Gestion des traits temporels
  - ▶ utilisation de timers,  
4 commandes :  $start(t)$ ,  $wait(t, k)$ ,  $reset(i, k)$  et  $cancel(i)$
  - ▶ ajout de comparaisons de minuteurs dans les tests

$$S1 = \left\{ \begin{array}{llll} l_0 & : & \text{getUser}() & \rightsquigarrow l_1 \\ l_1 & : & \text{M1.prendre}() & \rightsquigarrow l'_1 \\ l'_1 & : & \text{reset}(i, 25) & \rightsquigarrow l_2 \\ l_2 & : & \text{M2.prendre}() & \rightsquigarrow l_3 \\ l_3 & : & \text{S1.calcul}() & \rightsquigarrow l_4 \\ l_4 & : & \text{M2.relacher}() & \rightsquigarrow l_5 \\ l_5 & : & \text{M1.relacher}() & \rightsquigarrow l'_6 \\ l'_6 & : & \text{cancel}(i) & \rightsquigarrow l_6 \\ l_6 & : & \text{endUser}() & \rightsquigarrow l_0 \end{array} \right\}$$

# Conclusion

- ▶ Contribution
  - ▶ langage d'aspect dédié à la prévention des denis de services
  - ▶ l'approche en général
    - ▶ aspects comme des propriétés temporelles
    - ▶ tissage par produit d'automates
- ▶ Travaux futurs :
  - ▶ finaliser la preuve de correction associée
  - ▶ approfondir l'étude de la prévention des interblocages