

Aspect de gestion de ressources pour la disponibilité

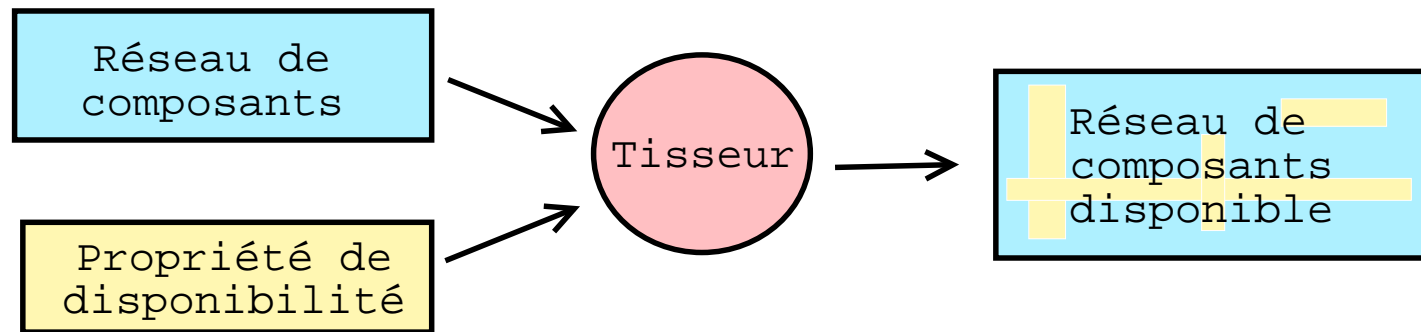
Stephane HONG TUAN HA

IRISA/INRIA, projet Lande

Contexte et Plan

▶ Objectif

- ▶ tisser des propriétés de disponibilité

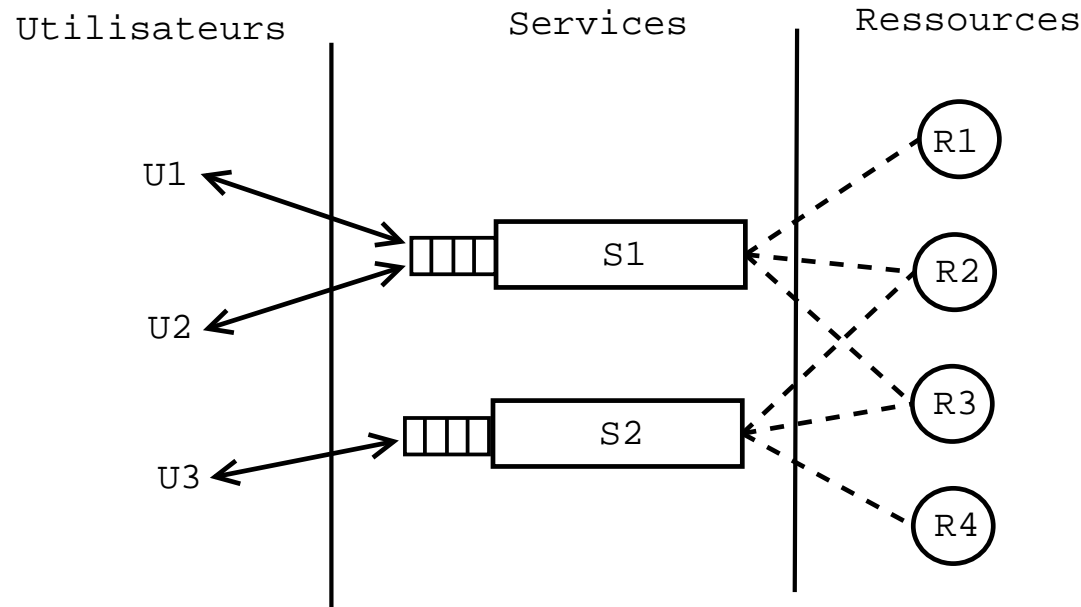


▶ Les points étudiés :

- ▶ modèle pour la disponibilité
- ▶ langage d'aspect de gestion de ressources pour la disponibilité
- ▶ modélisation et vérification avec UPPAAL

Modèle pour la disponibilité

► Modèle en 3 couches



- services en concurrence pour l'accès aux ressources
- par rapport à Yu&Gligor, possibilité de tisser les services
- nombre fixé de ressources et de services

Le langage d'aspect

- ▶ Description de la politique de disponibilité que les services doivent respecter
 - ▶ contraintes temporelles sur les opérations utilisant les ressources
- ▶ Spécification de la ressource (son état, son interface, et gestion des files d'attente)
 - ▶ pour générer la ressource à partir de la spécification ou pour tester ou vérifier l'implémentation
 - ▶ nécessaire pour déduire des propriétés de disponibilité
- ▶ Modélisation par des automates temporisés étendus avec des compteurs bornés

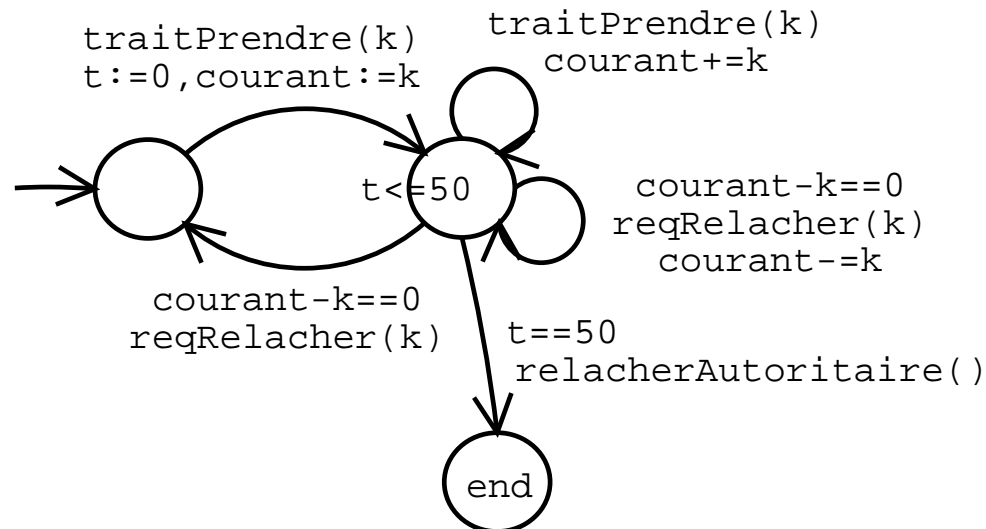
Présentation d'un exemple

La ressource 'partageable en k'

- ▶ Etat :
 - ▶ la taille de la ressource
 - ▶ un tableau qui tient à jour le nombre d'unités possédées par chaque service
- ▶ Interface :
 - ▶ *reqPrendre(k)* pour demander k unités
 - ▶ *traitPrendre(k)* quand la requête est accordée
 - ▶ *relacher(k)* pour relacher k unités
 - ▶ *relacherAutoritaire()* pour forcer la libération de toutes les unités d'un service. Si une requête *reqPrendre(k)* est en cours, elle est annulée.
- ▶ Gestion des files d'attente :
 - ▶ les *reqPrendre(k)* sont accordées selon un ordre fifo
 - ▶ les *relacher(k)* sont accordées immédiatement

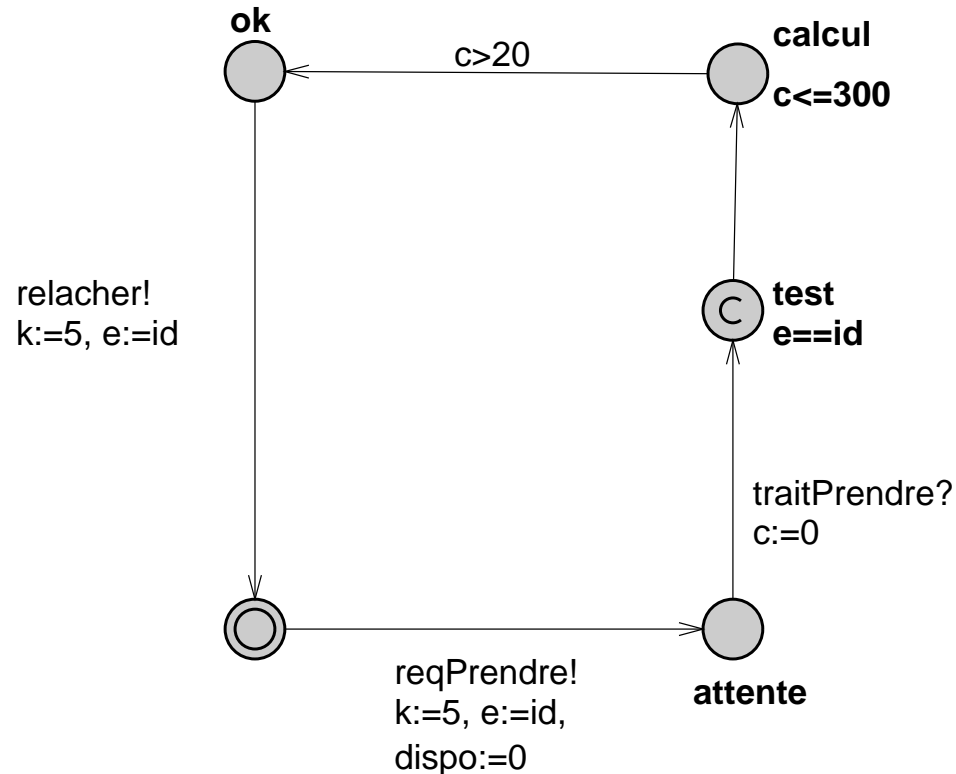
Une politique de disponibilité

- ▶ Un service qui effectue un *reqPrendre(k)* doit effectuer un *relacher(k)* avant 25 secondes sinon un *relacherAutoritaire* sera effectué avant de terminer le service.
- ▶ Modélisation avec un automate temporisé



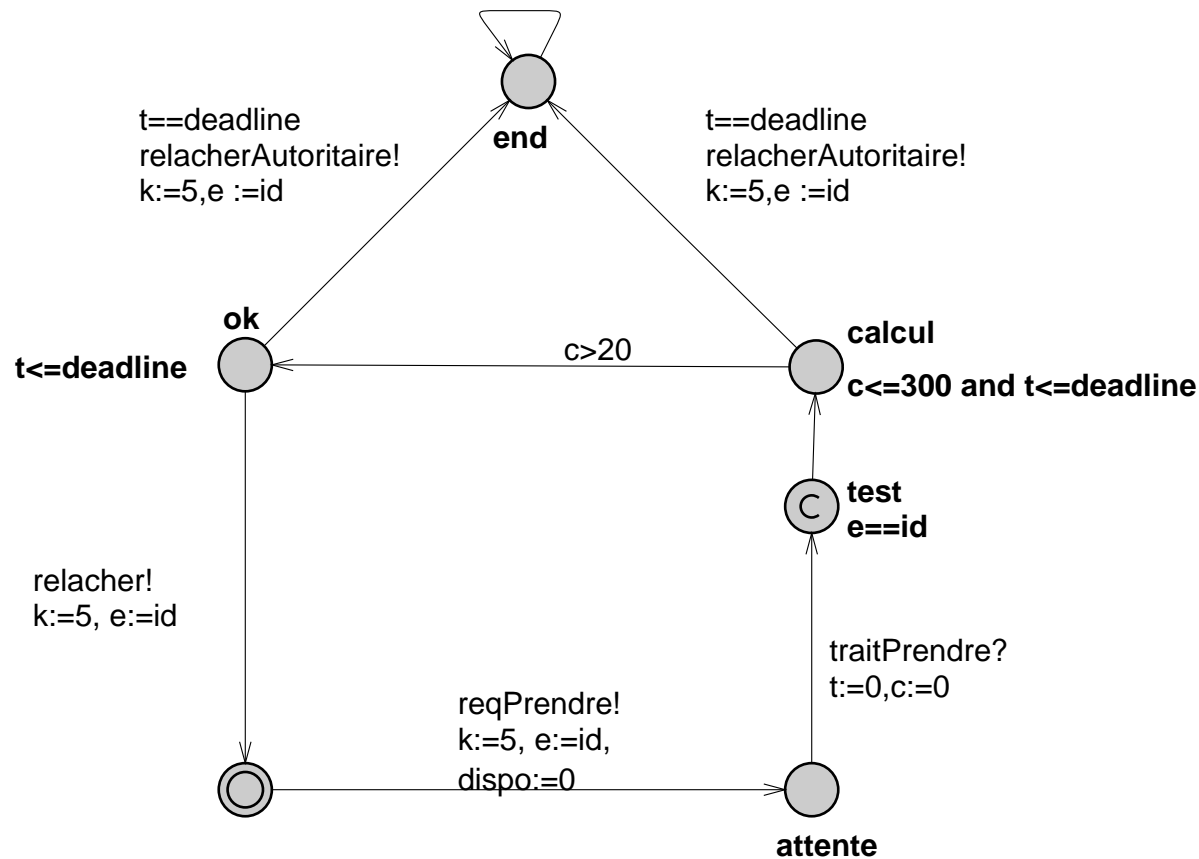
Un service

- ▶ Le service demande 5 unités, fait un calcul qui dure de 20 à 300 secondes, relache les 5 unités, et recommence.
- ▶ Modélisation avec UPPAAL :



Tissage de l'aspect sur le service

- ▶ Tissage sur la spécification du service par un 'produit' avec l'aspect



- ▶ Le tissage optimise l'aspect pour un service donné
 - ▶ analyse 'wcet' pour limiter le nombre de timers

Modélisation et vérification avec UPPAAL

- ▶ Soit un système constitué d'une ressource 'partageable en k' (taille de 10 unités) et de 3 fois le service tissé précédent
- ▶ On montre :
 - ▶ *A[] not deadlock* : le système est bien temporisé et n'arrive pas à des situations de deadlocks
 - ▶ *A[] service1.test imply service1.dispo ≤ 25* : le temps d'attente pour obtenir la ressource est inférieur ou égal à 25

Conclusion

- ▶ Un modèle pour la disponibilité en 3 couches (utilisateurs, services, et ressources)
- ▶ Un langage d'aspect
 - ▶ rendre le langage plus haut niveau (ne pas parler directement des timers et des deadlines)
 - ▶ étudier d'autres ressources (en particulier des ressources avec priorités)
 - ▶ étudier les problèmes spécifiques dans le cas de plusieurs ressources en même temps (le service ne réussit pas à accéder simultanément aux 2 ressources)

Questions

- ▶ Est-ce que le modèle pour la disponibilité est raisonnable ?
- ▶ Existe-t-il d'autres model-checker qu'UPPAAL ?
- ▶ Le langage graphique d'UPPAAL est-il acceptable pour un utilisateur ou est-ce préférable d'avoir un langage textuel ?
- ▶ ...

Annexe 2

- Spécification de la ressource ensemble avec UPPAAL (la queue qui stocke les requêtes en attente)

