

Vers le tissage de propriétés de Disponibilité

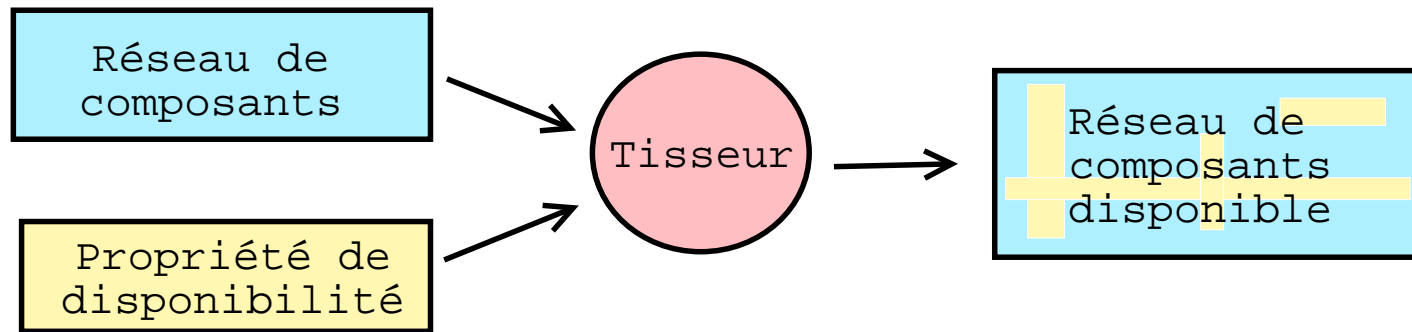
Stephane HONG TUAN HA

ACI DISPO

Inria/Irisa, projet Lande

Cadre

- ▶ Appliquer la programmation par aspects à la disponibilité



- ▶ Disponibilité de service : prévention des dénis d'accès (DoS) non autorisés aux informations
- ▶ Plusieurs type de dénis de service : matériel, par les ressources et distribué
- ▶ On se concentre sur les DoS par les ressources

Modèles pour la disponibilité

- ▶ Service comme un allocateur de ressources qui fournit un accès aux ressources
- ▶ Modèle de Yu&Gligor
 - ▶ modèle de spécification d'allocateur de ressources
 - ▶ invariant sur les ressources
 - ▶ propriété d'équité
 - ▶ contrat d'utilisateur
 - ▶ politique sur un temps d'attente fini
 - ▶ assurer la disponibilité d'un système en prouvant la politique sur le temps d'attente
- ▶ Modèle de Millen
 - ▶ modèle de moniteur qui surveille les demandes de ressources des utilisateurs pour détecter des DoS

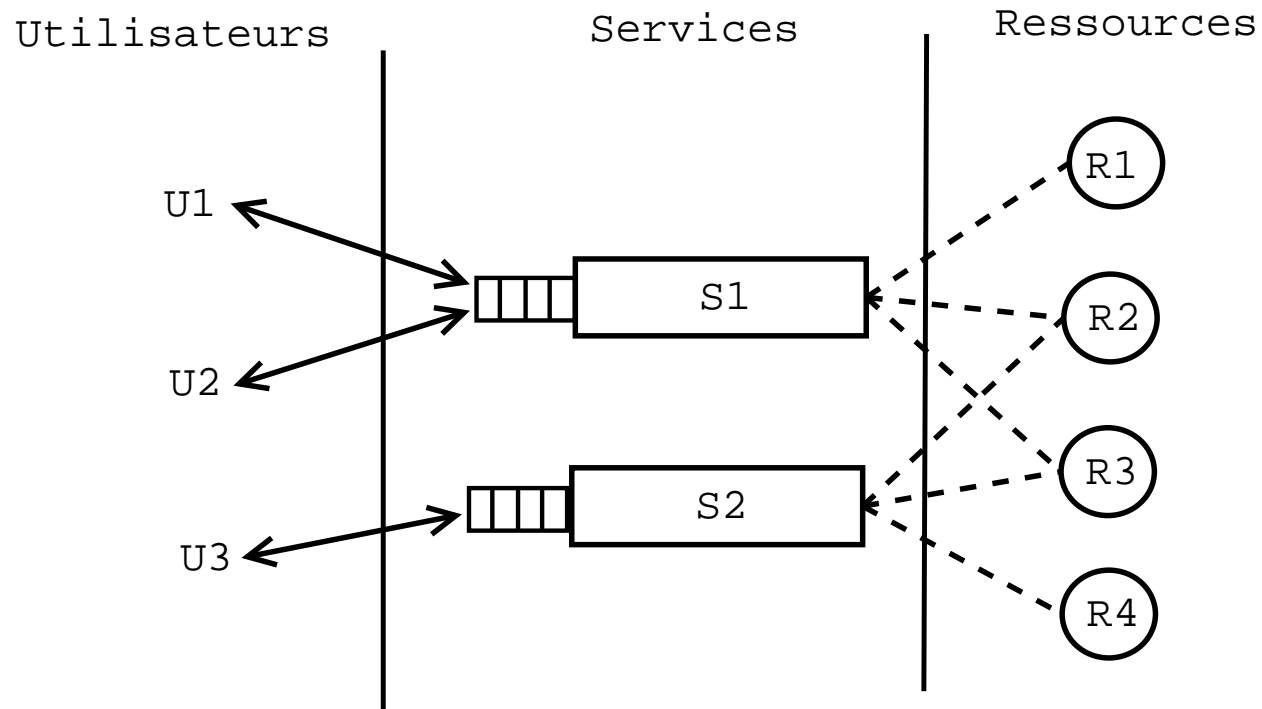
Problèmes et Approche

- ▶ Problème des utilisateurs :
 - ▶ le comportement des utilisateurs n'est pas formalisé
 - ▶ les utilisateurs doivent respecter leur contrat d'utilisateur
 - ▶ les utilisateurs ne sont pas tissables
- ▶ L'approche :
 - ▶ notion de service pour
 - ▶ formaliser l'utilisation des ressources
 - ▶ tisser le code des services

Notre cadre (1)

- ▶ Répondre aux DoS par les ressources dans une modélisation client-serveur
- ▶ Modèle en 3 couches
 - ▶ **les ressources**
 - ▶ **les services** qui nécessite les ressources pour fonctionner
 - ▶ le modèle définit un nombre fini de services
 - ▶ chaque service peut accepter un client à la fois
 - ▶ les utilisateurs attendent qu'un service soit libre pour l'utiliser (attente géré par une file FIFO)
 - ▶ **les utilisateurs** potentiellement nuisibles qui communiquent avec un service
 - ▶ nombre non borné
 - ▶ non modélisé

Notre cadre (2)



Notre cadre (3)

- ▶ Différences par rapport à Yu&Gligor
 - ▶ une vraie notion de service
- ▶ Assurer la disponibilité des ressources pour le fonctionnement des services
 - ▶ assurer que les services peuvent toujours progresser
- ▶ Assurer la disponibilité des services pour les utilisateurs

Exemple

Un serveur avec :

- ▶ un service de pages web
 - ▶ une page web avec un questionnaire de sauvegarder dans un fichier dans le répertoire 'save'
 - ▶ utilise la ressource carte réseau (r_1) et le répertoire 'save' (r_2)
- ▶ un service de surveillance
 - ▶ permet à l'utilisateur de récupérer les questionnaires du répertoire 'save'
 - ▶ utilise la carte réseau (r_1) et le répertoire 'save' (r_2)

Les ressources

- ▶ En nombre fixe
- ▶ De type accès exclusif (mutex)
- ▶ Chaque ressource $r \in \mathbf{Res}$ est utilisable par 2 instructions :
 - ▶ *take* r pour prendre la ressource r
 - ▶ *free* r pour relacher la ressource r .
- ▶ L'état d'une ressource est définie par :

$$\mathbf{StateRes} = \{lo^{cked}(p); unlo^{cked}\}$$

où $p \in P$ est le processus qui possède la ressource

Les services (1)

▶ Les services sont déterministes

▶ Un service est un ensemble de commandes :

$$P ::= \{C_1, \dots, C_n\}$$

▶ Les commandes :

$$C ::= l_1 : G \mid A \rightsquigarrow l_2$$

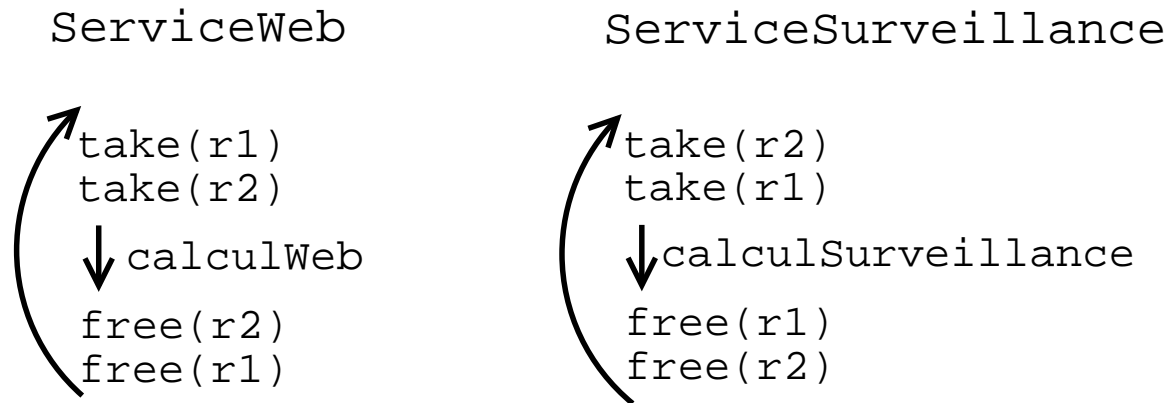
où l_1, l_2 sont des labels ($\in \text{Label}$), G une garde et A une action.

▶ Les actions :

$$A ::= f?x \mid f!x \mid \text{Take}(r) \mid \text{Free}(r) \mid I$$

Exemple

- ▶ Modélisation des services web et surveillance :



Quelques définitions

- ▶ Un store ($\text{Var} \rightarrow \text{Val}_\perp$)
- ▶ La sémantique des actions internes et des gardes donnée par les fonctions

$$\mathcal{I} : I \rightarrow \text{Sto} \rightarrow \text{Sto} \text{ et } \mathcal{G} : G \rightarrow \text{Sto} \rightarrow \mathbb{B}$$

- ▶ La sémantique d'un service par une fonction de transition entre des états constitués d'un label et d'un store
- ▶ Une trace :

$$(l_0, s_0) \xrightarrow{c_0}_p (l_1, s_1) \xrightarrow{c_1}_p \dots$$

Les services (2)

- ▶ Propriétés à vérifier
 - ▶ libération uniquement de ressources en sa possession

$$\forall t = \alpha_0 \xrightarrow{c_1}_k \dots \xrightarrow{c_i}_k \alpha_i \xrightarrow{Free(r)}_k \dots, \exists j \leq i. c_j = Take(r) \wedge \forall k \in [j + 1; i]. c_k \neq Free(r)$$

- ▶ pas de demande de ressource déjà en sa possession

Le système

- ▶ $systeme = \{(P_1), \dots, (P_n)\}$
- ▶ Représentation d'un état du système par une paire (π, β) avec :
 - ▶ $\pi : \mathbf{Prog} \rightarrow \mathbf{Sto}$: associe à chaque processus son état courant
 - ▶ $\beta : \mathbf{Res} \rightarrow \mathbf{StateRes}$: associe à chaque ressource son état

Sémantique d'un réseau de composants

- ▶ La relation de transition \longrightarrow_k du système définie par des règles d'inférences ; par exemple :

Prise d'une ressource

$$\frac{\pi \ p \xrightarrow{c}_p (l', s'), \ c = l : g \mid \text{Take}(r) \rightsquigarrow l', \ \beta \ r = \text{unlocked}}{(\pi, \beta) \xrightarrow{c}_k (\pi[p \mapsto (l', s')], \beta[r \mapsto \text{locked}(p)])}$$

- ▶ Trace :

$$(\pi_0, \gamma_0) \xrightarrow{\alpha_1}_k (\pi_1, \gamma_1) \xrightarrow{\alpha_2}_k \dots$$

- ▶ Sémantique non déterministe

Ordonnement

Formalisation de l'équité entre les services

- ▶ Nous nous intéressons uniquement aux traces équitables
- ▶ Equité forte (strong fairness) :
Si une action à partir d'un état est infiniment fois 'enabled' alors elle doit finalement être exécutée.

$Fair(\alpha_0 \xrightarrow{c_1}_k \dots \xrightarrow{c_n}_k \alpha_n \xrightarrow{c_{n+1}}_k \dots) = \forall n \in \mathbb{N}, \forall c \in Enabled_k(\alpha_n), (c : \text{infiniment fois enabled}), \exists m > n. c_m = c$
avec

$Enabled_k(\pi, \gamma) = \{c = l : g \mid a \rightsquigarrow l' \mid p \in P \wedge c \in Enabled(\pi p) \wedge (a = Take(r) \Rightarrow \beta r = unlocked)\}$

Notion de disponibilité

Formalisation du progrès d'un service

- ▶ Une propriété de disponibilité exprimée sur les traces
- ▶ Toute action 'enabled' même bloquée dans l'état α_i sera finalement exécutée

$$\forall t \in Traces(k), \forall i, t = \alpha_0 \xrightarrow{c_1}_k \dots \xrightarrow{c_i}_k \alpha_i \xrightarrow{c_{i+1}}_k \dots, \forall c \in Enabled_{all}(\alpha_n), \exists m > n. c_m = c$$

avec

$$Enabled_{all}(\pi, \gamma) = \{c \mid p \in P \wedge c \in Enabled(\pi p)\}$$

- ▶ La propriété à prouver pour montrer qu'un service progresse

Le contrat d'utilisation des ressources

- ▶ Propriétés à vérifier par les services
 - ▶ propriétés à imposer par des aspects
 - ▶ obliger la libération des ressources
 - ▶ obliger la terminaison des requêtes
 - ▶ absence de deadlocks

Libération des ressources

- ▶ Obliger la libération des ressources

$$\forall t \in \text{Traces}(R), \forall c_i = \text{Take}(r), t = \alpha_0 \xrightarrow{c_1}_k \dots \xrightarrow{c_i}_k \\ \alpha_i \xrightarrow{c_{i+1}}_k \dots \exists j, j > i, c_j = \text{Free}(r)$$

- ▶ Assurer cette condition en imposant qu' un service devra relaché une ressource avant un temps t_{max}
- ▶ Idée de l'instrumentation
 - ▶ gérer un compteur
 - ▶ RAZ, inc et test

Libération des ressources - Exemple

- ▶ Imposer la libération de r_1 et r_2 avant un maximum 0.5 secondes
- ▶ Différentes possibilités d'implémentation du timer
 - ▶ utilisation du matériel : contraintes relâchées (scheduler)
 - ▶ par appel d'une fonction qui retourne le temps

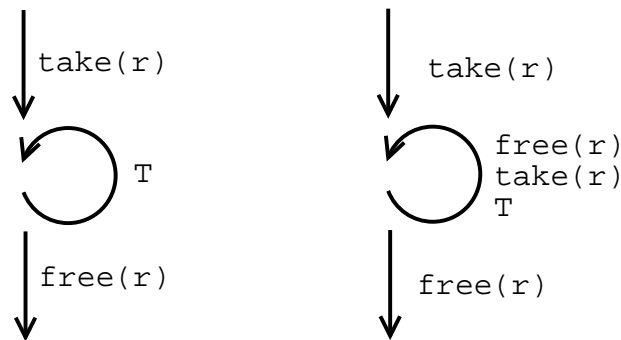
Terminaison des requêtes

Assurer la terminaison du traitement d'une requête avant un temps t_{max}

- ▶ comme pour imposer la libération d'une ressource avant un temps t_{max}
- ▶ gestion avec un compteur

Absence de deadlocks

- ▶ Pas de boucles dans les demandes de ressource
 $\nexists (\pi, \beta), \beta(r_1) = p_1, \pi(p_1) = Take(r_2), \dots, \beta(r_n) = p_n, \pi(r_n) = Take(r_1)$
- ▶ Imposer l'absence de boucles dans les demandes de ressource
 - ▶ Aspect inter-services
 - ▶ Interdire certaines actions dans certaines configurations
 - ▶ sur exemple : interdiction pour le service web de prendre r_1 si service surveillance possède r_2
 - ▶ Casser les boucles



Ce qu'il reste à faire

Les notions à ajouter/non présentes dans le modèle.

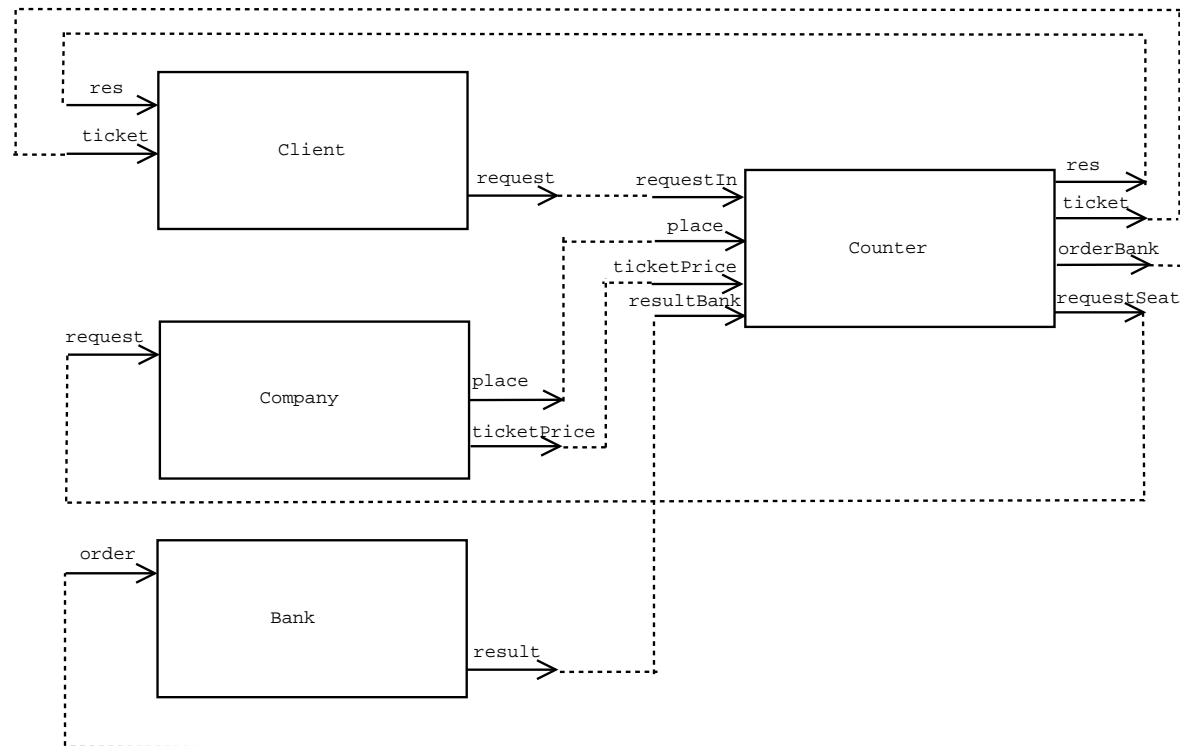
- ▶ Langage de propriété de disponibilité
 - ▶ temps maximum d'utilisation d'une ressource
 - ▶ temps maximum d'une requête
- ▶ Modélisation de la communication du service avec les utilisateurs - Comment le service dialogue avec son utilisateur ?
- ▶ Des propriétés et des preuves ... pour arriver à la preuve globale de la protection du système contre les denis de service

Extensions

- ▶ Gérer d'autres types de ressources
 - ▶ des lecteurs/rédacteurs
 - ▶ des ressources autorisant l'entrelacement des exécutions (comme CPU)
- ▶ Déclarer les services comme des KPNs
- ▶ Avoir un modèle plus proche de la réalité
 - ▶ création dynamique d'un service pour chaque utilisateur

Cas d'étude

- ▶ Modélisation de l'exemple de la billetterie



- ▶ Question : où est la disponibilité ?
- ▶ Question : exemple bien adapté à la disponibilité ?