

ACI Dispo - Aspects de disponibilité

Stéphane HONG TUAN HA, Pascal FRADET

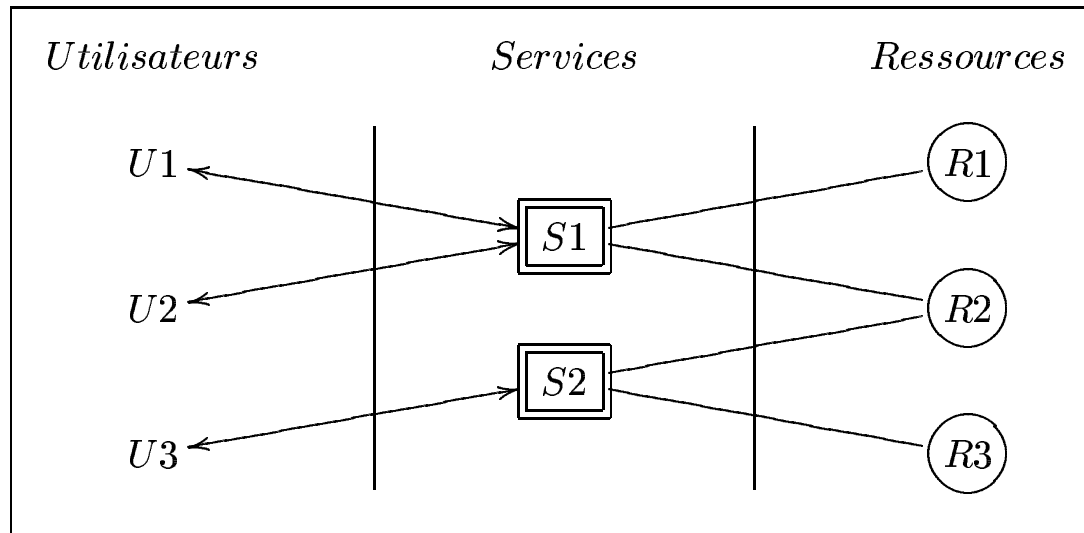
IRISA/INRIA Rennes & INRIA Rhône-Alpes

Contexte et Motivation

- ▶ Problématique de la disponibilité
 - ▶ "assurer l'accès aux informations"
 - ▶ dénis de services logiciels ou matériels
 - ▶ politiques et propriétés de disponibilité
- ▶ 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"
 - ▶ utilisation des automates temporisés comme formalisme sous-jacent

Modèle pour la disponibilité

- ▶ Modèle structuré en 3 couches

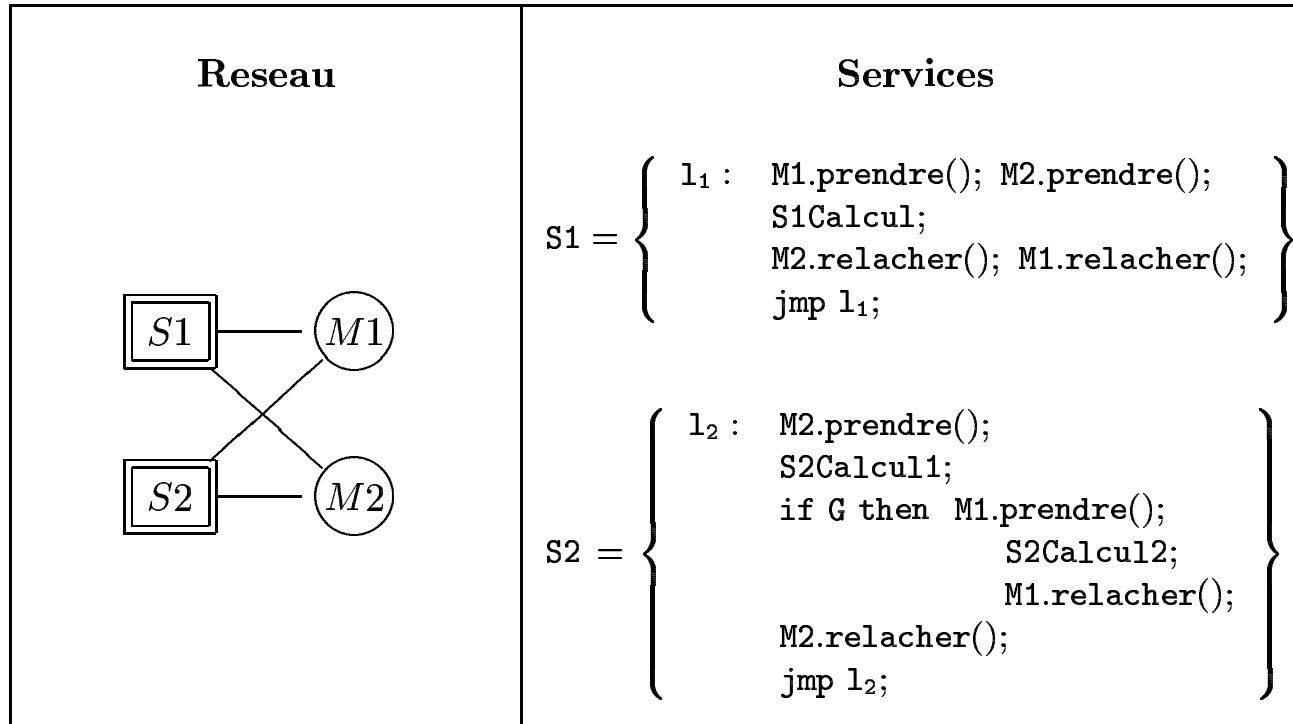


- ▶ correspond à un modèle client-serveur
- ▶ services traitent les requêtes des utilisateurs
- ▶ services en concurrence pour l'accès aux ressources
- ▶ possibilité de tisser les services

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 prouver que le système est disponible
- ▶ 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

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
- ▶ Ajout de règles dédiées à la disponibilité
 - ▶ $(C \text{ before } x \text{ else } I)$: attend un événement C avant x secondes ou sinon exécute I au bout de x secondes
 - ▶ $(C \text{ after } x)$: attend un événement qui correspond à C et si celui-ci est reconnu avant x secondes endort le service le complément de temps

Le langage d'aspect (2)

► Grammaire :

$A ::= \mu a.A$	<i>; definition recursive</i>
$(C \triangleright I) ; A$	<i>; pattern sur C</i>
$(C \text{ before } x \text{ else } I) ; A$	<i>; pattern sur C</i>
$(C \text{ after } x) ; A$	<i>; + condition temporelle</i>
$A_1 \square A_2$	<i>; choix</i>
a	<i>; fin de sequence</i>

- Restriction à une seule action (en plus du *skip*)
 - 'Reset' pour libérer les ressources et terminer le traitement de l'utilisateur courant du service

Le langage d'aspect (3)

- ▶ Exemples de politiques de disponibilité :
 - ▶ relacher ressource avant 25 secondes sinon Reset

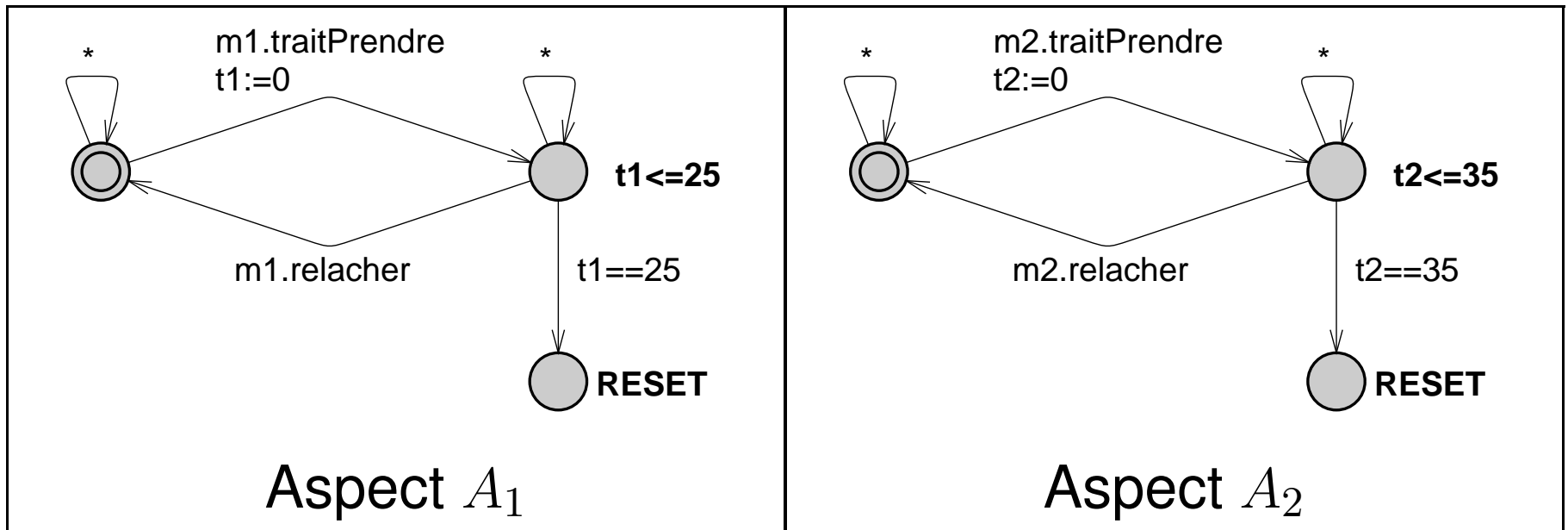
$$A_1 = \mu a1.\text{prendre} \triangleright \text{skip};$$
$$(\text{relacher before 25 else Reset}); a1$$

- ▶ attendre 20 secondes entre chaque prise de la ressource

$$A_2 = \text{prendre} \triangleright \text{skip};$$
$$\mu a1.\text{prendre after 20}; a1$$

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



Abstraction des services

- ▶ Abstraction du service en un automate classique
- ▶ Ajout d'annotations temporelles pour obtenir un automate temporisé
 - ▶ hypothèse d'une fonction de coût qui donne le BCET et le WCET

$$f_{cout}(S2calcul1) = [1, +\infty]$$

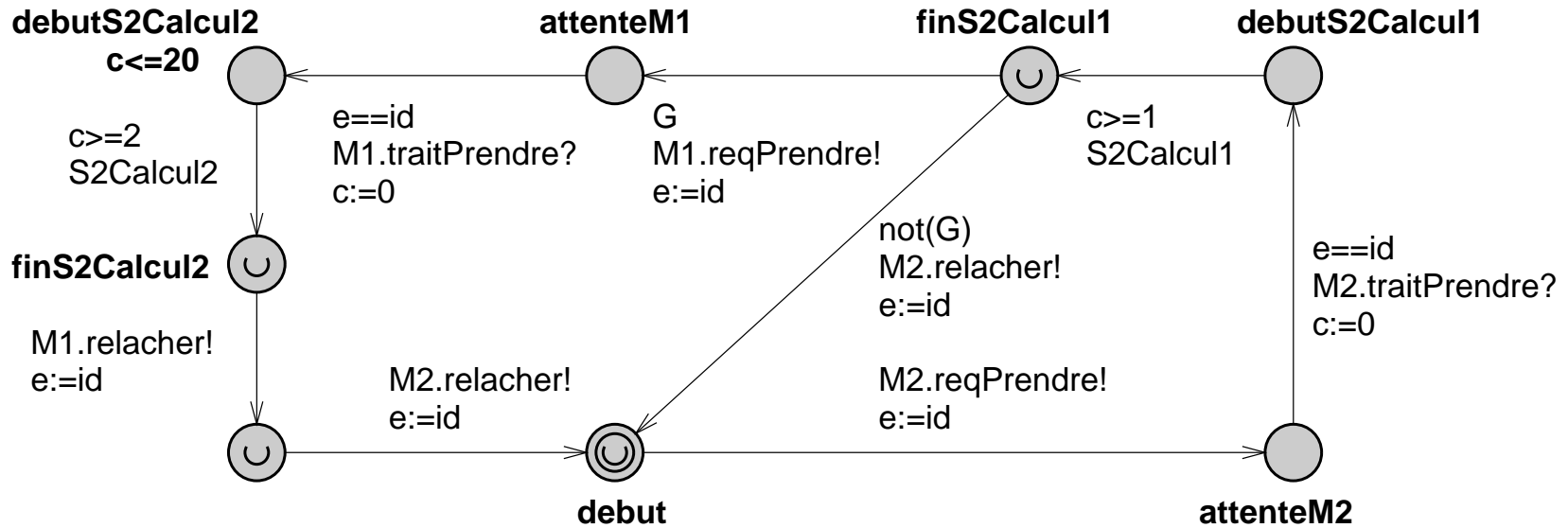
$$f_{cout}(S2calcul2) = [2, 20]$$

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

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

...

Abstraction du service S2



Tissage

1. Produit d'automates temporisés
 - ▶ (*service* × *aspect*)
2. Transformation adhoc pour gérer l'état puits *RESET*
3. Optimisation de l'automate temporisé produit
 - ▶ enlever les états non atteignables, les horloges et gardes inutiles, etc.
4. Génération de code à partir de l'automate temporisé
 - ▶ gestion des traits temporels en utilisant des timers, 3 instructions : *new timer()*, *schedule(r, d)* et *cancel()*

Tissage du service S2

$$S2 = \left\{ \begin{array}{l} l_2 : M2.prendre(); \\ \quad tim = new Timer(); tim.schedule(routineReset, 35); \\ \quad S2Calcul1; \\ \quad if G then M1.prendre(); \\ \quad \quad \quad S2Calcul2; \\ \quad \quad \quad M1.relacher(); \\ \quad M2.relacher(); tim.cancel(); \\ \quad jmp l_2; \end{array} \right\}$$
$$routineReset = \left\{ \begin{array}{l} relacherRessources(); \\ jmp l_2; \end{array} \right\}$$

- ▶ Ajout d'un timer pour l'aspect A_2
- ▶ Pas de nécessité d'un timer pour l'aspect A_1

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)
 - ▶ vérifier des propriétés de disponibilité
- ▶ Propriétés de disponibilité sur l'exemple après tissage :
 - ▶ pas d'interblocages (grâce aux deadlines sur la libération des ressources)
 - ▶ attente bornée pour prendre les ressources $M1$ et $M2$
 - ▶ pas de denis de service pour le service $S1$

Conclusion

- ▶ Un modèle pour la disponibilité en 3 couches (utilisateurs, services, et ressources)
- ▶ Un cadre
 - ▶ pour imposer des politiques de disponibilité exprimées comme des aspects
 - ▶ pour vérifier des propriétés de disponibilité
- ▶ Travaux futurs :
 - ▶ finaliser le véritable langage d'aspect
 - ▶ terminer la formalisation des différentes étapes (en particulier, le tissage et la génération de code)
 - ▶ tester l'approche sur un véritable cas d'étude

Spécifications des ressources

- ▶ Nécessaire pour déduire des propriétés de disponibilité
- ▶ Définition de l'interface et du comportement de la ressource
 - ▶ modélisation par des automates temporisés
 - ▶ pour générer la ressource à partir de la spécification
- ▶ Expérimentation avec des ressources 'mutex' et 'partageable en k'
- ▶ Possibilité d'exprimer des ressources plus élaborées comme des ressources avec priorités