Dispo

Thomas Jensen
IRISA

22 novembre 2005

Plan

Plan

DISPO

DISPO : Disponibilité de services dans des composants logiciels

Partenaires :

- École des Mines de Nantes, équipe Obasco
- ENSTB dépt. RSM
- Irit/SVF
- Irisa, projet Lande et Inria Rhône-Alpes, projet Pop-Art.

Objectifs :

Intégrer la sécurité dans la conception de logiciels.

Sécurité de systèmes informatiques

- Confidentialité
- Intégrité
- **Disponibilité**

Disponibilité :

Aptitude à répondre à une demande d'un service, d'une ressource en garantissant des contraintes de performances

Disponibilité

Modèles de disponibilité :

- Yu et Gligor
  - Fondé sur la logique temporelle (*finite waiting time*)
  - *User requirements* : des contraintes sur le comportement des utilisateurs.
- Millen
  - moniteurs d'allocations de ressources
  - modèle basé sur des systèmes de transitions
- Cuppens et Saurel
  - logiques temporelles et déontiques
  - cohérence d'une politique de disponibilité
- Le modèle Nomad

Disponibilité et génie logiciel

La disponibilité a plusieurs aspects :

- matériel : défaillances et attaques physiques ;
- logique : erreurs de génie logiciel, logiciel malveillant.

l'action DISPO se focalise sur le deuxième point

Intégrer la disponibilité dans la conception modulaire de logiciels :

- composants et interfaces,
- conception et programmation par aspects

Développer des techniques de validation de l'usage de ressources :

- analyse statique,
- vérification de modèles,
- tissage de moniteurs.

Plan

Nomad

Formalisme pour exprimer des politiques de sécurité.

- Contrôle d'accès aux services (Permission, Interdiction)
- Contrôle d'usage des services (Obligation)
- Prise en compte du facteur temps
  - remplir les obligations en respectant les délais
  - sinon violation
- Prise en compte du contexte

Sémantique des mondes possibles et définition d'une axiomatique complète

Une logique temporelle . . .

La logique Nomad comprend

- un ensemble d'actions $a \in A$, fermé sous composition séquentielle (;) et parallèle (|),
- des formules ϕ comme *req a*, *start a*, *doing a*, *done a*,. . .
- des modalités temporelles $\oplus\phi$ (*next*) et $\Box\phi$ (*always*)
- des modalités dérivées de délais, comme $\bigcirc^d\phi$, $\bigcirc^{\leq d}\phi$.

Axiomes :

$$start\ a \leftrightarrow \bigcirc^{duree(a)} done\ a$$
$$start\ (a \mid b) \leftrightarrow start\ a \wedge start\ b$$

. . . avec des opérateurs déontiques

Les politiques de disponibilité expriment

- des obligations (de répondre à une requête)
- l'interdiction (de faire une action)
- la permission

L'obligation est exprimée avec la modalité $\mathcal{O}\phi$
Obligation de remplir une tâche dans un délai d : $\mathcal{O}^{\leq d}\phi$
Obligation de faire A si condition C : $\mathcal{O}(A \mid C)$

Modalités dérivées :

$$\begin{array}{lrcl} \text{Interdiction} & \mathcal{I}\phi & \equiv & \mathcal{O}\neg\phi \\ \text{Permission} & \mathcal{P}\phi & \equiv & \neg\mathcal{O}\neg\phi \end{array}$$

Axiomatisation

L'axiomatisation est étendue aux modalités déontiques :
Par exemple :

$$\begin{array}{lcl} \mathcal{O}A & \rightarrow & \mathcal{P}A \\ \mathcal{O}(A \mid C) & \rightarrow & \Box(C \rightarrow \mathcal{O}(A)) \\ \ldots & & \end{array}$$

Interprétation de la logique dans des structures de Kripke avec deux relations de transition (temporelle et déontique).

Nomad pour la disponibilité

Expression d'une politique de disponibilité en Nomad :

Empêcher les accès et usages abusifs de services

- Interdiction des demandes impossibles à satisfaire
- Interdiction des demandes illégitimes

Garantir les accès et usages légitimes

- Obligations de fournir le service en respectant les délais
- Obligations de libérer le service au bout d'un certain délai

Exemples

Une demande de connection quand une place est disponible est acceptée :

$$Syn_i \wedge Free_{=k} \rightarrow \oplus(Syn\_Ack_i \wedge Busy_{k,i})$$

Une ressource doit être libérée au plus 10 secondes après être utilisée :

$$\Box(utilise \rightarrow \mathcal{O}^{\leq 10} libere)$$

Obligation de supprimer des demandes non-acquittées après un certain temps :

$$\mathcal{O}(\bigcirc^d RESET_i \mid Syn\_Ack_i \wedge \bigcirc^{\leq d} \neg Ack_i)$$

Cohérence de politiques Nomad

Développement en cours d'une procédure de décision pour une logique proche de NOMAD qui permettra

- de vérifier la **cohérence** d'une politique de disponibilité
  $\Rightarrow$ rien n'est à la fois obligatoire et interdit, ni permis et interdit
- de vérifier le **respect** (ou non violation) de la politique par une modélisation d'un système
  $\Rightarrow$ ce qui est obligatoire arrive effectivement
  $\Rightarrow$ ce qui est interdit n'arrive pas
  $\Rightarrow$ ce qui arrive est permis

Possibilité d'obtenir une procédure semblable pour *NOMAD*

Plan

Analyse de ressources

Analyser l'utilisation des ressources dans une architecture à composants:
Deux analyses ont été étudiées :

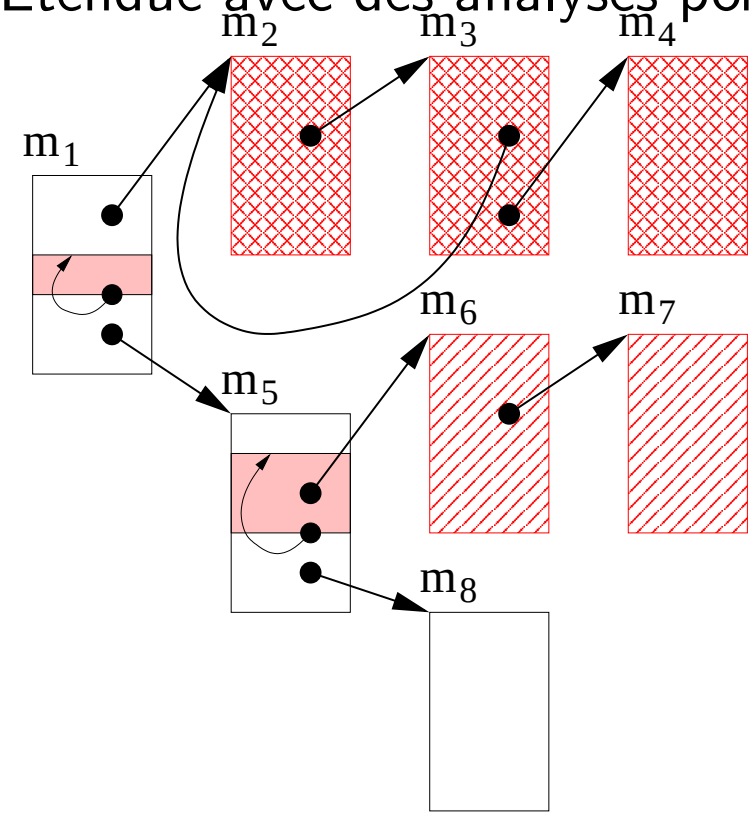
- borner les boîtes à lettre pour les communications asynchrones entre composants,
- détecter la consommation non-bornée de cellules mémoire dans un composant,

Les deux analyses sont basées sur la détection de cycles.
Bornes "existentielles" $\Rightarrow$ dépasse les propriétés de sûreté classiques.

Analyse de ressources "intra-composants"

Détection de boucles intra- et interprocédurales :

- Formalisé comme une analyse statique
- Étendue avec des analyses portant sur les valeurs numériques



Compter l'allocation des ressources ou des services

- Utilisation de machines à états et compteurs ($c_{1 \leq i \leq n}$)
- Systèmes potentiellement infinis, la "boundedness" permet de vérifier si on reste dans un cadre fini
  - dimensionnement des ressources
  - model-checking classique
- Principe de recherche d'un cycle d'accumulation dans le graphe des configurations
  - Condition décidable suffisante pour la "bornitude" des FIFO dans des composants communicants de façon asynchrone
  - Procédure de décision de la boundedness avec garde $c_i \geq k_i$ et action $c_i = \sum_1^n a_j * c_j \pm b_i$ sur les compteurs
- Résultat à rapprocher de ceux sur les double net et transfer net

Aspects de disponibilité

Une politique Nomad fournit des interdictions et des obligations.

Réalisation :

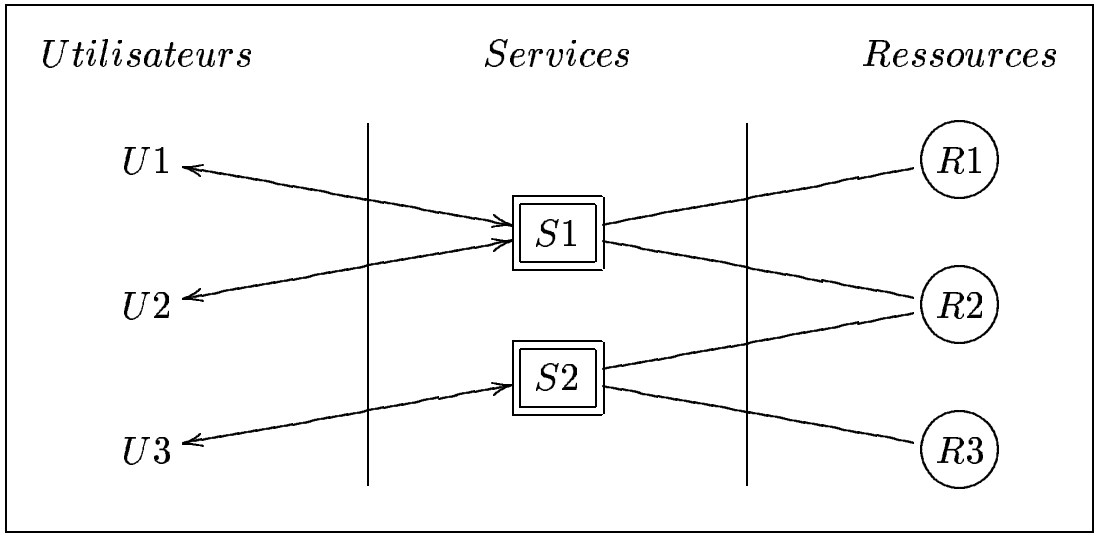
- par vérification statique (avant exécution)
- par contrôle dynamique

Insertion d'un contrôleur dans un code

- par codage manuel
- par tissage d'aspects

Modèle orienté services

- Modèle structuré en 3 couches



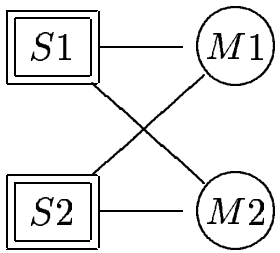
  - correspond à un modèle client-serveur
  - services traitent les requêtes des utilisateurs
  - services en concurrence pour l'accès aux ressources
  - possibilité de tisser les services

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

Reseau



Services

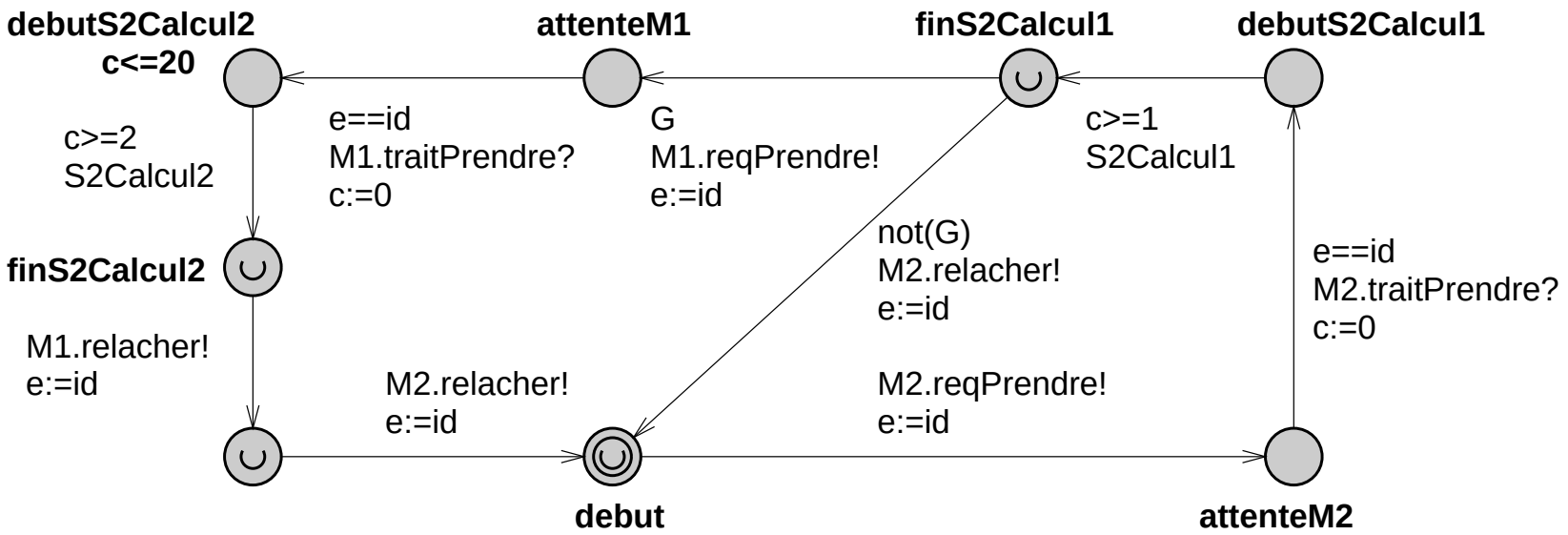
$$S1 = \left\{ \begin{array}{ll} l_1 : & \texttt{M1.prendre(); M2.prendre();} \\ & \texttt{S1Calcul;} \\ & \texttt{M2.relacher(); M1.relacher();} \\ & \texttt{jmp } l_1\texttt{;} \end{array} \right\}$$

$$S2 = \left\{ \begin{array}{ll} l_2 : & \texttt{M2.prendre();} \\ & \texttt{S2Calcul1;} \\ & \texttt{if G then M1.prendre();} \\ & \qquad\qquad\quad \texttt{S2Calcul2;} \\ & \qquad\qquad\quad \texttt{M1.relacher();} \\ & \texttt{M2.relacher();} \\ & \texttt{jmp } l_2\texttt{;} \end{array} \right\}$$

- 2 possibilités de dénis de service :
  - dénis de service de type famine si *S2Calcul*1 ne termine pas
  - inter-blocage pour la prise des 2 ressources

Abstraction des services

Les services sont abstraits par des automates temporisés :
Exemple : abstraction du service S2

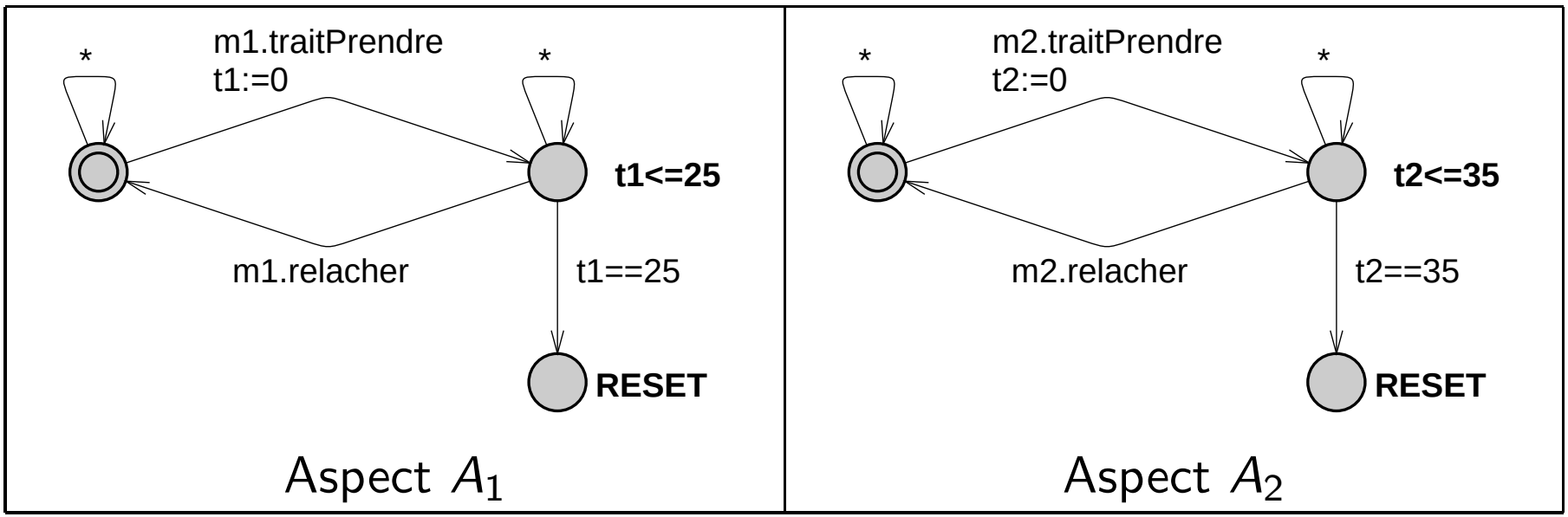


Aspects temporels (1)

- Description des politiques que les services doivent respecter
  - politiques de type "temps borné"
  - politiques locales qui seront tissée sur le service
- Contraintes temporelles sur les opérations utilisant les ressources
  - décrit par un automate temporisé
- Une seule action
  - 'Reset' pour libérer les ressources et terminer le traitement de l'utilisateur courant du service

Aspects temporels (2)

- Exemple d'aspects pour le service S2 :
  - A_1 : relâcher M1 avant 25 secondes sinon Reset
  - A_2 : relâcher M2 avant 35 secondes sinon Reset



Tissage

1. Produit entre l'automate temporisé du service et celui de l'aspect
2. Transformation ad-hoc 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

$$
\texttt{S2} = \left\{ \begin{array}{ll} \texttt{l}_2 : & \texttt{M2.prendre}(); \\ & \texttt{tim} = \texttt{new Timer}();\ \texttt{tim.schedule}(\texttt{routineReset}, 35); \\ & \texttt{S2Calcul1}; \\ & \texttt{if G then}\ \ \texttt{M1.prendre}(); \\ & \qquad\qquad\qquad \texttt{S2Calcul2}; \\ & \qquad\qquad\qquad \texttt{M1.relacher}(); \\ & \texttt{M2.relacher}();\ \texttt{tim.cancel}(); \\ & \texttt{jmp l}_2; \end{array} \right\}
$$

$$
\texttt{routineReset} = \left\{ \begin{array}{l} \texttt{relacherRessources}(); \\ \texttt{jmp l}_2; \end{array} \right\}
$$

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

Vérification

- Utilisation du model-checker "UPPAAL" pour analyser le système (ressources + services tissés)
  - vérifier des propriétés de disponibilité
  - vérifier la cohérence entre les aspects
- Propriétés de disponibilité sur l'exemple de système après tissage :
  - pas d'inter-blocages (grâce aux deadlines sur la libération des ressources)
  - attente bornée pour prendre les ressources $M1$ et $M2$
  - pas de dénis de service pour le service $S1$

Un véritable langage d'aspect (1)

- Langage de haut niveau traduisible en automates temporisés
  - $(C \triangleright I)$: filtre la trace d'exécution ; 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
  - $(C\ \mathit{before}\ x\ \mathit{then}\ I_1\ \mathit{else}\ I_2)$: exécute I_1 si un événement correspond à C avant x secondes ou sinon exécute I_2 au bout de x secondes
  - $(C\ \mathit{after}\ x)$: attend un événement qui correspond à C et si celui-ci est reconnu avant x secondes endort le service le complément de temps

Un véritable langage d'aspect (2)

- Grammaire :

$$
\begin{array}{lcll}
A & ::= & \mu a.A & ; \textit{ définition récursive} \\
 & | & (C \triangleright I)\ ;\ A & ; \textit{ pattern sur } C \\
 & | & (C \textit{ before } x \textit{ then } I_1 \textit{ else } I_2)\ ;\ A & ; \textit{ pattern sur } C \\
 & | & (C \textit{ after } x)\ ;\ A & ; \ + \textit{ condition temporelle} \\
 & | & A_1 \Box A_2 & ; \textit{ choix} \\
 & | & a & ; \textit{ fin de séquence}
\end{array}
$$

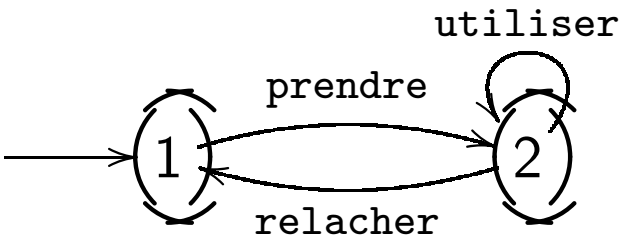
Un véritable langage d'aspect (3)

- Exemples de politiques de disponibilité :
  - relâcher ressource avant 25 secondes sinon Reset
    $A_1 = \quad \mu a1.\texttt{prendre} \triangleright \mathit{skip};$
    $(\texttt{relacher}\ \mathit{before}\ 25\ \mathit{then\ skip\ else\ Reset}); a1$
  - attendre 20 secondes entre chaque prise de la ressource
    $A_2 = \quad \texttt{prendre} \triangleright \mathit{skip}\ ;$
    $\mu a1.\texttt{prendre}\ \mathit{after}\ 20\ ;\ a1$

Au delà du "temps borné"

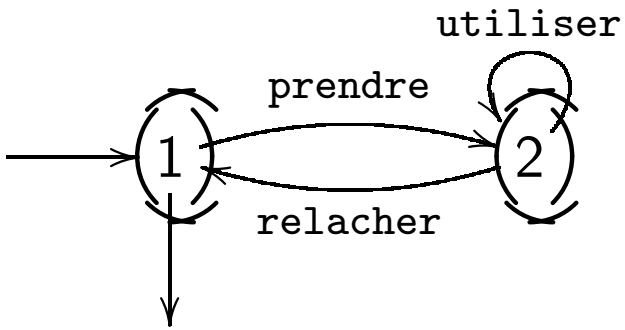
- Les propriétés de disponibilité de type "temps borné" sont des propriétés de sûreté.
  Exemple : relâcher une ressource avant x secondes, sinon RESET
- C'est inadapté lorsqu'on ne connaît pas le temps d'exécution (lorsqu'on modélise le système à un niveau abstrait, lorsqu'on utilise un composant en boîte noire, etc.).
- Si le temps n'est plus borné, la disponibilité s'exprime alors par des propriétés ayant une composante de vivacité.
  Exemple : toute ressource utilisée doit être relâchée.

Exemple : un protocole simple



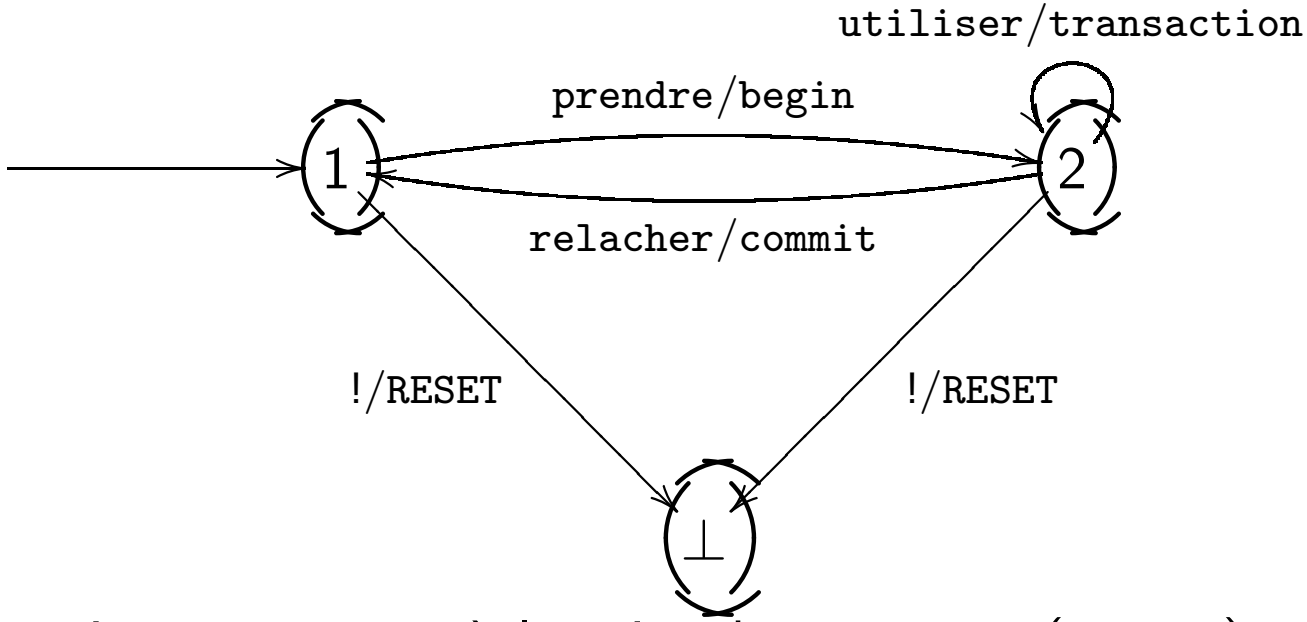
Prendre, utiliser puis relâcher

Une propriété de disponibilité



La condition d'arrêt en l'état 1 exprime la disponibilité : la ressource doit être régulièrement relâchée.

Une solution avec des transactions



La transaction commence à la prise de ressource (`begin`), se poursuit lors de l'utilisation (`transaction`), et est exécutée lorsque la ressource est relâchée (`commit`).
Généralisation des travaux de Bauer, Ligatti et Walker sur les "edit automata".

Plan

Résultats

- Nomad : un formalisme pour décrire des politiques de disponibilité
  - logique temporelle
  - logique déontique
- vérification de propriétés
  - de cohérence
  - de finitude (intra- et inter-composants)
- contrôleurs de ressources
  - aspects de disponibilité
  - tissage avec protocoles entre composants

En cours

- développement d'un exemple "syn flooding" en Nomad
- cohérence des politiques de Nomad avec l'outil Uppaal
- langage d'aspects enrichi
- extension du modèle des moniteurs pour prendre en compte les ressources (compteurs et transactions)
- synthèse de ces modèles dans le cadre d'une infrastructure prototype de composants Java actifs communiquant par envoi de messages.
- implémentation: code Java standard (composants primitifs) ou composition de composants (composants structurés)