Moniteur d'exécution formel et Tolérance aux Fautes Pratique

Thomas Robert, LTCI, Télécom ParisTech
Thomas.Robert@telecom-paristech.fr

Séminaire Mefosyloma

Motivations et contexte

Motivations

- **Contexte:**

  Conception et intégration de la Tolérance aux fautes (TaF) logicielle pour des systèmes temps réels

- **Les moyens :**
  - Des motifs architecturaux *(redondance active/passive …)*
  - Des protocoles de communication & synchronisation fiables *(diffusion, checkpointing)*
  - Des disciplines de programmation et conception *(exceptions, modes et protocoles de changement de modes)*
- **Motivation :**
  - Comment améliorer les démarches MDE pour la conception de mécanismes de Tolérance aux Fautes

Démarche

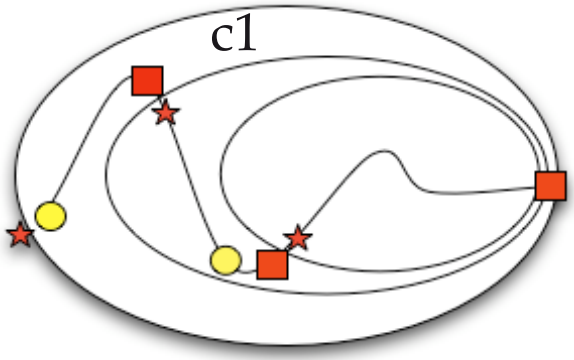
- **Modèles disponibles**
  - Comportements Valides
  - Modèles de fautes (occurrences, manifestation …)
  - Modèles architecturaux de l'implémentation
  - Exigences sur des Métriques (~QoS)

- **Utiliser ces modèles pour générer le code de Détection en fonction du mécanisme de recouvrement**
  - Pb : Peu ou pas de mécanismes de recouvrement indépendants de la détection
  - Pb : Peu ou pas de modèles du processus complet détection/ recouvrement

Mise en place de la Tolérance aux Fautes

- **But : maitriser les conséquences des états erronés**

C



Identifier les erreurs pour les confiner

- **Erreurs** : défaillance d'une fraction du Système
  - Domaine : valeurs, comportement
  - Observabilité : Qui, Comment, Sous quelles hypothèses
- **Mise en œuvre :**
  - Réactive: détection => recouvrement <=
  - Solution à base de Redondance (exécution concurrente)

Spécifier les signatures d'erreurs

- La signature d'une erreur == séquence d'observations menant à l'identification d'une erreur

Modèle explicite

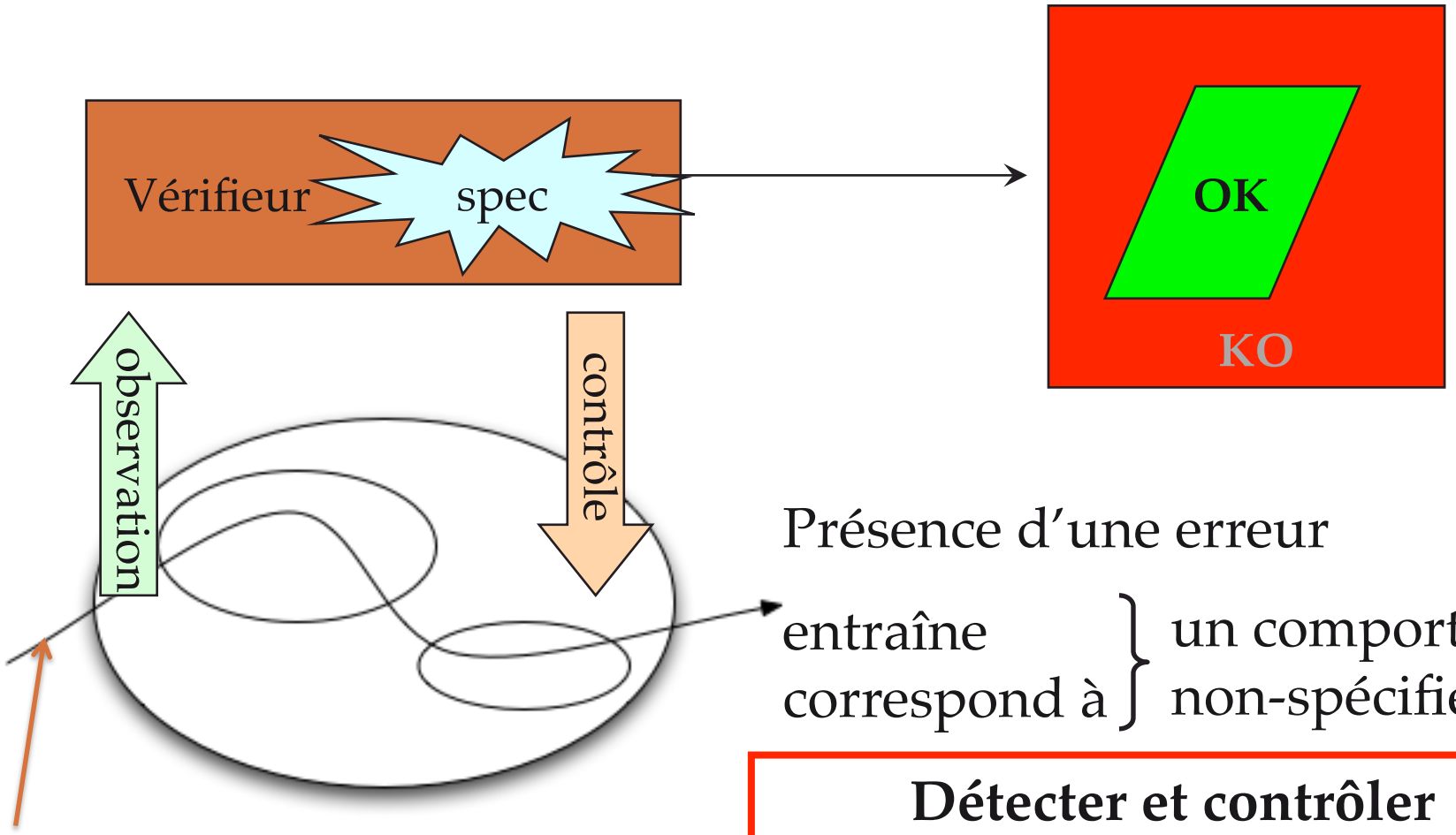
- Définition de signatures de détection
- Définition du domaine affecté
- Caractérisation du confinement

Modèle implicite

- Déduction de signatures de défaillance par défaut
- Sélection du domaine à observer
- Aucune assurance du succès du recouvrement

- 1 modèle implicite == 1 spécification comportementale de ma cible

Principe de l'observateur formel



Présence d'une erreur

entraîne } un comportement
correspond à } non-spécifié

Détecter et contrôler ces comportements non-prévus

Trajectoire d'exécution

Génération des fonctions de détections

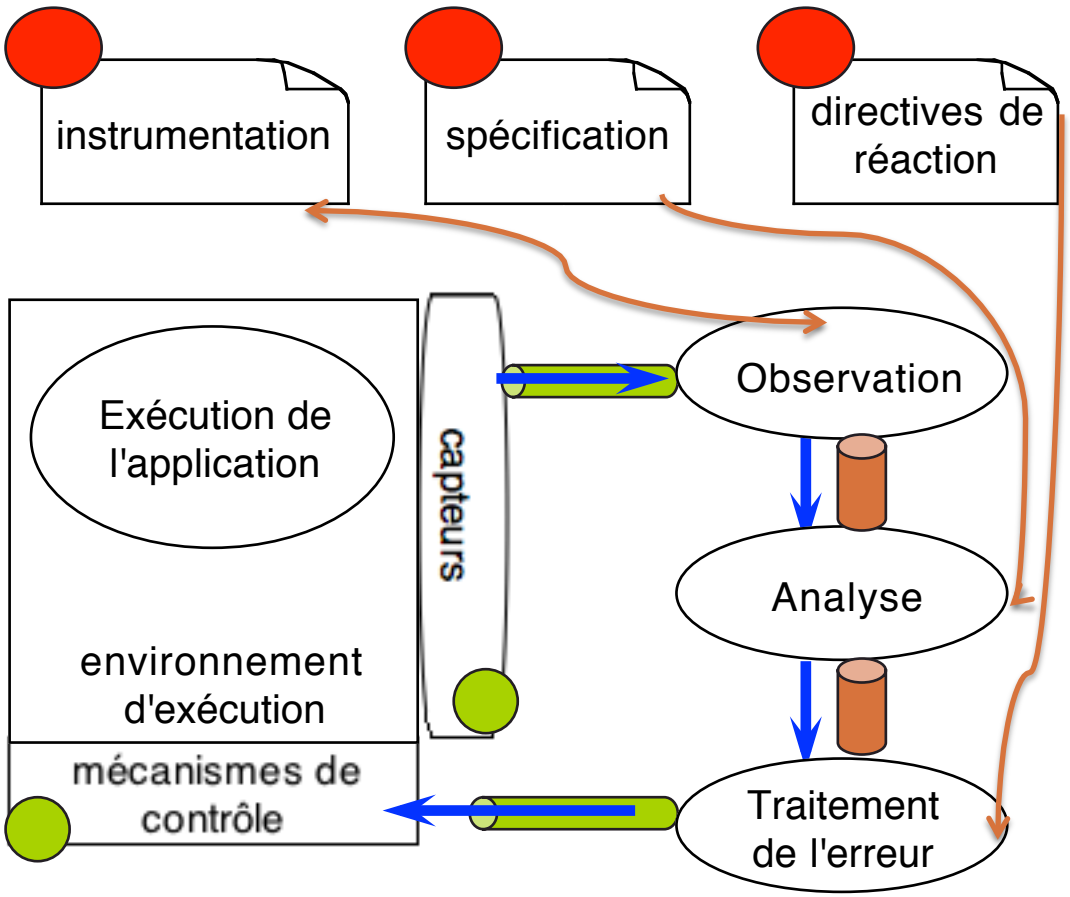
- **Objectif : Synthétiser les fonctions de détection d'erreurs**
- **Hypothèses :**
  - Spécification => signatures d'erreurs valides (Spec. sans défaut)
- **Contraintes :**
  - Maitriser le coût à l'exécution
  - Borner la latence de détection
  - Maîtriser le confinement d'erreur
- **Cible : lot de tâches**
  - Interface zone de confinement == API système
  - Domaines d'observation: types et enchaînement des appels

Bilan sur l'existant
Les tâches et leurs caractéristiques

- **Moniteur => 3 fonctions : Observer, Interpréter, Réagir**
- **Observation => instrumentation**
  - Mise en œuvre logicielle ou matérielle
  - Insertion manuelle, semi-automatique, automatique
  - Propagation synchrone / asynchrone de l'information

  ⇒Affecte : la latence, la fiabilité des observations
- **Détection => Analyse du « flux » d'observations**
  - Représentation de l'algorithme d'analyse (in-lining de systèmes à transition, algorithmes généraux)
  - Type de rapports d'erreur (différenciés, non-typés …)

Bilan sur l'existant
Architecture [Delgado *et al.* 2004]



- Description de la tâche de vérification en ligne
- Synthèse des composants
- L'intégration
- La dynamique

Vers des mécanismes de TaF prédicitbles

- **Impact de la latence :**
  - Disponibilité
  - Fiabilité des composants utilisant le composant défaillant
- **Assemblage du mécanisme de tolérance au fautes**
  - Compromis coût en régime nominal / capacité de confinement
  - 2 paramètres clés :
    la maitrise de la latence et la précision de la détection
- **En pratique :**
  - Déterminer le fonctionnement de chaque bloc
  - Fixer la méthode de synchronisation
  - Produire un modèle comportemental analysable de l'assemblage

Les solutions de la communauté « Sécurité »

- **Proposition : utiliser une sorte de transducteur pour modéliser l'application d'une politiques de sécurité**
- **Adaptation : une politique de sécurité ~ la spécification des comportements corrects du système**
- **Pb :**
  - Explosion combinatoire de l'espace d'état lors de l'analyse formelle
  - Modèle sans temps quantitatif
- **Réduire le pouvoir d'expression à une famille très réduite d'altérations de la trace originale**

Collecte des observations

Approches disponibles

- **Hyperviseur & Machines virutelles : Extraction**
  - 1 service de collecte de trace fourni par le support
  - Evénements observé == événements liés à l'utilisation des services du support d'exécution par l'application
  - Avantage : possibilité de faire « croire au système » qu'il n'est pas observé [Rogriguez2002]
- **Sondes logicielles :**
  - 1 stratégie de placement (aspects, annotations simples …)
  - 1 chaîne de génération du code d'instrumentation
- **Approches mixtes :**
  - Placement des instructions d'observation par la VM …

Isolation du couple détecteur cible

- **Objectif :**
  - Le mécanisme de détection ne doit pas être affecté par les erreurs qu'il doit détecter
- **Conséquences :**
  - L'approche par extraction reste populaire malgré son coût
  - Analyser les risques de propagation d'erreur dans le détecteur

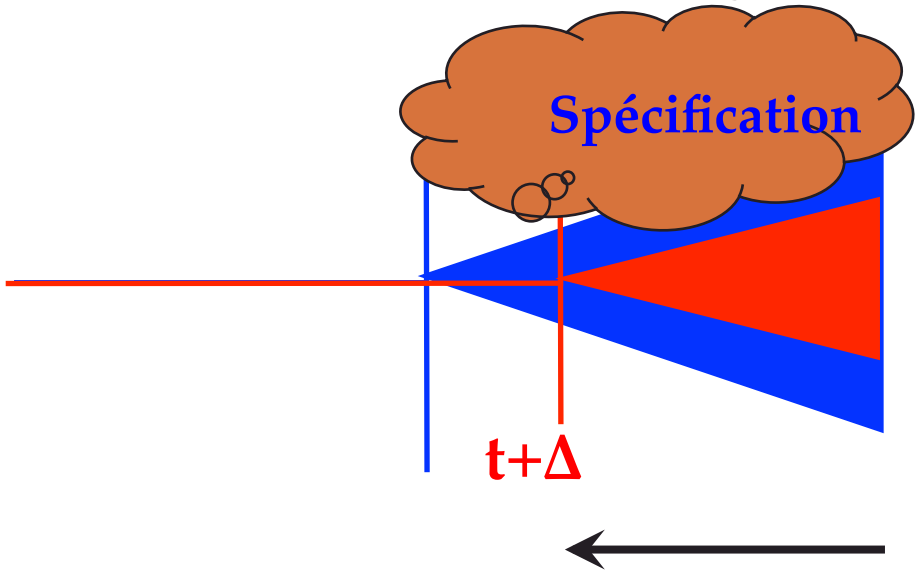
Synthèse de la fonction d'analyse

Grandes lignes

- Principe de déduction des signatures d'erreur
- Typage des erreurs
- Obtention du modèle comportemental du détecteur

Déduction des signatures d'erreur

- **Si** *Spec == comportement valide => ? Comportement invalide ?*
  - Trace tronquée == quantité d'information perçue
  - Préfixe d'une trace == passé
  - Plus petit == remonter le temps jusqu'à …



… trouver la frontière
correct/incorrect

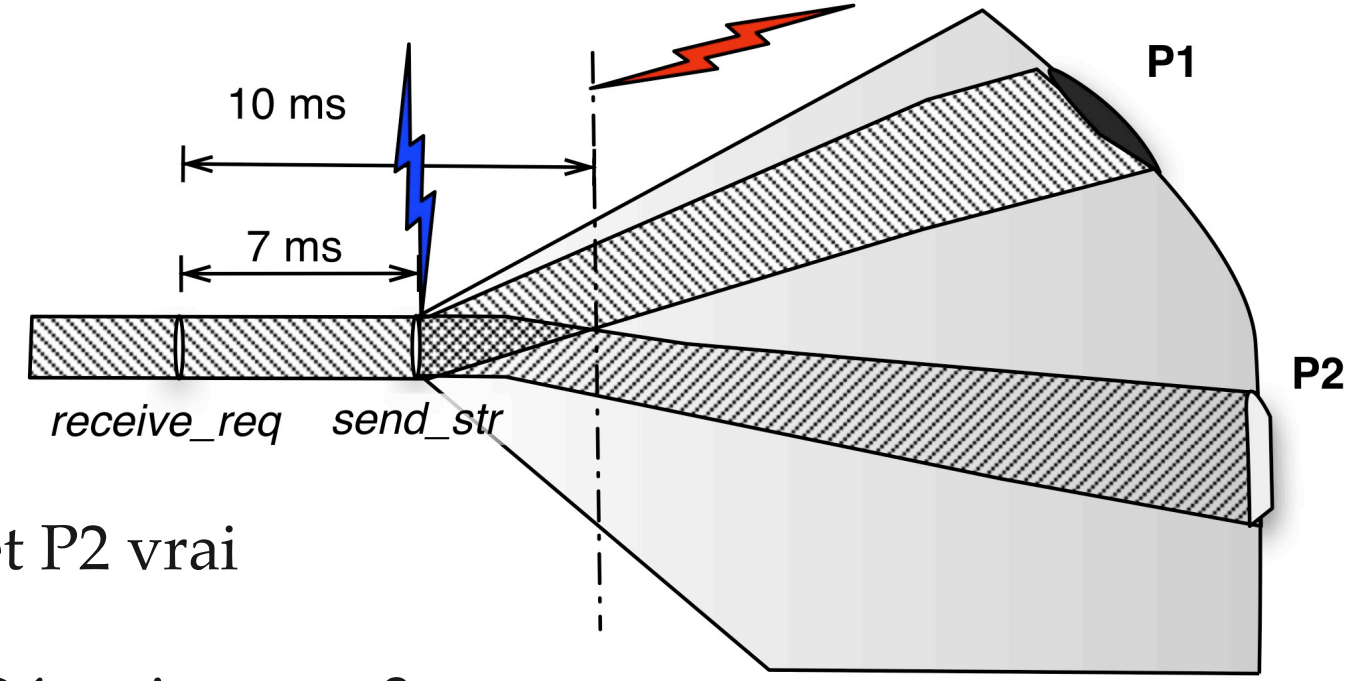
Propriétés des signatures pour la TaF

- Correction:
  - S'assurer que le signalement d'une erreur est monotone
- Latence de réaction bornée :
  - Décider avec un nombre fini d'observation
  - Consommer l'information au plus tôt
- Invariance % classe d'équivalence :
  - Si Spec 1 ~ Spec 2 alors même comportement de l'observateur
- **Identification de {signatures} par un lecteur de traces**

=> l'analyseur devient le lecteur de traces (séquences d'observations)

Interprétation syntaxique vs plus petit préfixes

- Spécification $P1 \wedge P2 \quad \sim \neg(P1 \vee P2)$
- P1- si *receive_req* alors *send_os* avant 10 ms
- P2- si *send_str* alors aucun événement pendant 4 ms



« Pire cas » :
P1 sera fausse et P2 vrai

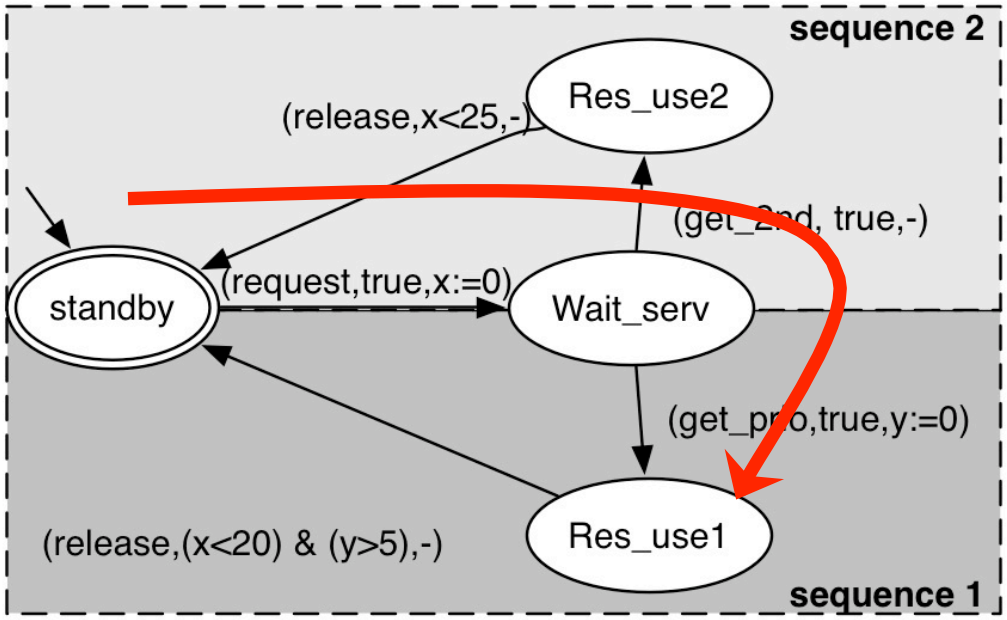
$\text{Détection}_{\text{Sem}} - \text{Détection}_{\text{Syn}} = 3$ ms

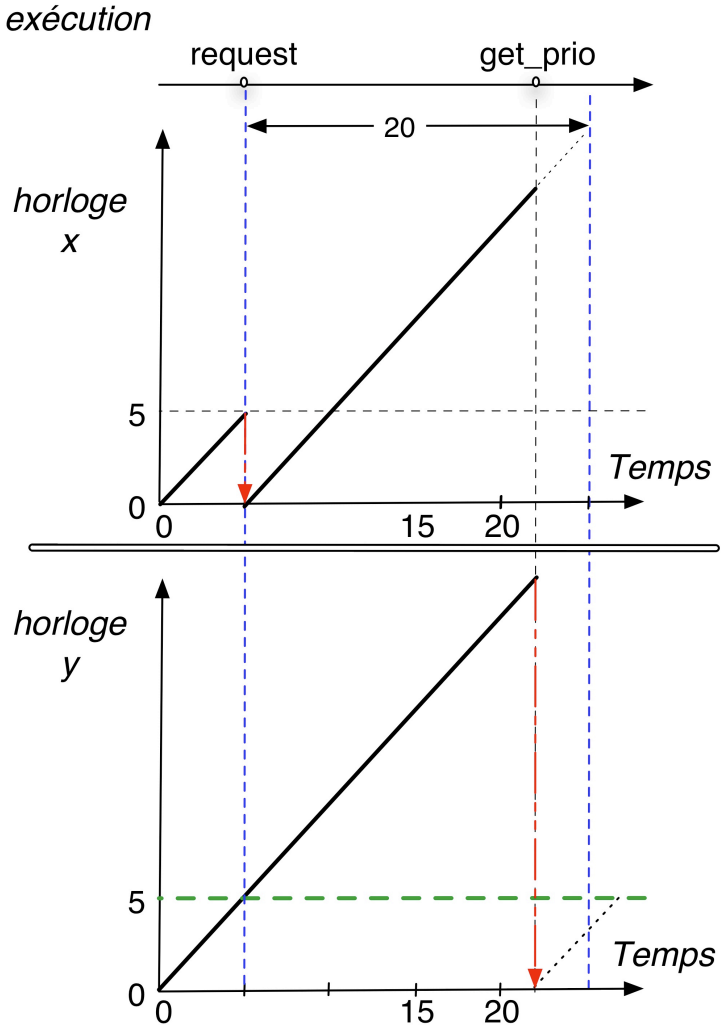
Identification des contextes d'application

- **L'analyse sémantique != analyse syntaxique si :**
  - L'évaluation de la validité du comportement possède un état

  Ex: un invariant instantané (Toujours x <0)
  => aucune mémoire nécessaire
- **Les propriétés significatives :**
  - vivacité bornée (e.g. réponse en temps borné)
  - Toute spécification équivalente à une propriété de vivacité bornée

Un exmeple : les automates temporisés





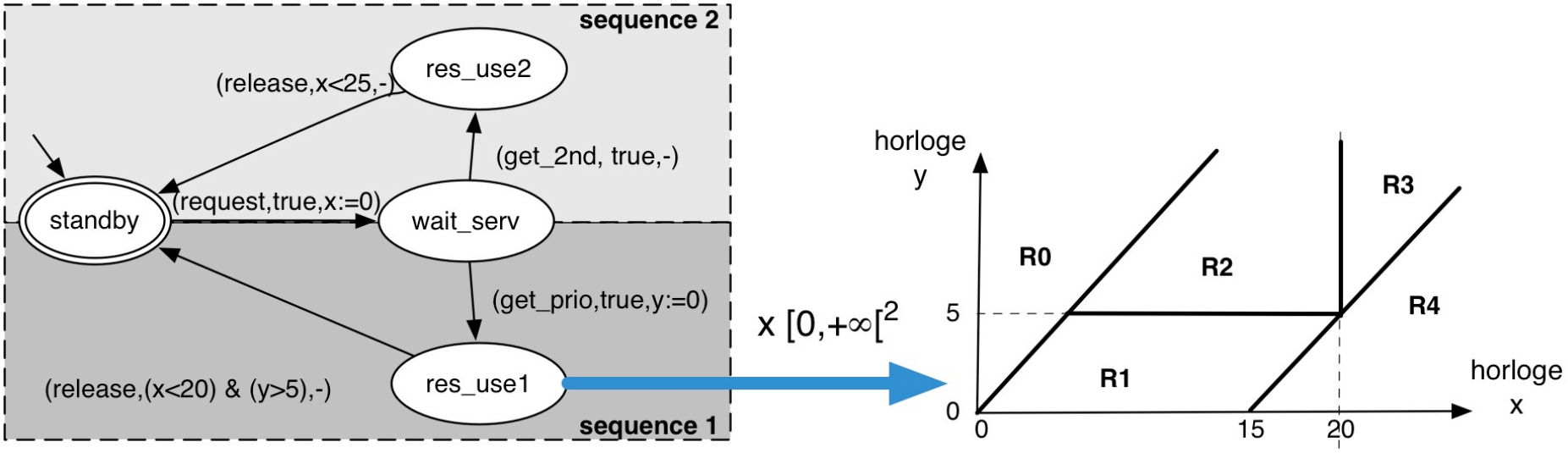
Sureté – vivacité bornée et accessibilité

- **[Bensalem05] Génération d'un automate dérivé de l'automate temporisé tel que**
  - Les transitions ne sont plus gardés
  - Seuls les états sont contraints temporellement : si l'on reste « trop longtemps » dans état ~ 1 erreur est détectée

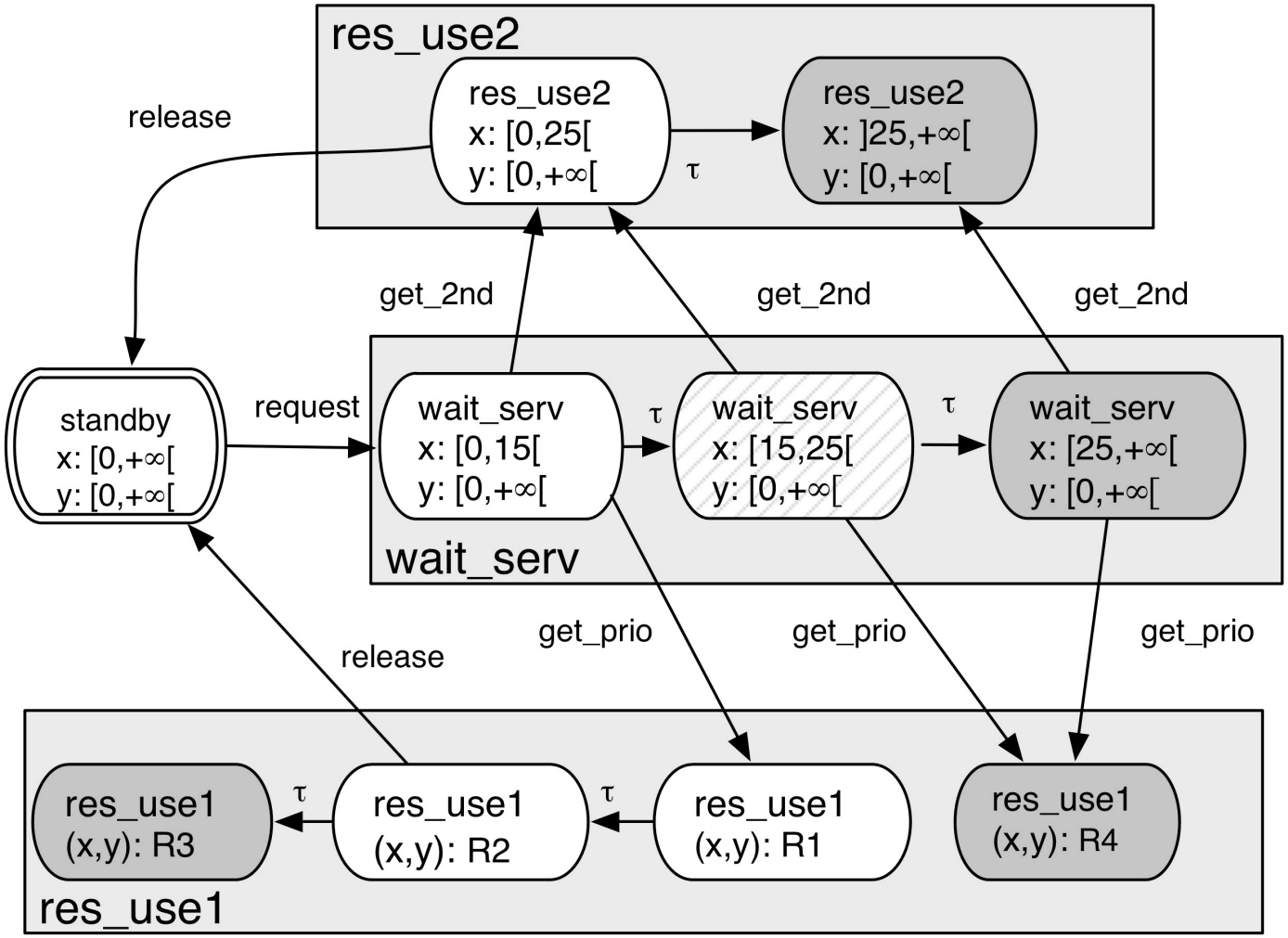
  Utilise l'analyse d'accessibilité et la génération de l'abstraction temporelle

  Le moniteur est implémentable à travers un unique timer et des tables de transition simples

De l'abstraction au moniteur (1)



De l'abstraction au moniteur (2)



Un moteur fixe et un modèle analysable

Algorithme 1 Simulation de la transition $s \xrightarrow{\Delta} s'$ depuis s

```
1  Algorithme Update(Δ, γ_s, S_s)
2  { L'état s correspond au lieu l_s et au vecteur γ_s. S_s est la classe d'équivalence
   associé à s. S et γ définissent conjointement un état. S détermine un lieu et γ le
   vecteur d'horloge correspondant. }
3  Variable
4      TempsRestant : durée
5      S : entier   { index de classe }
6      γ : entier[NC]   { vecteur d'horloge courant }
7      d : durée
8  Début
9      TempsRestant ← Δ
10     S ← S_s
11     γ ← γ_s
12     d ← Dist(γ, S)
13     TantQue (TempsRestant > d) ET (d > 0) Faire
14         TempsRestant ← TempsRestant – Dist(γ, S)
15         S ← succ_τ(S)
16         AdditionVect(γ, d)
17         d ← Dist(γ, S)
18     FinTantQue
19     AdditionVect(γ, TempsRestant)
20 Fin
21 { L'algorithme termine lorsque le «temps restant» n'est pas suffisant pour
   franchir une τ-transition supplémentaire. }
```

Algorithme 2 Calcul de la durée séparant l'instant courant de la nouvelle échéance

```
1  Algorithme Echeance(S'', γ_s'' : classe, vecteur)
2  { Cette fonction met directement à jour l'échéance existant sur l'occurrence du
   prochain événement. Cette mise à jour repose sur le calcul d'une date relative Δ
   définie depuis l'état courant (l'état courant après validation de l'événement
   correspond à la date 0) }
3  { s'' est l'état de l'automate juste après avoir franchi la transition correspondant
   à l'événement qui vient d'être validé. γ_s'' est le vecteur d'horloge de s'', et S'' est
   la classe de s'' }
4  { Chaque classe est une structure qui possède un champ ddl qui est un vecteur
   de coordonnées représentant les valeurs maximales acceptables pour chaque
   horloge pour S'' et ses successeurs temporels valides }
5  Constante
6      DernierEvt = der_evt   { date de l'événement venant d'être validé }
7  Début
8      Δ ← max(S''.ddl)   { Extraction de la borne supérieure de Δ, cf 1) }
9      Si Δ > 0 Alors   { existence d'une échéance, cf 1) }
10         Pour i variantDe 1 à N Faire   { Pour chaque horloge de γ_s'' }
11             { Calcule la durée disponible avant l'échéance induite par la }
12             { i^me horloge, x_i }
13             Si ddl[i] ≠ −1 Alors   { existence d'une échéance selon x_i, cf 1) }
14                 Δ ← min(Δ, ddl[i] − γ_s''[i])   { Mise à jour de Δ }
15             FinSi
16         FinPour
17         Δ← Δ − (high_res_clock() − DernierEvt)   { Correction de Δ, cf 2) }
18         rt_alarm_start(Δ)   { Configuration de l'alarme }
19     Sinon
20         rt_alarm_stop()   { Désactivation de l'alarme en cours }
21     FinSi
22 Fin
```

Synchronisation Système Observateur observateur - Recouvrement

- **On peut considérer deux modèles de communication**
  - Rendez-vou (~Synchrone == a priori confinement maximal)
  - Messages (~ Asynchrone == a priori confinement non maitrisé)
- **Pb : le but est d'empêcher les erreurs de se propager**

⇒ **Si aucune connaissance à priori : Rendez-vous**

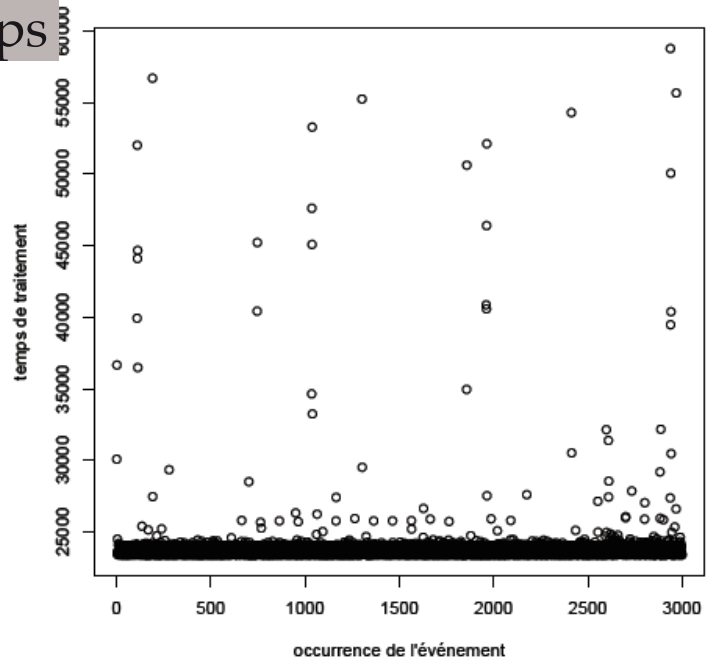
⇒ **Si l'on dispose d'une information identifiant les événements susceptible de propager l'erreur => il est possible de relâcher la synchronisation forte**

  - Mise en file d'attente de messages pour les observations « ne propageant pas » les erreurs
  - Synchronisation forte sur les autres

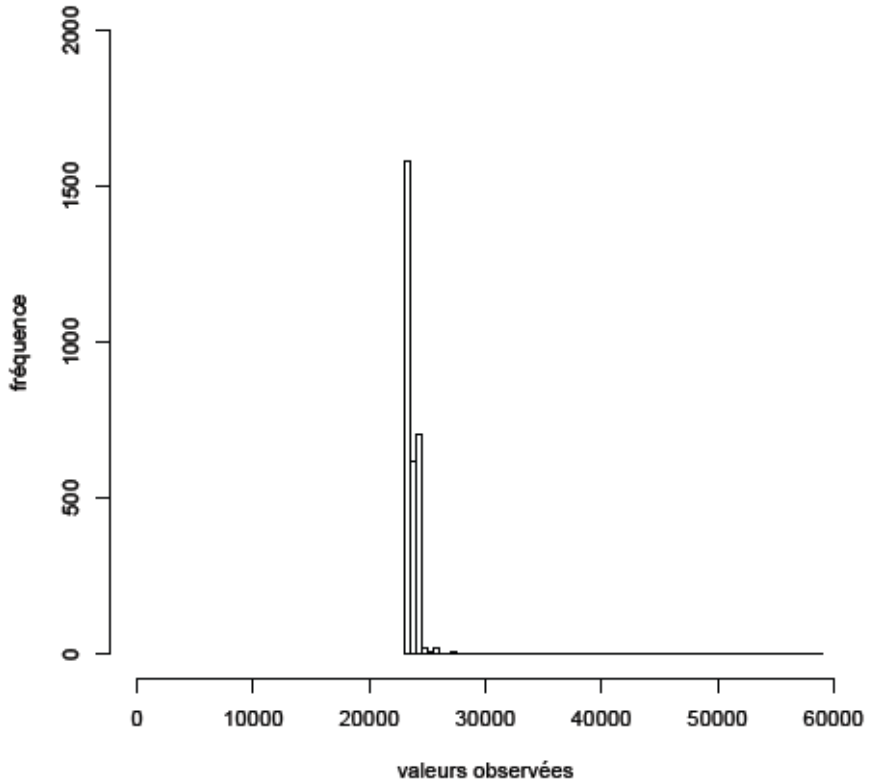
Mesures

- Peu d'événements mais sur un modèle très complexe

Temps



Expériences

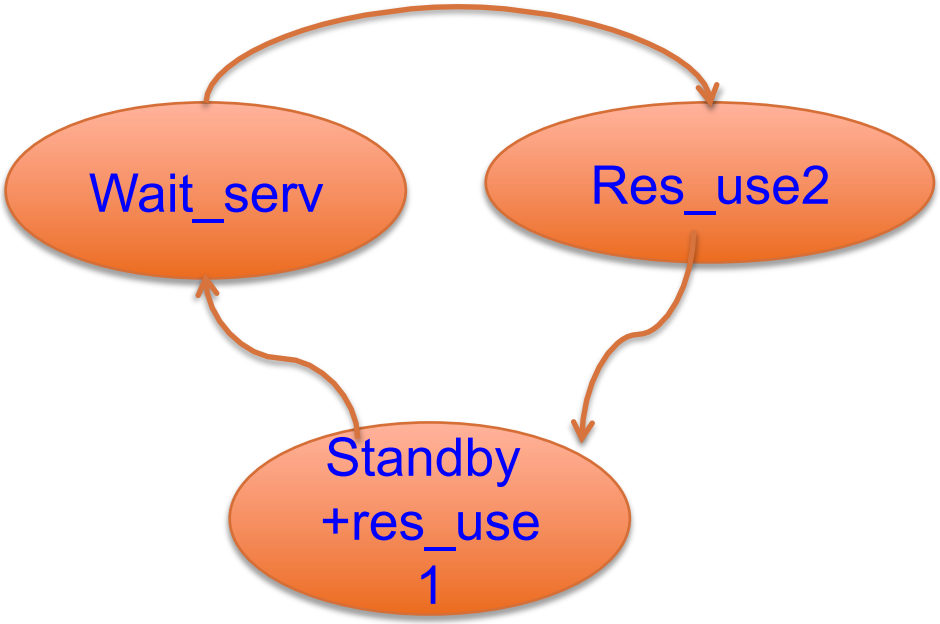


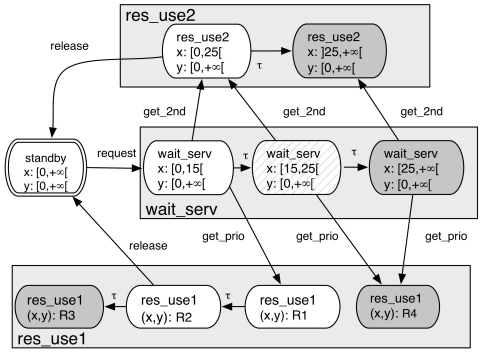
Temps

Caractérisation des paramètre sur les études de cas UPPAAL

- **La profondeur reste raisonnable < 20**
- **borne le temps d'analyse d'un événement à 60 µs pour une implémentation du Xenomai(PIII 800 Mhz, sans gestion d'énergie)**
- **Optimisations envisagées :**
  - Synchronisation mixte
  - Mode de fonctionnement asynchrone :
    le moniteur devient une tâche sporadique.

Evaluation par scénarios stochastiques





Définition des taux d'erreur : état par état

Conclusion

- Modélisation possible dans beaucoup de formalismes
- Pb 1 : efficacité discutée de l'approche
- Solutions : fournir un moyen d'évaluer à priori leur efficacité sur des scénarios de fautes
  (par ex. par synchronisation avec des modèles stochastiques d'occurrence de fautes)
- Pb 2 : trouver de bonnes signatures d'erreurs
- Deux alternatives:
  - Des modèles statistiques rendus discrets pour générer des détecteurs de seuils => suppose que le modèle de faute est connu (méthode des résidus)
  - Des modèles discret qui capture tout comportement anormal (hors norme avec une définition formelle ou non de la norme)

Références

Run-time monitoring pour automates et modèles discrets:

- Delgado, N.; Gates, A. Q. & Roach, S., « A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools », *IEEE Trans. Software Eng,* 2004*, 30*, 859-872
- Bensalem, S.; Bozga, M. & and Stavros Tripakis, M. K. "Testing Conformance of Real-Time Applications by Automatic Generation of Observers », *Electr. Notes Theor. Comput. Sci,* 2005*, 113*, 23-43
- Havelund, K. & Rosu, G., "Synthesizing Monitors for Safety Properties », *Proceedings of TACAS'2002, Springer-Verlag, Berlin,* 2002*, 2280*, 342-356
- Bauer, A.; Leucker, M. & Schallhart, C., 'Monitoring of Real-Time Properties',*FSTTCS, Springer,* 2006*,* Arun-Kumar, S. & Garg, N. *(ed.) 4337*, 260-272

Runtime monitoring pour les modèles probabilistes

- Sammapun, U., « Monitoring and checking of real-time and probabilistic properties », *University of Pennsylvania,* 2007

Prise en compte formelle du recouvrement :

- Easwaran, A.; Kannan, S. & Sokolsky, O., "Steering of Discrete Event Systems: Control Theory Approach », *Electr. Notes Theor. Comput. Sci,* 2006*, 144*, 21-39

Robert, T.; Fabre, J.-C. & Roy, M., « On-line Monitoring of Real Time Applications for Early Error Detection », In Proceedings *PRDC, IEEE Computer Society,* 2008, 24-31