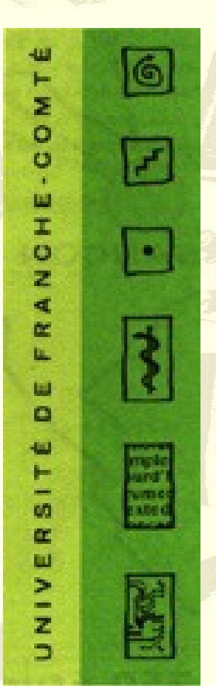
Réunion projet ACI GECCOO
10 mars 2006



Génération automatique de tests à partir de spécifications JML

Fabrice Bouquet, Frédéric Dadeau, Bruno Legeard,
Julien Groslambert, Jacques Julliand

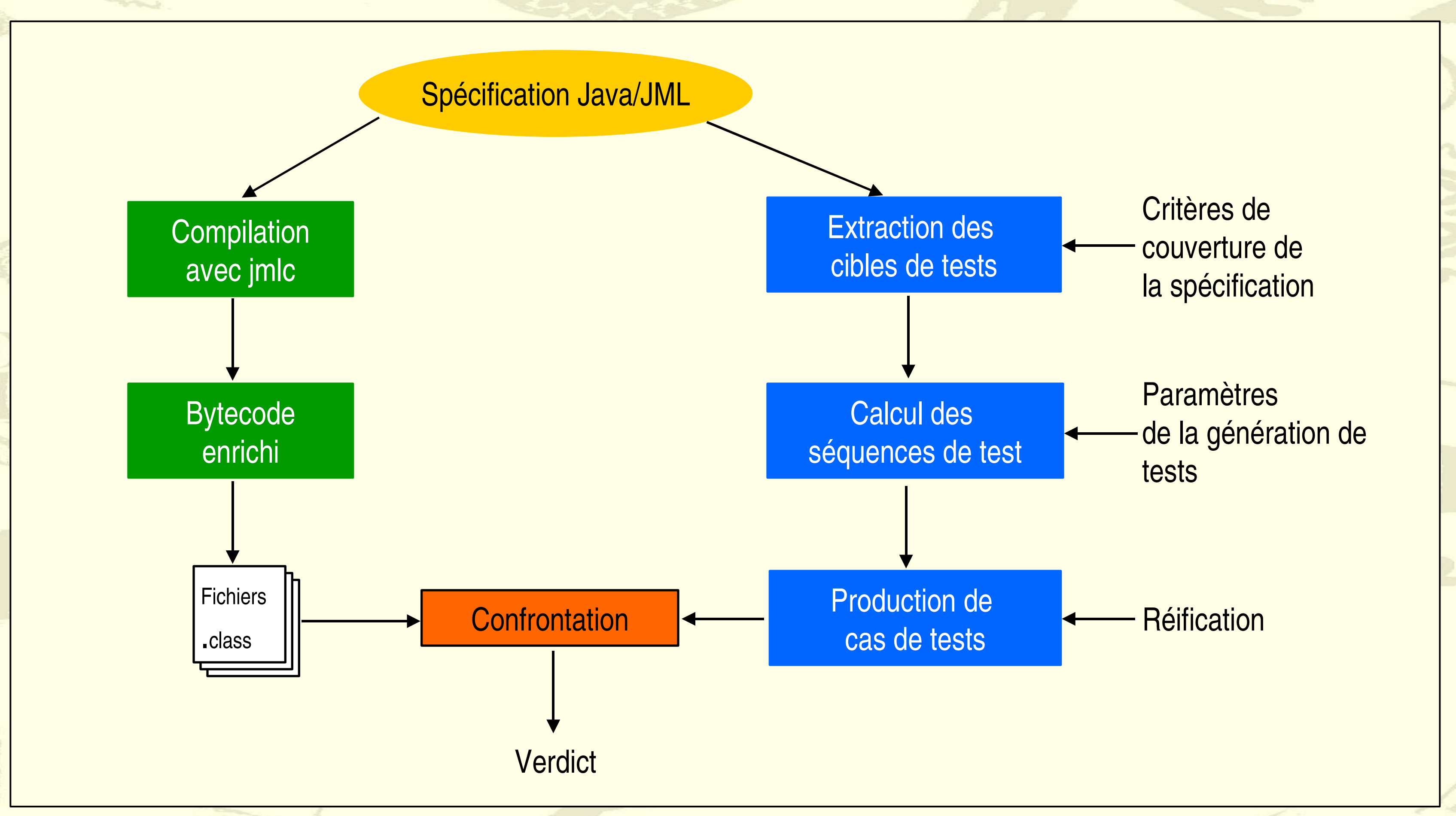
Motivations

- Tester un programme à partir d'une spécification :
  - Confronte le programme à sa spécification
  - Vérifier que le programme est conforme à sa spécification
- Utilisation de la spécification :
  - Cadre du test fonctionnel boite noire
  - Produire les cas de test
  - Permettre de comparer les résultats du programme avec la spécification

Motivations

- Intérêt majeur de JML dans le test :
  - Mélange spécification et implantation :
    → variables et signatures de méthodes identiques
  - Exécution du CT sur le système sous-test directe et gratuite
  - Nombreux outils déjà existants
- Deux approches présentées :
  - Approche aux limites (extension BZ-TT)
  - Approche à partir des propriétés temporelles (complémentarité JAG)
- Objectif : produire de manière automatique des tests pour Java à partir de JML

Motivations



Plan de la présentation

- Quelques approches de test avec JML
- Approche du test aux limites à partir de JML
- Approche à partir de propriétés temporelles
- Conclusion et perspectives

Le test en Java/JML (1/4)

- JML permet de faire du test fonctionnel pour Java
- Oracle donné par le Runtime Assertion Checker (RAC)
  - Pas d'assertions JML violées ➔ test OK
  - Présence d'assertions JML violées ➔ test KO
- Point de vue « en général » du test en Java
  - « The more test cases, the better »
  - Production massive de tests
  - Cas de tests insignifiants filtrés par le RAC

Le test en Java/JML (2/4)

Principe de fonctionnement du Run-Time Assertion Checker :

```
//@ invariant I ;

/*@ requires P ;
  @ ensures Q ;
  @ signals (Exception) S ;
  @*/
Type method(…) {
   body ;
}
```

```
Type method(…) {
    [Évaluation des expressions \old]
    [Vérification de l'invariant I]
    [Vérification des préconditions P]
  try {
    body ;
  }
  catch (JMLPreconditionException e) {
    throw new JMLInternalPreconditionException( );
  }
  catch (Exception e) {
    [Vérification des postconditions exceptionnelles S]
  }
  finally {
    [Vérification des postconditions normales Q]
    [Vérification de l'invariant I]
  }
}
```

Le test en Java/JML (3/4)

La référence JMLUnit

- Environnement d'exécution de tests unitaires de classes Java
- Combinaison
  - Runtime Assertion Checker des JML tools
  - JUnit
- Facilite la création de tests unitaires JUnit à partir de classes JML
- Utilisé par tous les outils de génération de test pour exécuter les tests
- Sélection des données de test manuelle ☹ mais quelques automatisations
  - Entiers : 0, 1, -1, Integer.MIN_VALUE, Integer.MAX_VALUE, + valeurs aléatoires
  - Objets : null, constructeur par défaut
  - Tableaux : null, aucun élément, un seul élément

Le test en Java/JML (4/4)

- Jartege : Java Random Test Generator
  - C. Oriat – LSR Grenoble
  - Choix aléatoire des appels de méthodes
  - Choix aléatoire des valeurs des paramètres
  - Filtrage des cas de tests non significatifs à la construction
- Tobias : Test Combinatoire
  - Y. Ledru et al. – LSR Grenoble
  - Combine les séquences de tests correspondant à un schéma défini par l'utilisateur
  - Définition manuelle des valeurs d'entrées
  - Filtrage des cas de tests inconclusifs à l'exécution
- TestEra / Korat : automated testing based on Java predicates
  - C. Boyapati et al., MIT – Cambridge
  - Génère les valeurs d'entrées satisfaisant un prédicat Java donné
  - Nécessite des limitations (taille) sur les structures
  - Cas de tests composés d'une création d'objet et d'une invocation de méthode

Plan de la présentation

- Quelques approches de test avec JML
- Approche du test aux limites à partir de JML
- Approche à partir de propriétés temporelles
- Conclusion et perspectives

Définition des cibles de tests

Rappel : le test à partir de spécifications B

- Extraction du but aux limites :
  - A partir d'un comportement de la spécification
    - Découpage d'une opération
    - Prédicat avant/après représenté sous forme d'un graphe
  - Recherche d'un état qui permet l'activation du comportement
    - En utilisant l'animation symbolique de la spécification
  - Minimisation/maximisation des variables d'état
    - En fonction de leur type (atomes, entiers, ensembles, etc.)
  - Activation du comportement avec des valeurs des paramètres aux limites
  - Inconvénient : état limite peut-être inatteignable
- Récemment modifié :
  - Préambule vise à atteindre un état permettant d'activer un comportement
  - Prédicat de spécialisation (éventuellement visant des valeurs limites)
  - Toujours des valeurs aux limites pour les paramètres

Définition des cibles de test

- Objectif du test JML
  - Tester une classe (ou un ensemble de classes)
  - Bouchonner les autres classes (considérer qu'elles sont correctes)
  - Activer les comportements des méthodes avec des paramètres pertinents
- Critères de couvertures
  - Couverture des comportements
  - Couverture des décisions
  - Couverture des données
- Critère d'arrêt
  - Activation de chaque comportement de chaque méthode de chaque classe

Définition des cibles de test
Couverture des comportements

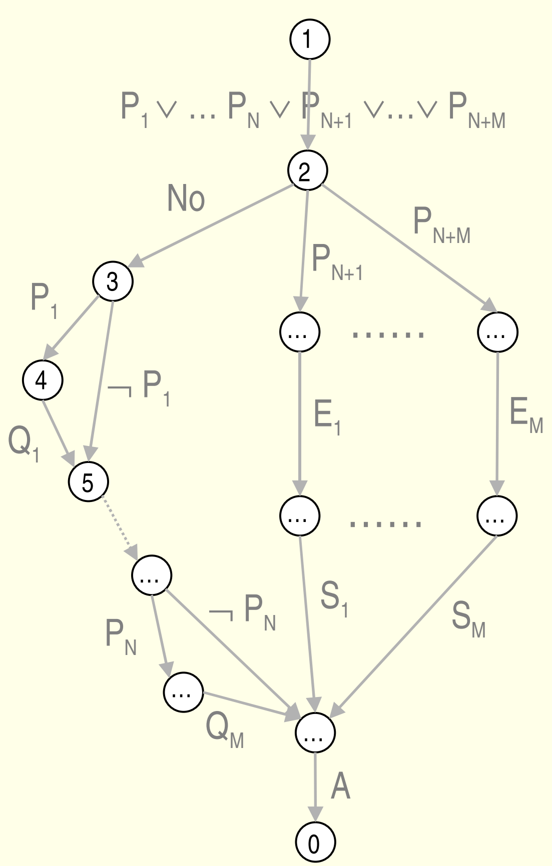
Extraction des comportements

```
/*@  requires P1 ;
  @  assignable A ;
  @  ensures Q1 ;
  @ also
  @  ...
  @ also
  @  requires PN
  @  assignable A ;
  @  ensures QN ;
  @ also
  @  requires PN+1 ;
  @  assignable A ;
  @  signals (E1) S1 ;
  @ also
  @  ...
  @ also
  @  requires PN+M ;
  @  assignable A ;
  @  signals (EM) SM ;
  @*/
Type methode(T1 p1, ...) { ... }
```

Hypothèse de travail :

Exclusion mutuelle entre comportements

- Normal
- Chacun des comportements exceptionnels



Définition des cibles de test
Couverture des décisions

- Réécriture des disjonctions dans les comportements $P_1 \vee P_2$
  - Réécriture 1: $P_1 \vee P_2$
    → Statement Coverage & Decision Coverage
  - Réécriture 2 : P_1 [] P_2
    → Decision/Condition Coverage
  - Réécriture 3 : $P_1 \wedge \neg P_2$ [] $\neg P_1 \wedge P_2$
    → Full Predicate Coverage
  - Réécriture 4 : $P_1 \wedge \neg P_2$ [] $\neg P_1 \wedge P_2$ [] $P_1 \wedge P_2$
    → Multiple Decision Coverage

Définition des cibles de test

Couverture des données

- Valeurs limites pour les paramètres de la méthode dans le contexte (partie avant) du comportement testé

  - Paramètres numériques : bornes du domaine de définition du paramètre permettant d'activer le comportement

    ```
    ex. //@ requires p > 0;
        void method(short p)        p = 1 [] p = 32767
    ```

  - Objet *p* de type *C* aux limites : valeurs systématiques « limites »
    1. Référence *null*
    2. Référence *this* (si *\typeof(this) <: \type(C)*)
    3. Objet *p* tel que : *p != null && p != this && \typeof(p) == \type(C)*
    4. Objet *p* tel que : *p != null && p != this && \typeof(p) < : \type(C)*
    5. Objet *p* tel que : *p == p'* avec *p'* autre paramètre

  - Pour les cas 2,3,4,5 : combinaison avec la mise aux limites des attributs numériques du paramètre utilisés dans le comportement

Adaptation du test aux limites à JML (1/3)

- Calcul d'un objectif de test avec la technologie contrainte CLPS-BZ

  - Considérer un état contraint du système (représentation symbolique)
    - Environnement : ensemble objets → attributs → valeurs
    - Quelque soient les objets existants (restriction : ensemble des adresses d'objets borné)
    - Contraint par les ensembles de définition des types des attributs

4. Poser les contraintes d'invariant de classe
   - Réduire les domaines des valeurs des attributs des classes

6. Poser la partie avant du comportement issu de la méthode à tester
   - Pose des contraintes d'existence de *this*

7. Application des règles de production des cas de test pour les objets
   - Pose des contraintes d'existence des objets paramètres (cas 3,4,5 du slide précédent)
   - Production d'un prédicat de spécialisation $Pspe_{Obj}$

8. Application des règles de production pour les données numériques
   - Minimize/Maximize les valeurs numériques
   - Production d'un prédicat de spécialisation $Pspe_{Num}$

Adaptation du test aux limites à JML (2/3)

- Travail avec les prédicats de spécialisation $Pspe = Pspe_{Obj} \wedge Pspe_{Num}$
- $Pspe$ = objectif de test
  - S'assurer de la consistance *Pspe* avec le contexte
  - Préambule du cas de test :
    trouver par animation symbolique un état satisfaisant *Pspe*
- Le calcul de préambule contiendra :
  - La création d'un objet de la classe testée (guidé)
  - La création des objets correspondant aux paramètres objets de la méthode testée (guidé)
  - L'invocation de méthodes pour placer l'objet testé dans un état satisfaisant l'activation du comportement (algorithme Best-First)
  - L'invocation de méthodes pour placer les objets en paramètres aux limites (algorithme Best-First)

Adaptation du test aux limites à JML (3/3)

- Une fois le préambule calculé, on obtient un cas de test abstrait
- Cas de test réifié pour produire du code Java exécutable
- Le JML Run-Time Assertion Checker vérifiera les assertions (oracle) lors de l'exécution

Plan de la présentation

- Quelques approches de test avec JML
- Approche du test aux limites à partir de JML
- Approche à partir de propriétés temporelles
- Conclusion et perspectives

Approche à partir de propriétés temporelles

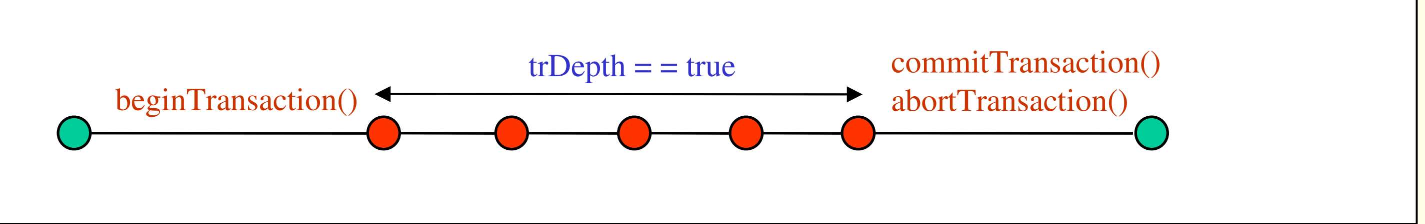
- Principe : utiliser les annotations générées par JAG
  - Implantation du langage de M. Huisman & K. Trentelman
    - exprimant des propriétés temporelles \always, \before, \unless, \until, …
  - Basé sur la notion d'événements
    - des appels de méthode, terminaison normale, exceptionnelle
  - Rajoute des annotations :
    - variables ghost,
    - invariants,
    - contraintes historiques
  - Permet l'expression de propriété de sûreté et de vivacité

Approche à partir de propriétés temporelles

Problématique : générer des tests pertinents vis-à-vis de la propriété

- Exemple :

  after beginTransaction() **called always** trDepth == true **unless**
  commitTransaction() **called**, abortTransaction() **called**



- Toutes les exécutions ne contenant pas d'appel à beginTransaction() ne sont pas pertinentes.

Approche à partir de propriétés temporelles

- A partir les propriétés de sûreté et de vivacité :
  - Sûreté :

    //@ **invariant** $Pred_{contexte} \Rightarrow Propriete$

  - Vivacités :

```
//@ invariant   Variant >= 0 ;
//@ constraints \old(Pred_Entrée) && !(Pred_Sortie) ==> \old(Variant) > Variant ;
//@ constraints \old(Pred_Entrée) && !(Pred_Sortie) ==> requires(m_1) || …|| requires(m_N) ;
```

    où $m_1, m_2, \ldots, m_N$ sont des méthodes de progrès

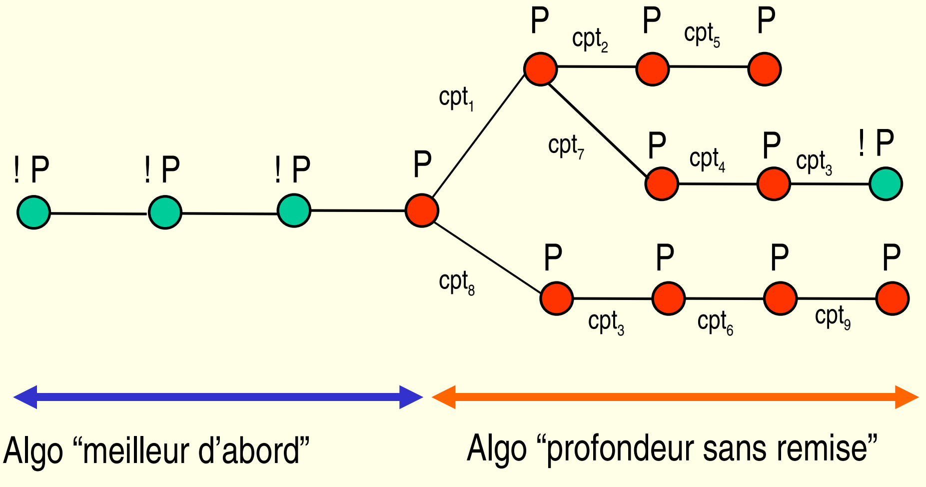
- Objectif du test :
  - activer les comportements de la spécification
  - dans le contexte : $\backslash old(Pred_{Entrée})$ && $!(Pred_{Sortie})$ avec différentes couvertures
- Le Runtime Assertion Checker vérifie que la propriété n'est pas violée sur le code

Approche à partir de propriétés temporelles

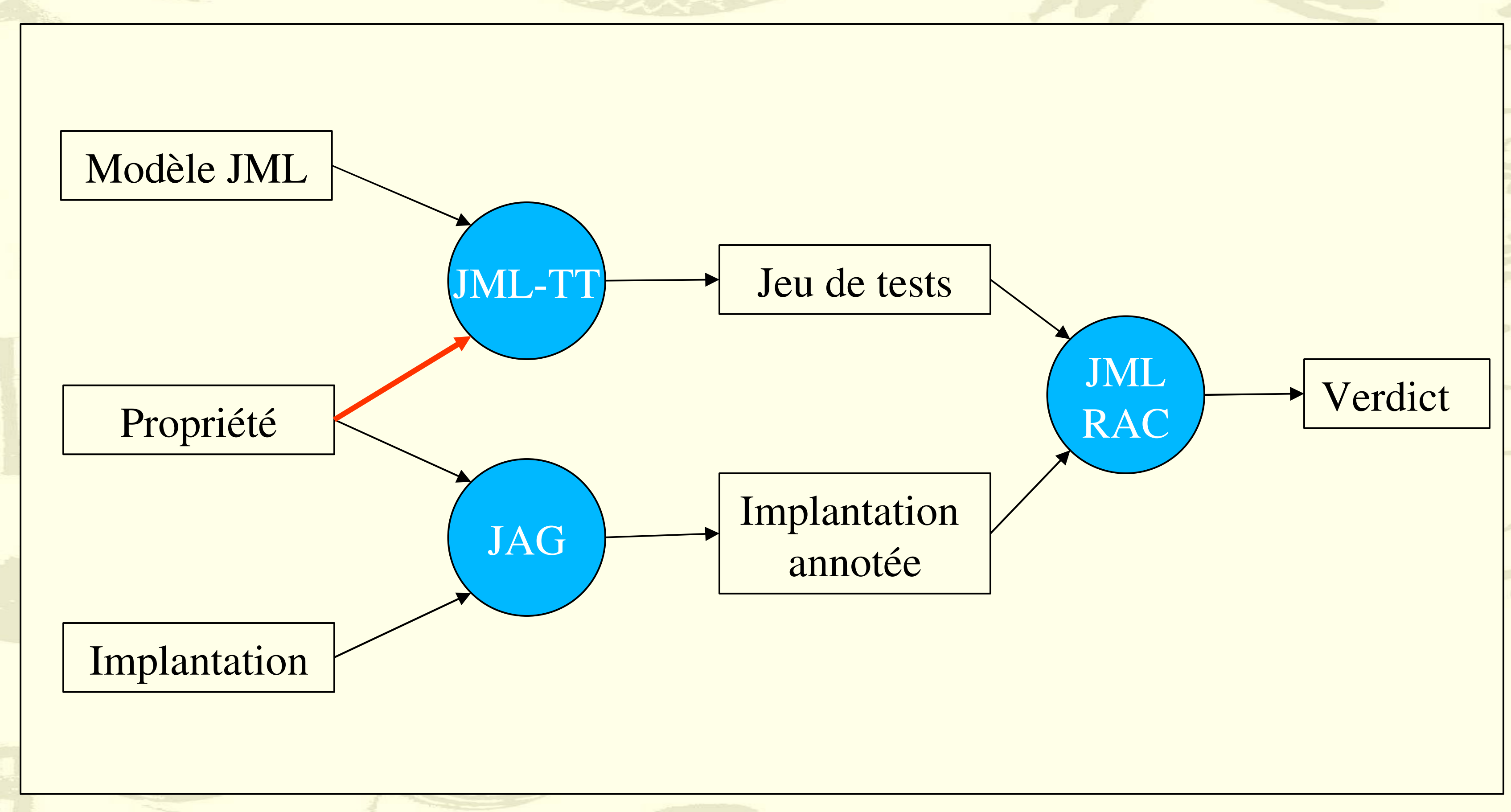
- Principe de construction des cas de tests :
  - Atteindre un état satisfaisant le contexte
    - algorithme « best-first »
  - Activation des comportements par :
    - parcours en profondeur sans remise du graphe d'état
      - algorithme de parcours de graphe [Col05]
      - cherche à activer une liste de comportements
      - dès qu'on comportement a été activé, on l'enlève de la liste
    - tant que le contexte est satisfait
  - Cas d'arrêts possibles :
    - Plus de comportements activables
    - Tous les comportements couverts
    - Le contexte n'est plus satisfait
    - → Réification du cas de test utilisant les valeurs aux limites des paramètres

Approche à partir de propriétés temporelles

- Illustration du principe de construction des cas de tests :
  - Comportements à activer cpt1, cpt2, cpt3, cpt4, cpt5, cpt6, cpt7, cpt8, cpt9
  - Contexte P



Collaboration JAG/JML-TT



Plan de la présentation

- État de l'art du test avec Java basé sur JML
- Approche du test aux limites à partir de JML
- Approche à partir de propriétés temporelles
- Conclusion et perspectives

Conclusion

- Originalité par rapport à l'existant
  - Génération automatique de tests basés sur la spécification
    - Pas une séquence aléatoire (≠ Jartege)
    - Complètement automatique (≠ Tobias)
  - Utilisation des valeurs limites
    - Classique pour les valeurs numériques
    - Nouveauté pour les objets
  - Calcul du préambule guidé
    - Par l'activation d'un ou plusieurs comportement issus de la spécification
    - Par les valeurs aux limites d'attributs spécifiques

Perspectives

- Évaluer la qualité des tests produits

- Adapter une autre méthode de génération des tests
  - Test combinatoire + valeurs limites (approche à la Tobias)
  - Test aléatoire + valeurs limites (approche à la Jartege)

- Comparer les approches