Spécification de Demoney en JML par raffinement

Pierre-Alain Masson, Julien Groslambert

LIFC Besançon

Réunion GECCOO - 10 mars 2006

UNIVERSITÉ DE FRANCHE-COMTÉ

L I F C

Laboratoire d'Informatique de l'Université de Franche-Comté

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

FRE 2661

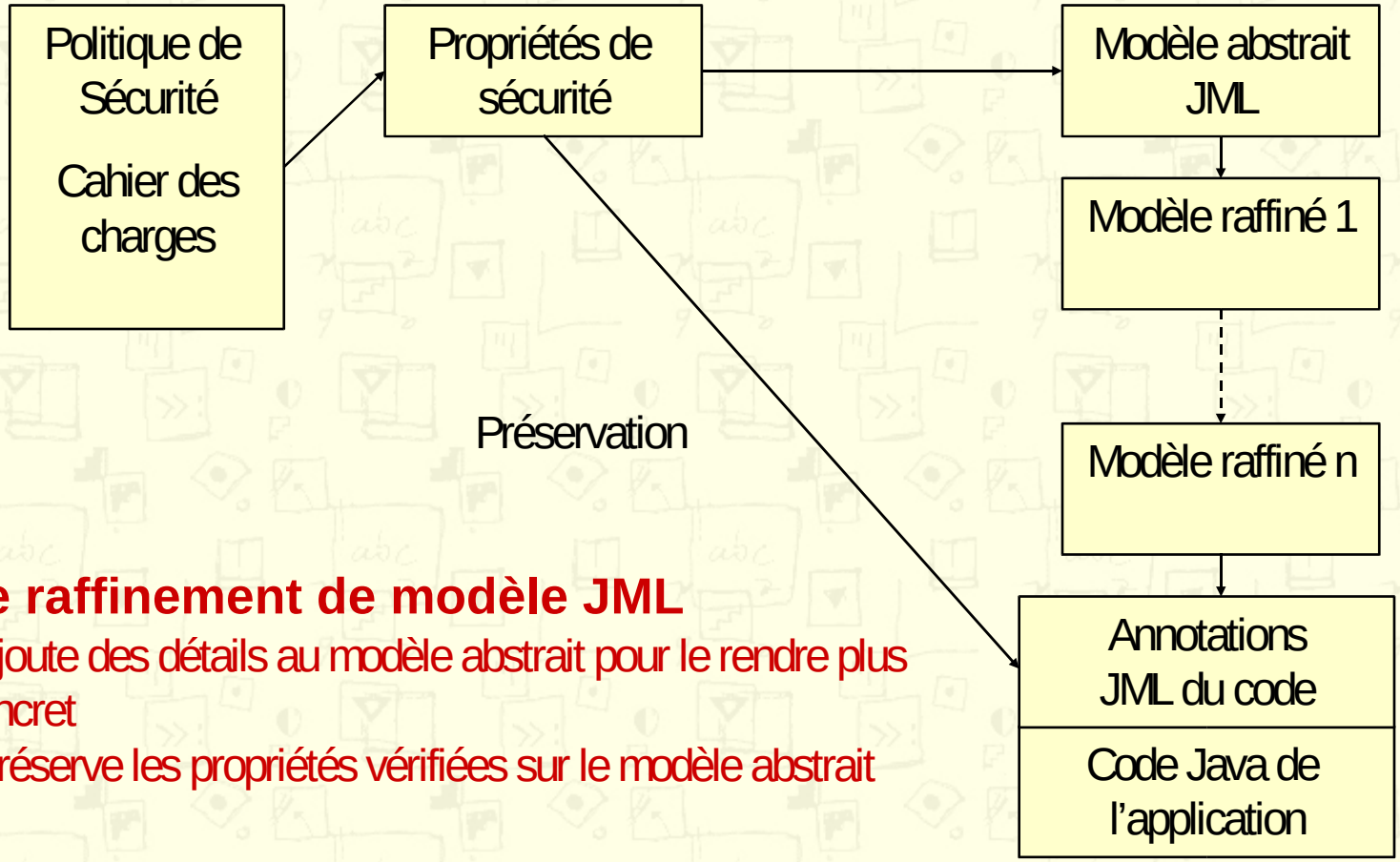
Plan de l'exposé

- Rappels sur le raffinement JML
- Spécification JML par raffinement de Demoney
- Perspectives d'utilisation du modèle de Demoney

Plan de l'exposé

- **Rappels sur le raffinement JML**
- Spécification JML par raffinement de Demoney
- Perspectives d'utilisation du modèle de Demoney

Spécification par raffinement en JML



Le raffinement de modèle JML

- ajoute des détails au modèle abstrait pour le rendre plus concret
- préserve les propriétés vérifiées sur le modèle abstrait

Raffinement de données

- Notations
  - **C_r raffine C_a** $\quad C_a \supseteq C_r$
  - **C_a vérifie une propriété φ** $\quad \Sigma_{Ca} \models \varphi$

- Raffinement des données
  - Prédicat JML I_c
  - Lien entre variables abstraites et raffinées
  - Changement d'espace d'état
  - Ajout de nouvelles variables durant le raffinement

Raffinement de méthodes: schéma

```
/*@ behavior
  @ requires Pa;
  @ diverges Da;
  @ assignable Aa;
  @ ensures Qa;
  @ signals (Ea e) Ra
  @*/
Ma();
```

```
/*@ behavior
  @ requires Pr;
  @ diverges Dr;
  @ assignable Ar;
  @ ensures Qr;
  @ signals (Er e) Rr
  @*/
Mr();
```

Raffinement de méthodes : clauses JML

- Renforcement des pre-conditions — @ requires
- Renforcement des conditions de divergence — @ diverges
- Pas d'ajout d'effet de bord — @ assignable
- Renforcement des post-conditions — @ ensures
- Inclusion des types d'exceptions — @ signals
- Renforcement des post-conditions exceptionnelles — @ signals

Raffinement de méthodes : exemple

```
/*@ behavior
  @  requires estInscrit;
  @  ensures true;
  @  signals (Exception e) true
  @*/
vote();
```

```
/*@ behavior
  @  requires estInsrit && ! aVote;
  @  assignable aVote;
  @  ensures aVote;
  @ also
  @ requires estInscrit && aVote;
  @ signals (Exception e) true;
  @*/
 vote();
```

```
estInscrit && !aVote ||  estInscrit && aVote ==>  estInscrit;
\forall aVote . aVote ==> true;
```

Plan de l'exposé

- Rappels sur le raffinement JML
- **Spécification JML par raffinement de Demoney**
- Perspectives d'utilisation du modèle de Demoney

Spécification par raffinement de Demoney : démarche adoptée

- Expérience de spécification par raffinement JML
- Démarche de spécification « pure »
  - Spec. indépendante de l'idée d'implantation
- Fonctionnalités basiques de JML utilisées
  - Variables model
  - Invariant
  - Variables modifiables
  - Pre et post conditions
  - Exceptions et post-conditions exceptionnelles

Spécification par raffinement de Demoney : méthodologie

- Point de départ : liste des commandes Demoney
- Ajout des détails dans un ordre intuitif
  - Nécessite une bonne connaissance globale du cahier des charges
  - Création d'un nouveau modèle par détail introduit
- Hiérarchie des modèles
  - Des aspects haut-niveau vers bas-niveau de la spécification

- **Peu de retours en arrière nécessaires**

Spécification par raffinement de Demoney : Modélisation (1)

- Un package pour les constantes
- Un package pour les exceptions
- Un package pour les différents niveaux du modèle
  - Model 0 : liste des commandes de Demoney
  - Model 1 : personnalisation
  - Model 2 : niveaux d'accès
  - Model 3 : définition du PIN
  - Model 4 : modification du solde de la carte
  - Model 5 : vérification du code PIN

Spécification par raffinement de Demoney : Modélisation (2)

- Les différents niveaux du modèle (suite)
  - Model 6 : séquencement des fonctions d'authentification
  - Model 7 : séquencement des fonctions de transaction
  - Model 8 : messages d'erreur (exceptions)
  - Model 9 : cycle de vie de l'application
  - Model 10 : limitation du nombre de transactions
  - Model 11 : limitation du nombre d'essais pour le code PIN
  - Model 12 : journalisation des transactions
  - Model 13 : communication par APDU

Exemple (model 8) : méthode de personnalisation `storeData()`

```
/*@
@ behavior
@   requires personnalized == false && accessLevel == AL.ADMIN
@       && authenticateSequence == SEQ.OK
@       && transactionSequence == SEQ.OK;
@   assignable personnalized;
@   ensures personnalized == true;
@ also
@   requires personnalized == true;
@   signals(ConditionOfUseNotSatisfiedException) true;
@ also
@   requires accessLevel < AL.ADMIN;
@   signals(SecurityStatusNotSatisfiedException) true;
@*/
void store_data() throws ConditionOfUseNotSatisfiedException,
   SecurityStatusNotSatisfiedException
{}
```

Résultats de la modélisation

- 13 niveaux de raffinement
- Toutes les commandes de Demoney spécifiées
  - Jusqu'au niveau communication par APDU
- Taille des modèles
  - Avant APDU : ~ 350 lignes
  - Avec APDU : ~ 900 lignes
  - Code Java Trusted Logic : ~ 1300 lignes (sans les commentaires)

Bilan de la modélisation

- Modèle en accord avec les exigences de sécurité
- Exigences de raffinement respectées
- Modèle validé et animé
  - JML-TT animator
- Aspects bas niveau (APDU) difficiles à modéliser

Exemple de modélisation APDU (Model 13)

```
/*@ requires      comMsg.cla == 0x84 - 0x7F && comMsg.ins == 0xE2 - 0x7F && comMsg.p1 == 0x80 - 0x7F && comMsg.p2 == 0x00 - 0x7F && comMsg.lc == comMsg.data.length && comMsg.data[1] == 1
@                 && comMsg.data[7] == 2 && comMsg.data[11] == 3  && comMsg.data[15] == 4 && comMsg.data[19] == 5 && comMsg.data[23] == 6 && comMsg.data[25 + comMsg.data[24]] == 7;
@ assignable      \nothing;
@ ensures         repMsg.sw1 == 0x90 - 0x7F && repMsg.sw2 == 0x00 - 0x7F;
@ signals         (InconsistentTLVStructureException)       comMsg.lc != comMsg.data.length  && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x85 - 0x7F;
@ signals         (IncorrectParametersInTheDataFieldException)       comMsg.data[1] < 1 || (comMsg.data[1] > 7 && comMsg.data[1] < 14) || comMsg.data[1] > 17
@                 && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x80 - 0x7F;
@ signals         (IncorrectParametersInTheDataFieldException)       comMsg.data[7] < 1 || (comMsg.data[7] > 7 && comMsg.data[7] < 14) || comMsg.data[7] > 17
@                 && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x80 - 0x7F;
@ signals         (IncorrectParametersInTheDataFieldException)       comMsg.data[11] < 1 || (comMsg.data[11] > 7 && comMsg.data[11] < 14) || comMsg.data[11] > 17
@                 && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x80 - 0x7F;
@ signals         (IncorrectParametersInTheDataFieldException)       comMsg.data[15] < 1 || (comMsg.data[15] > 7 && comMsg.data[15] < 14) || comMsg.data[15] > 17
@                 && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x80 - 0x7F;
@ signals         (IncorrectParametersInTheDataFieldException)       comMsg.data[19] < 1 || (comMsg.data[19] > 7 && comMsg.data[19] < 14) || comMsg.data[19] > 17
@                 && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x80 - 0x7F;
@ signals         (IncorrectParametersInTheDataFieldException)       comMsg.data[23] < 1 || (comMsg.data[23] > 7 && comMsg.data[23] < 14) || comMsg.data[23] > 17
@                 && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x80 - 0x7F;
@ signals         (IncorrectParametersInTheDataFieldException)       comMsg.data[25 + comMsg.data[24]] < 1
@                 || (comMsg.data[25 + comMsg.data[24]] > 7 && comMsg.data[25 + comMsg.data[24]] < 14) || comMsg.data[25 + comMsg.data[24]] > 17 && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x80 - 0x7F;
@ signals         (FunctionNotSupportedException)       comMsg.data[1] >= 14 && comMsg.data[1] <=17  && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x81 - 0x7F;
@ signals         (FunctionNotSupportedException)       comMsg.data[7] >= 14 && comMsg.data[7] <=17  && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x81 - 0x7F;
@ signals         (FunctionNotSupportedException)       comMsg.data[11] >= 14 && comMsg.data[11] <=17   && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x81 - 0x7F;
@ signals         (FunctionNotSupportedException)       comMsg.data[15] >= 14 && comMsg.data[15] <=17   && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x81 - 0x7F;
@ signals         (FunctionNotSupportedException)       comMsg.data[19] >= 14 && comMsg.data[19] <=17  && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x81 - 0x7F;
@ signals         (FunctionNotSupportedException)       comMsg.data[23] >= 14 && comMsg.data[23] <=17   && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x81 - 0x7F;
@ signals         (FunctionNotSupportedException)       comMsg.data[25 + comMsg.data[24]] >= 14 && comMsg.data[25 + comMsg.data[24]] <=17 && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x81 - 0x7F;
@ signals         (IncorrectParametersInTheDataFieldException)       comMsg.data[2] != 4  && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x80 - 0x7F;
@ signals         (IncorrectParametersInTheDataFieldException)       comMsg.data[8] != 2  && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x80 - 0x7F;
@ signals         (IncorrectParametersInTheDataFieldException)       comMsg.data[12]!= 2 && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x80 - 0x7F;
@ signals         (IncorrectParametersInTheDataFieldException)       comMsg.data[16]!= 2 && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x80 - 0x7F;
@ signals         (IncorrectParametersInTheDataFieldException)       comMsg.data[20]!= 2 && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x80 - 0x7F;
@ signals         (IncorrectParametersInTheDataFieldException)       comMsg.data[24]!= 0  && comMsg.data[24]!= 16  && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x80 - 0x7F;
@ signals         (IncorrectParametersInTheDataFieldException)       comMsg.data[26 + comMsg.data[24]] != 0  && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x80 - 0x7F;
@ signals         (IncorrectParametersInTheDataFieldException)   comMsg.data[26 + comMsg.data[24]] <5  && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x80 - 0x7F;
@ signals         (IncorrectParametersInTheDataFieldException)   comMsg.data[26 + comMsg.data[24]] > 16 && repMsg.sw1 == 0x6A - 0x7F && repMsg.sw2 == 0x80 - 0x7F;
@*/
void store_data()
throws      InconsistentTLVStructureException,
            IncorrectParametersInTheDataFieldException,
            FunctionNotSupportedException {}
```

Plan de l'exposé

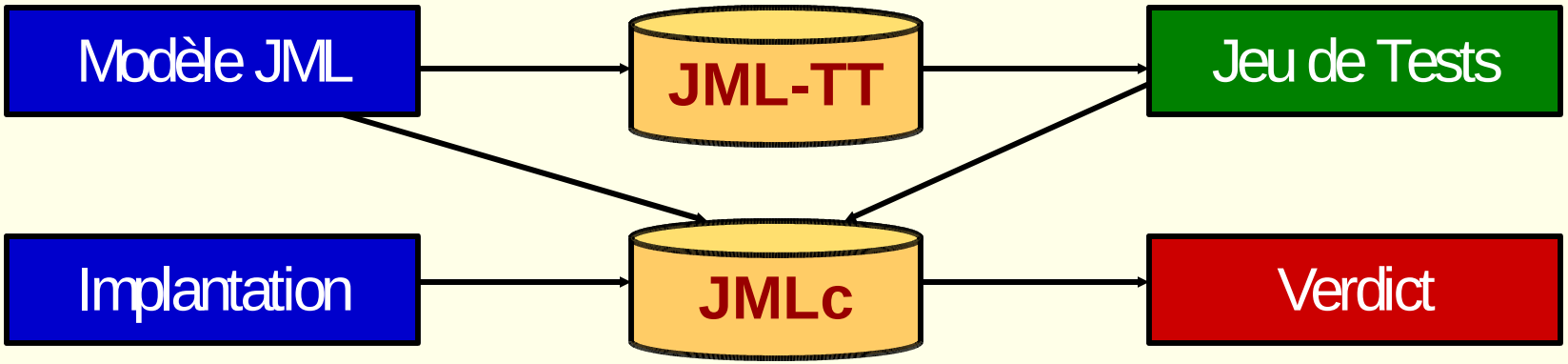
- Rappels sur le raffinement JML
- Spécification JML par raffinement de Demoney
- **Perspectives d'utilisation du modèle de Demoney**

Perspectives : confrontation au code Java

- Confrontation spec. JML et code Java Trusted Logic
  - Vérification à la volée avec JMLc
  - Vérification par preuve avec Jack et/ou Krakatoa
  - Besoin : Adaptation de la spec. à l'API de l'applet (concrétisation)
- Phase de concrétisation automatique (couche d'adaptation)

Perspectives : génération de tests

- JML-TT animator
  - Animation et validation ✔
    - Adaptations mineures du modèle
    - Découverte de quelques erreurs dans le modèle
  - Génération automatique de tests orientés par notre modèle



Perspectives : Vérification du modèle

- Cohérence interne du modèle
  - Méthode proposée dans ZB'05 (limitée)
  - Génération d'OP pour Jack, Krakatoa
- Preuve de raffinement des modèles
  - Besoin d'un outil de génération d'OP

Perspectives : propriétés temporelles

- Propriétés de sûreté et de vivacité pour Demoney
- Spécification de ces propriétés dynamiques
  - Logique temporelle pour JML
- Vérification des propriétés sur le modèle
  - Utilisation de JAG pour ajouter des annotations sur le modèle
  - Vérification de la cohérence du modèle enrichi
  - Préservation de la propriété lors du raffinement

Conclusion

- Démarche concluante de spécification en JML par raffinement
- Aspects bas niveaux difficiles à modéliser
- Phase de concrétisation à développer