



# Technologie Smartesting Certifylt

Applications aux composants de sécurité, cartes à puces et e-transactions

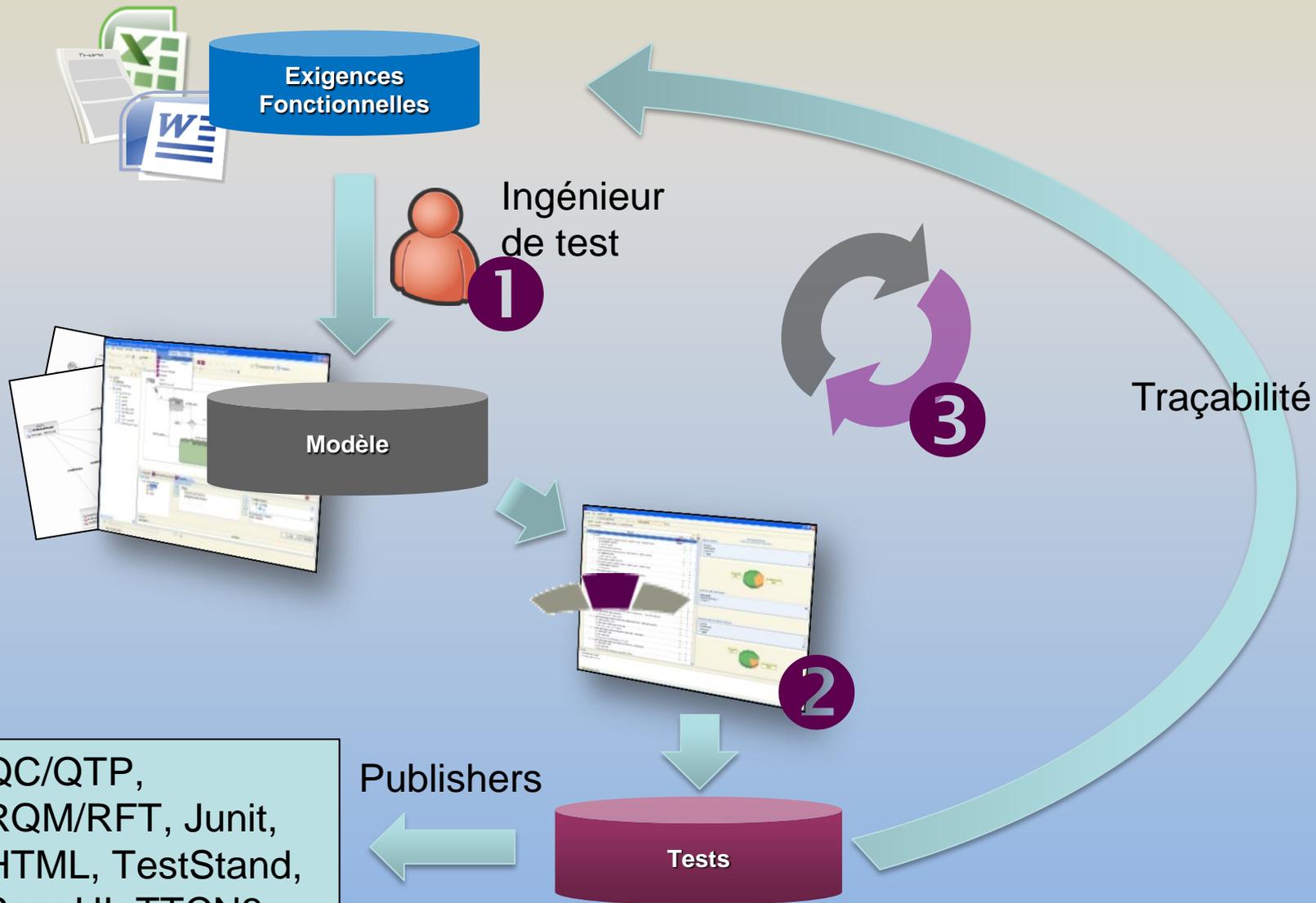
**Bruno Legiard**

# Agenda

---

- ① **Technologie CertifyIt: du modèle aux tests**
- ② **Pilotage de la génération: complémentarité des approches**
- ③ **Exemple : Composant de sécurité**
- ④ **Conclusion**

# Smartesting Certifylt





Le modèle est une abstraction du système sous test, permettant d'en exprimer le comportement attendu.

## Diagramme de classes :

→ *représentation des objets métier manipulés par le système sous test*

→ *les classes portent les opérations appelables sur le système sus test*

## Langage d'actions :

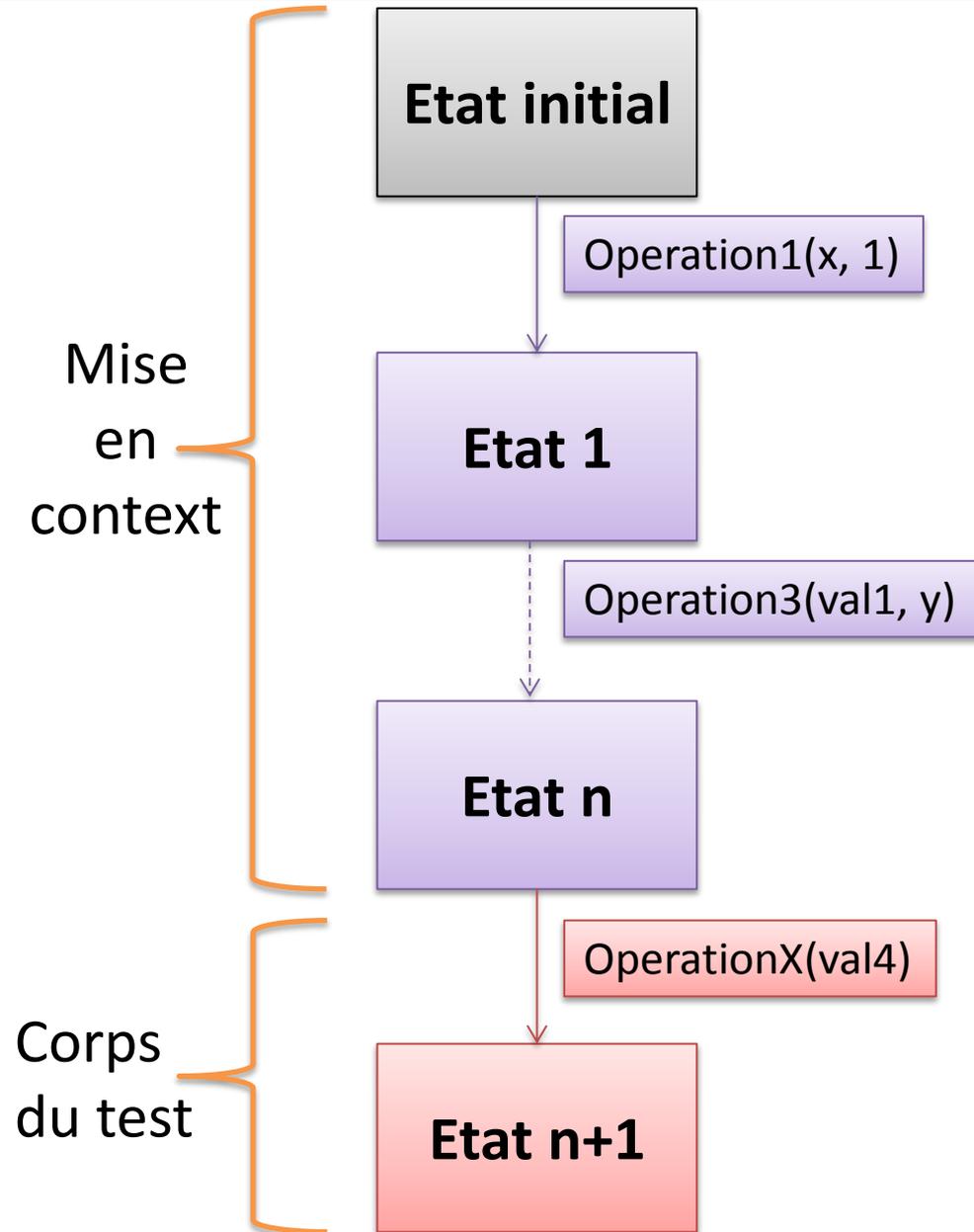
→ *représentation des comportements attendus du système sous test, porté par les opérations présentes dans le diagramme de classes*

## Configuration de l'état initial :

→ *représentation de l'état initial du système sous test pour une suite de tests*



Un test est une séquence d'opérations sur le système sous test, permettant d'ammener celui-ci de son état initial, jusqu'au context d'activation du comportement à tester.





## Spécification fonctionnelles DAB

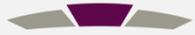
Le distributeur automatique de billets doit répondre à un ensemble d'exigences fonctionnelles listées ci-dessous.

### Gestion de l'état du distributeur

ETAT001	Déclencher sa mise hors service lorsque sa caisse comporte un solde inférieur à 900
ETAT002	Déclencher sa mise hors service, lorsqu'il détecte une anomalie
ETAT003	La mise en service est faite manuellement
ETAT004	L'opérateur est chargé du rechargement de la caisse
ETAT005	Il revient en mode veille après la fin de la transaction
ETAT006	Toute sollicitation d'un client entraîne une commutation en mode transaction
ETAT007	Mis hors service, il interdit l'insertion de carte

### Fonctions de contrôle

CTRL001	Vérification de la lisibilité des informations des cartes présentées
CTRL002	Vérification de la date de péremption par rapport à la date du jour
CTRL003	Vérification du code saisi par le client par rapport à celui enregistré sur la carte
CTRL004	Demande d'autorisation de la carte, ainsi que la provision du compte correspondant
CTRL005	Redemande le code, s'il est erroné et si le nombre d'essais est inférieur ou égal à 2



Le modèle est une abstraction du système sous test, permettant d'en exprimer le comportement attendu.

## Diagramme de classes :

→ *représentation des objets métier manipulés par le système sous test*

→ *les classes portent les opérations appelables sur le système sus test*

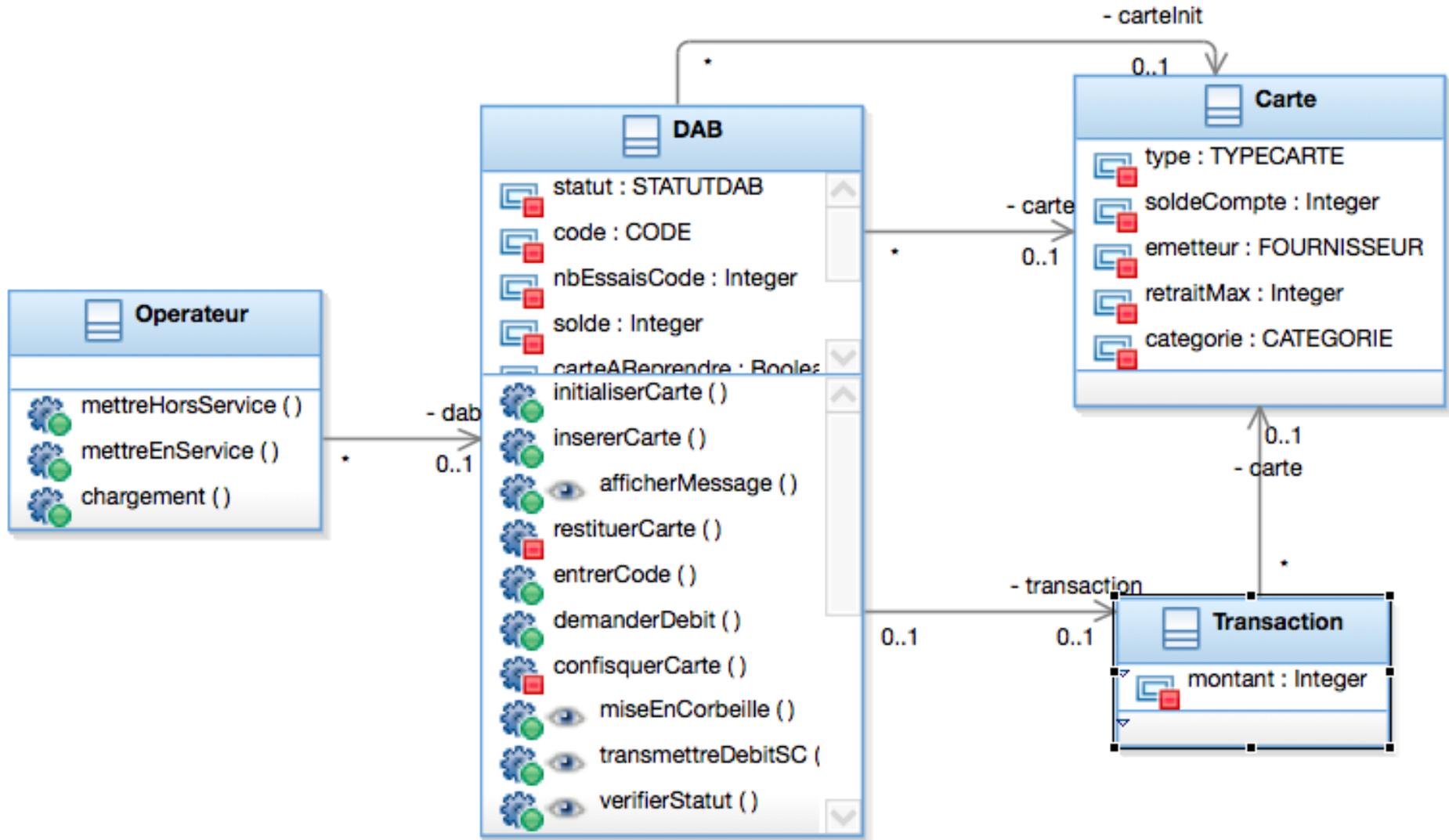
## Langage d'actions :

→ *représentation des comportements attendus du système sous test, porté par les opérations présentes dans le diagramme de classes*

## Configuration de l'état initial :

→ *représentation de l'état initial du système sous test pour une suite de tests*

# Exemple DAB : Diagramme de classes





Le modèle est une abstraction du système sous test, permettant d'en exprimer le comportement attendu.

## Diagramme de classes :

→ *représentation des objets métier manipulés par le système sous test*

→ *les classes portent les opérations appelables sur le système sus test*

## Langage d'actions :

→ *représentation des comportements attendus du système sous test, porté par les opérations présentes dans le diagramme de classes*

## Configuration de l'état initial :

→ *représentation de l'état initial du système sous test pour une suite de tests*



### Fonctions de contrôle

CTRL001	Vérification de la lisibilité des informations des cartes présentées
CTRL002	Vérification de la date de péremption par rapport à la date du jour
CTRL003	Vérification du code saisi par le client par rapport à celui enregistré sur la carte
CTRL004	Demande d'autorisation de la carte, ainsi que la provision du compte correspondant
CTRL005	Redemande le code, s'il est erroné et si le nombre d'essais est inférieur ou égal à 2



```
1|--@REQ: CTRL003-Vérification du code saisi par le client par rapport à celui enregistré sur la carte
2if (p_Code=CODE::FAUX)then
3  ---@REQ: INFO006-Informe le client que son code est erroné
4  afficherMessage(MESSAGE::CODE_ERRONE) and
5  nbEssaisCode = nbEssaisCode+1 and
6  if (nbEssaisCode >= 3) then
7    ---@REQ: GEST006-Confisque la carte au bout de la 3ème tentative de saisi de code avec la même carte
8    ---@REQ: INFO004-Signaler au client que sa carte est confisquée
9    ---@NAME: Trois essais code faux
10   confisquerCarte()
11  else
12    ---@REQ: CTRL005-Redemande le code, s'il est erroné et si le nombre d'essais est inférieur ou égal à 2
13    afficherMessage(MESSAGE::ENTRER_CODE)
14  endif
15else
16  code = CODE::OK and
17  ---@REQ: INFO002-Signaler au client de saisir sa somme
18  afficherMessage(MESSAGE::ENTRER_MONTANT)
19endif
20
```



Le modèle est une abstraction du système sous test, permettant d'en exprimer le comportement attendu.

## Diagramme de classes :

→ *représentation des objets métier manipulés par le système sous test*

→ *les classes portent les opérations appelables sur le système sus test*

## Langage d'actions :

→ *représentation des comportements attendus du système sous test, porté par les opérations présentes dans le diagramme de classes*

## Configuration de l'état initial :

→ *représentation de l'état initial du système sous test pour une suite de tests*

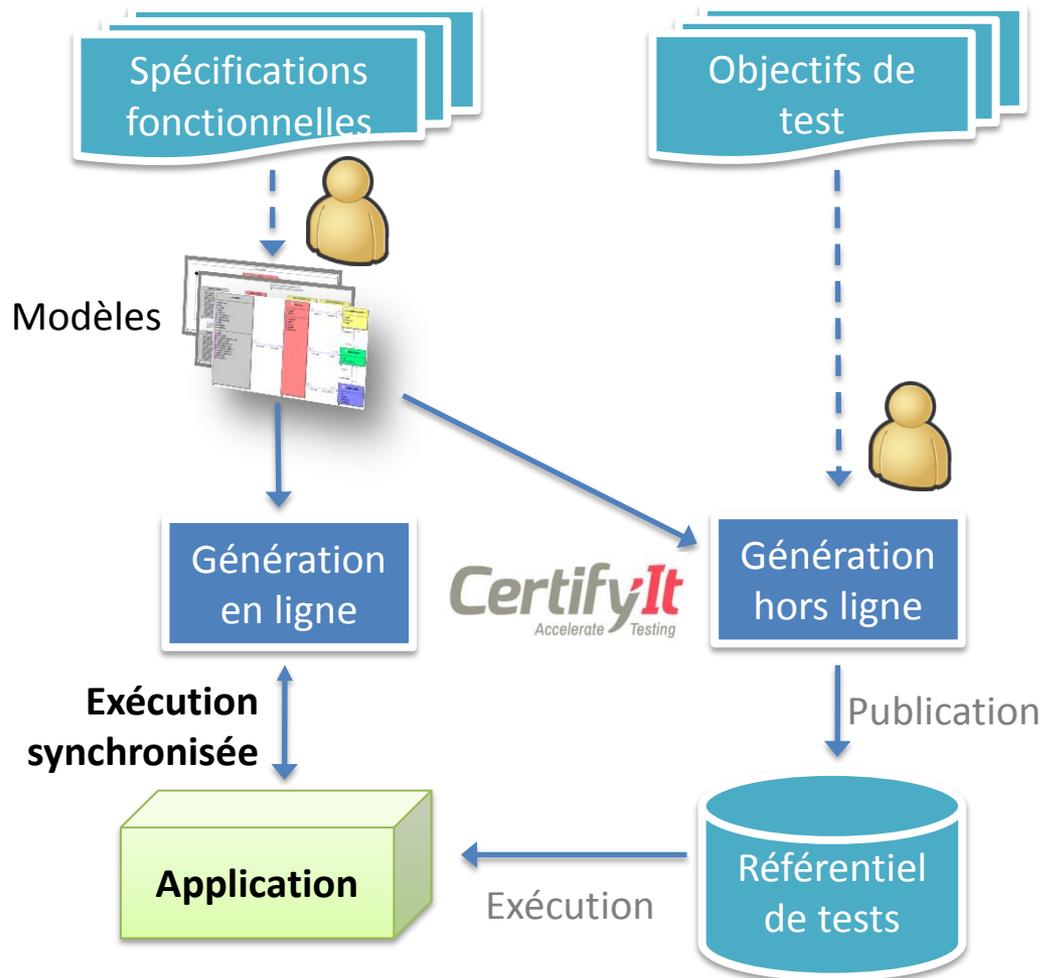


## Instances of DABModel::InitialData

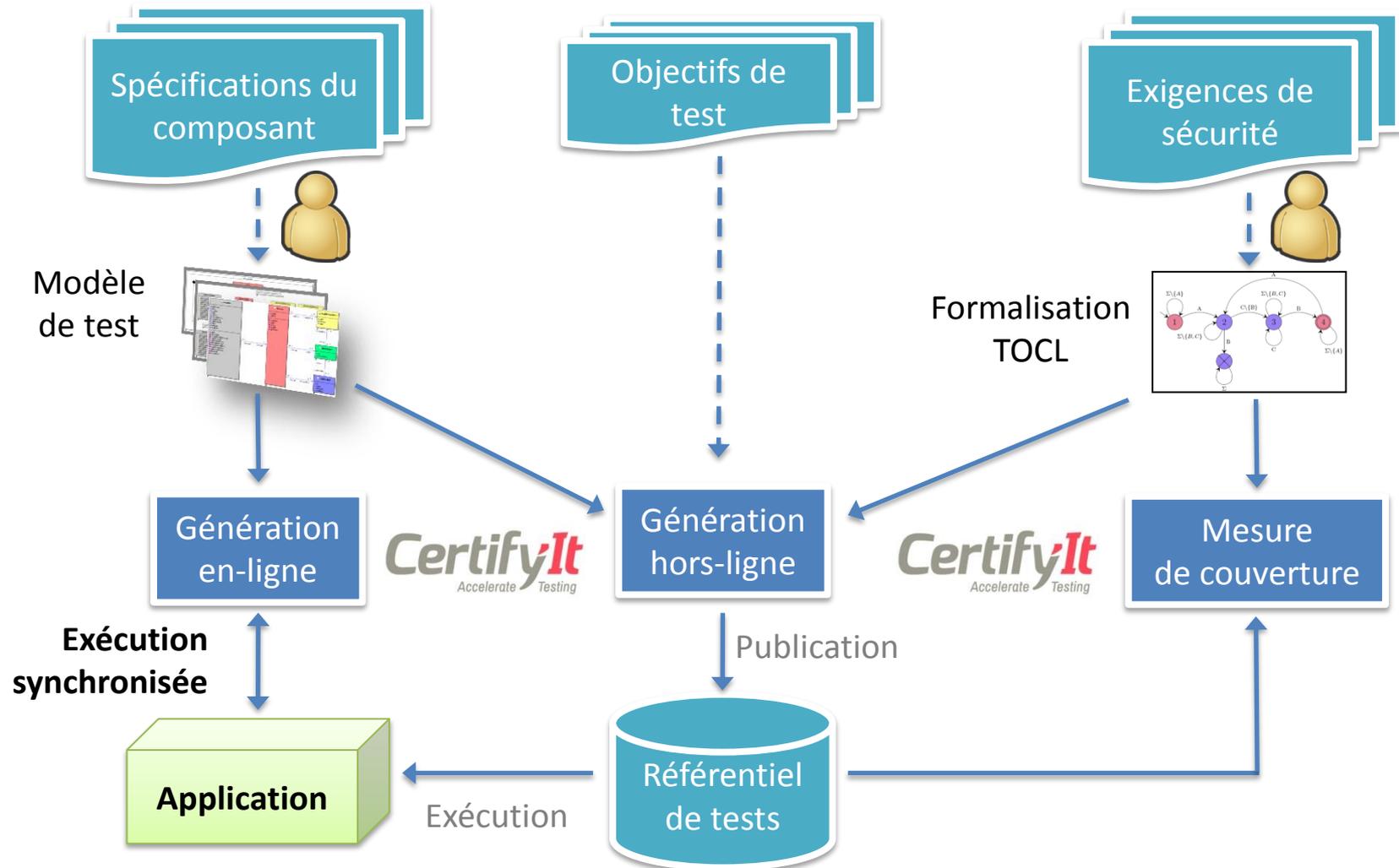
Manage instances of DABModel::InitialData

- ▼ DABModel
  - ▼ Carte (7)
    - ▼ carteFaibleSolde
      - ☐ categorie = NORMALE (default)
      - ☐ emetteur = BANQUE
      - ☐ retraitMax = 500
      - ☐ soldeCompte = 10
      - ☐ type = VALIDE
      - ▶ carteFortSolde
      - ▶ cartellisible
      - ▶ carteInconnue
      - ▶ carteInterditBancaire
      - ▶ cartePerimee
      - ▶ carteVierge
  - ▼ DAB (1)
    - ▼ leDab
      - ☐ anomalie = AVANTDETECTION (default)
      - ☐ billetsAPrendre = false (default)
      - ☐ carteAReprendre = false (default)
      - ☐ cartelnit = ∅
      - ☐ carte = ∅
      - ☐ code = INCONNU (default)
      - ☐ nbFoisCode = 0 (default)

# Génération & exécution des tests (en ligne / hors ligne)



# Couverture fonctionnelle + sûreté & sécurité

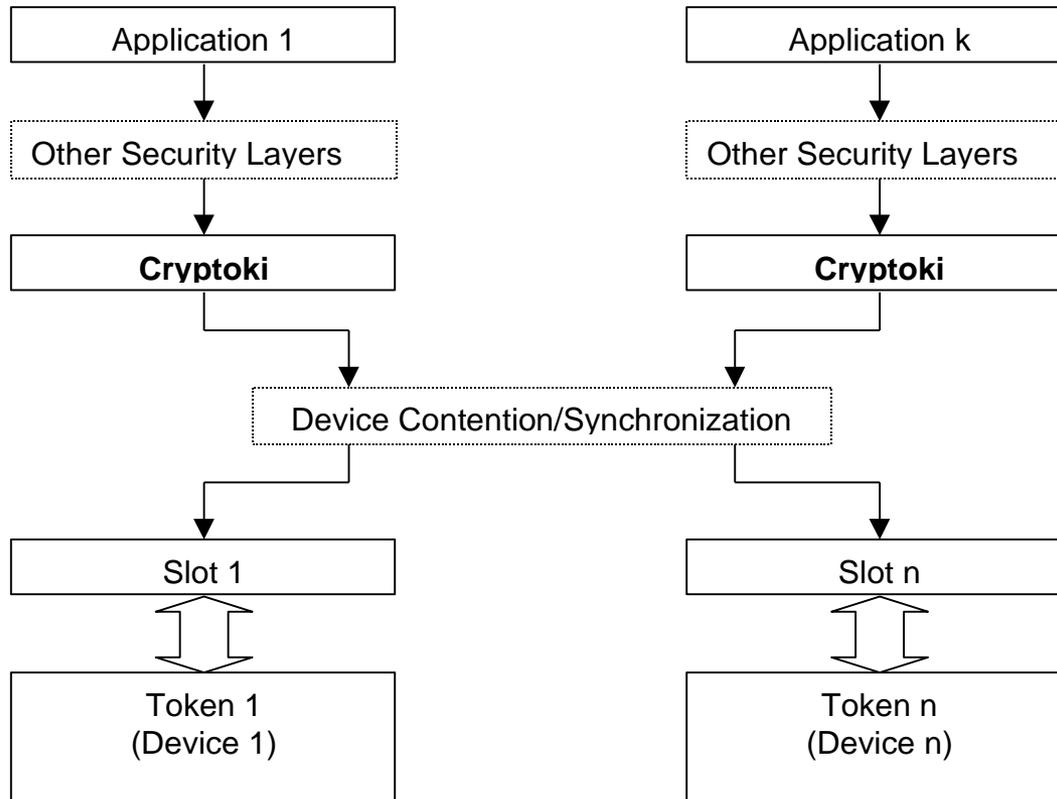


# Cas d'étude - PKCS#11

---

- PKCS#11 est une spécification qui définit un standard d'interfaçage des dispositifs cryptographiques.
- Elle définit une API, nommée « Cryptoki », avec un ensemble de fonctions en C, dont des fonctions cryptographiques.

# Modèle général de Cryptoki

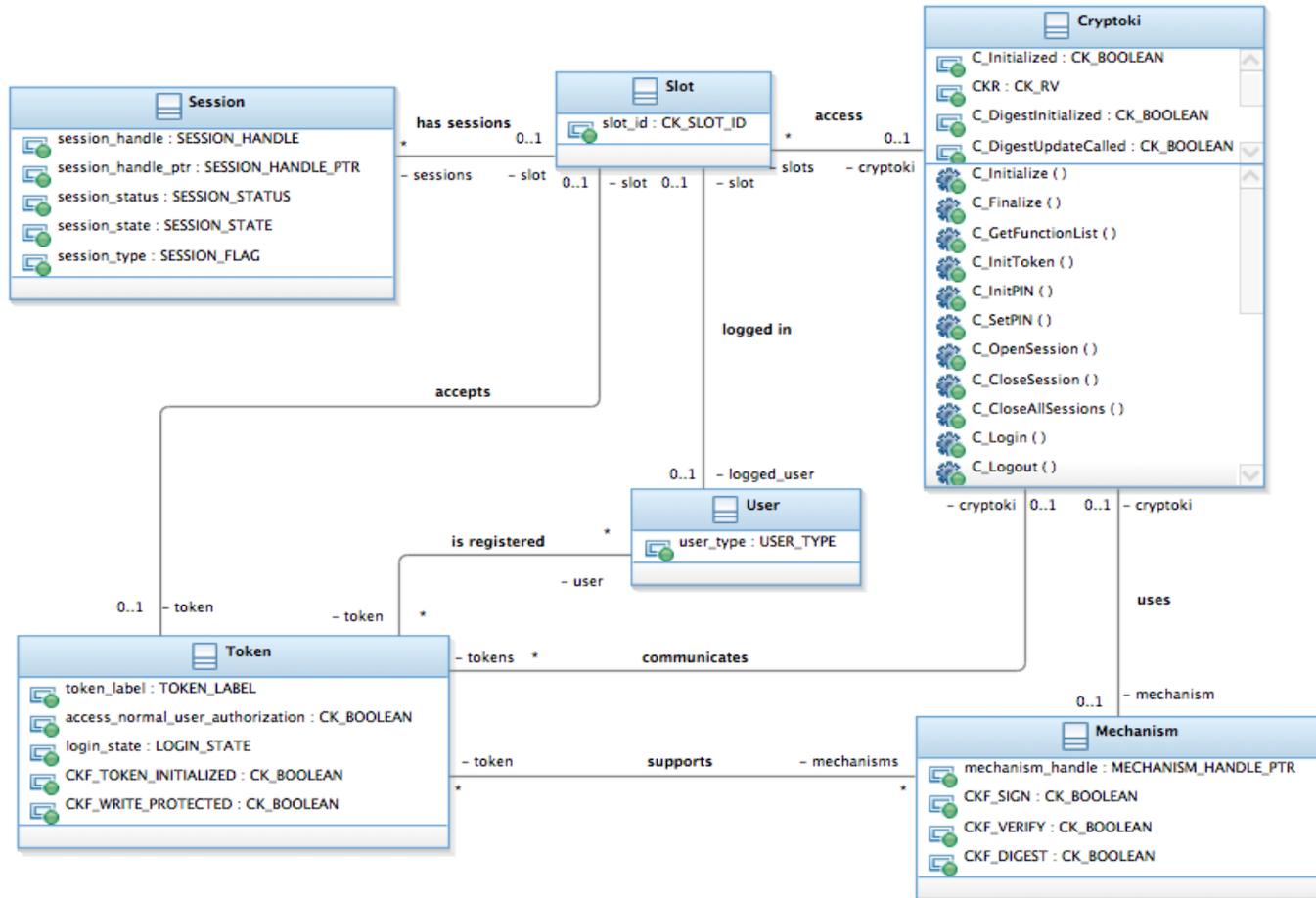


# Périmètre PKCS#11

---

- Générales
  - Initialisation, terminaison de communication
- Gestion de session
  - Ouverture, fermeture de session, connexion et déconnexion.
- Gestion de token
  - Initialisation du token, pin, modification de pin.
- Fonctions cryptographiques
  - Signature, vérification, hachage, création de clef de cryptage

# Modèle PKCS#11



# CertifyIt

The screenshot displays the CertifyIt application interface, which is divided into several sections:

- Project Preferences Help:** Located at the top left, with a 'Run test generation' button and an 'HTML publisher' dropdown.
- Search stories:** A search bar with a magnifying glass icon.
- Artifacts Table:** A table listing test artifacts with columns for 'Artifacts', 'Status', and 'Tests'. The table contains multiple rows for 'Cryptoki::C\_SetPIN()', 'Cryptoki::C\_Sign()', and 'Cryptoki::C\_SignFinal()', each with a green checkmark in the status column and a count in the tests column.
- Test detail:** A section on the right showing the execution steps of a test. The steps include:
  - Default model instance
  - Initialized model instance
  - CryptokiInstance.setUp()
  - CryptokiInstance.C\_Initialize(VOID\_NULL\_PTR) = CKR\_OK
  - CryptokiInstance.C\_OpenSession(SLOT\_VALID, CKF\_SERIAL\_SESSION\_CKF\_RW\_SESSION, VOID\_NULL\_PTR, VOID\_NULL\_PTR) = CKR\_OK
  - CryptokiInstance.C\_Login(HANDLE\_NEW\_RW\_USER\_PUBLIC\_SESSION, CKU\_USER, PIN\_USER\_TOKEN\_INIT) = CKR\_OK
  - CryptokiInstance.nominal\_generateKey(HANDLE\_NEW\_RW\_USER\_PUBLIC\_SESSION, CK\_TRUE, CK\_TRUE, KEY\_ID1) = CKR\_OK
  - CryptokiInstance.C\_SignInit(HANDLE\_NEW\_RW\_USER\_PUBLIC\_SESSION, CKM\_MDS\_HMAC\_PTR, KEY\_ID1) = CKR\_OK
    - CryptokiInstance.checkResult() = CKR\_OK
  - CryptokiInstance.C\_Sign(HANDLE\_NEW\_RW\_USER\_PUBLIC\_SESSION, DATA\_ONE\_PART\_NORMALLEN, BUF\_NORMAL\_LEN, ...) = CKR\_OK
    - CryptokiInstance.checkResult() = CKR\_OK
  - CryptokiInstance.tearDown()
- Point of view:** A section below the test detail showing the tags of the suite reached by the test, with the current tag '+ C\_Sign/OK' in bold. Other tags include 'C\_OpenSession/OK', 'C\_SignInit/OK', and 'GENERATE\_KEY/OK'. Below this, it shows 'CASE: CREATE\_USER\_PRIVATE\_KEY\_WITH\_SIGNVERIFY\_NOT\_SUPPORTED', 'CKR: + OK', and 'REQ: C\_Initialize, C\_Login'.
- Console:** A small window at the bottom left labeled '1 : Console'.

# Propriétés TOCL

### Properties

- C\_SignFinal\_2
- C\_SignInit\_1\_1
- C\_SignInit\_1\_2
- C\_SignInit\_1\_3
- C\_SignInit\_1\_4
- C\_SignUpdate\_1
- C\_SignUpdate\_2
- C\_SignUpdate\_2\_never
- C\_Sign\_1
- C\_Sign\_1\_bis
- C\_Sign\_2
- C\_Sign\_3\_never
- C\_VerifyFinal\_1
- C\_VerifyFinal\_2
- C\_VerifyInit\_1\_1
- C\_VerifyInit\_1\_2
- C\_VerifyInit\_1\_3
- C\_VerifyInit\_1\_4
- C\_VerifyUpdate\_1
- C\_VerifyUpdate\_2
- C\_VerifyUpdate\_2\_never
- C\_Verify\_1
- C\_Verify\_1\_bis
- C\_Verify\_2\_never
- access\_CF\_C\_Sign**
- access\_CF\_C\_Verify

### Definition

**eventually isCalled**(CryptokiInstance.C\_Login, including: {@CKR:OK})  
**at least 1 times**  
**before**  
**isCalled**(CryptokiInstance.C\_SignInit, including: {@CKR:OK})

**i** Well-formed property (4 reachable objectives expected)

### Graph

The graph shows a sequence of states. A red node with a cross is on the left. A blue node labeled '0' is in the middle, with a green self-loop and a green arrow labeled 'E0' pointing to a blue node labeled '1'. From node '1', a green arrow labeled 'E1' points to a blue node labeled '2'. Node '2' has a green self-loop. Above the nodes are event sets:  $\Sigma$  above the red node,  $\Sigma - \{E0, E1\}$  above node '0',  $\Sigma - \{E1\}$  above node '1', and  $\Sigma$  above node '2'. Transitions are labeled 'E1' between the red node and node '0', and 'E0' between node '0' and node '1'.

### Test Scenarios

- ? CryptokiInstance.C\_VerifyInit(HANDLE\_NEW\_RW\_US
- ? CryptokiInstance.C\_Verify(HANDLE\_NULL\_SESSION,
- ? CryptokiInstance.tearDown()
- ▼ **C\_SignFinal (7f-f1-73)**
  - ? CryptokiInstance.setUp()
  - ? CryptokiInstance.C\_Initialize(VOID\_NULL\_PTR) = Ck
  - ? CryptokiInstance.C\_OpenSession(SLOT\_VALID, CKF,
  - ? CryptokiInstance.C\_Login(HANDLE\_NEW\_RW\_USER,
  - ✓ CryptokiInstance.C\_SignInit(HANDLE\_NEW\_RW\_USEI
  - ✓ CryptokiInstance.C\_SignUpdate(HANDLE\_NEW\_RW\_U
  - ✓ CryptokiInstance.C\_SignFinal(HANDLE\_NEW\_RW\_US
  - ✓ CryptokiInstance.tearDown()
- ▼ ? C\_VerifyFinal (7f-b5-29)
  - ? CryptokiInstance.setUp()
  - ? CryptokiInstance.C\_Initialize(VOID\_NULL\_PTR) = Ck
  - ? CryptokiInstance.C\_OpenSession(SLOT\_VALID, CKF,
  - ? CryptokiInstance.C\_Login(HANDLE\_NEW\_RW\_USER,
  - ? CryptokiInstance.C\_VerifyInit(HANDLE\_NEW\_RW\_US
  - ? CryptokiInstance.C\_VerifyUpdate(HANDLE\_NEW\_RW\_U
  - ? CryptokiInstance.C\_VerifyFinal(HANDLE\_NEW\_RW\_U
  - ? CryptokiInstance.tearDown()
- ▼ ? C\_VerifyFinal (7f-1b-57)
  - ? CryptokiInstance.setUp()
  - ? CryptokiInstance.C\_Initialize(VOID\_NULL\_PTR) = Ck
  - ? CryptokiInstance.C\_OpenSession(SLOT\_VALID, CKF,
  - ? CryptokiInstance.C\_Login(HANDLE\_NEW\_RW\_USER,

# Configuration du test en ligne

The screenshot displays the MBeeTle v3.0.0 application window. At the top left, there are two buttons: 'Animer' (Animate) and 'Stop'. Below these is the 'Paramétrage' (Configuration) section, which includes several settings:

- Modèle:** A text field containing 'anches\pkcs11\model\_crypto\model\_crypto.smtmodel' and a browse button (...).
- Suite:** A text field containing 'PKCS11'.
- Export:** A dropdown menu set to 'Simple Exporter' with a gear icon for settings.
- Animateur:** A dropdown menu set to 'Smartesting Animator 6.3.2' with a gear icon.
- Service:** A dropdown menu set to 'UnifiedService' with a gear icon.
- Générateur:** A dropdown menu set to 'Random' with a gear icon.
- Stratégie:** A dropdown menu set to 'strategieexemple' with a gear icon, a green plus icon, and a red minus icon.

To the right of the configuration section is the 'Progression' section, which displays test statistics:

- Temps écoulé: 0
- Trace(s) valide(s): 0
- Trace(s) invalide(s): 0
- Opérations: \_\_\_\_\_
- Tag(s) couvert(s): \_\_\_\_\_

Below the statistics, there are two circular progress indicators. Each indicator is a yellow circle with a horizontal line through it, and a label 'UNREACHED 100%' is positioned to its left.

# Exécution de tests sur SoftHSM

---

- Tests hors ligne → 285 en 3 suites
  - Reprise de tests manuels de SoftHSM : **20, dont 1 non conforme à la spécification.**
  - Couverture des comportements :
    - Temps de génération 7 minutes
    - Résultats d'exécution : **158, dont 7 failed**
  - Couverture des propriétés TOCL :
    - Temps de génération 12 minutes
    - Résultats d'exécution : **107, dont 16 failed**
- Tests en ligne → test de robustesse
  - Génération à la volée avec une stratégie simple : **100 tests de 30 pas, dont 1 failed**
  - Temps de génération & exécution : **1minute 34 secondes.**

# Synthèse

---

- Couverture des comportements modélisés
  - Assure la couverture des **exigences fonctionnelles**
- Propriétés temporelles
  - **Expression** des exigences de sécurité sans ajouter l'information superflue dans le modèle
  - Montrent l'insuffisance de **couverture** des exigences de sécurité par des tests fonctionnels (environ 35% de propriétés TOCL ne sont pas couverts par ces tests).
- Test en ligne
  - Mise en oeuvre avec un **coût minimal** (réutilisation du modèle et de l'harnais de test)
- **Complémentarité** des approches en ligne et hors ligne et du pilotage de la génération.
- **Détection de bugs** de type différent.