femto-st SCIENCES & TECHNOLOGIES

Smartesting

Technologie Smartesting CertifyIt

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

Bruno Legeard

Séminaire Test & Méthodes formelles – LAAS-CNRS – Toulouse – 16 juin 2015

Agenda

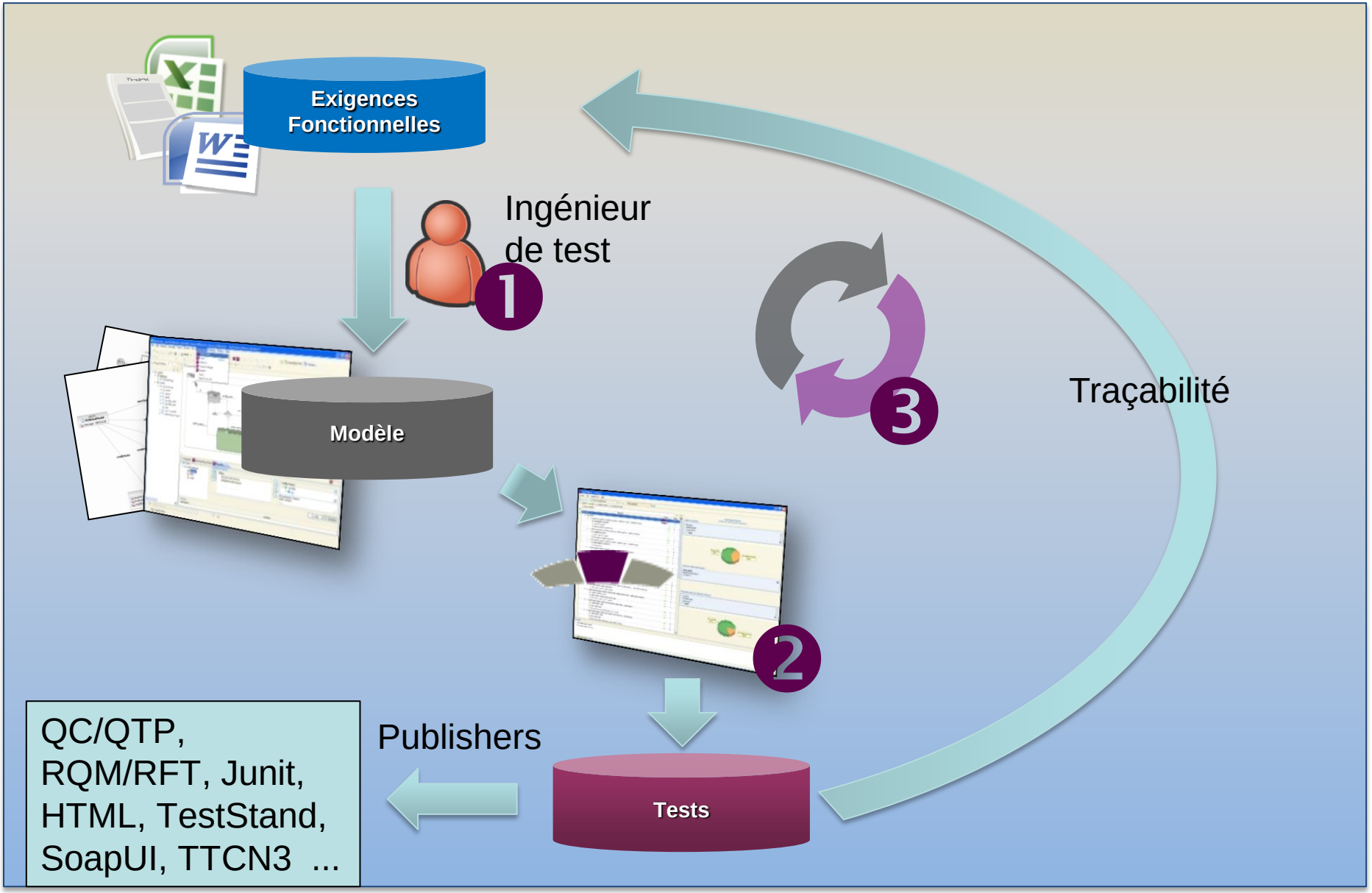
① **Technologie CertifyIt: du modèle aux tests**

② **Pilotage de la génération: complémentarité des approches**

③ **Exemple : Composant de sécurité**

④ **Conclusion**

Smartesting CertifyIt



Modèle de test

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 :

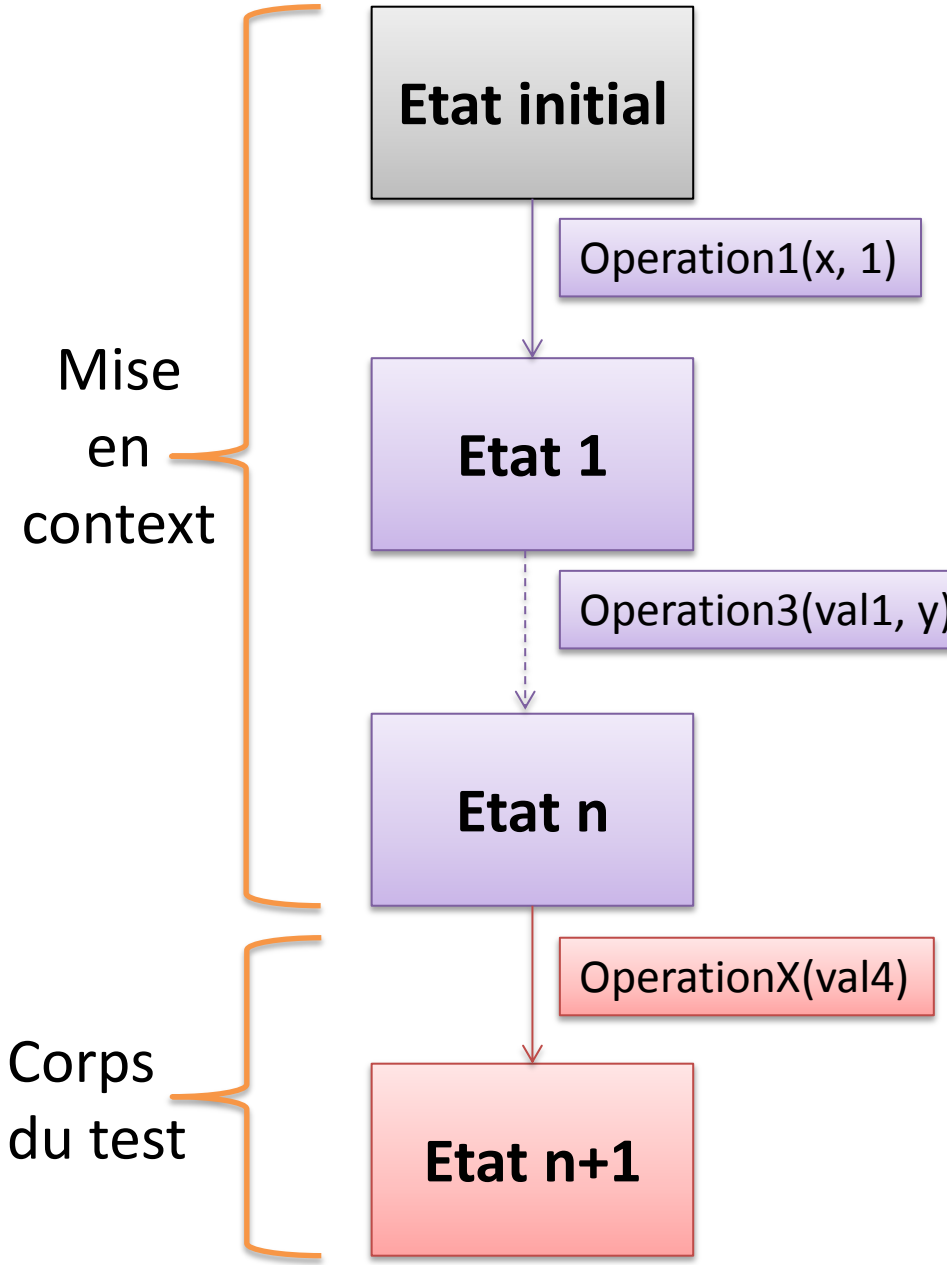
→ *représentation des comportements attendus du système sous test, porté par les opérations présentes dans le diagramme de classes*

Configuration de l'état initial :

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

Génération 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*

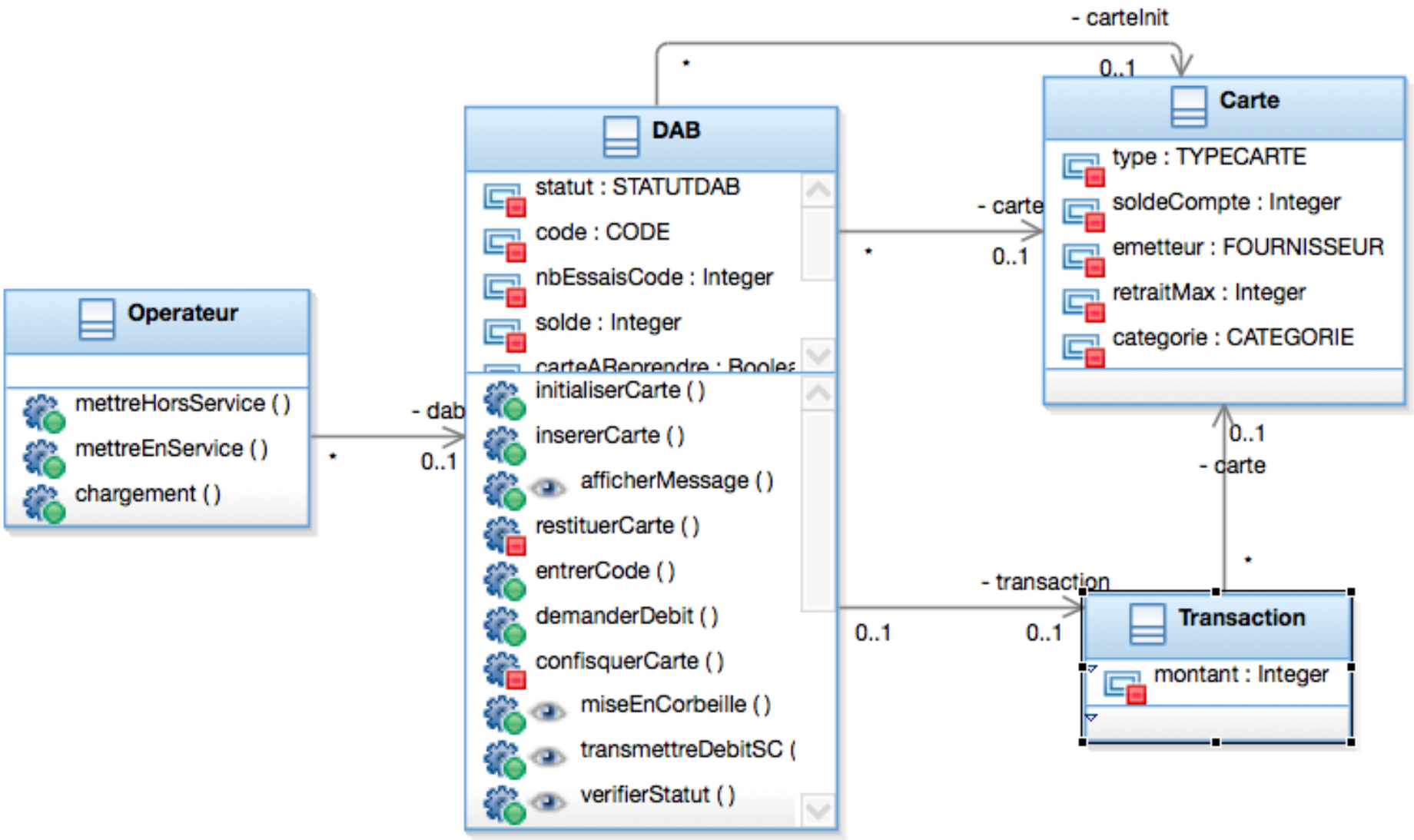
Langage d'actions :

→ *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*

Exemple DAB : OCL, opération enterCode

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
2  if (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
15 else
16     code = CODE::OK and
17     ---@REQ: INFO002-Signaler au client de saisir sa somme
18     afficherMessage(MESSAGE::ENTRER_MONTANT)
19 endif
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*

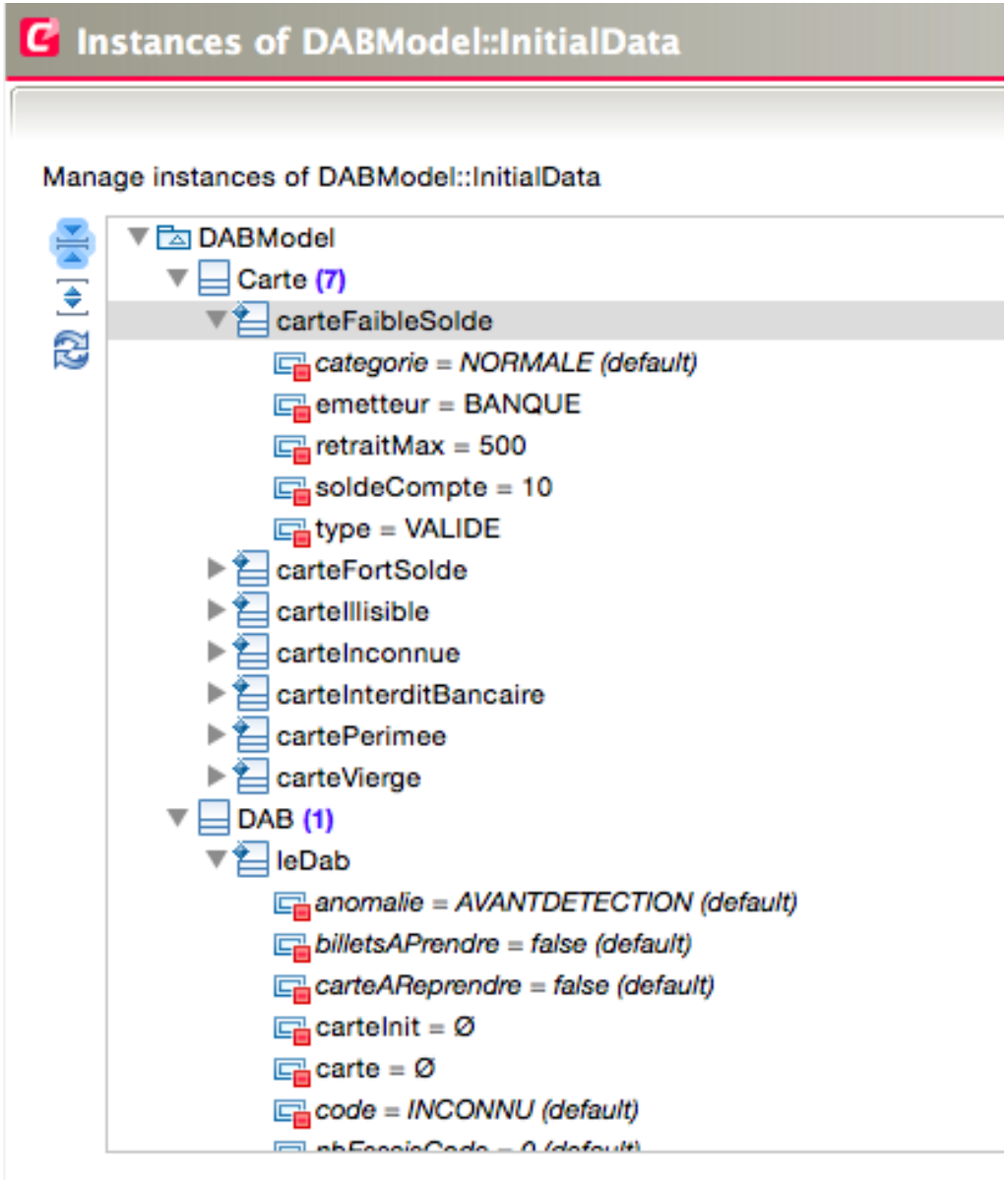
Langage d'actions :

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

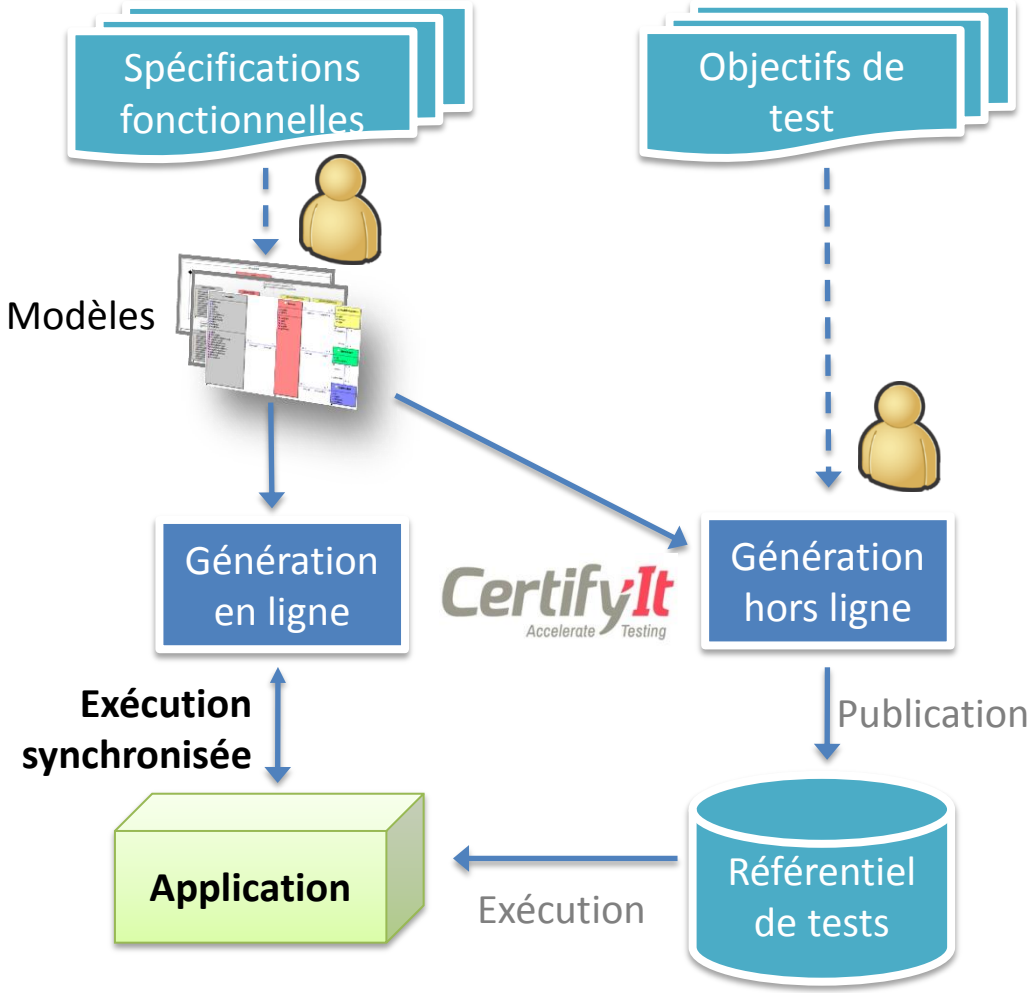
Configuration de l'état initial :

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

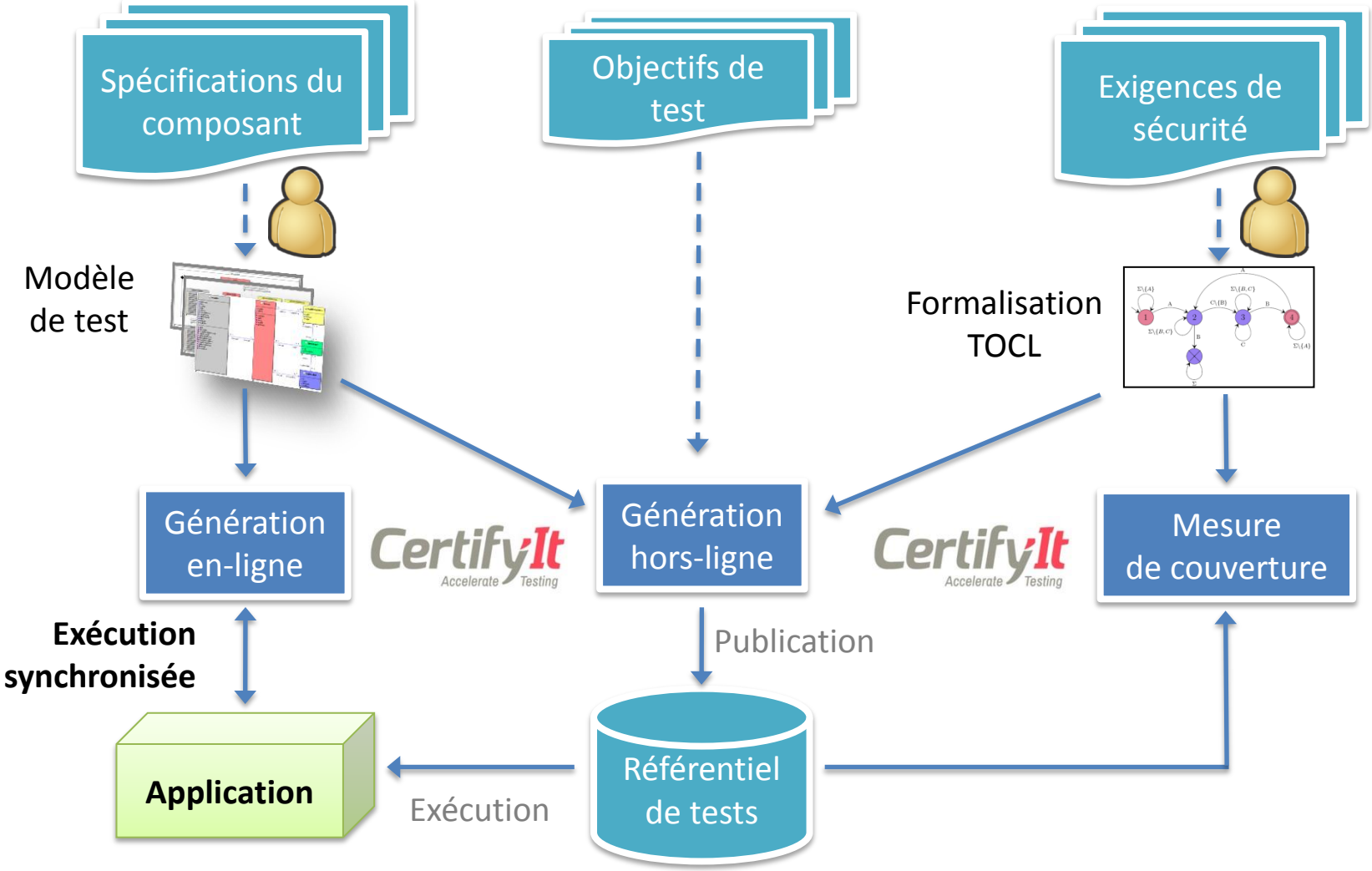
Exemple DAB : Etat initial



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



Couverture fonctionnelle + sureté & sécurité

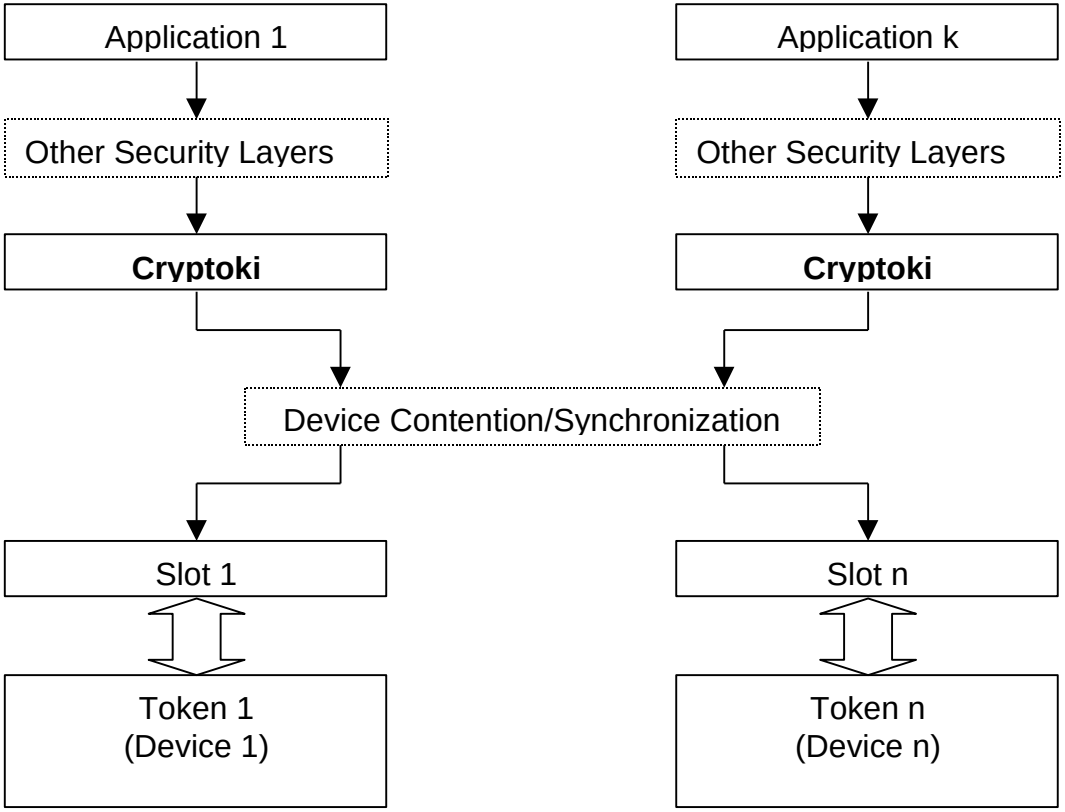


Cas d’étude - PKCS#11

- 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.

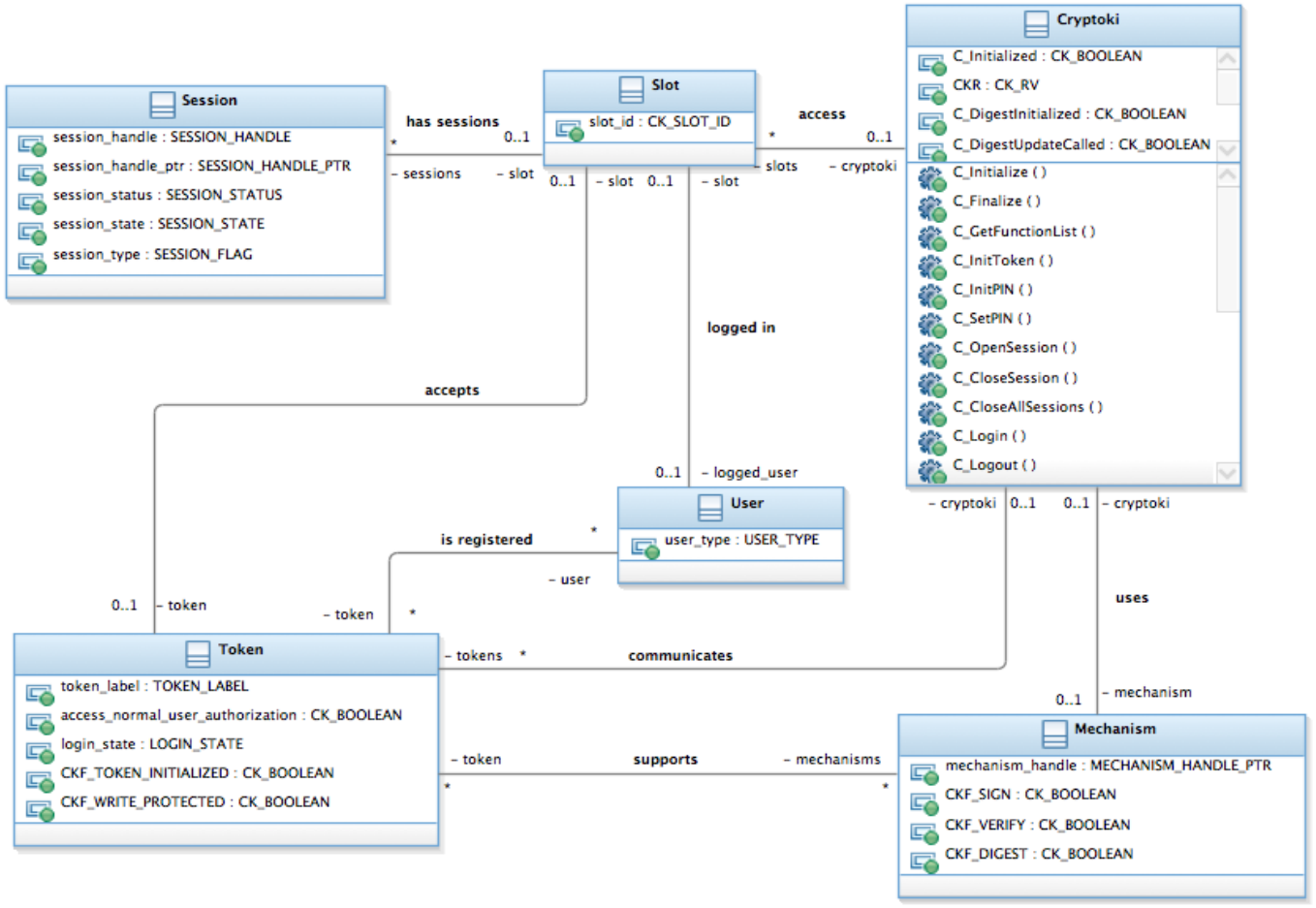
Modèle général de Cryptoki



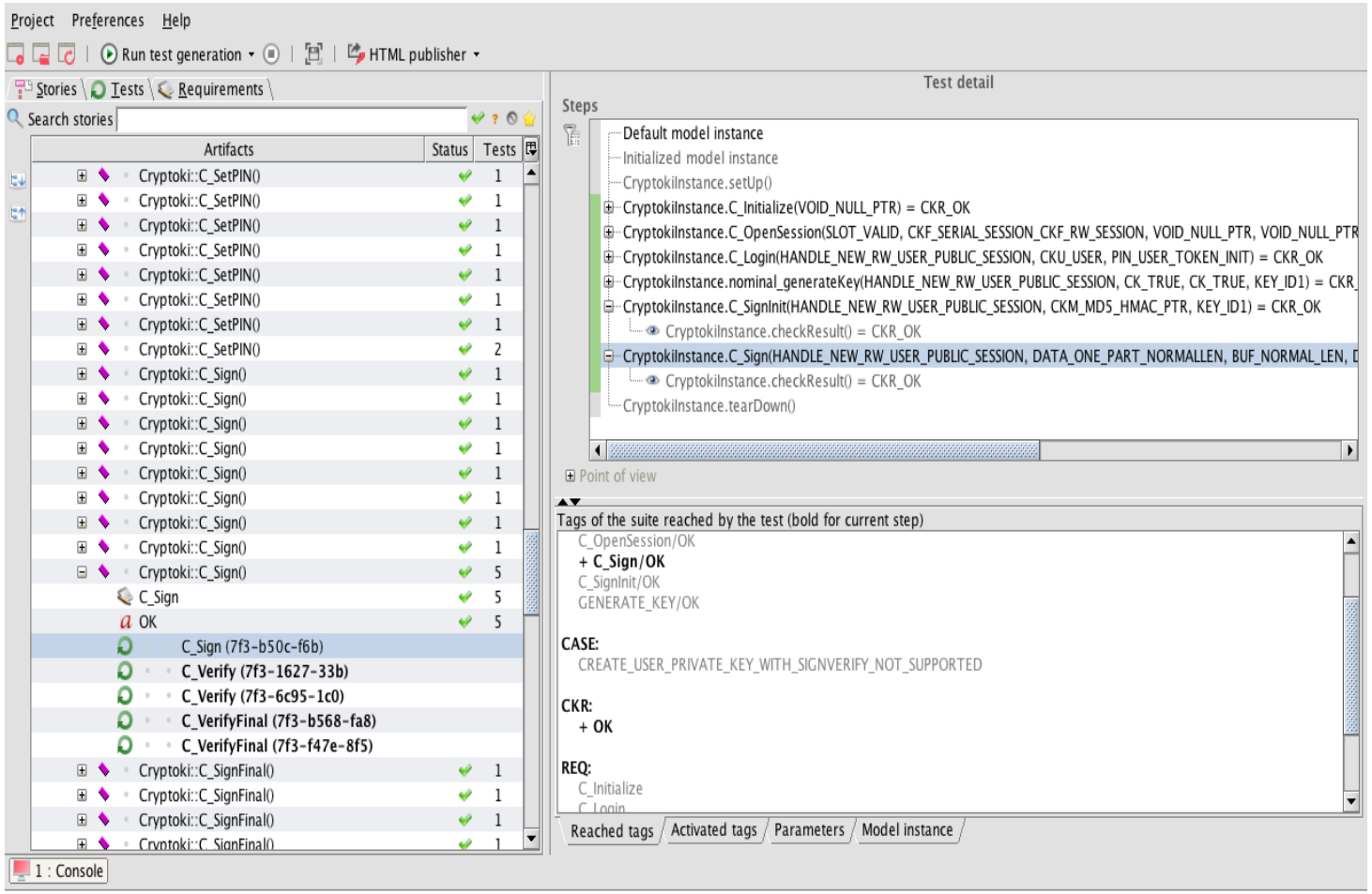
Périmètre PKCS#11

- 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

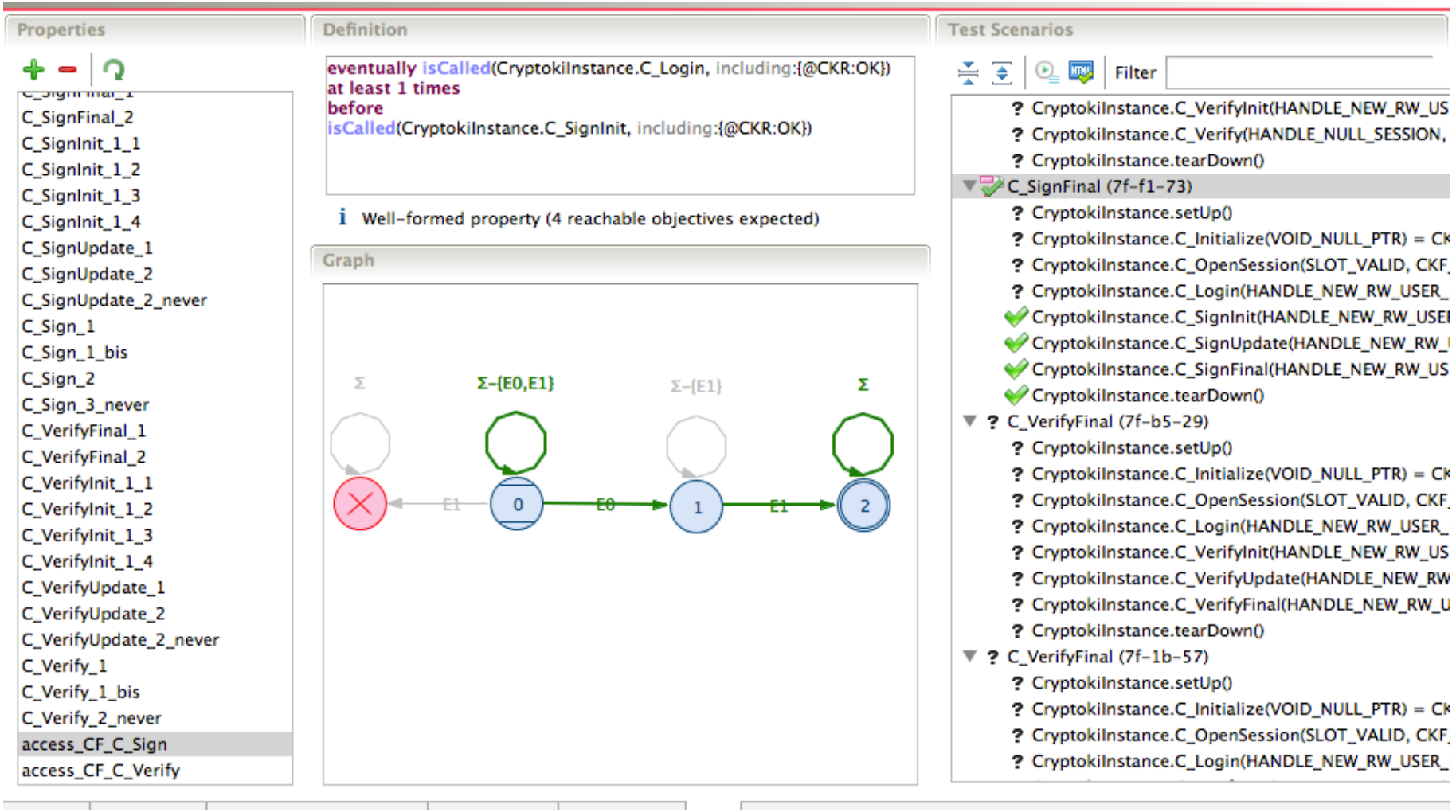
Modèle PKCS#11



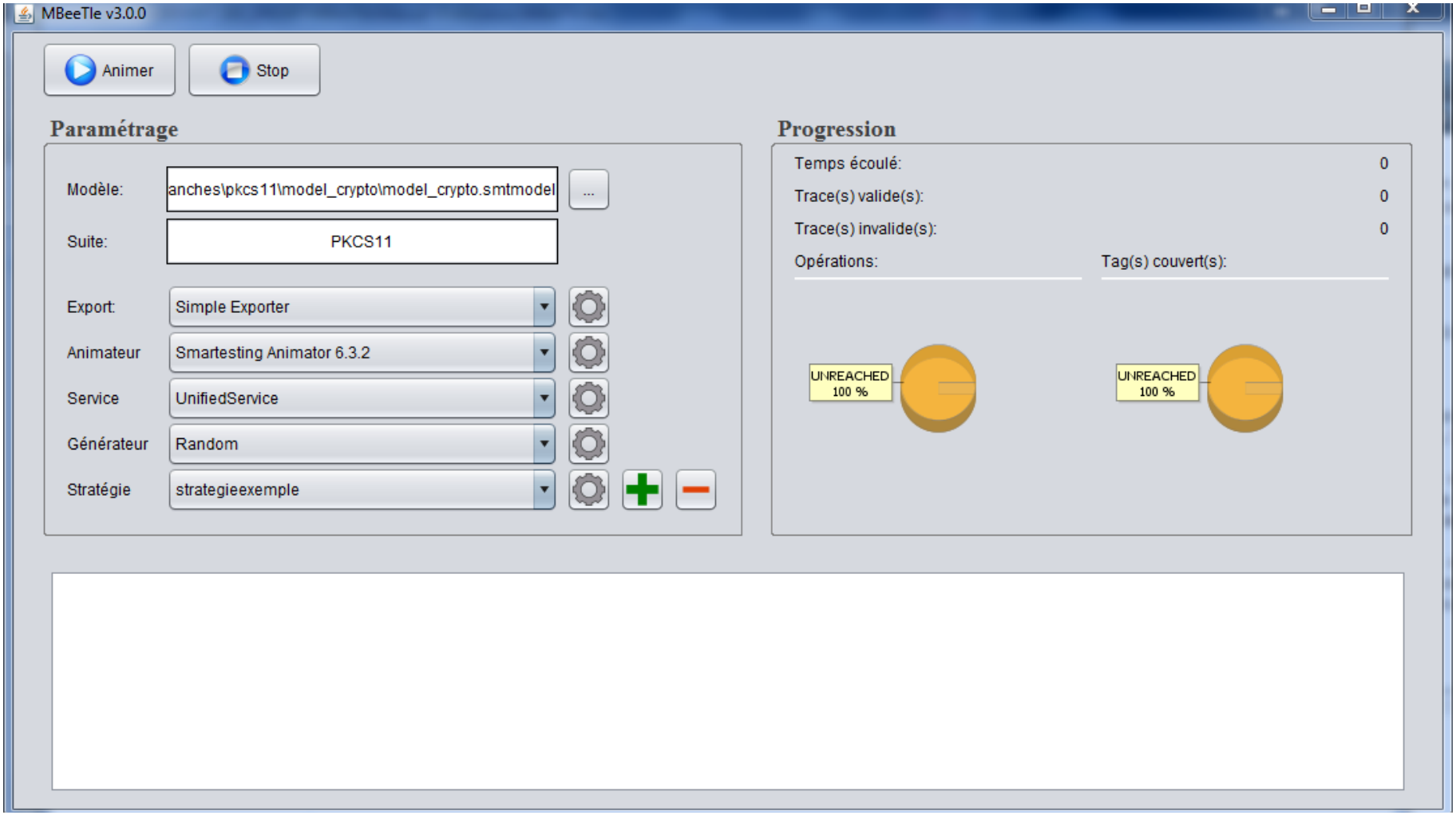
CertifyIt


Project
Preferences
Help
Run test generation
HTML publisher
Stories
Tests
Requirements
Search stories
Artifacts
Status
Tests
Cryptoki::C_SetPIN()
Cryptoki::C_Sign()
C_Sign
OK
C_Sign (7f3-b50c-f6b)
C_Verify (7f3-1627-33b)
C_Verify (7f3-6c95-1c0)
C_VerifyFinal (7f3-b568-fa8)
C_VerifyFinal (7f3-f47e-8f5)
Cryptoki::C_SignFinal()
1 : Console
Test detail
Steps
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
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_
CryptokiInstance.C_SignInit(HANDLE_NEW_RW_USER_PUBLIC_SESSION, CKM_MD5_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,
CryptokiInstance.checkResult() = CKR_OK
CryptokiInstance.tearDown()
Point of view
Tags of the suite reached by the test (bold for current step)
C_OpenSession/OK
+ C_Sign/OK
C_SignInit/OK
GENERATE_KEY/OK
CASE:
CREATE_USER_PRIVATE_KEY_WITH_SIGNVERIFY_NOT_SUPPORTED
CKR:
+ OK
REQ:
C_Initialize
Reached tags
Activated tags
Parameters
Model instance

Propriétés TOCL



Configuration du test en ligne



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 ouvre 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.