Rapport GL

Systeme d'identification

Etude de cas

9 janvier 2007

Olivio Mariano - olivio.mariano@gmail.com

Table des matières

Introduction

On présente ici une étude de cas d'un système d'identification de pièces. En utilisant le langage B et la méthode B on s'intéressera aux spécifications de cette machine abstraite à l'aide d'un formalisme mathématique de haut niveau. On décrira les données et les traitements basiques pour que la machine soit prouvable et qu'elle puisse aboutir à un modèle concret.

L'utilisation de preuves formelles nous permettra de vérifier la cohérence du modèle abstrait et la conformité de l'ensemble des implantations concrètes avec le modèle abstrait.

Chapitre 1

Choix de modélisation

Nous présenterons tout d'abord les principales fonctionnalités de ce projet. Nous justifierons ensuite l'intérêt de l'Étude Système que nous allons entreprendre. Puis nous mènerons le développement jusqu'à son terme, c'est-à-dire jusqu'à l'obtention d'une architecture complète.

But du système :
On souhaite construire un système d'identification de pièces chargé de contrôler et gérer les différentes composantes pour la vérification du poids et de la couleur des pièces insérés dans le système.

1.1 Composantes

La machine en question est composée par :

- une balance - permettant de peser les pièces
- un scanner - pouvant déterminer la couleur des pièces
- deux pinces - permettant de charger et décharger, respectivement, balance et scanner
- une benne - ou les pièces non identifiées sont amenés
- une sortie - ou les pièces identifiés sont amenés

Le contrôle s'effectue sur la base de deux composantes principales : scanner et balance. La phase d'identification amène donc nécessairement à l'utilisation de la balance et en option à celle du scanner. Deux pinces sont modélisés aussi dans le système pour pouvoir assurer les mouvements des pièces dans le système.

1.2 Etats

Chaque composant "mobile" du système possède un état pour interagir correctement avec les autres "acteurs" de la machine. Une pièce peut posséder différents états selon la phase d'identification passée. Une pince peut posséder différents états selon sa positon dans le système. Cette choix de modélisation a permis d'écrire la machine en affectant à chaque opérations des préconditions sur la base des états décrits.

1.3 Répartition du contrôle

Une des principales questions qui se posent pour ce projet concerne la distribution, plus ou moins importante, des opérations dans une telle machine. Les diagrammes *statecharts* et d'action en fig 1.1 et fig 1.2 montrent la répartition des opérations.

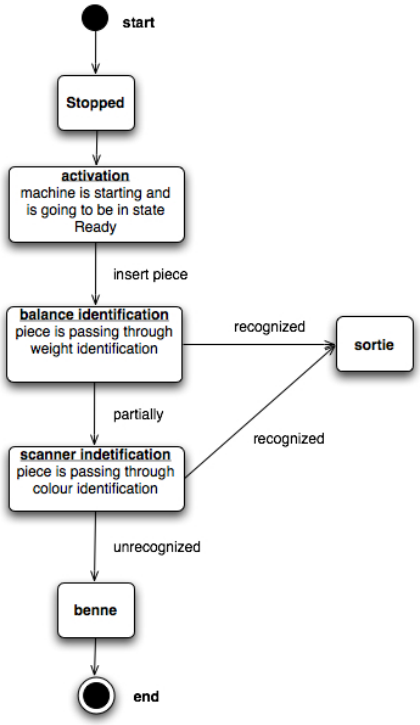

FIG. 1.1 – Action chart diagram

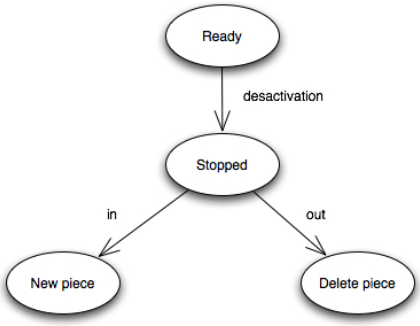

FIG. 1.2 – State chart diagram pour les opération d'insertion d'une nouvelle piece et elimination

Comme expliqué par les diagrammes, il existe, des situations et des opérations "intermédiaires" qui s'occupent uniquement de activer ou désactiver la machine. Cela parce-que il est précognisé par le système la possibilité de mémoriser de nouveaux types de pièces à identifier ainsi que d'en supprimer ; dans ce cas le système doit forcement être inactif.

1.4 Procédure systématique

Obtenir un fonctionnement correcte de la machine d'identification obéit à une procédure systématique composée d'une suite d'événements :

- La machine est activée
- Une pièce pas encore indentifiée est insérée dans la machine.
- La pièce, dans l'état "in" et "unidentified", est sur la balance.
- La balance, par rapport à son database, change l'état de la piéces selon son poids en "identified" ou "partially". Si la balance a reconnu la pièce elle sera amené en sortie sinon au scanner car identifiée partiellement. La pince 1 effectuera les déplacements nécessaires.
- Le scanner prendra en entrée uniquement les pièces en état "partially" et selon la couleur de ces dernières les placera en sortie ou dans la benne. La pince 2 effectuera les déplacements nécessaires.

Il est important donc de bien spécifier de quelle manière les états de la machine sont gérés. Les opérations vont être déclenchées suivant les états et les états de la machine varieront selon l'opération effectuée.

1.5 Conception d'un modèle fermé

L'argumentation technique qui peut conduire à une décision spécifique dans la modélisation du système ne peut s'effectuer qu'en analysant le système dans son ensemble. L'Étude Système va donc consister essentiellement à construire un modèle fermé de notre futur système et à prouver que les principales propriétés caractéristiques de ce système (qu'il va donc falloir expliciter avec précision) sont bien assurées.

1.6 Comportement matériel

Le résultat de l'Etude Système concerne les spécifications comportementales du système d'identification de pieces. Pour bien mener cette étude, il est donc important d'introduire dans le modèle un certain nombre d'hypothèses concernant la réponse de notre machine aux différentes cas que la machine peut rencontrer à l'insertion d'une pièce. Cette option constitue une hypothèse sous laquelle le modèle du logiciel peut être prouvé et peut fonctionner correctement.

À l'issue de l'Étude Système, nous aurons ainsi effectué un certain nombre de choix de comportement du matériel. Ces choix devront être consignés dans un Cahier des Charges Matériel comportant, entre autres, une procédure de recette visant à vérifier que le matériel effectivement mis en place possède bien les propriétés que l'on attend de lui. Si ce n'était pas le cas, le mariage du logiciel et du matériel n'aurait aucune chance d'aboutir à un système fonctionnant correctement comme cela aurait été démontré sur le modèle (sous les hypothèses concernées).

Le test du comportement matériel de la machine sera effectué à l'aide de l'outil d'animation *LEIRIOS Test Generator* (LTG) .

Chapitre 2

Le code

2.1 Sets, Constants et Variables

```
SETS
PIECES = {eu1, eu2, cent5, cent10, cent20, cent50};
COULEUR = {rouge, vert, jaune, violet, blanc, noir};
ETAT_PIN1 = {etat_benne1, etat_scanner1, etat_balance1, etat_sortie1};
ETAT_PIN2 = {etat_scanner2, etat_sortie2, etat_benne2};
MESSAGES = {ok,ko, not_recognized};
ETAT_MACHINE = {ready, stopped};
ETAT_PIECE = {recognized, unrecognized, partially};
ETAT_PIECE2 = {in, out}

CONSTANTS
MAX_POIDS,
POIDS

PROPERTIES
MAX_POIDS = 10 &
POIDS = 1..10

VARIABLES
pince1, pince2, balance2, benne2, sortie2, scanner2, colore, couleur,
peso, piece, etat_pince1, etat_pince2, state, etat_pezzo1, pezzo, etat_pezzo2
```

- Les SETS sont des ensembles qui n'ont aucun élément en commun. On les appelle aussi « ensembles de base ». Ce sont des ensembles finis et non vides. On construit des ensembles à l'aide du produit cartésien et de l'opérateur ensemble des parties. Ici, les sets `PIECES, COULEUR, ETAT_PIN1, ETAT_PIN2, MESSAGES, ETAT_MACHINE, ETAT_PIECE , ETAT_PIECE2` sont définis en tant qu'ensembles.
- Les parties CONSTANTS et PROPERTIES modélisent le poids maximum des pièces traités. Dans CONSTANTS on défini les constantes et dans PROPERTIES on défini les contraintes sur ces constantes.

2.2 Invariant et Initialisation

```
INVARIANT
pince1 <:  PIECES &
pince2 <: PIECES &
balance2 <: PIECES &
benne2 <: PIECES &
sortie2 <: PIECES &
scanner2 <: PIECES &
piece <: PIECES &
etat_pince1 <: ETAT_PIN1 &
etat_pince2 <: ETAT_PIN2 &
state <: ETAT_MACHINE &
couleur <: COULEUR &
peso <:  POIDS &
colore : piece --> couleur &
pezzo :  piece --> peso &
etat_pezzo :  piece -->  ETAT_PIECE &
etat_pezzo2 : piece --> ETAT_PIECE2


INITIALISATION
couleur := {rouge, vert, jaune, blanc, violet, noir} ||
piece := {eu1, eu2, cent5, cent10, cent20, cent50} ||
peso := 1..10 ||
colore := {eu1 |-> rouge, eu2 |-> vert, cent5 |-> jaune,
cent10 |-> blanc, cent20 |-> violet, cent50 |-> noir} ||
pezzo := {eu1 |->  2, eu2 |-> 1, cent50 |-> 3, cent20 |-> 4,
cent10 |-> 5, cent5 |-> 3} ||
etat_pezzo2 := {eu1 |-> out, eu2 |-> out, cent50 |-> out,
cent20 |-> out, cent10 |-> out, cent5 |-> out} ||
etat_pezzo := {eu1 |-> unrecognized, eu2 |-> unrecognized,
cent50 |-> unrecognized, cent20 |-> unrecognized,
cent10 |-> unrecognized, cent5 |-> unrecognized} ||
etat_pince1 := {etat_balance1} ||
etat_pince2 := {etat_scanner2} ||
state := {stopped} ||
balance2 := {} ||
benne2 := {} ||
sortie2 := {} ||
scanner2 := {} ||
pince1 := {} ||
pince2 := {}
```

Les propriétés, devant être assurées en permanence par le système, sont modélisées dans les invariants B. L' invariant est un prédicat. L' invariant d'un composant B est un ensemble de propriétés qui doivent être respectées en permanence. Sachant que l' invariant dépend des variables d'état du composant B, et que seules les initialisations et les opérations du composant

peuvent modifier ces variables, il suffit de montrer que l'invariant est établi par l' initialisation, et préservé par chaque opération.

2.3 Opérations

2.3.1 Activation, desactivation, insertion

Trois opérations fondamentales sont modélisés pour activer, désactiver et accueillir une pièce dans la machine.

Pour une activation correcte on exige que la machine soit en état de stop et que la quantité des pièces insérée dans la machine soit à 0. La machine est alors mise en état de fonctionnement "ready".

```
msg <-- machine_active =
PRE
state = {stopped} /\
card (etat_pezzo |> {in} ) = 0
THEN
state := {ready} ||
msg := {ok}
END;
```

Dans l'opération de desactivation, à l'image de l'activation, on exige que la machine soit en état de marche "ready".

```
msg <-- machine_stopped =
       PRE
       state = {ready}
       THEN
       state:= {stopped} ||
       msg:= {ok}
       END;
```

Dans l'opération d'insertion il est nécessaire que la machine soit en état de marche et que la pièce passée en paramètre (piece1) appartienne à l'ensemble de pièces modélisé. Bien entendu, la pièce passée en paramètre de l'opération ne doit pas appartenir à l'ensemble balance modélisant l'endroit où les pièces vont être amenés. Avec ces conditions, l'état de la pièce est mis à "in" et piece1 est placé dans l'ensemble balance2.

```
msg <-- inserer_piece(piece1) =
       PRE
       piece1: PIECES &
       piece1 : piece &
       card(etat_pezzo2 |> {in}) : 0..1 &
       state = {ready} &
       piece1 /: balance2
       THEN
```

```
etat_pezzo2 (piece1):= in ||
balance2:= balance2 \/ {piece1} ||
msg:= {ok}
END;
```

2.3.2 Identifications

On modélise dans le système deux opérations d'identification : balance et scanner.

L'opération identifierbalance se charge de l'identification de la pièce quand cette dernière est sur la balance. Pour chaque pièce sur la balance dans l'état "in" :

```
IF card(dom(pezzo |> {poids})) > 1
THEN
etat_pezzo(piece1) := partially ||
etat_pince1 := {etat_balance1} ||
msg := {ko}
ELSE
IF pezzo(piece1) = poids
THEN
etat_pezzo(piece1) := recognized ||
etat_pince1 := {etat_balance1} ||
msg := {ok}
ELSE
etat_pezzo(piece1) := unrecognized ||
etat_pince1 := {etat_balance1} ||
msg := {not_recognized}
```

On vérifie la présence de son poids dans la base de données et on change d'état à la pièce à réponse positive. Si par contre il existe une pièce que a le même poids de la pièce passée en paramètre on change d'état de la pièce en "partially" pour qu'elle soit passée au scanner.

En tous cas la piece1 de cette opération sortira du traitement avec trois possibles états : recognized, partially ou unrecognized. Dans le deuxième cas, la mains est passée au scanner qui, à l'image de l'opération d'identification sur la balance effectuera le traitement sur l'ensemble de couleurs.

2.3.3 Les pinces

Pour chaque pince on a deux types d'opération modélisée.

Une opération qui s'occupe de charger la balance selon l'état de cette dernière quand la pince (transfertpince1 ou transfertpince2) est sur la balance ou sur le scanner. On modélise les pinces pour éviter toutes collisions entre elles. Aprés avoir chargée la pince on la déplace pour qu'elle soit prête a à être déchargée.

```
transfertpince1(piece1) =
PRE
state = {ready} &
piece1 : piece &
piece1 /: (pince1)&
etat_pince1 = {etat_balance1} &
```

```
card(etat_pezzo2 |> {in}) = 1
THEN
IF
etat_pezzo(piece1) =  recognized
THEN
pince1:= pince1 \/ {piece1}||
balance2 := balance2 -{piece1} ||
etat_pince1:= {etat_sortie1} ||
etat_pince2:= {etat_benne2}
ELSE
IF etat_pezzo(piece1) = partially
THEN
pince1:= pince1 \/ {piece1} ||
balance2 := balance2 -{piece1} ||
etat_pince1:= {etat_scanner1} ||
etat_pince2:= {etat_benne2}
ELSE
pince1:= pince1 \/ {piece1} ||
balance2 := balance2 - {piece1} ||
etat_pince1:= {etat_benne1} ||
etat_pince2:= {etat_scanner2}
END
END
END;
```

Pour décharger les pinces on modélise trois opérations pour la pince numéro 1 ; pince1 vers scanner, pince1 vers sortie, pince1 vers benne. Pour la pince numéro 2 on modélise deux opérations vue l'impossibilité de cette dernière à effectuer des opérations de déchargement vers la balance ; pince2 vers sortie, pince2 vers benne. Ces 5 opérations sont conçue selon ce modèle systématique :

```
pince1scanner(piece1) =
PRE
state = {ready} &
piece1 : piece &
piece1 /: (scanner2) &
piece1 : dom(etat_pezzo |> {partially}) &
etat_pince1 = {etat_scanner1} &
card(etat_pezzo2 |> {in}) = 1
THEN
pince1 := pince1 - {piece1} ||
scanner2 := scanner2 \/ {piece1}
END;
```

2.3.4 Ajout et suppression d'une pièce

Pour l'ajout d'une nouvelle pièce on met en état de stop la machine et on ajoute une couple associant la nouvelle pièce à un poids, à une couleur et à un état.

```
msg <-- nouvelle_piece(pp,cc,po,et,et2) =
PRE
pp : PIECES &
cc : couleur  &
po : peso &
et : ETAT_PIECE &
et2 : ETAT_PIECE2 &
state = {stopped} &
card(etat_pezzo2 |> {in}) : 0..1
THEN
piece:= piece \/ {pp} ||
pezzo:= pezzo <+ {pp |-> po} ||
colore:= colore <+ {pp |-> cc} ||
etat_pezzo:= etat_pezzo <+ {pp |-> unrecognized } ||
etat_pezzo2 := etat_pezzo2 <+ {pp |-> out } ||
msg:= {ok}
END;
```

L'opération inverse supprime toutes les couples associés à la pièce à supprimer.

```
msg <-- supprimer_piece(pp) =
PRE
pp: PIECES &
state = {stopped} &
card(etat_pezzo2 |> {in}) : 0..1
THEN
piece:= piece -{pp} ||
pezzo:= {pp} <<| pezzo ||
colore:= {pp}<<| colore ||
etat_pezzo := {pp} <<| etat_pezzo ||
etat_pezzo2 := {pp} <<| etat_pezzo2 ||
msg := {ok}
END
```

Chapitre 3

Preuve

La cohérence des modèles aux différentes étapes et la conformité du programme au modèle initial, sont garanties par des preuves mathématiques. A l'ultime étape, les programmes réalisés sont corrects par construction.

Pour vérifier que la machine soit correcte, plusieurs étapes sont nécessaires :

– La machine est d'abord testée avec les outils fournis par l' UFR : la balbulette ou JEdit avec le plugin B.jar. On effectue un typecheck et on génère les *proof obligations* (boutons [ty] et [po]). La balbulette nous indiquera la présence des obligations de preuve.
Les éventuelles erreurs de compilation sont vérifiées :

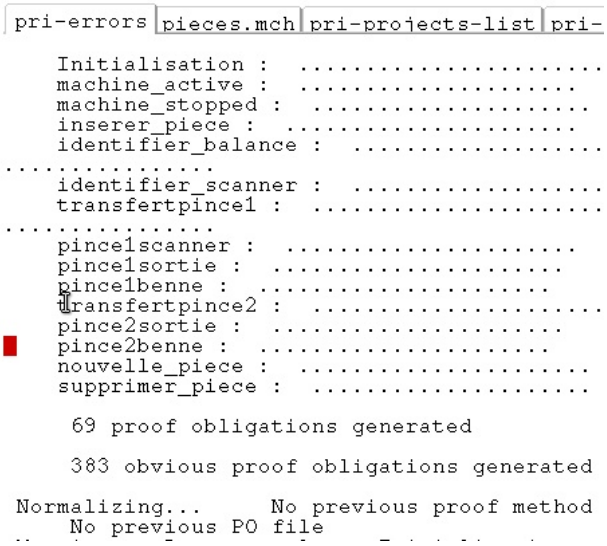

FIG. 3.1 – La compilation de la machine

– Lancement des prouveurs automatiques : [p0] et [p1] . Ces outils vont tenter de prouver le plus possible. La preuve s'effectue d'abord avec
`[p0]` .
Dans notre machine le résultat avec cette première preuve est montré en figure 3.2.
On observe la présence de 2 preuves non effectuées. [p0] a parfois du mal parce qu'il ne cherche pas assez d'informations. Pour le faire marcher correctemment il est souvent utile de lui donner le typage des variables (des fonctions par exemple). Si la preuve avec [p0] montre encore des "-" il sera nécessaire de passer à l'utilisation de [p1] ou [ip].
– Avec
`[ip]`

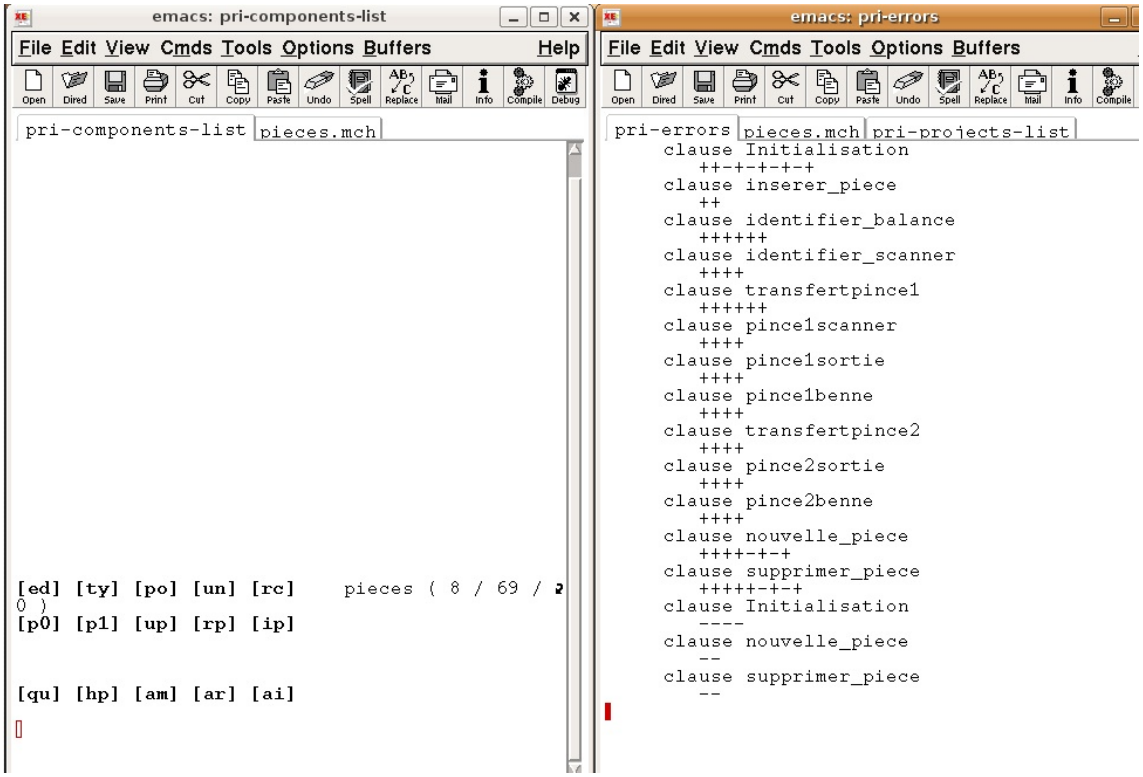

FIG. 3.2 – La preuve effectuée avec [p0]

on pourra prouver de manière plus efficace la machine car cet outil cherchera automatiquement une hypothèse si besoin est. On montre le résultat de la preuve en Fig 3.3.

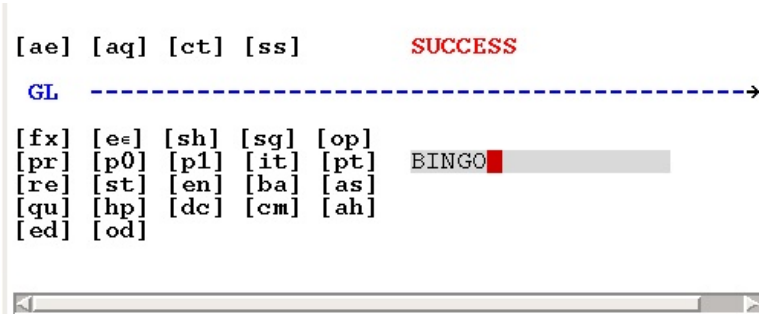

FIG. 3.3 – La preuve effectuée avec [ip]

Chapitre 4

Génération de tests

4.1 Tests

L'outil *LEIRIOS Test Generator* (LTG) permet la génération automatique de tests à partir de modèles écrits en notations B, UML/OCL ou Statecharts. La génération de cas de tests fonctionnels à partir du modèle permet :

- de consolider les spécifications, en déterminant au plus tôt les anomalies/incohérences,
- de construire un modèle de haut niveau : facilitant sa modification et sa mise à jour ultérieurement
- de pouvoir régénérer des cas de tests de façon incrémentale.

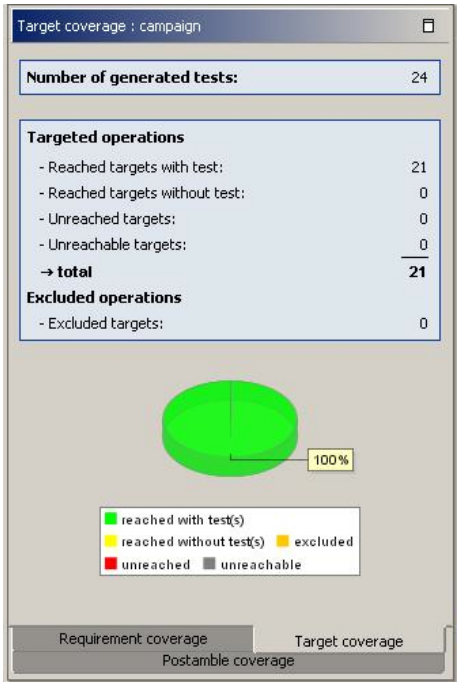

FIG. 4.1 – Le test effectué avec LTG

La notation par machine abstraite B est particulièrement adéquate pour réaliser des modèles formels de haut niveau : des normes, applications et OS carte à puce. Ces modèles permettent de formaliser et de vérifier les propriétés de sécurité et possèdent toutes les bonnes propriétés pour la génération automatique des tests de validation.

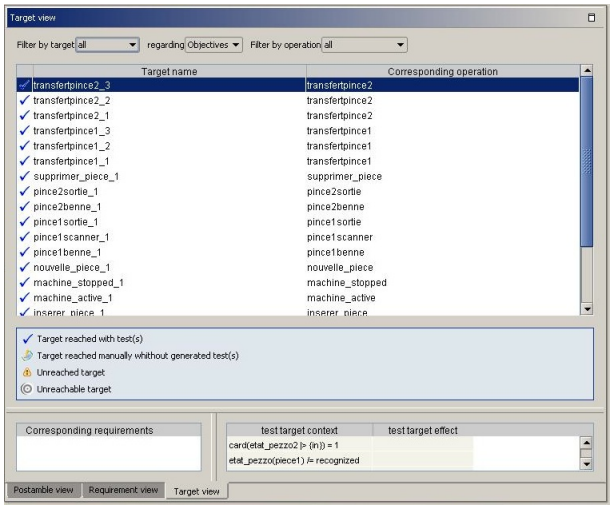

FIG. 4.2 – La liste des operations testées

Comme montré en figure 4.1 et 4.2 le test effectué sur notre machine spécifique a montré la validité de notre système en affichant un graphe de validation de test complètement vert. Les opérations et les objectifs (*targets*) de la machine ont étés vérifiés donc *reached*.

Le système de gestion d'identification a été validé par cette génération automatique de tests. Dans le cycle de développement du système logiciel, la validation a été une étape fondamentale qui a representé presque 60 % du coût global en terme de temps. Au coeur des activités de validation ont figuré la conception des test fonctionnels. Ces test ont permis de garantir la conformité de notre application par rapport aux spécifications initiales.

4.2 Exemple d'utilisation

Un exemple typique d'utilisation de la machine :
- Activation de la machine (machineactive)
- Insertion d'une pièce (insererpiece)
- Identification sur la balance (identifierbalance)
- Chargement de la pince 1 (transfertpince1)
- Déchargement de la pince 1 vers le scanner vu l'état "partially" de la pièce identifiée par la balance. (pince1scanner)
- identifierscanner (on selectionnera la couleur noir pour la piece cent50)
- transfertpince2
- pince2sortie (l'operation sera possible car le système a bien identifiée une pièce de poids "3" et de couleur "noir".

4.3 Interface d'utilisation

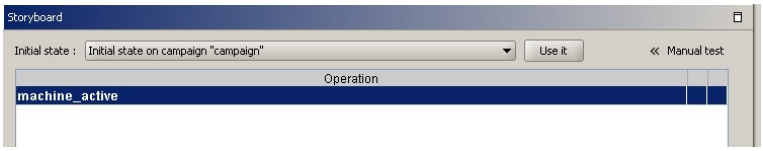

FIG. 4.3 – Activation de la machine

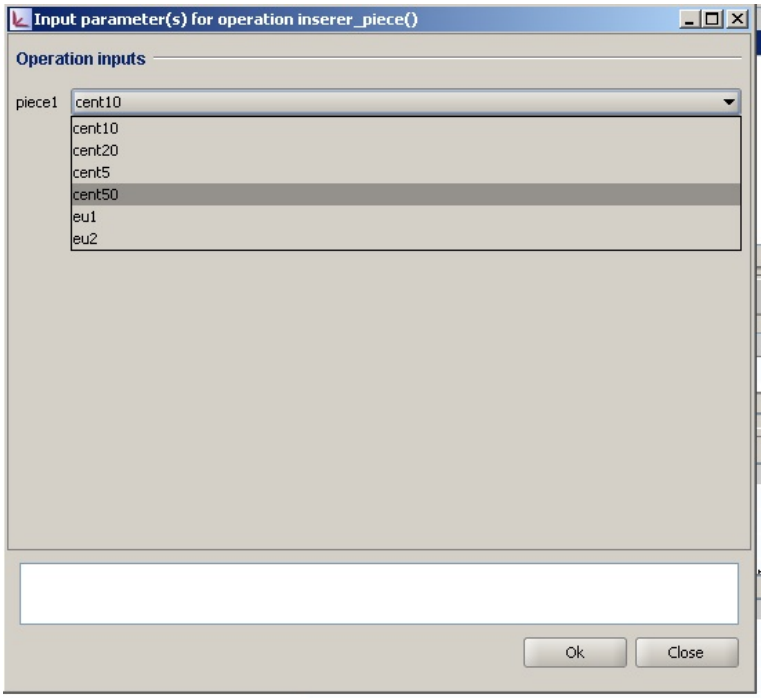

FIG. 4.4 – Opération d'insertion d'une pièce

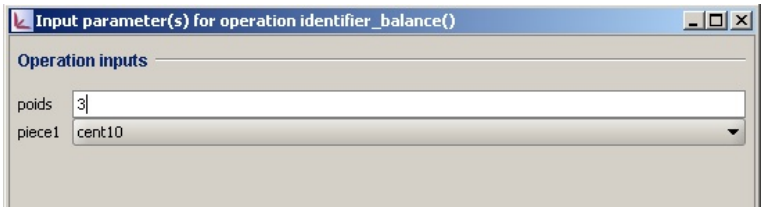

FIG. 4.5 – Identification sur la balance ; dans ce cas pour le cent 50 on met le poids 3. Le même poids que la piece cent 5 ; il sera nécessaire de passer au scanner.

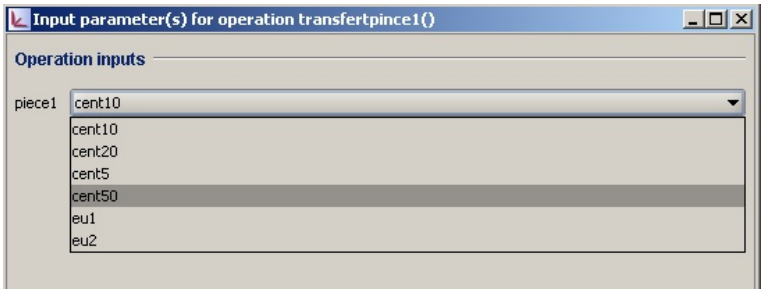

FIG. 4.6 – Chargement de la pince 1

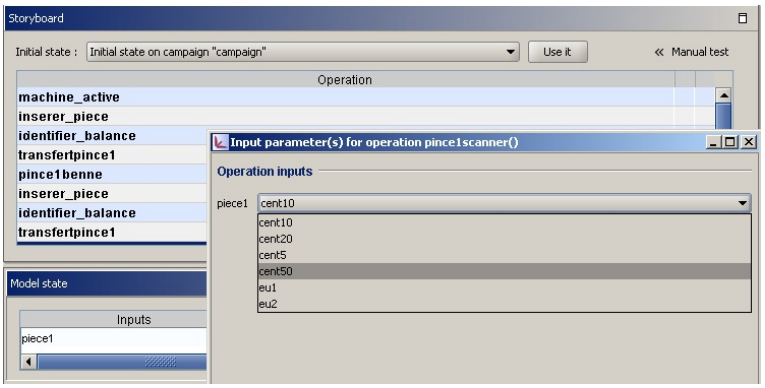

FIG. 4.7 – Déchargement de la pince 1 vers le scanner.

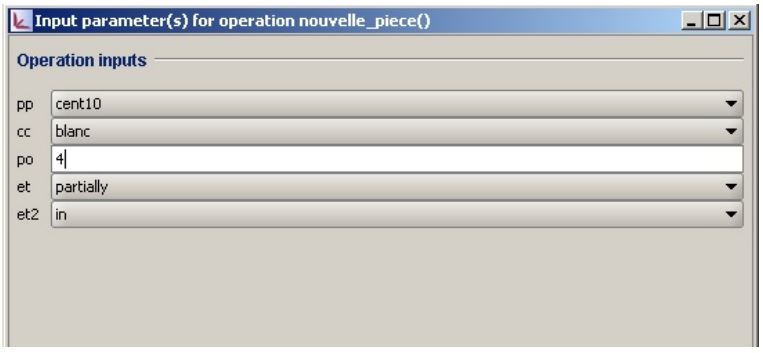

FIG. 4.8 – Exemple de création d'une nouvelle pièce, après avoir désactivé la machine (machinestopped) :