Rapport de Labo — ANSYS SCADE Suite

Contrôleur portes/passerelle, Q1–Q7

Étudiant : Oliver Fundu
Cours : Vérification de modèles (INFOM471)
Encadrant : Schobbens P.

Table des matières

Introduction **2**

1 Question 1 — Opérateur `Button` **3**

2 Question 2 — Machine à états de la porte **5**

3 Question 3 — Machine à états de la passerelle **7**

4 Question 4 — Synchronisation porte et passerelle **9**

5 Question 5 — Commandes d'ouverture de porte et de déploiement de passerelle en station **10**

6 Question 6 — Fermeture de porte et rétraction de passerelle lors du départ immédiat **12**

7 Question 7 — Autorisation de départ **14**

Introduction

Ce rapport présente la réalisation des **Questions 1 à 7** autour d'un contrôleur porte/passerelle dans *ANSYS SCADE Suite*.

J'utilise la **version Student** de SCADE sur **macOS**. Certaines fonctionnalités ne sont pas entièrement disponibles ou ne fonctionnent pas comme dans la version Campus/Industrie. **Pour pallier ces limites**, j'ai documenté des *contournements* (variables locales initialisées jouant le rôle de constantes, notes dans les états pour affecter les sorties, etc.). L'objectif de ce rapport est d'expliquer **clairement la logique fonctionnelle** et les choix de modélisation malgré ces contraintes.

Chapitre 1

Question 1 — Opérateur `Button`

Objectif

Concevoir un opérateur `Button` qui mémorise l'appui d'un bouton :
— lorsque `press` passe de `false` à `true`, la sortie `req` devient `true` ;
— `req` reste ensuite `true` même après relâchement ;
— aucune remise à zéro (*reset*) n'est prévue dans cette version.

Méthode

La solution a été scindée en deux sous-parties :

1) Détection de front montant

— Blocs utilisés : `previous` (`pre`), `Init` (initialisation à `false`) et `not`.
— Rôle : détecter la transition `press : false → true`.
— La sortie de cette chaîne est le signal `rise` (déclenchement au clic).

2) Mémorisation (latch)

— Variable locale : `latched`, destinée à conserver son état.
— Principe logique (référence Lustre) :

```
latched = (false -> pre(latched)) or rise;
req     = latched;
```

— Réalisation schématique : second couple `previous + Init` suivi d'un `or`.
— Contrainte version Étudiante : impossibilité pratique d'une unique variable locale en mode Write/Read.
  — Contournement : utilisation d'une étiquette `latchedWrite` branchée à la sortie du `or`.
  — Conséquence : messages d'erreurs au *check* (`variable non définie` / `unused flow`), sans impact sur l'intention fonctionnelle.

Résultat attendu (simulation)

— Démarrage (`press = false`) → `req = false`.
— `press = true` sur un cycle → `req` devient `true`.
— `press = false` ensuite → `req` reste `true` (comportement latch).

Limitation rencontrée

La version Étudiante de SCADE limite la définition/lecture d'une même variable locale (`latched`). Le contournement via `latchedWrite` explique les avertissements/erreurs au *check*, tout en respectant le comportement attendu.

Conclusion

L'opérateur `Button` mémorise correctement un appui : détection sur front montant puis maintien via une boucle de rétroaction. Malgré les restrictions de la version Étudiante (gestion des variables locales et constantes), la logique fonctionnelle est conforme et observable en simulation.

Chapitre 2

Question 2 — Machine à états de la porte

Objectif

Spécifier le comportement d'une porte au moyen d'une **machine à états** qui renseigne en sortie son statut `doorStatus`. Les statuts possibles sont :

— `Closed` (fermée),
— `Opening` (en cours d'ouverture),
— `Open` (ouverte),
— `Closing` (en cours de fermeture).

Méthode

1. Création d'un type énuméré `DoorStatus` avec les quatre valeurs ci-dessus.
2. Création d'un *Node Operator* `Door` avec :
   — **Entrées** :
      — `openDoor : bool` (commande d'ouverture),
      — `closeDoor : bool` (commande de fermeture).
   — **Sortie** :
      — `doorStatus : DoorStatus`.
3. Conception d'une **machine à états** contenant les quatre états : `Closed`, `Opening`, `Open`, `Closing`.
   — L'état initial est `Closed`.
   — Transitions définies :
      — `Closed` → `Opening` si `openDoor = true`,
      — `Opening` → `Open` (transition immédiate dans la version simplifiée),
      — `Open` → `Closing` si `closeDoor = true`,
      — `Closing` → `Closed` (transition immédiate dans la version simplifiée).
4. Dans la version Étudiante de SCADE, l'éditeur n'affiche pas directement l'onglet *Actions/Activity*.

- Contournement : l'assignation de la sortie `doorStatus` a été placée dans une *note* à l'intérieur de chaque état.
- Exemple :
  - `Closed` : `doorStatus := Closed;`,
  - `Opening` : `doorStatus := Opening;`,
  - `Open` : `doorStatus := Open;`,
  - `Closing` : `doorStatus := Closing;`.

Résultat attendu (simulation)

- Au démarrage : `doorStatus = Closed`.
- Si `openDoor = true` : la machine passe par `Opening`, puis `Open` (`doorStatus = Opening` puis `Open`).
- Si `closeDoor = true` : la machine passe par `Closing`, puis `Closed` (`doorStatus = Closing` puis `Closed`).

Conclusion

La machine à états de la porte a été réalisée avec quatre états correspondant aux différents statuts possibles. L'utilisation de *notes* dans chaque état pour assigner la valeur de `doorStatus` permet de pallier les limites de la version Étudiante de SCADE, tout en obtenant le comportement attendu.

Chapitre 3

Question 3 — Machine à états de la passerelle

Objectif

Spécifier le comportement de la passerelle au moyen d'une **machine à états** donnant en sortie son statut `bridgeStatus`. Les statuts possibles sont :

— `Retracted`,
— `Deploying`,
— `Deployed`,
— `Retracting`.

Méthode

1. Définition du type énuméré `BridgeStatus`.
2. Création d'un *Node Operator* `Bridge` avec :
   — **Entrées** :
      — `deployBridge : bool` (commande de déploiement),
      — `retractBridge : bool` (commande de rétraction).
   — **Sortie** :
      — `bridgeStatus : BridgeStatus`.
3. Conception d'une **machine à états** avec 4 états :
   — État initial : `Retracted`.
   — Transitions définies :
      — `Retracted` → `Deploying` si `deployBridge = true`,
      — `Deploying` → `Deployed` (transition immédiate dans la version simplifiée),
      — `Deployed` → `Retracting` si `retractBridge = true`,
      — `Retracting` → `Retracted` (transition immédiate dans la version simplifiée).
4. Dans la version Étudiante de SCADE, l'éditeur n'affiche pas directement l'onglet *Actions/Activity*.

- Contournement : l'assignation de la sortie `bridgeStatus` a été placée dans une *note* à l'intérieur de chaque état.
- Exemple :
  - `Retracted` : `bridgeStatus := Retracted;`,
  - `Deploying` : `bridgeStatus := Deploying;`,
  - `Deployed` : `bridgeStatus := Deployed;`,
  - `Retracting` : `bridgeStatus := Retracting;`.

Résultat attendu (simulation)

- Au démarrage : `bridgeStatus = Retracted`.
- Si `deployBridge = true` → passage par `Deploying` puis `Deployed`.
- Si `retractBridge = true` → passage par `Retracting` puis `Retracted`.

Conclusion

La machine à états de la passerelle a été réalisée de façon analogue à celle de la porte, avec des états adaptés. L'utilisation de *notes* pour assigner `bridgeStatus` dans chaque état permet d'obtenir un modèle fonctionnel malgré les limitations de la version Étudiante de SCADE.

Chapitre 4

Question 4 — Synchronisation porte et passerelle

Objectif

Assurer la synchronisation entre la porte et la passerelle afin de garantir la sécurité des passagers.

Solutions mises en place dans le contrôleur

1. **Ouverture de porte conditionnée** : la porte ne peut s'ouvrir que si la passerelle est déjà déployée (`bridgeStatus = Deployed`).
2. **Blocage du mouvement de la passerelle** : la passerelle ne peut ni se déployer ni se rétracter lorsque la porte est `Open` ou `Opening`.
3. **Fermeture de porte** : elle peut se produire indépendamment de la passerelle, dès que l'ordre est donné.
4. **Rétraction de passerelle** : autorisée uniquement si la porte n'est pas ouverte.

Rôle du contrôleur

Le contrôleur agit comme un filtre logique :

— il reçoit les demandes brutes (`openDoorCmd`, `closeDoorCmd`, `deployBridgeCmd`, `retractBridgeCmd`) ainsi que les statuts (`doorStatus`, `bridgeStatus`),

— il décide s'il est autorisé ou non de transmettre la commande aux automates de la porte et de la passerelle.

Conclusion

Le contrôleur garantit que la passerelle est toujours déployée avant l'ouverture de la porte et qu'elle reste immobile tant que la porte n'est pas refermée. Cette logique de synchronisation assure la cohérence des commandes et la sécurité des passagers.

Chapitre 5

Question 5 — Commandes d'ouverture de porte et de déploiement de passerelle en station

Objectif

Spécifier l'envoi des commandes d'ouverture de la porte et de déploiement de la passerelle lorsque la rame est en station et qu'une demande a été effectuée par un passager (porte ou passerelle). La commande doit être maintenue tant que l'action n'est pas réalisée (porte effectivement ouverte ou passerelle effectivement déployée).

Méthode

1. **Entrées considérées** :
   — `inStation : bool` (position de la rame),
   — `reqDoor : bool` (demande de porte),
   — `reqBridge : bool` (demande de passerelle),
   — `doorStatus : DoorStatus` (état courant de la porte),
   — `bridgeStatus : BridgeStatus` (état courant de la passerelle).
2. **Sorties générées** :
   — `deployBridge : bool` (commande de déploiement),
   — `openDoor : bool` (commande d'ouverture),
   — `resetBridgeReq : bool` (réinitialisation de la demande passerelle),
   — `resetDoorReq : bool` (réinitialisation de la demande porte).
3. **Implémentation graphique dans un Node Operator** :
   — Déploiement de passerelle :

```
deployBridge = inStation
            AND (reqDoor OR reqBridge)
            AND NOT( (bridgeStatus = Deployed) OR (bridgeStatus
               = Deploying) )
```

— Ouverture de porte :

```
openDoor = inStation
        AND (reqDoor OR reqBridge)
        AND (bridgeStatus = Deployed)
        AND NOT( (doorStatus = Open) OR (doorStatus = Opening) )
```

— Resets :

```
resetBridgeReq = (bridgeStatus = Deployed);
resetDoorReq   = (doorStatus = Open);
```

— Pour réaliser les comparaisons, des **variables locales initialisées** à des valeurs énumérées (`vDeployed`, `vDeploying`, `vOpen`, `vOpening`) ont été utilisées comme contournement, la version Étudiante ne permettant pas de créer directement des constantes d'énumération.

Résultat attendu (simulation)

— **Déploiement** : quand la rame est en station et qu'un passager demande porte ou passerelle, la passerelle commence à se déployer. La commande `deployBridge` reste active jusqu'à ce que `bridgeStatus = Deployed`.
— **Ouverture** : dès que la passerelle est déployée, la commande `openDoor` devient active. Elle reste active tant que `doorStatus` n'est pas `Open`.
— **Resets** : une fois la passerelle déployée (`bridgeStatus = Deployed`), la demande associée est réinitialisée ; de même pour la porte lorsqu'elle est ouverte (`doorStatus = Open`).

Conclusion

Le contrôleur en station garantit une séquence correcte :

1. Déploiement de la passerelle,
2. puis ouverture de la porte.

Les commandes sont maintenues jusqu'à exécution complète et les demandes passagers sont réinitialisées automatiquement.

Chapitre 6

Question 6 — Fermeture de porte et rétraction de passerelle lors du départ immédiat

Objectif

Spécifier l'envoi des commandes de fermeture de la porte et de rétraction de la passerelle lorsqu'un départ immédiat est déclenché, afin d'assurer la sécurité avant le mouvement de la rame.

Méthode

1. **Entrées considérées** :
   — `immDeparture : bool` (signal indiquant le départ immédiat),
   — `doorStatus : DoorStatus` (état de la porte),
   — `bridgeStatus : BridgeStatus` (état de la passerelle).
2. **Sorties générées** :
   — `closeDoor : bool` (commande de fermeture),
   — `retractBridge : bool` (commande de rétraction).
3. **Logique de décision** :
   — **Fermeture de porte** :
   ```
   closeDoor = immDeparture AND NOT(doorStatus = Closed)
   ```
   La commande est envoyée tant que la porte n'est pas encore fermée.
   — **Rétraction de passerelle** :
   ```
   retractBridge = immDeparture AND NOT(bridgeStatus = Retracted)
   ```
   La commande est envoyée tant que la passerelle n'est pas encore rétractée.
4. Pour réaliser ces comparaisons, des **variables locales initialisées** à des valeurs énumérées (`vClosed`, `vRetracted`) sont utilisées, comme pour les questions précédentes, en remplacement de constantes (indisponibles dans la version Étudiante).

Résultat attendu (simulation)

- Si `immDeparture = true` :
  - et que la porte est ouverte (`Open` ou `Opening`), la commande `closeDoor` est activée jusqu'à ce que `doorStatus = Closed`.
  - et que la passerelle est déployée (`Deployed` ou `Deploying`), la commande `retractBridge` est activée jusqu'à ce que `bridgeStatus = Retracted`.
- Une fois les deux équipements sécurisés, les commandes retombent automatiquement à `false`.

Conclusion

Le contrôleur en cas de départ immédiat garantit la fermeture de la porte et la rétraction de la passerelle si nécessaire. Les commandes sont maintenues jusqu'à exécution complète, ce qui assure une séquence sécurisée avant la mise en mouvement de la rame.

Chapitre 7

Question 7 — Autorisation de départ

Objectif

Spécifier le signal d'**autorisation de départ** du contrôleur, en fonction de l'état de la porte et de la passerelle. Trois cas sont à considérer :

1. La porte est déjà fermée et la passerelle rétractée → le tramway peut redémarrer.
2. La porte est ouverte ou en cours d'ouverture → attendre sa fermeture avant le départ.
3. La passerelle est déployée ou en cours de déploiement → attendre sa rétraction et la fermeture de la porte.

Méthode

1. **Entrées considérées** :
   — `doorStatus : DoorStatus`,
   — `bridgeStatus : BridgeStatus`.
2. **Sortie générée** :
   — `departAuth : bool`.
3. **Implémentation graphique dans un Node Operator** :
   — Comparateur `doorStatus = Closed` → booléen,
   — Comparateur `bridgeStatus = Retracted` → booléen,
   — `and` des deux résultats → `departAuth`.
4. Formule :

```
departAuth = (doorStatus = Closed) AND (bridgeStatus = Retracted)
```

Résultat attendu (simulation)

— Si la porte est `Closed` et la passerelle `Retracted` → `departAuth = true` → départ autorisé.
— Si la porte est `Open` ou `Opening` → `departAuth = false` jusqu'à fermeture.
— Si la passerelle est `Deployed` ou `Deploying` → `departAuth = false` jusqu'à rétraction complète (et fermeture de la porte).

Conclusion

L'autorisation de départ repose sur une condition de sécurité simple : **porte fermée ET passerelle rétractée**. Cette logique couvre les trois cas de l'énoncé et s'intègre au contrôleur conçu aux étapes précédentes.