Modélisation des Acteurs

Marion Ly
Thibaut Smith

2013 - 2014

Table des matières

1 Interface pour les acteurs

Le module **actor_contract** fournit à la fois la spécification d'un acteur (les fonctions que l'acteur doit implémenter) et une liste de fonctions exportées permettant de simplifier les actions les plus basiques tels que création/recherche d'information dans les métadonnées d'un acteur.

Voici une liste des fonctions que doivent implémenter chaque acteur pour assurer une compatibilité avec le système de contrôle.

1.1 Fonction create/0

Cette fonction renvoi un acteur avec des valeurs mises par défaut ou générés aléatoirement, tels que le nom de l'acteur (en général le nom du module), l'identifiant de l'acteur créé (unique).

Elle ne prend aucun argument.

1.2 Fonction answer/2

Cette fonction permet de **caractériser** le comportement du module en fonction d'une entrée précise (qui peut être un produit, mais pas nécessairement). C'est cette fonction qui doit modifier/lire le produit si il y a lieu.

Elle prend deux arguments en entrée :

1. La configuration de l'acteur
2. La requête demandé qui s'exprime, pour toutes les *requêtes*, par le protocole suivant : {ConfigurationActeurA, {NomActeurB, ConfigurationActeurB}} ActeurA correspond à l'acteur auquel on envoie la requête, et ActeurB correspond au flux physique, dans notre cas le produit.

Cette fonction représente aussi le temps de travail via une pause forcée (en mode temps-réel), c'est pourquoi à chaque fois que cette fonction est appelée, elle est encapsulée dans un noeud extérieur qui prévient l'acteur quand le travail est terminé. Cet aspect nous permet de contrôler plusieurs requêtes en parallèle.

La réponse est de la forme suivante :

{NouvelleConfigurationActeurA, {NomActeurB, ConfigurationActeurB, Information}, Destinaton}

1. On retrouve les mêmes informations que tout à l'heure, mais légèrement modifiées : la nouvelle configuration de l'acteur A se retrouve en première position.
2. le nom et la nouvelle configuration de l'acteur B en deuxième position, et une information s'est rajoutée. Cette information peut représenter l'information détectée par l'acteur (exemple : le lecteur RFID met l'identifiant du produit ici), ou juste le nom de l'action qui a été exécuté sur l'acteur B.

3. Enfin vient la destination de la requête, qui est soit dirigée à l'acteur suivant, soit au superviseur. (Cette valeur est beaucoup utilisée par l'aiguilleur.)

Cette fonction décrit donc à la fois les messages correspondant au flux physique et au flux logique. En fonction de la destination, on peut faire la différence.

La dernière tâche de cette fonction est d'inscrire des informations utiles à la fois dans la base de données du produit et de l'acteur actuel.

L'interface **actor_contract** comprend également différentes fonctions implémentées pour accéder et modifier les données des configurations ainsi que des réponses types pouvant être utilisées par chaque acteur. Il est cependant toujours possible d'annuler ce comportement via programmation.

Configuration	Module	Id	opt (options)	State	Worktime	list_data
Convoyeur	actor_conveyor	Manuel/Hasard	sortie	on/off	Temps de transport	Produits transférés
Aiguilleur	actor_railway	Manuel/Hasard	entrée/sortie/règle de priorité	in, out (son état physique)	Temps d'aiguillage	Produits transférés, état aiguilleur
RFID	actor_rfid	Manuel/Hasard	entrée	on/off	Temps de détection	Produits détectés
Station de travail	actor_workstation	Manuel/Hasard	entrée/sortie	on/work/off	Temps de fabrication	Liste des produits crées, avec la qualité
Produit	actor_product	Manuel/Hasard	Qualité demandée	raw/Q1/Q2/Q3	Aucun	Emplacements parcourus
File d'attente basique	actor_basic_queue	Manuel/Hasard	Pas d'options	on/processing/off	Temps minimum avant qu'un produit soit disponible	Liste de produits avec temps d'entrée et temps de sortie
Scanner	actor_scanner	Manuel/Hasard	Pas d'options	on/off	Temps de détection de 2 secondes	Liste de produits détectés, avec la qualité
Station de travail d'assemblage	actor_workstation_assembly	Manuel/Hasard	Configuration à insérer dans le produit suivant	on/processing/off	Temps d'assemblage de 12 secondes	Liste de produits transformés
Station de travail finition	actor_workstation_finish	Manuel/Hasard	Pas d'options	on/processing/off	5 secondes	Liste des produits avec la qualité

2 Objectifs du module container

Le module **container** permet de créer un noeud Erlang qui attend les messages des autres acteurs. Les fonctions du container ne dépendent uniquement de la configuration d'un acteur. (C'est pourquoi il est important que les acteurs implémentent l'interface.) Dans toute la suite de cette section, quand on dira 'l'acteur', il faudra en fait comprendre 'le module container lancé avec la configuration de tel acteur'.

Le container utilise des fonctions pour la gestion de son acteur. Voici une liste des choses qu'il doit gérer :

1. Recevoir les messages des autres acteurs
2. Recevoir et envoyer les messages au superviseur
3. Les noeuds qui représentent le travail sur les produits
4. Les envois de produits aux autres acteurs

2.1 Recevoir les messages des autres acteurs

Les acteurs s'envoient principalement des produits entre eux, le seul flux physique que nous avons.

Les acteurs ayant à chaque instant un comportement asynchrone, ils peuvent gérer les autres situations qui peuvent arriver.

2.2 Recevoir et envoyer les messages au superviseur

L'acteur envoie constamment ses changements de configurations au superviseur. Pour l'acteur aiguilleur, il y a un dialogue qui lui permet d'orienter le produit vers la bonne sortie.

2.3 Les noeuds qui représentent le travail sur les produits

Dès qu'un acteur reçoit un produit, le travail supposé être fait par l'acteur est effectué par la fonction *answer/2*. Comme le travail à effectuer est potentiellement long, il est lancé dans un autre noeud en parallèle.

Une fois que ce noeud a terminé, l'acteur est prévenu via les fonctions *end_of_physical_work/3* (**flux physique**) et *end_of_logical_work/3* (**flux logiques**) et le produit est rajouté dans la liste des produits à envoyer.

2.4 Les envois de produits aux autres acteurs

Cependant, pour conserver une bonne gestion des produits vis-à-vis des blocages du aux capacités maximum des acteurs, on utilise le protocole suivant quand un acteur souhaite envoyer

un produit :

1. Le **premier acteur** envoie une notification de produit en attente au **second acteur**. Le produit est enregistré dans le premier acteur, et il faut **attendre la confirmation** du second acteur avant d'envoyer le produit. [1]
2. Sur réception de cette notification, le second acteur examine sa liste de produits qu'il a sous sa garde : les produits en cours de travail **et** les produits qui sont en attente d'être envoyé à l'acteur suivant. Deux issues possibles :
   — Si le second acteur a de la place disponible, il envoie une confirmation au premier acteur en utilisant le 'PidPremierActeur' pour le contacter.
   — Si il n'a pas de place, il n'envoie rien. Dès que de la place se libère, il enverra une confirmation au premier acteur.
3. Quand le premier acteur reçoit une confirmation, il envoie le **premier produit arrivé** (FIFO).

3 Acteurs actuellement implémentés

Voici des acteurs qui ont déjà été implémentés dans le but de pouvoir mettre en place le scénario donné par nos encadrants.

3.1 Lecteur RFID

Lorsque le lecteur reçoit un produit, nous avons utilisé le temps de travail comme temps de détection (qui peut-être mit à zéro par défaut, s'il le faut) qui crée une pause. Puis le lecteur enregistre alors les données du produit dans sa base et ses données à lui dans la base du produit. Le lecteur envoie alors l'Id du produit qu'il a lu en réponse.

3.2 Convoyeur

Lorsqu'un produit arrive au convoyeur, une pause correspondant au temps de travail de la configuration de l'acteur se passe, c'est le temps de transport du produit. Les données du produit sont enregistrées dans la base du convoyeur, celles du convoyeur dans la base du produit. Le produit est alors envoyé à l'acteur suivant.

3.3 Aiguilleur

À partir du moment, où l'aiguilleur reçoit le produit, sa réponse dépendra des entrées sorties de celui-ci :

— plusieurs entrées ou plusieurs sorties, un message est envoyé au superviseur, contenant la description du conflit et les données du produit et de l'aiguilleur pour que le superviseur

1. Format : {PidPremierActeur, awaiting_product}

puisse prendre sa décision (un ordre de priorité contenu dans les options de l'aiguilleur, et la qualité demandé du produit peut l'aider).
— une entrée et une sortie, on ajoute le produit ainsi que l'état de l'aiguilleur à la base de l'aiguilleur et on envoie le produit à l'acteur suivant.

S'il y a eu conflit, l'aiguilleur recevra alors la décision du superviseur sur l'aiguillage du produit. On pourra ajouter les informations de la même façon qu'avec une entrée et une sortie. L'aiguilleur changera d'état (entrée/sortie) et il y aura alors un temps de travail. l'aiguilleur enverra alors ensuite le produit au destinataire choisi par l'aiguilleur. Si la décision choisi est identique qu'à l'état actuelle de l'aiguilleur celui-ci n'a donc pas à changer d'état et il n'y a donc pas de temps de travail.

3.4 Station de travail

Les stations de travail peuvent recevoir un seul produit à la fois, et permettent de modifier l'état du produit. On ne peut cependant pas le connaître tant que le produit n'est pas arrivé à un acteur Scanner qui pourra confirmer l'état du produit réel.

Cette acteur doit être configuré avec l'option 'workstation_luck', correspondant à la qualité voulue des produits arrivant sur la station. Voici des exemples de valeurs :

— **{'Q2', 33}** : 33% de chance d'accomplir un objet de qualité Q2 (ceci est équivalent à une équiprobabilité entre les trois qualités, Q1, Q2 et Q3).
— **{'Q1', 98}** : 98% de chance d'accomplir un produit de qualité Q1 (configuration de la station de travail numéro deux dans le scénario final).

3.5 File d'attente basique

Les files d'attentes sont juste avant les stations : elles permettent de contrôler l'arrivée des objets selon la capacité de la station.

Un temps minimum de transport est possible dans le cas où la file d'attente est utilisé comme convoyeur : le temps représente le déplacement du produit. Cet élement est à zéro par défaut, soit le fonctionnement de base.

Plus tard, il sera possible d'implémenter des files spéciales à plusieurs entrées.

3.6 Le produit

Comme dit précédemment, le produit est un flux physique donc il n'a pas particulièrement besoin d'un protocole. On a donc désactivé plusieurs type de requêtes qui permettent de le modifier. On lui demandera alors juste de répondre par ses données. Ce type de réponse pourra aussi être utilisé par les autres acteurs si on veut récupérer/modifier certaines de leurs données de manières simplifié.

Le produit est composé de la palette en bois et de la disposition des tablettes. Il y a quatre

emplacements possibles. De plus, la qualité peut être augmenté une fois à 100% grâce à un acteur.

La qualité initiale du produit (palette en bois) est choisie de manière équiprobable.

3.7 Le scanner

L'acteur Scanner permet de s'assurer de la qualité d'un produit. En effet, les stations de travail ne peuvent que travailler sur un produit et accomplir leur commande avec leur probabilité d'échouer. Le scanner se charge d'analyser la qualité du produit pour aider les superviseurs à diriger les acteurs.

Note : il sera possible d'ajouter un pourcentage d'erreur à la détection.

3.8 Station de travail d'assemblage

Cette acteur ajoute les tablettes dans le produit. On le configure avec l'option 'order'. Voici des exemples :

— **{1, 0, 1, 0}**
— **{1, 1, 1, 1}**
— **{0, 0, 0, 1}**

3.9 Station de travail pour la finition

Cette acteur se trouve dans la dernière zone du scénario final. Il permet d'augmenter la qualité d'un produit de un si la commande client le demande. De même que pour la station de travail d'assemblage, la commande est à inscrire dans les options.

4 Tests unitaires

Sur demande de Arnould Guidat, nous avons programmé des tests unitaires pour le plus de fonctions possibles. Cela permet d'être sûr du comportement des fonctions et de remarquer très rapidement les régressions dans le code.