PROJET SAR 2018-2019

Producteurs-Consommateur

Par : Axel Acuna, Shirel Matti, Corentin Fanton, Lisa Aït-Mouloud et Samuel Arzur

Sommaire :

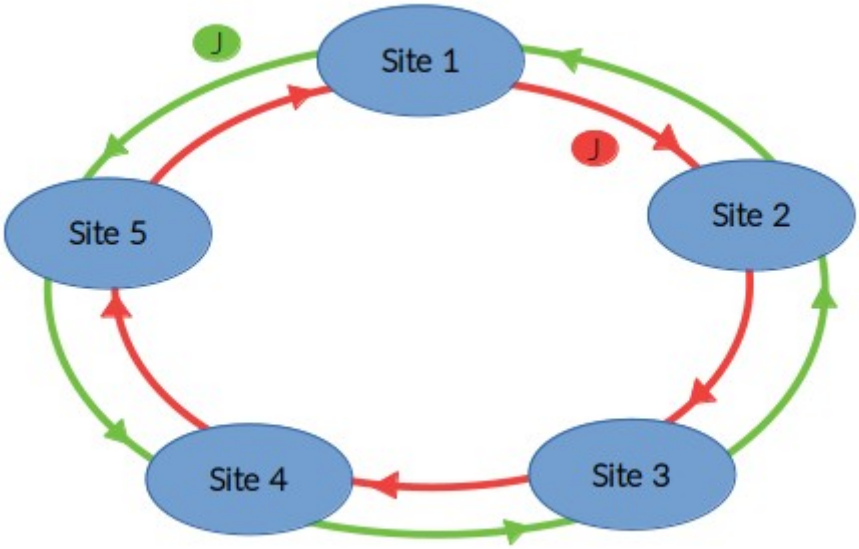
Introduction :

Dans le cadre du module algorithmes et systèmes répartis, il nous a été demandé d'implémenter le concept des producteurs/consommateur.
Pour notre cas, le système réparti implémenté se résume à la couche service, et donc peut être relié à toute application de la couche applicative utilisant le concept producteurs/consommateur.

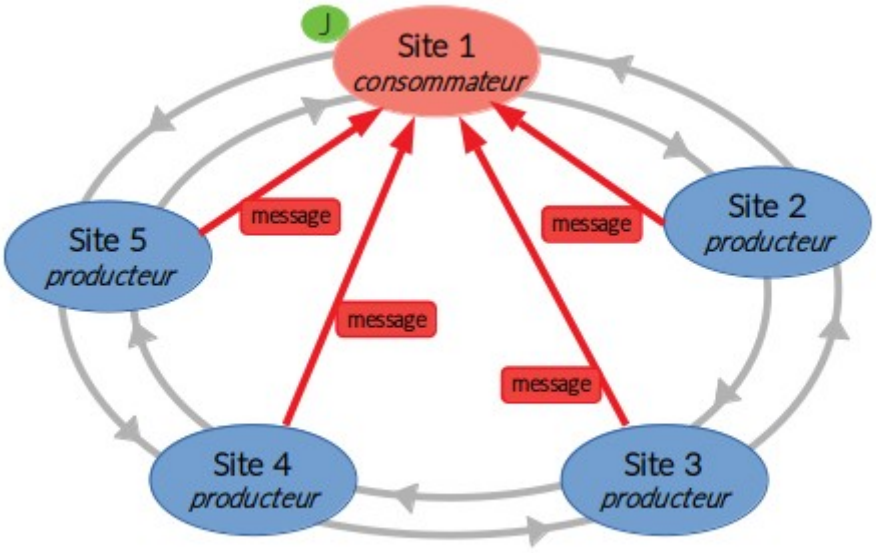
Ce rapport présente tout d'abord la modélisation de notre système ainsi que des explications sur les choix techniques, puis il fournit le pseudo code de nos algorithmes répartis et des algorithmes d'élection, et se termine par un guide d'utilisation du programme.

Graph de communication et hypothèses :

- Graph de communication :



Graph de communication : construction de l'anneau de base



Graph de communication : après établissement des liens entre producteurs et consommateur

- Hypothèses :

Ainsi, contrairement à l'hypothèse n°1 du cours, notre graphe de communication n'est pas complet -voir figure graph de communication-. Néanmoins, nous conservons l'hypothèse n°2 du cours qui stipule que les lignes de transmission ont les propriétés suivantes :
- Les messages ne sont pas altérés
- Les messages ne sont pas perdus
- Le canal est FIFO.

Il est important à souligne que le délai de transit des messages est quelconque et que de ce fait, notre réseau est asynchrone, tout comme l'hypothèse n°3 du cours le précise.
En outre, la réception de message est synchrone c'est-à-dire que des qu'un site souhaite recevoir un message, il fait explicitement appel à la primitive SRD(id j , message m) et se bloque dans l'attente de ce message. Contrairement à l'envoi, qui lui est asynchrone, c'est-à-dire qu'un site qui émet un message ne se bloque pas dans l'attente de ce message.
Toutefois, comme l'hypothèse n°4 du cours ; nous pouvons dire que la communication est asynchrone puisque nous avons émuler la primitive synchrone SRD(id j , message m) à partir de primitives asynchrones. Enfin, l'application est exécutée sur un unique processus pour les producteurs mais non pour le consommateur qui se doit de gérer plusieurs requêtes à la fois, entre autre, la réception des messages de plusieurs producteurs. Donc, nous mentionnons le fait qu'à l'opposé de l'hypothèse n°5 du cours, notre application fonctionne sur plusieurs processus.

Choix techniques :

- Justification des choix techniques :
    - Justification du choix du contexte

La classe Context est une classe commune à l'ensemble de sites celle si contient les informations nécessaires à toute connexion. C'est-à-dire, adresse IPV4 et Port. Chaque machine du réseau est enregistrée au sein de cette classe dans une string ayant la structure suivante «xx.xx.xx.xx :PORT ».
Si cette classe Context n'est pas commune à l'ensemble l'application ne marchera pas. Il est donc nécessaire de définir les adresses et les ports des machines participantes avant de lancer le programme.

- Un choix de réserver le port 4010 aux échanges entre producteurs et consommateur.

Cette hypothèse sera expliquée dans la suite du rapport.
Chaque Site lors de son lacement, crée une liaison avec son prédécesseur et son successeur définis dans le contexte, d'où les canaux bidirectionnels dans le graph de communication.

- Nous supposons que nos machines, tout comme des servers, ont des adresses IP fixes, ce qui nous permet de mettre en place un contexte statique défini une fois seulement. Car les servers, ne changent jamais d'adresse IP.

    - Justification de l'architecture en anneau :

Le choix s'est porté sur une architecture en anneau en premier lieu, sur laquelle vont s'ajouter des canaux d'échange entre producteurs et consommateurs qui seront présentés dans la suite du rapport.
L'idée d'implémenter une architecture en graph complet a été rejetée, car le problème à résoudre consiste en une élection puis une communication entre les producteurs et le

consommateur. Les producteurs ne communiquent qu'avec leur successeur et leur prédécesseur, et non pas avec tout les autres producteurs.
Tout d'abord, toutes les machines sont définies par leur adresses IP et leur port d'échange dans le contexte.
Nous supposons ce qui suit, pour assurer un fonctionnement continu sans interruption ni arrêt :

- Nous supposons que nos machines ne seraient jamais hors-service (down), ce qui nous assure une architecture opérationnelle. Car l'architecture en anneau est connue pour être down dans le cas où un des sites de l'anneau est down.

    - Justification des types de connexion utilisées :

Comme nous pouvons le voir au sein de la classe Producteur et Consommateur, nous utilisons des sockets TCP car leur liaison est permanente durant toute la durée d'envoi des messages.
Il en est de même pour les sockets qui relient en anneau le consommateur avec tout les producteurs. Des Socket TCP sont utilisées aussi pour assurer la circulation du jeton qui met à jour le nombre d'autorisation et le nombre de messages à consommer.

- Fonctionnement des protocoles :
    - Création de l'anneau

    Nous avons pris la décision de faire un anneau en créant un serveur par site. Chaque site devra gerer la connexion au sein de son serveur et devra se connecter chez son prédécesseur en créant un Socket Client. Ainsi, le site gère son serveur et la connexion à son prédécesseur.
    Un site est composé d'une class Client et Serveur qui permettront de créer l'anneau.Le site 5 se connecte au serveur du site 1, le site 1 au serveur du site 2, etc. Cette structure nous permet d'ouvrir des canaux grâce au socket qui font la liaison entre le serveur et le client.
    - Producteurs / consommateur

      Nous avons pris la décision d'utiliser deux serveurs pour l'algorithme du producteur consommateur. La classe producteur sera comme un client pour le consommateur. La classe consommateur créera un nouveau serveur qui administrera tous les producteurs et leurs messages de façon asynchrone. Un objet jeton circulera au sein de l'anneau pour distribuer des autorisations d'envoi de message.
        - Producteur :

Le producteur correspond à un client qui va chercher à se connecter au sein du serveur « élu comme consommateur» (choisi durant la phase d'élection). Il utilise la classe Client pour générer un socket permettant la liaison et la création d'un flux entre le serveur du consommateur l'élu et le site producteur.
Concernant les autres sites non élus, ils feront appel à la classe Producteur qui se chargera de gérer l'envoie des messages vers le consommateur et l'envoie et réception du jeton de son successeur et son prédécesseur.
La circulation du jeton et l'envoie de messages sont deux tâches exécutées en parallèle par deux threads différents.

- Consommateur :

Le consommateur ouvre un serveur pour que toutes les machines productrices se connectent. L'adresse de connexion du serveur est « IP :4010 ». Nous avons fait ce choix

pour faciliter la connexion des autres machines. Les connexions, des sockets crées par liaison du producteur au consommateurs, seront traité par la classe Thread Producteur, cette classe permet la réception et le traitement des messages envoyés par les producteurs en asynchrone.

Par exemple lors de la réception d'un message le Thread qui a reçu le message se chargera de prévenir le consommateur pour effectuer une sur_reception_de().
Lors de son exécution, le consommateur lance un jeton qui circulera parmi les producteurs. Les messages reçus par le consommateur seront consommés au moment où celui-ci aura décidé de le faire. L'envoie et la réception, ainsi que la consommation des messages seront des tâches en parallèle à la réception de message.
Nous avons fait le choix de synchroniser certaine tache (incrémentation, nbmess, etc). Les monitors nous ont servis à restreindre l'accès et maintenir la cohérence au sein des threads producteurs qui ont, dans une certaine mesure, accès aux variables du consommateur.
Exemple d'utilisation du monitor :

MonitorNBCell : Permet que le thread qui se charge de la consommation et de la réception du jeton n'ai pas accès à une modification simultanée et risquée de cette variable.
La classe consommateur correspondra ainsi à un serveur qui va gérer les messages producteurs.

Algorithmes répartis d'élection :

Concernant l'élection, les trois méthodes les plus connus de l'élection dans une architecture en anneau ont été implémentés et ainsi lors de l'exécution, un choix sera fait dans le contexte qui précisera l'élection à exécuter.
Les trois algorithmes prennent comme critère d'élection les identifiants des sites et les compare pour élire un consommateur.

- Algorithme de Lelann :
  Le principe de l'algorithme de LeLann consiste en le fait que dans un anneau, une sorte de jeton est envoyé par chaque site pour mettre à jour une variable min qui prend la plus petite valeur entre sa valeur et la valeur de l'id du site sur lequel passe le jeton.
  Cette opération s'arrête sur un site quand il finit par recevoir la valeur de son identifiant, autrement dit, quand le jeton envoyé termine de faire le tour de l'anneau.
  Chaque machine à ce moment compare son id avec la valeur du min qui représente la valeur minimale dans l'anneau. Si son id est égal au min, alors elle est élue, sinon elle est non-élue.

Algorithme 1 : Algorithme de LeLann (1977)

```
début
    min := mon_id ;
    id_courant := mon_id ;
    répéter
        envoyer ⟨id_courant⟩ à suivant ;
        recevoir ⟨id_courant⟩ de précédent ;
        si id_courant < min alors
            min := id_courant ;
    jusqu'à id_courant == mon_id;
    si min == mon_id alors
        statut := élu
    sinon
        statut := non-élu ;
fin
```

- Algorithme de Chang et Roberts
  Le principe de cet algorithme est lui aussi très simple. En effet, il consiste à que chaque site envoie à son suivant don id. Les sites comparent la valeur reçue avec leur propre id, et n'envoient que la plus petite valeur des deux.
  Autrement dit, un site dont l'id est supérieur à son précédent, se servira qu'à faire passer le message de la plus petite valeur rencontrée jusqu'ici.
  L'opération s'arrête au moment où un site reçoit la valeur de son propre id, ce qui indique que ce site en question est celui qui possède le plus petit identifiant et qui est donc l'élu comme consommateur. Tout de suite après cela, le site élu, va faire circuler un message de fin pour prévenir les autres sites qu'il a été élu pour arrêter l'opération d'élection.

Algorithme 2 : Algorithme de Chang et Roberts (1979)

```
début
    envoyer ⟨tok, mon_id⟩ à suivant ;
    répéter
        recevoir m de précédent ;
        si m == ⟨tok, id_courant⟩ alors
            si id_courant < mon_id alors
                statut := non-élu ;
                envoyer m à suivant ;
    jusqu'à id_courant == mon_id ou m == ⟨fin⟩;
    envoyer ⟨fin⟩ à suivant ;
    si id_courant == mon_id alors
        statut := élu ;
        recevoir ⟨fin⟩ de précédent ;
fin
```

- Algorithme de Franklin

Le principe de cet algorithme est pratiquement le même que l'algorithme de Chang et Roberts. La seule différence, est que celui-ci est en bidirectionnel et donc envoie l'id du site au précédent et au suivant et reçoit l'id du précédent et du suivant aussi.
De même que pour l'algorithme de Chang et Roberts, celui-ci n'envoie que le minimum entre le min et l'id du site au suivant et au précédent.
L'algorithme s'arrête au moment où un site reçoit la valeur de son id, et à ce moment prévient les autres sites qu'un site a été élu en tant que consomatteur.

Algorithme 3 : Algorithme de Franklin (1982)

```
début
    statut := actif ;
    fini := faux ;
    tant que ¬fini faire
        si statut == actif alors
            envoyer ⟨mon_id⟩ à suivant ;
            recevoir ⟨id_prec⟩ de précédent ;
            envoyer ⟨mon_id⟩ à précédent ;
            recevoir ⟨id_suiv⟩ de suivant ;
            si id_prec < mon_id ou id_suiv < mon_id alors
                statut := non-élu ;
            si id_prec == mon_id et id_suiv == mon_id alors
                statut := élu ;
                fini := vrai ;
        sinon
            recevoir ⟨id_prec⟩ de précédent ;
            envoyer ⟨id_prec⟩ à suivant ;
            recevoir ⟨id_prec⟩ de suivant ;
            envoyer ⟨id_prec⟩ à précédent ;
            si id_prec == id_suiv alors
                fini := vrai ;
fin
```

Algorithme réparti de communication producteur-consommateur :

Pour cette partie de système, nous nous sommes basés sur la partie du cours qui aborde la gestion de flux dans un concept producteurs/consommateur.

- Producteur :

Algorithm 1 Algorithme du producteur i

$\mathbf{T_i[0, N\text{-}1]}$: tableau contenant les messages produits par le producteur i. où, in_i : est l'indice d'insertion dans le tableau, initialisé à 0. et, out_i : est l'indice d'exraction du tableau, initialisé à 0. $\mathbf{nbMess_i}$: nombre de messages stockés dans le tableau, initialisé à 0. $\mathbf{nbAut_i}$: nombre d'autorisations d'envoie de messages, initialisé à 0. $\mathbf{succ_i}$: l'id su successeur du site i. Si i=p, la valeur de $succ_i$ est l'id du consommateur, sinon c'est i+1.

produire(m)
begin
attendre($NbMess_i < N_i$);
$T_i[in_i] = m$;
$in_i = (in_i + 1)\%N$;
$nbMess_i + +$;
end;

surReceptionDe(j, (jeton, val))
begin
$temp_i = min(nbMess_i - nbAut_i, val)$;
$nbAut_i += temp_i$;
$val -= temp_i$;
$envoyerA(succ_i, (jeton, val))$;
end;

facteur()
begin
while *true* **do**
attendre($nbAut_i > 0$);
envoyerA(C, (m,$T_i[out_i]$));$out_i = (out_i + 1)\%N$;
$nbAut_i - -$;
$nbMess_i - -$;
end while
end;

- Consommateur :

Algorithm 1 Algorithme du consommateur

$\mathbf{T_C[0, N\text{-}1]}$: tableau contenant les messages des sites producteurs sur le tampon du consommateur. où, $\mathbf{in_C}$: est l'indice d'insertion dans le tableau, initialisé à 0. et, $\mathbf{out_C}$: est l'indice d'exraction du tableau, initialisé à 0. $\mathbf{nbMess_C}$: nombre de messages stockés dans le tableau et non encore consommés, initialisé à 0. $\mathbf{nbCell_C}$: nombre de cellules libérées entre deux passages de jeton, initialisé à 0.

consommer(m)
begin
attendre($NbMess_c > 0$);
$m = T_c[out_c]$;
$out_c = (out_c + 1)\%N$;
$nbMess_c - -$;
$nbCell_c + +$;
end;

surReceptionDe(j, (jeton, val))
begin
$val+ = nbCell_c$;
$nbCell_c = 0$;
$envoyerA(1, (jeton, val))$;
end;

surReceptionDe(j, (mess, m))
begin
$T_c[in_c] = m$;
$in_c = (in_c + 1)\%N$;
$nbMess_c + +$;
end;

Documentation utilisateurs et captures d'écran :

Il est à noter que le projet a été créé en utilisant Maven, ce qui justifie la séparation en packages dans le projet. Ce choix a été fait, pour pouvoir exécuter le projet sur toutes les plateformes.

Lien du répertoire GitHub du projet : https://github.com/axelacu/SAR-Project/

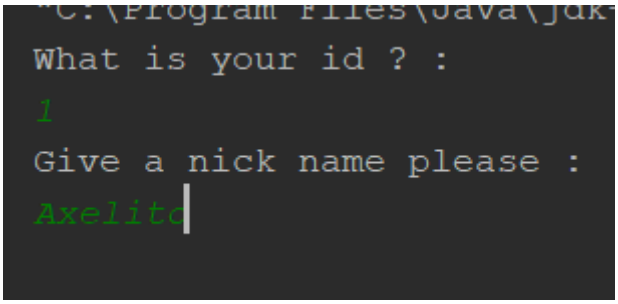
Tout d'abord, il faut ouvrir le projet, aller dans le package **projet** puis ouvrir la classe **App.java**, où est défini le contexte des machines participantes.

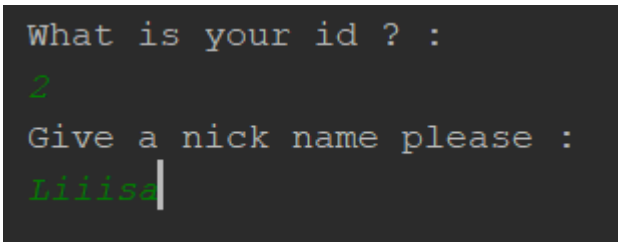
- Si le teste se fait en local, il suffit de mettre la même adresse IP pour le nombre de machines que l'on souhaite faire participer.

- Si le test se fait sur plusieurs machines différentes, il suffit de mettre les adresses IP des différentes machines.

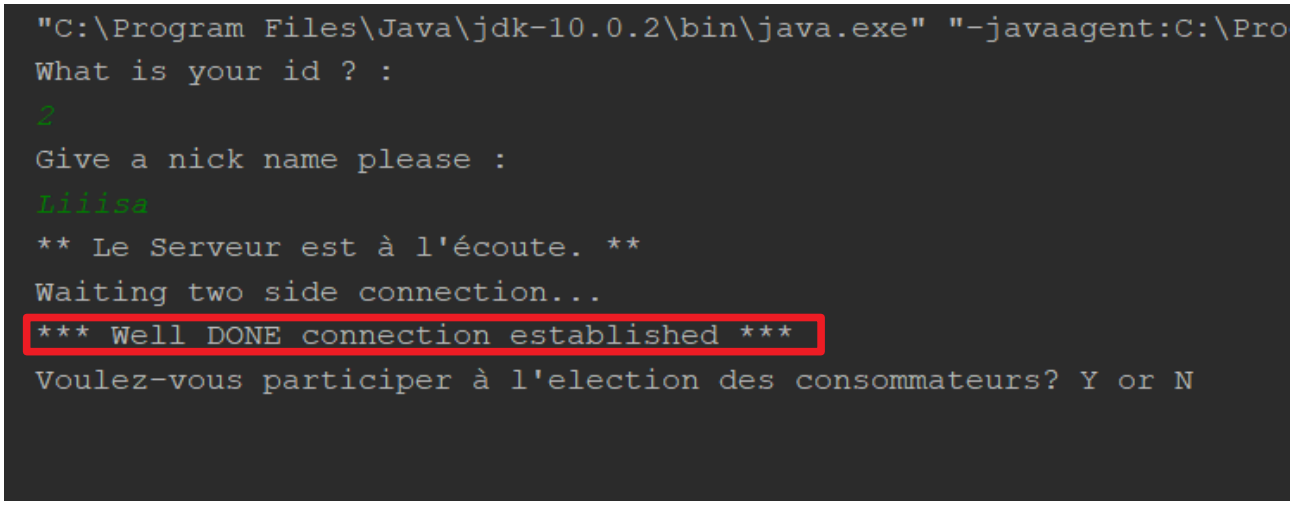
Il est à noter qu'il faut éviter d'utiliser le port 4010, qui lui est réservé a la communication consommateur/producteurs.

Dès le lancement de la classe App.java, nous obtenons pour chaque machine l'interface suivante :





Une fois que les machines se sont identifiées en se définissant un pseudo qui les définira plus tard dans la communication et l'échange de messages, place à la connexion entre les serveurs et la création de l'anneau.



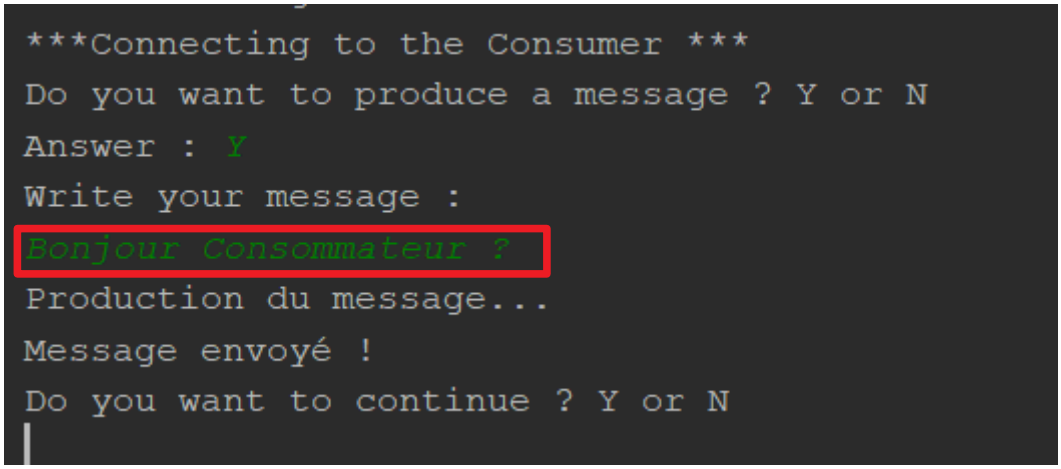
Le programme donne la main aux machine de choisir de participer ou non à l'élection du consommateur.

```
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe" "-javaagent:C:\Pro
What is your id ? :
1
Give a nick name please :
Axelito
** Le Serveur est à l'écoute. **
Waiting two side connection...
Waiting predecessor...
Waiting predecessor...
Waiting predecessor...
Waiting predecessor...
Waiting predecessor...
*** Well DONE connection established ***
Voulez-vous participer à l'election des consommateurs? Y or N
Y
*** L'election a été lancée ****
*** L'election de LeLann est en cours ***
*** Vous candidatez pour être elu ****
```

Une fois le choix fait, toutes les machines saurons qui a été élu.

```
Give a nick name please :
Axelito
** Le Serveur est à l'écoute. **
Waiting two side connection...
Waiting predecessor...
Waiting predecessor...
Waiting predecessor...
Waiting predecessor...
Waiting predecessor...
*** Well DONE connection established ***
Voulez-vous participer à l'election des consommateurs
Y
*** L'election a été lancée ****
*** L'election de LeLann est en cours ***
*** Vous candidatez pour être elu ****
Le chef a été elu il correspond a : 1
** Le Serveur est à l'écoute. **
*** Well Done; all connection etablished ***
```

Après cela, les producteurs auront la main pour envoyer les messages.



Et on retrouve ce message sur l'interface du consommateur avec le pseudo du producteur ayant envoyé le message.