Sommaire :

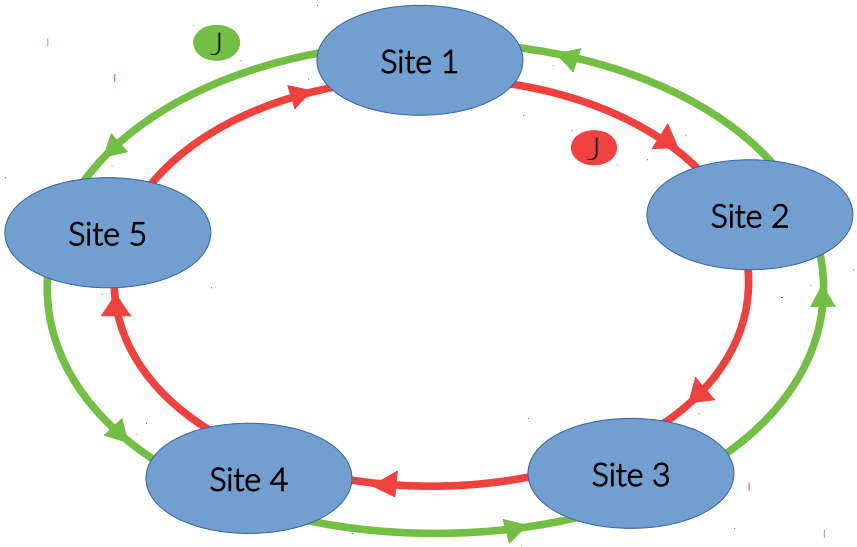
Introduction :

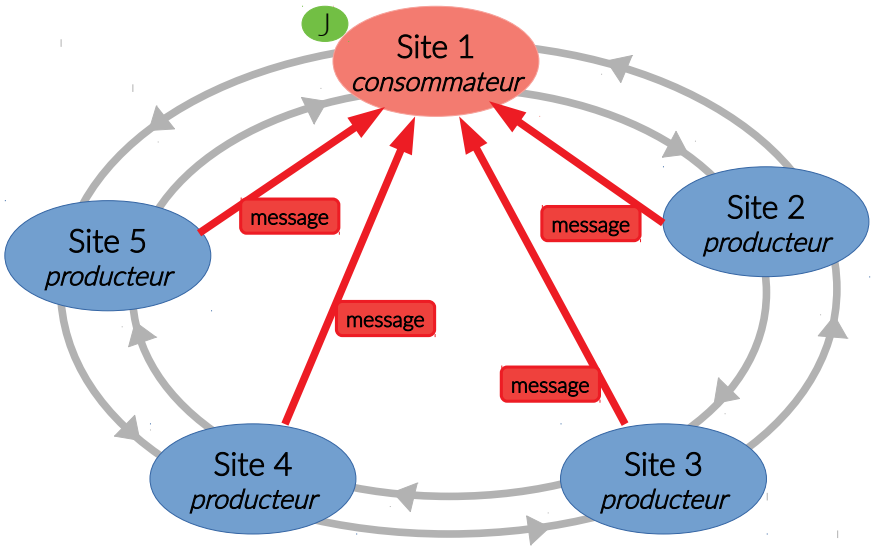
Dans le cadre du module algorithmes et systèmes répartis, il nous a été demandé d'implémenter le concept des producteurs/consommateur.
Pour notre cas, le système réparti implémenté se résume à la couche service, et donc peut être relié à toute application de la couche applicative utilisant le concept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





- 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 :

- Diagramme de classes :

- Justification des choix techniques :
  - Justification de l'architecture en anneau et du contexte statique :

    Le choix s'est porté sur une architecture en anneau en premier lieu, sur laquelle vont s'ajouter des canaux qui seront présentés dans la suite du rapport.
    Tout d'abord, nous avons un nombre de machines définies dans le contexte du réseau dans une liste qui contient les adresses IP des machines et leur ports d'échange.
    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.
    - Nous supposons également 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 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.

- (rajouter ce que vous voyez nécessaire).

Algorithmes répartis d'élection :

- Algorithme de Lelann :

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

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

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
```