Université Larbi Ben M'Hidi
Faculté des sciences exactes et des sciences de la nature et de la vie
Département M.I. - 1ière Année Master Architectures Distribuées

16 Jan. 2025
Dr. Kouah S.

Contrôle d'Ingénierie des Logiciels Distribués

Questions de Compréhension (8 Points)

1. Expliquez les concepts de **"transparence"** et **"hétérogénéité"** dans les systèmes distribués.
2. Lequel des éléments suivants n'est PAS une caractéristique des systèmes distribués ?
   a) Transparence
   b) Homogénéité
   c) Hétérogénéité
   d) Concurrence
3. Quel est le but principal du middleware dans les systèmes distribués ?
   a) Agir comme un système de gestion de base de données
   b) Faciliter la communication entre différents systèmes
   c) Assurer une horloge globale pour tous les nœuds
   d) Garantir l'homogénéité entre les nœuds
4. Mettez en correspondance les termes suivants avec leurs descriptions :

a)	Middleware	**1.**	1) Couche facilitant la communication entre différents systèmes
b)	Transparence	**2.**	2) Systèmes avec diverses configurations matérielles et logicielles
c)	Hétérogénéité	**3.**	3) Systèmes apparaissant comme une entité unique à l'utilisateur

5. Quels sont les principaux modèles de communication dans les systèmes distribués, vus en cours ?
6. Qu'est-ce qu'un système distribué, donner deux exemples de systèmes distribues ?
7. Quelle est la différence principale ente l'automate de Muller et l'automate de Büchi ? expliquer leur rôle ?

Exercice 1 (9 Points) :

On veut spécifiez un scénario où un client communique avec deux serveurs, tel que : Le client envoie une requête. Les deux serveurs répondent simultanément avec leurs résultats. Le client combine les deux réponses pour produire un résultat final.

Exemple d'un scénario détaillé :

- **Client** : Le client envoie une requête contenant une donnée **data** via la porte **req**. Le client attend deux réponses, **val1** et **val2**, provenant respectivement du serveur 1 et du serveur 2.
- Une fois les deux réponses reçues, le client calcule un **résultat final (val1 + val2)** qu'il offre via la porte **result**.
- **Serveur 1** : Le serveur 1 attend une donnée **x** via la porte **req**. Il effectue un traitement simple (x * 2), et renvoie cette valeur sur le canal **resp1**.
- **Serveur 2** : Le serveur 2 attend une donnée **y** via la porte **req**. Il effectue un traitement différent (y + 5), et renvoie cette valeur sur la porte **resp2**.

1. Modélisez ce scenario d'interactions en LOTOS.
2. Comment utiliser le langage LOTOS pour spécifier un système distribue ? Expliquez les principaux opérateurs.
3. Transformer la spécification LOTOS précédente en une spécification au moyen d'automates, en utilisant les annotations suivantes pour les primitives d'envoi et de réception de messages sur une porte « a » :
   a : ! x représente l'envoi de la valeur x sur le canal a.
   a : ? x représente l'attente d'une valeur x sur le canal a.
4. Donner deux propriétés formelles pour l'automate résultant, une en LTL, l'autre en CTL.
5. Comment spécifier un système distribué au moyen d'automates ?

Exercice 2 (3 Points) :

1. Expliquez le rôle des classes Java suivantes : InetAddress, Thread, ServerSocket et Socket.

2. Expliquez brièvement comment créer une application basée sur RMI.

3. Comment pouvons-nous créer des threads en Java, expliquez tous les cas possibles ?

Bon courage

Université Larbi Ben M'Hidi
Faculté des sciences exactes et des sciences de la nature et de la vie
Département M.I. - 1ière Année Master Architectures Distribuées

16 Jan. 2025
Dr. Kouah S.

Corrigé-Type du Contrôle d'Ingénierie des Logiciels Distribués

Questions de Compréhension (8 Points)

1. La **transparence** dans les systèmes distribués est la propriété qui permet aux utilisateurs et aux applications de ne pas être conscients de la répartition des ressources et des services. L'objectif est de masquer la complexité de la répartition et de rendre le système distribué perçu comme un système centralisé.
   L'hétérogénéité dans les systèmes distribués fait référence à la diversité des composants du système, tels que les architectures matérielles, les systèmes d'exploitation, les langages de programmation et les protocoles de communication. **(2 Pts)**
2. b) Homogénéité **(0.5)**
3. b) Faciliter la communication entre différents systèmes (**0.5 Pt)**
4. (1, A), (2, C), (3, B) **(0.5 Pt)**
5. Les principaux modèles de communication dans les systèmes distribués sont : **(0.5 ×4 Pts)**
   - Communication par messages
   - Communication par flux
   - Communication par mémoire partagée
   - Appel de procédure à distance (RPC)
   - Communication par événements et notifications
   - Invocation de méthodes à distance
   - Communication de groupe
6. Un système distribué est un ensemble de nœuds autonomes, connectés via un réseau, qui coopèrent pour atteindre un objectif commun tout en apparaissant comme un système unique pour l'utilisateur. **(1)**
   Exemples **:** systèmes de messagerie en ligne, systèmes de gestion de fichiers distribués,… **(0.75 Pt)**
7. L'automate de Büchi et l'automate de Muller sont équivalents a un certain niveau d'abstraction. L'automate de Büchi est généralement non déterministe, alors que l'automate de Muller peut être déterministe. Les deux automates sont essentiels en logique temporelle et en vérification formelle pour analyser et modéliser des comportements infinis dans des systèmes distribués. **(0.75 Pt)**

Exercice 1 :

1. Modélisation en LOTOS

```
process ClientServerInteraction[req, resp1, resp2, result] : noexit :=      (1 Pt)
  Client[req, resp1, resp2, result]
  |[req, resp1, resp2]|
  (Server1[req, resp1] ||| Server2[req, resp2])
where
  process Client[req, resp1, resp2, result] :=                              (1Pt)
    req ! data;            -- Le client envoie une requête contenant "data"
    resp1 ? val1;          -- Le client reçoit une réponse "val1" du serveur 1
    resp2 ? val2;          -- Le client reçoit une réponse "val2" du serveur 2
    result ! (val1 + val2); -- Le client combine les réponses et offre le résultat final
    exit
  endproc
  process Server1[req, resp1] :=                                            (0.75 Pt)
    req ? x;               -- Le serveur 1 attend une donnée "x"
    resp1 ! (x * 2);       -- Le serveur 1 renvoie "x * 2"
    exit
  endproc
  process Server2[req, resp2] :=                                            (0.75 Pt)
    req ? y;               -- Le serveur 2 attend une donnée "y"
    resp2 ! (y + 5);       -- Le serveur 2 renvoie "y + 5"
    exit
  endproc
endproc
```

2. Utilisation de LOTOS pour spécifier un système distribué (1 Pt)

- LOTOS est un langage puissant pour la modélisation des systèmes distribués. Considéré comme une algèbre de processus, il permet de représenter les différents composants d'un système distribué comme des processus distincts qui interagissent entre eux via des canaux de communication. Chaque nœud du système est modélisé comme un processus, vu comme étant une boîte noire, et qui peut être décomposé en sous-processus. Chaque processus est une séquence d'actions. Les interactions sont représentées par des "rendez-vous", c-à-d des synchronisations explicites entre processus, réalisées via des portes d'interaction spécifiant les canaux de communication utilisés.
- Les opérateurs de LOTOS permettent de spécifier la concurrence, l'ordre d'exécution et les différents types de synchronisation entre processus. Ils offrent également la possibilité de modéliser des comportements conditionnels, préemptifs, cachés, et alternatifs. Par

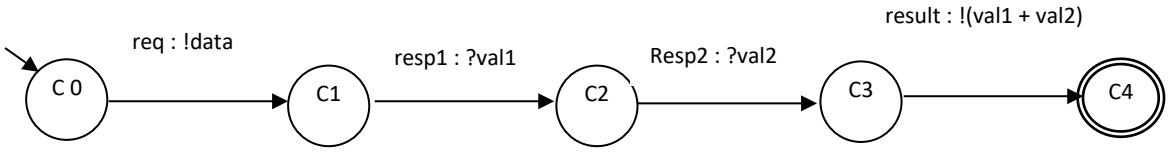
conséquent, LOTOS fournit une méthode formelle et modulaire pour spécifier les systèmes distribués, facilitant ainsi leur conception, leur vérification et leur analyse.

Opérateurs principaux : (0.75 Pt) (au moins 3 operateurs)

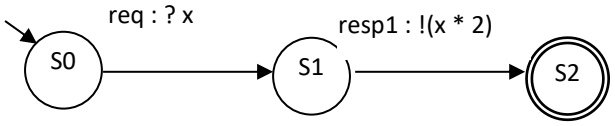
- |[canaux]| : Synchronisation sur les canaux spécifiés.
- ||| : Exécution concurrente sans synchronisation obligatoire.
- ? : Attente d'une donnée en entrée.
- ! : Offre d'une donnée en sortie.
- ; : Ordre séquentiel entre les actions.
- >> ordre séquentiel entre processus
- Stop : Fin du comportement du processus.
- Exit : terminaison avec succès
- [ ] : choix entre processus.
- [> : préemption
- Hide : cacher des portes

3. Spécification au moyen d'automates

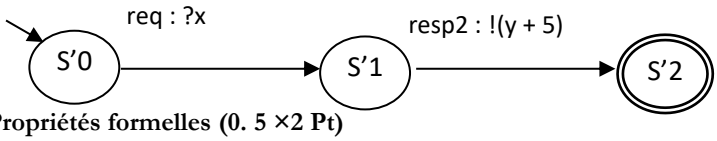
Automate Client : (1 Pt)



Automate Serveur 1 : (0.5 Pt)



Automate Serveur 2 : (0.5 Pt)



4. Propriétés formelles (0. 5 ×2 Pt)

a) **En LTL** : Chaque requête (req) conduit toujours à un résultat (result):
Formule : G(req→F(result)) *(Toujours, si une requête est émise, alors éventuellement un résultat est produit).*

b) **En CTL** : Il existe un chemin où la réponse du serveur 1 et du serveur 2 est reçue avant le calcul du résultat.
Formule : ∃G(resp1∧resp2→result) *(Il existe un chemin où, après les réponses des serveurs, le résultat est calculé).*

5. Spécification d'un système distribué par automate (0.75 Pt) : Un système distribué peut être spécifié avec des automates en :

- Décomposant chaque composant (tels que : client, serveur 1, serveur 2) en un automate individuel.
- Synchronisant leurs transitions sur des actions communes (ex. : req, resp1, resp2).
- Utilisant des annotations pour indiquer les envois (!) et réceptions (?) de messages.
- Chaque automate décrit un composant isolé mais interagissant avec les autres via des **transitions synchronisées**.

Exercice 2 :

1. (1 Pt)

- **InetAddress:** Représente une adresse. Permet d'identifier des hôtes réseau via leurs adresses IP ou noms de domaine.
- **Thread:** Représente un thread d'exécution dans un programme Java. Permet l'exécution simultanée de plusieurs tâches.
- **ServerSocket:** Utilisé pour créer un serveur écoutant les connexions clients entrantes. Écoute sur un port spécifique pour établir des connexions via des objets Socket.
- **Socket:** Représente une connexion côté client vers un serveur. Utilisé pour envoyer et recevoir des données entre un client et un serveur.

2. Création d'une application basée sur RMI: (1 Pt)

- **Définir une interface distante** : Créer une interface qui hérite de java.rmi.Remote.
- **Implémenter l'interface distante** : Créer une classe qui implémente l'interface distante.
- **Créer le serveur RMI** : Associer l'objet distant à un nom dans le registre RMI.
- **Créer le client RMI** : Rechercher l'objet distant dans le registre et appeler ses méthodes.
- **Exécuter l'application** : Compiler toutes les classes. Exécuter le serveur puis le client.

3. Création **de threads en Java:** Il existe **deux principales méthodes** pour créer des threads en Java : **(1 Pt)**

- **Étendre la classe Thread** : Créer une sous-classe de Thread et redéfinir la méthode run(), puis Démarrer le thread à l'aide de la méthode start().
- **Implémenter l'interface Runnable** : Créer une classe qui implémente Runnable et définir la méthode run(). Utiliser un objet Thread pour exécuter le Runnable.