A small synthesis about Interaction Models between distributed Objects

Java RMI and CORBA as concrete examples

Remote Method Invocation Protocol

- Java RMI and CORBA implement this protocol (RMI), also SOAP/XML-RPC
- What happens exactly in the normal situation (no failure, nor from the client, nor the server, nor the network)
  - Caller object : doOperation (calling method on an interface) which is blocking
  - Network: transports the request
  - Called object : gets the request, delegates it to the effective object, prepares transportation of the reply if any (or ACK)
  - Network: transports the reply (or signal to unblock the caller)
  - Reception of the reply and delivery to the calling object
- *Exactly-once* semantics (case of any Object-based Method I. in a centralized setting)
  - Means the called method is called exactly one time

Remote Method Invocation Protocol: failure case

- Caller object : doOperation (calling method on an interface) which is blocking. Several possible failure scenarios:

i. Network transports the request, but fails (eg if not TCP, or network conn. crashes), or called object has failed. Request does not reach the called object
   - Timeout from the caller process: several retries to send request
   - After expired timeout: signals an error to the caller (caller side)
     - Method has not been executed
- Network transports the request, eventually succeeds… One or several requests reach the called object :
   - Called object filters duplicate requests, ie only one method call executed

ii. Called object sends reply to the network which may fail
   - Timeout from the called process: several retries to send reply
   - After expired timeout: signals an error to the caller (done at caller side)
   - Method has been executed once (but caller does not know it)
- Network transports the reply, eventually succeeds… :
   - Caller receives a reply: → method has been executed one time

At most-once semantics from the caller viewpoint, if it receives a reply, it means the called method has been executed one time and no more than one time (it is the most costly protocol due to both sides retries, plus filter)
Case of JavaRMI and Corba synchronous method invocation model

Université Nice SOPHIA ANTIPOLIS

Remote Method Invocation Protocol: failure case

- Caller object : doOperation (calling method on an interface) which is blocking. Several possible failure scenarios:

i. Network transports the request, but fails … or called object has failed. Request does not reach the called object
   - Timeout from the caller process: several retries to send request
   - After expired timeout: signals an error to the caller (caller side)
     - Method has not been executed
   - Network transports the request, eventually succeeds… One or several requests reach the called object
     - Called object executes duplicate requests (no filtering)

ii. Called object sends reply(ies) to the network which may fail
   - Method has been executed but caller does not know it
   - Network transports the reply(ies), eventually succeeds… Caller receives first reply: → method has been executed one or several times

At least-once semantics : from the caller viewpoint, if it receives a reply, it means the called method has been called at least one time (traditional RPC systems, suited to stateless server).

It is less costly compared to the "at most once protocol" but additional overheads can come from the application layer (several executions of same method)

Remote Method Invocation Protocol: failure case

- Caller object : doOperation (calling method on an interface) which is blocking. Several possible failure scenarios:

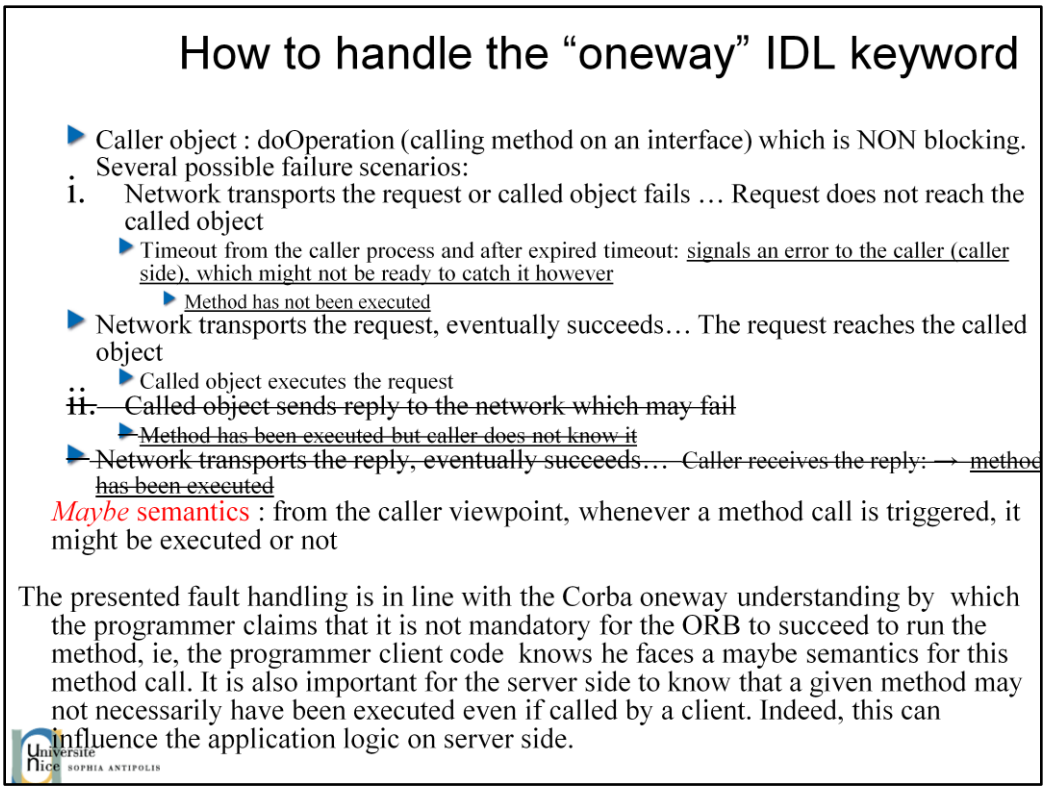
i. Network transports the request or called object fails … Request does not reach the called object
   - Timeout from the caller process and after expired timeout: signals an error to the caller (caller side)
     - Method has not been executed
- Network transports the request, eventually succeeds… The request reaches the called object
   - Called object executes the request

ii. Called object sends reply to the network which may fail
   - Method has been executed but caller does not know it
- Network transports the reply, eventually succeeds… Caller receives the reply: → method has been executed

This protocol does not include any fault-handling mechanism (not costly!)

Maybe semantics : from the caller viewpoint, whenever a method call is triggered, it might be executed or not (eg Corba oneway)

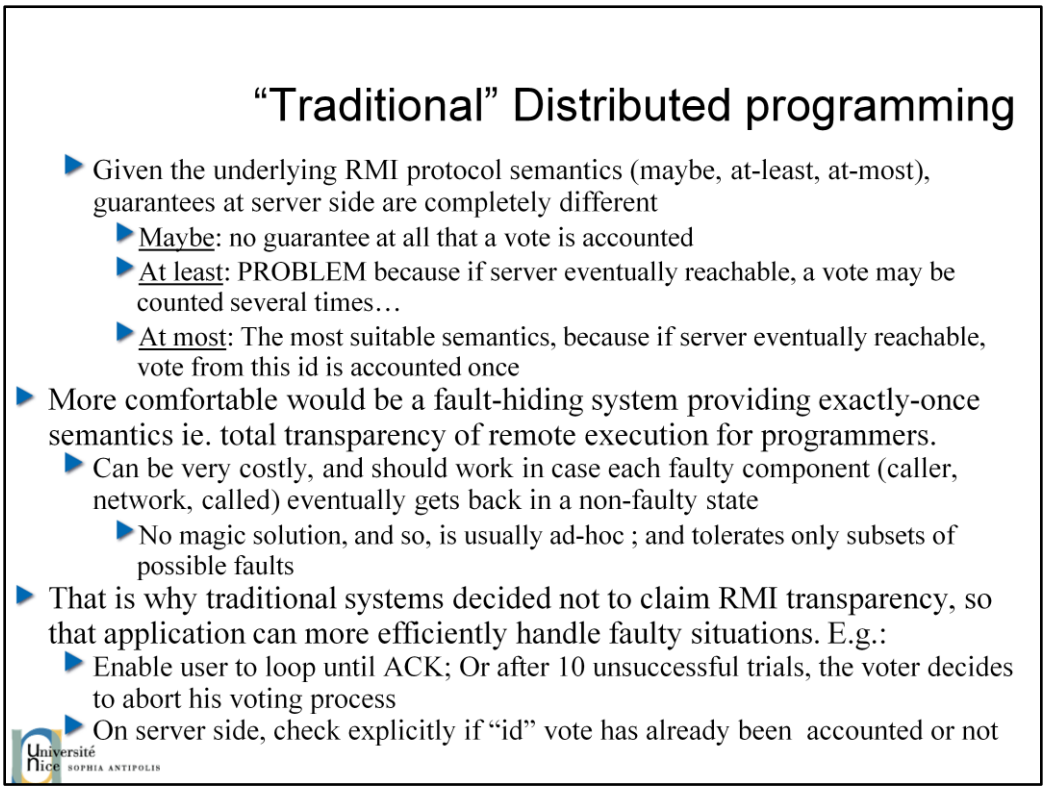
A propos de Corba oneway, voir les détails donnés dans le transparent suivant.



La spec CORBA de oneway dit juste que ce type d’opération n’aura ni retour ni exception métier. Charge à l’ORB de le supporter de différentes façons plus ou moins couteuses et pertinentes. Par exemple, JacORB démarre un thread pour que l’appelant soit non bloquant. Dans ce cas, le code client perçoit une sémantique maybe , même si, ensuite, la thread qui est branchée à l’ORB peut garantir une sémantique plus riche. Si par exemple, la thread se repose sur l’ORB de SUN il est fort possible que ce soit une sémantique at most once. En effet…IDLJ (mapping IDL to Java) transforme une op. oneway en un appel bloquant synchrone au niveau du stub, donc, l’appelant est quand même bloqué et peut donc savoir si l’appel a échoué, et si il ne reçoit pas d’exceptions du tout, il sait même que l’appel a réussi. Dans ce cas, la sémantique de l’op oneway sans remontée d’erreur est donc la même que celle que l’ORB offre pour une op non oneway., qui est pour les ORB et JavaRMI une sémantique at most once.

“Traditional” Distributed programming

- Technical additional duties (e.g. extends java.rmi.Remote, etc)
- More importantly: requires additional logic in the application (for handling failure situations).
- Imagine a distributed voting system:
  - Each voter (client) has a unique id
  - He votes once by sending a vote to the remote voting system (if failure, already gone…) and he is given only one chance (i.e., the web front-end application is storing the fact that he has voted with his id)
  - Even if abstention is a very common phenomena, the voting system should make its best to account votes!
- Assume the application is not ready to be changed in order to handle faulty scenarios,
  - Receiving a reply (ACK) at client side: OK, nothing special to program
  - Receiving a signal error at client side:
    - Too bad, but client already doing something else



Si on implémente cette spec. de système de vote, sur un middleware supportant les appels de méthode distantes, il faut bien faire attention à la sémantique offerte pour raisonner sur le comportement obtenu côté serveur. Si la sem. est maybe, on n'a même aucune assurance que le vote sera pris en compte; si la sem. est at least, le vote côté serveur risque d'avoir été compté plusieurs fois (puisque la méthode déclenchée à distance peut l'être plusieurs fois par le middleware); si la sem. est at most, là, le code du serveur est correct puisque un vote n'est déclenché qu'une seule fois. Mais, au prix d'un surcoût au sein du middleware qui aura géré les fautes en évitant les duplicats (en filtrant). Malgré cela, l'appli cliente devrait tout de même être modifiée pour gérer les erreurs signalant un echec dans le vote. C'est pour cela que le plus confortable ce serait d'avoir une sémantique exactly one. Mais on sait que c'est très coûteux. C'est pour cette raison que les middleware n'offrent jamais exactly once, mais, au mieux at most once. Ce qui, en pratique, se traduit par l'obligation du programmeur de distinguer les cas problématiques du cas normal. D'où le fait que les méthodes RMI ne sont jamais totalement transparentes. Ce serait une illusion trop coûteuse. Et même si non transparent, c'est parfois plus efficient de reposer sur une sémantique moins puissante que at most once, par exemple, at least est suffisant si on implémente le test 'est ce que cet id a déjà voté', plutôt qu'avoir un middleware qui filtre les duplicats de requêtes quelles qu'elles soient.

Summary: featured behaviour of underlying middleware offering remote method invocation

Caller Knowledge about occurrence number of Method execution in remote	NO ERROR signaled to the caller	ERROR signaled to the caller
Exactly once semantics	**1**	Not applicable, however caller is stuck
At most once semantics	1	0 or 1
At least once semantics	1 or +	0, 1 or +
Maybe semantics	0 or 1	Not always applicable and if yes: 0 or 1

- Bricks used for implementing RMI « advanced » semantics
  - Retry sending the request message whenever no answer
  - Filter duplicate requests
  - Retransmit reply thanks to an history, without re-execution of method
  - Concrete examples: WS-ReliableMessaging specification.