UFR 919 Sciences de l'ingénieur
Master Informatique – Spécialité STL
UE CPS – Composants

Système minimal de publication/souscription en BCM4Java

Jacques Malenfant
professeur des universités

version 1.2 du 10/02/2020

Résumé

L'objectif de ce projet est de comprendre l'utilisation des concepts, des approches et des techniques du génie logiciel des architectures à base de composants concurrents et répartis. Cet objectif sera poursuivi par le développement d'un système de publication/souscription. Le projet se développe en plusieurs étapes permettant d'aborder successivement l'implantation d'un système minimal fonctionnel, puis d'augmenter sa performance par l'utilisation de *pools de threads*, d'en assurer la répartition sur plusieurs processus (voire sur plusieurs ordinateurs en réseau) et enfin d'en mesurer la performance pour affiner ses paramètres de configuration au déploiement. Ces étapes se veulent représentatives d'un projet de développement de bout-en-bout.

1 Généralités

Un système de communication par publication/souscription est un intergiciel permettant de créer, envoyer, recevoir et consommer des messages (voir figure 1). L'organe principal du système est le ***courtier*** (en anglais ***broker***) qui gère la transmission des messages entre ses clients. Les clients peuvent endosser deux rôles (non exclusifs l'un de l'autre) : le rôle de ***publieur*** (***publisher***) qui crée et publie des messages et le rôle de ***souscripteur*** (***subscriber***) qui s'abonne, reçoit des messages et les consomme (les lit, les traite, etc.).

Le modèle de communication par publication/souscription est un modèle de multiplicité 1:N, c'est-à-dire qu'un message publié par un producteur peut être reçu par N consommateurs ($N \geq 0$). Dans ce modèle, les producteurs ne connaissent pas les consommateurs de leurs messages ni les consommateurs le producteur de ceux qu'ils consomment ; le courtier est chargé de faire le lien entre producteurs et consommateurs en recevant les messages publiés et en les acheminant aux souscripteurs. La communication est *asynchrone* : le producteur

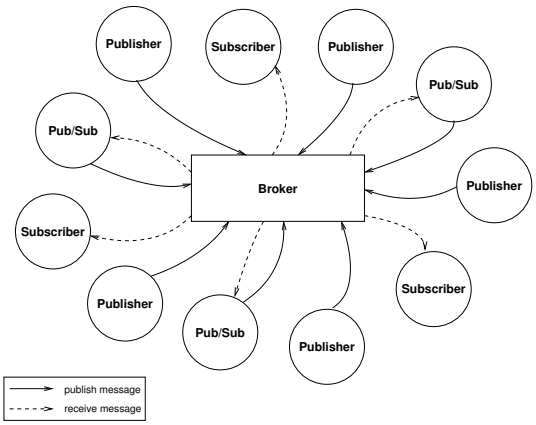

FIGURE 1 – Fonctionnement général d'un système de publication/souscription.

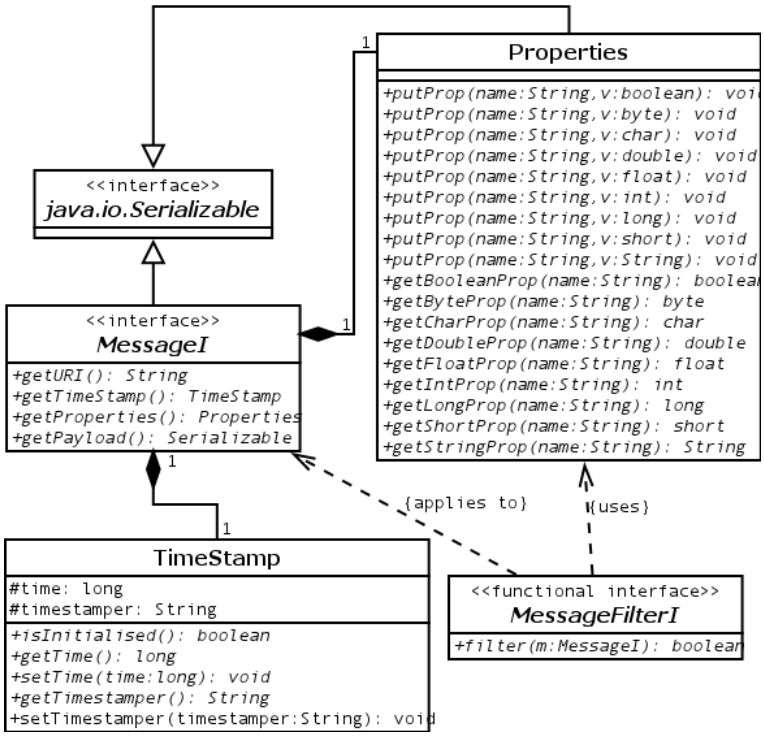

FIGURE 2 – Diagramme de classes présentant les messages.

qui publie un message continue immédiatement son exécution et ne reçoit pas de réponse (de résultat). Le consommateur reçoit les messages correspondant à ses abonnements par un mécanisme similaire à la notification. Le courtier garantit la livraison des messages et le fait que chaque consommateur reçoit chacun des messages qui lui revient une et une seule fois.

La publication et la souscription sont organisées autour de ***sujets*** (*topics*) : les producteurs publient des messages sur des sujets et les consommateurs souscrivent à la réception de messages sur des sujets. Les consommateurs peuvent, à la souscription, fournir un ***filtre*** permettant de préciser parmi les messages publiés sur les sujet auxquels ils souscrivent lesquels les intéressent. Les filtres posent des conditions à satisfaire par un message pour que l'abonné le reçoive ; avant de détailler ces filtres, il faut examiner la forme des messages.

2 Messages

2.1 Définition

Un message publiable et acheminable par le système est un objet Java sérialisable dont la classe implante (au sens de Java) l'interface `MessageI` (voir figure 2). Il comporte quatre parties :

1. Un identifiant unique de message de type `String` qui permet de distinguer de manière unique chaque message (cet identifiant est utile principalement pour la mise au point des programmes).
2. Une estampille de temps (`TimeStamp`) en deux parties :
   (a) un entier `long` représentant le temps système Unix auquel le message a été publié et
   (b) l'identification du dateur sous la forme d'une valeur `String` (le nom de l'hôte ou son adresse IP).
3. Un ensemble de propriétés (`Properties`) ayant chacune un nom et une valeur pouvant être soit d'un type primitif Java (représenté par un objet de la classe correspondante par exemple, `Integer` pour `int`) ou du type `String`.
4. Un contenu (ou charge utile) pouvant être n'importe quel objet Java sérialisable.

2.2 Filtrage

Lors de la souscription, les consommateurs de messages peuvent fournir un filtre qui va sélectionner parmi les messages publiés sur le sujet ceux qui vont lui être transmis. Le filtre est un objet instance d'une classe ou d'une lambda experssion implantant l'interface fonctionnelle Java 8.0 `MessageFilterI` qui déclare une fonction booléenne `filter` qui prend en argument un message et qui retourne vrai si les valeurs des propriétés de ce message satisfont des contraintes qu'elle impose et faux sinon. Les contraintes s'expriment sous la forme d'expressions booléennes Java programmées dans le corps de la méthode ou de la lambda expression implantant la signature `filter`. Les conditions ne peuvent porter que sur les propriétés du message accessibles via les méthodes de la classe `Properties` (voir fig. 2) ou ses propriétés intrinsèques comme l'estampille de temps ou la présence/absence d'une charge utile.

2.3 Gestion des messages par le courtier

Le courtier assure la gestion des messages publiés :

— leur stockage depuis la publication jusqu'à la réception par tous les souscripteurs intéressés,
— leur destruction lorsqu'ils ont été reçus par tous les souscripteurs intéressés.

3 Publication

3.1 Gestion des sujets

Le système organise la publication des messages et la souscription à leur réception autour de sujets (*topics*). Un sujet est défini à l'initiative d'un producteur par une URI, c'est à dire une chaîne de caractères formant un identifiant unique sous lequel des messages seront transmis entre producteurs et consommateurs. Le courtier implante des opérations de gestion des sujets par l'interface `ManagementImplementationI` (voir figure 3) :

— création d'un sujet à partir d'une URI,
— destruction d'un sujet d'une URI donnée (un sujet détruit ne permet plus la publication, mais tous les messages déjà publiés doivent avoir été livrés avant la destruction effective du sujet),
— vérification qu'une URI correspond à un sujet existant pour le courtier,
— obtention de la liste des URI définissant des sujets existants pour le courtier,
— obtention de l'URI d'un port offrant l'interface `PublicationCI` auquel le composant publieur va pouvoir se connecter pour publier ses messages (voir ci-après la publication des messages).

Ces services sont offerts et requis via l'interface de composants `ManagementCI` qui applique, comme les autres interfaces de composants données ici, la patron de conception partageant les signatures d'une interface d'implantation (`ManagementImplementationI`) avec l'interface de composant.

3.2 Publication des messages

Les producteurs peuvent publier des messages sur n'importe quel sujet existant sur un courtier donné ou sur un nouveau sujet, ce qui entraîne la création de ce sujet. La publication se fait via des méthodes `publish` par appel asynchrone. La principale de ces méthodes prend en argument l'URI du sujet sur lequel la publication doit se faire et le message à publier. La publication peut également se faire sur plusieurs sujets à la fois et plusieurs messages peuvent aussi être publiés en même temps, ce qui est autorisé par les autres méthodes `publish`.

Les services de publication sont offerts et requis via l'interface de composants `PublicationCI`. La méthode `getPublicationPortURI` de l'interface `ManagementCI` permet aux composants souhaitant publier d'obtenir l'URI d'un port entrant du courtier sur lequel ils vont pouvoir publier les messages.

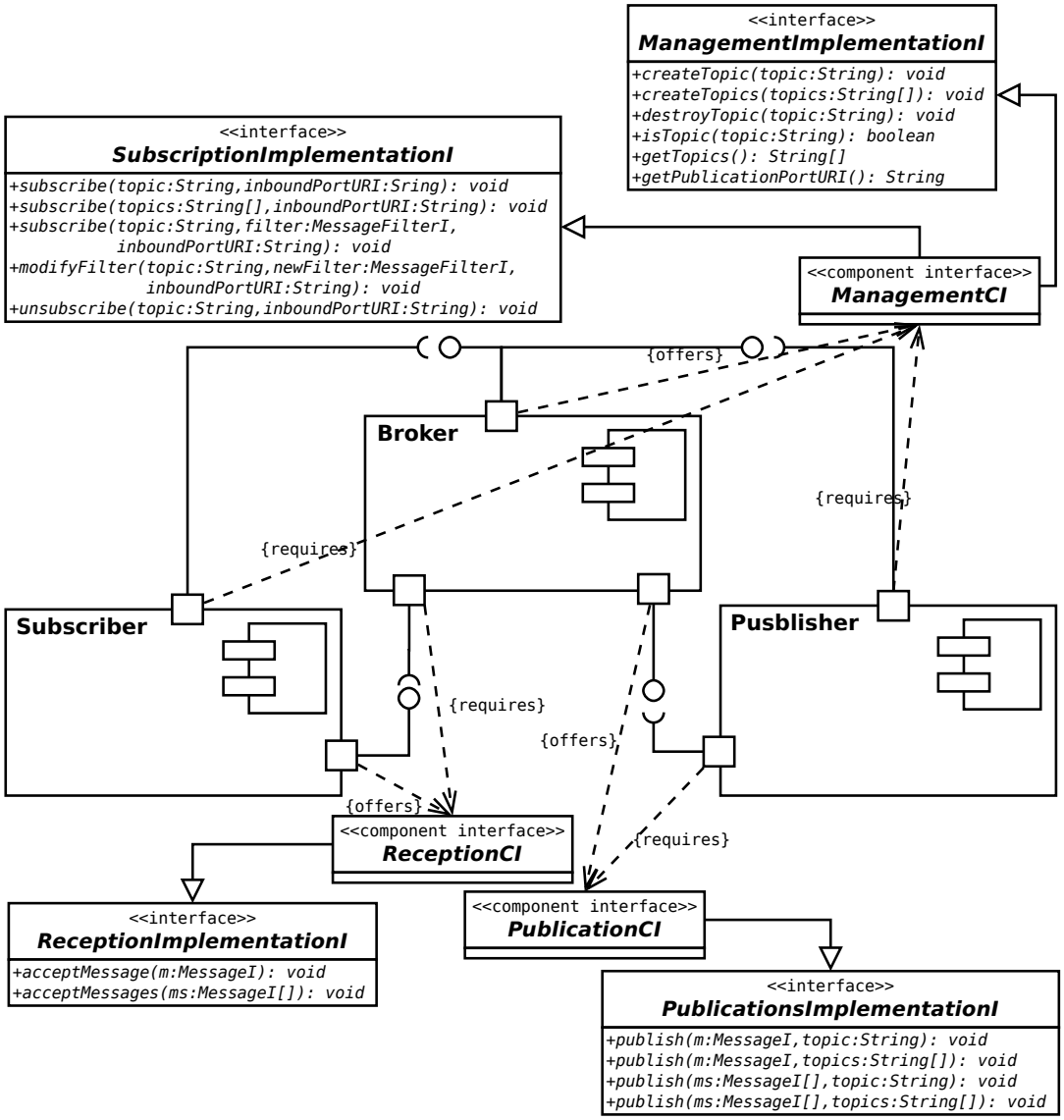

FIGURE 3 – Diagramme partiel de classes et de composants de la partie publication/souscription montrant les liens entre les composants publieurs, les composants souscriveurs (et récepteurs) et le composant courtier (*broker*) qui gère les échanges de messages.

4 Souscription

4.1 Gestion des souscriptions

Le courtier implante des opérations de souscription déclarées par l'interface `SubscriptionImplementationI` :

— souscription à la réception des messages publiés sur un sujet donné avec un filtre donné reçu via un port entrant donné,
— modification du filtre associé à une souscription,
— fin d'une souscription.

Les souscripteurs peuvent s'abonner à n'importe quel sujet existant sur un courtier ou à un nouveau sujet, ce qui entraîne la création de ce sujet.

Ces services sont offerts et requis via l'interface de composants `ManagementCI` qui, pour cela, étend l'interface `SubscriptionImplementationI`.

4.2 Réception des messages

Les consommateurs reçoivent les messages selon une approche qui rappelle un patron observateur, c'est-à-dire qu'ils sont appelés par le courtier sur un port entrant proposé lors de la souscription. Le port entrant offre l'interface `ReceptionCI` et il répercute l'appel sur le composant consommateur de telle manière que la réception soit traitée de manière asynchrone (interface `ReceptionImplementationI`).

Un consommateur reçoit une et une seule fois tous les messages et que les messages publiés sur le sujet auquel il a souscrit et qui satisfont le filtre qu'il impose au moment de la publication du message et ce depuis le début de sa souscription jusqu'à la fin de cette dernière. Les messages publiés sur le sujet avant le début de la souscription et après la fin de cette dernière ne sont pas reçus. Le système ne garantit pas que l'ordre de réception des messages par un consommateur soit le même que l'ordre de publication de ces messages. Il ne garantit pas non plus que tous les consommateurs recevant les mêmes messages vont les recevoir dans le même ordre.

5 Étape 1 — Première version fonctionnelle

La première étape du projet consiste à implanter des types de composants courtier, publieur et souscripteur en BCM4Java à partir de la spécification exposée dans les sections précédentes puis des les tester pour leurs fonctionnalités.

À partir du diagramme de la figure 3 et des interfaces qu'elle déclare, vous devez définir :

— toutes les ports entrants et sortants ainsi que les connecteurs utilisés pour gérer les appels entre le courtier et les clients ;
— un type de composant courtier avec toutes les interfaces requises et offertes, les ports entrants et sortants ainsi que les connecteurs qu'il utilise ;
— des greffons (*plug-ins*) pour chacun des rôles de producteur et consommateur à installer par les composants souhaitant utiliser la communication par publication/souscription.

Vous devez également programmer des tests unitaires et d'intégration démontrant le bon fonctionnement de ces éléments.

Modalités de réalisation

À titre de suggestion, en vous inspirant des exemples fournis avec la bibliothèque BCM4Java, vous devriez procéder dans l'ordre suivant :

1. Définir les interfaces de composants, les ports, les connecteurs et, dans une première version simplifiée, les trois types de composants (courtier, publieur, souscripteur). Dans cette première version, le courtier devrait immédiatement renvoyer tout message reçu à un souscripteur, sans utiliser le mécanisme des sujets, ni celui des abonnements, ni de la mémorisation temporaire des messages, ni du filtrage. L'objectif est simplement de pouvoir vérifier l'architecture des composants, leur interconnexion et la bonne livraison des messages.
2. Ajouter progressivement les différentes fonctionnalités (sujets, abonnements, mémorisation, filtrage, ...).
3. Factoriser les fonctionnalités des publieurs et des souscripteurs dans des greffons.

6 Étape 2 — Parallélisme au sein du courtier

En exécution sur une seule machine virtuelle Java, le système de publication/souscription utilise un seul composant courtier par lequel transitent tous les messages. La performance d'un système de publication/souscription est une de ses qualités majeures. La réalisation du projet doit donc en tenir compte pour assurer de bonnes performances au passage à l'échelle, c'est à dire lorsque le nombre de message publiés par seconde, le nombre de souscripteurs et la complexité des filtres croissent.

Pour atteindre cet objectif, l'implantation du courtier doit utiliser autant de *threads* que nécessaire en lançant des *threads* séparés pour découpler les traitements longs du noyau de son fonctionnement. Par exemple, le filtrage

et la livraison des messages aux souscripteurs sont des opérations qui prennent du temps. Jamais le courtier ne devrait bloquer l'exécution d'un publieur pendant qu'il filtre et livre des messages. De même, jamais le courtier ne devrait être complètement bloqué par un souscripteur qui ne rendrait pas la main lorsqu'on lui livre un message.

7 Étape 3 — Répartition du courtier

En exécution sur plusieurs machines virtuelles, la latence des appels entre JVM fait qu'il est plus efficace d'utiliser plusieurs composants courtiers, un par machine virtuelle, reliés les uns aux autres. Ces composants courtier ne sont pas indépendants les uns des autres mais gérent ensemble tous les composants utilisateurs, leurs souscriptions, leurs publications et la transmission des messages, y compris entre composants s'exécutant sur différentes machines virtuelles. Le courtier au sens logique est donc réparti entre les JVM exécutant l'application BCM.

Lors du lancement de l'application, un composant courtier est créé sur chaque machine virtuelle Java (JVM) utilisée par l'application BCM. Ces composants servent de point d'entrée au système de publication/souscription pour les composants clients de leur propre JVM. Ils sont liés entre eux pour acheminer les messages des producteurs d'une JVM aux consommateurs présents sur les autres JVM. De cette manière, un message publié sur une machine virtuelle A pour lequel il y a plusieurs souscripteurs sur une machine virtuelle B sera acheminé une seule fois sur B entre les deux composants courtiers puis redistribué en local à tous les souscripteurs concernés présents sur B (plutôt qu'acheminé entre machines virtuelles pour chaque souscripteur présent sur B).

8 Étape 4 — Tests de performance

Tout au long de son développement, le projet devra inclure des tests fonctionnels complets, incluant des tests unitaires implantés avec JUnit quand c'est possible et des tests d'intégration implantés comme des applications en BCM4Java. Pour cette quatrième étape, le projet devra également inclure des tests de performance (aussi implantés sous la forme d'applications BCM4Java).

Ces tests devront montrer comment se comporte le système de publication/souscription en fonction de la pression imposée par les clients, incluant un nombre croissant de messages publiés par seconde, un nombre croissant de souscripteurs et une complexité croissante des filtres utilisés par les souscripteurs. L'objectif de ces tests est de déterminer comment fixer les choix d'implantation (interconnexion des colposants courtiers répartis) et les paramètres de déploiement (nombre de *threads* utilisés dans chaque courtier pour exécuter ses différentes fonctionnalités – réception et stockage des messages publiés, traitement des souscriptions, filtrage et transmission des messages aux souscripteurs).

Les résultats de cette étape seront :
- un (petit) plan d'expérimentation fixant les cas à traiter et les mesures de performance à prendre (à décrire dans la Javadoc du paquetage regroupant tous ces tests) ;
- des résultats (par exemple sous la forme de tableaux) des mesures de performance obtenues (à présenter lors de la soutenance finale) ;
- des recommandations faites aux installateurs du système de publication/souscription pour la bonne configuration des paramètres selon leur situation (à inclure dans la Javadoc de la classe implantant le composant courtier.

9 Modalités générales de réalisation et calendrier des évaluations

- Le projet se fait **obligatoirement** en **équipe de deux étudiant·e·s**. Tous les fichiers sources du projet doivent comporter les noms (balise `authors`) de tous les auteurs en Javadoc. Lors de sa formation, chaque équipe devra se donner un nom et me le transmettre avec les noms des étudiant·e·s la formant au plus tard le **9 février 2020** à minuit.
- Le projet doit être réalisé avec Java SE 8. Attention, peu importe le système d'exploitation sur lequel vous travaillez, il faudra que votre projet s'exécute correctement sous Eclipse et sous Mac Os X/Unix (que j'utilise et sur lequel vous devrez me faire vos soutenances).
- L'évaluation comportera quatre épreuves : deux audits intermédiaires, une soutenance à mi-semestre et une finale, ces dernières accompagnées d'un rendu de code et de documentation. Ces épreuves se

dérouleront selon les modalités suivantes :

1. Les deux audits intermédiaires dureront 5 à 10 minutes (par équipe) et se dérouleront pendant les séances de TD/TME. Le premier audit se tiendra pendant les séances 2, 3 et 4 et il examinera plus particulièrement votre avancement sur l'étape 1. Le second audit se déroulera durant les séances 7, 8 et 9 et il examinera plus particulièrement votre avancement sur l'étape 3. Ils compteront chacun pour 5% de la note finale de l'UE.
2. La **soutenance à mi-parcours** d'une durée de 20 minutes portera sur l'atteinte des objectifs des *deux premières étapes* du projet. Elle se tiendra pendant la semaine des premiers examens répartis du **16 au 20 mars 2020** selon un ordre de passage et des créneaux qui seront annoncés sur le site de l'UE. Elle comptera pour 35% de la note finale de l'UE. Elle comportera une discussion des réalisations pendant une quinzaine de minutes (devant l'écran sous Eclipse) et une courte démonstration de cinq minutes sur un ordinateur fourni (sous Mac Os X) à partir du rendu du projet. Les rendus à mi-parcours se feront le **dimanche 15 mars 2020 à minuit** au plus tard (des pénalités de retard seront appliquées).
3. La **soutenance finale** d'une durée de 30 minutes portera sur l'ensemble du projet mais avec un accent sur les *troisième et quatrième étapes*. Elle aura lieu dans la semaine des seconds examens répartis du **18 au 22 mai 2020** selon un ordre de passage et des créneaux qui seront annoncés sur le site de l'UE. Elle comptera pour 55% de la note finale de l'UE. Elle comportera une présentation d'une douzaine de minutes (en utilisant des transparents), une discussion d'une dizaine de minutes également devant écran sur les réalisations suivie d'une démonstration sur un ordinateur fourni (sous Mac Os X) à partir du rendu final du projet. Les rendus finaux se feront le **dimanche 17 mai 2020 à minuit** au plus tard (des pénalités de retard seront appliquées).

— Lors des soutenances, les points suivants seront évalués :
  — le respect du cahier des charges et la qualité de votre programmation ;
  — l'exécution correcte de tests unitaires (JUnit pour les classes et les objets Java, scénario de tests des composants) ;
  — l'exécution correcte de tests d'intégration mettant en œuvre des composants de tous les types ;
  — la qualité et l'exécution correcte des scénarios de tests de performance, conçus pour mettre à l'épreuve les choix d'implantation versus la montée en charge ;
  — la qualité de votre code (votre code doit être commenté, être lisible - choix pertinents des identifiants, ...– et correctement présenté – indentation, ...– etc.) ;
  — la qualité de votre plan d'expérimentation et tests de performance (couverture de différents cas, isolation des effets pour indetifier leurs impacts sur la performance globale, etc.) ;
  — la qualité de la documentation (vos rendus devront inclure une *documentation Javadoc* des différents paquetages et classes de votre projet générée et incluse dans votre livraison dans un répertoire `doc` au même niveau que votre répertoire `src`).
— Bien que les audits et les soutenances se fassent par équipe, l'évaluation reste à chaque fois **individuelle**. Lors des audits et des soutenances, *chaque étudiant·e* devra se montrer capable d'expliquer différentes parties du projet, et selon la qualité de ses explications et de ses réponses, sa note peut être supérieure, égale ou inférieure à celle de l'autre membre de son équipe.
— Lors des soutenances, **tout retard** non justifié d'un·e ou des membres de l'équipe de plus d'un tiers de la durée de la soutenance (7 minutes à la soutenance à mi-semestre, 10 minutes à la soutenance finale) entraînera une **absence** et une note de 0 attribuée au(x) membre(s) retardataire(s) pour l'épreuve concernée. Si un·e des membres d'une équipe arrive à l'heure ou avec un retard de moins d'un tiers de la durée de la soutenance, il·elle passera l'épreuve seul·e.
— Le rendu à mi-parcours et le rendu final se font sous la forme d'une archive `tgz` si vous travaillez sous Unix ou `zip` si vous travaillez sous Windows que vous m'enverrez à `Jacques.Malenfant@lip6.fr` comme attachement fait proprement avec votre programme de gestion de courrier préféré ou encore par téléchargement avec un lien envoyé par courrier électronique (en lieu et place du fichier). Donnez pour nom au répertoire de projet et à votre archive celui de votre équipe (ex. : équipe LionDeBelfort, répertoire de projet `LionDeBelfort` et archive `LionDeBelfort.tgz`).
— **Tout manquement à ces règles élémentaires entraînera une pénalité dans la note des épreuves concernées !**
— Pour la deuxième session, si elle s'avérait nécessaire, elle consiste à poursuivre le développement du projet pour résoudre ses insuffisances constatées à la première session et donnera lieu à un rendu du code et de documentation puis à une soutenance dont les dates seront déterminées en fonction du calendrier du master.