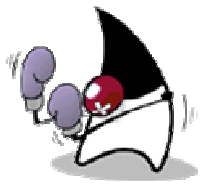
Laboratoire de Réseaux et technologie Internet (TCP-UDP/IP & HTTP en C/C++/Java) :

3ème Informatique de gestion
3ème Informatique et systèmes
opt. Informatique industrielle et Réseaux-télécommunications
2019-2020

Projet "**Happy Ferry Inpres**"

Claude Vilvens, Christophe Charlet, Sébastien Calmant
(Network programming team)



1. Préambule

L'Unité d'Enseignement "**Programmation réseaux, web et mobiles**" (10 ECTS - 135h) comporte des Activités d'apprentissage :

- AA: Réseaux et technologies Internet (dans toutes les options);
- AA: Programmation.Net (dans toutes les options);
- AA: Technologie de l'e-commerce et mobiles (informatique de gestion seulement);
- AA: Compléments de programmation réseaux (informatique réseaux-télécoms seulement).

Les travaux de programmation réseaux présentés ici constituent la description technique des travaux de laboratoire de l'AA "**Réseaux et technologie Internet**". Il s'agit ici de maîtriser les divers paradigmes de programmation réseau TCP/IP et HTTP dans les environnements UNIX et Windows, en utilisant les langages C/C++ et Java. Ces travaux ont été conçus de manière à pouvoir collaborer avec les travaux de laboratoire des AA "**Compléments programmation réseaux**" (3ème informatique réseaux-télécoms) et "**Technologies du e-commerce**" (3ème informatique de gestion); certains points correspondants sont donc déjà vaguement évoqués ici.

Les développements C/C++ UNIX sont sensés se faire avec les outils de développement disponibles sur la machine Sunray; on pourra aussi utiliser une machine virtuelle Sun Solaris ou Linux. Cependant, Code::Blocks sur les PCs peut vous permettre de développer du code en dehors de ces machines. Les développements Java sont à réaliser avec **NetBeans 8.***. Le serveur Web avec moteur à servlets/JSP utilisé est **Tomcat 7*/8*** sous Windows (PC) et/ou Unix (machines U2 ou INXS). La gestion des fichiers élémentaires se fera au moyen de fichiers à enregistrements classiques, de fichiers textes ou de fichiers CSV (dans le contexte C/C++) ou properties (dans le contexte Java). La gestion des bases de données intervenant dans cet énoncé se fera avec le SGBD **MySQL**, interfacé avec la ligne de commande ou, de manière plus attrayante, avec des outils comme **MySQL Workbench** ou **Toad for MySQL**.

Les travaux peuvent être réalisés

- soit par pour **une équipe de deux étudiants** qui devront donc se coordonner intelligemment et se faire confiance, sachant qu'il faut être capable d'expliquer l'ensemble du travail (pas seulement les éléments dont on est l'auteur);
- soit par un **étudiant travaillant seul** (les avantages et inconvénients d'un travail par deux s'échangent : on a moins de problèmes quand il faut s'arranger avec soi-même et on ne perd pas de temps en coordination).

2. Règles d'évaluation

Comme on sait, la note finale pour l'UE considérée se calcule par une moyenne des notes des AA constitutives, sachant que le seul cas de réussite automatique d'une UE est une note de 10/20 minimum dans chacune des AAs.

Pour ce qui concerne l'évaluation de l'AA "Réseaux et technologie Internet", voici les règles de cotation utilisées par les enseignants de l'équipe responsable de cette AA.

1) L'évaluation établissant la note de l'AA "Réseaux et technologie Internet" est réalisée de la manière suivante :

- théorie : un examen écrit en janvier 2020 (sur base d'une liste de points de théorie à développer fournis au fur et à mesure de l'évolution du cours théorique) et coté sur 20;
- laboratoire : 4 évaluations pondérées selon leur importance (les 3 premières en évaluation continue, non remédiables car visant à acquérir des notions de base, la dernière remédiable en 2ème session), chacune notée sur un total établi en fonction de l'importance des notions à assimiler et de la charge de travail correspondante; la moyenne de ces notes fournit une note de laboratoire sur 20;
- note finale : **moyenne de la note de théorie (poids de 50%) et de la note de laboratoire (poids de 50%)**.

Dans ces conditions, ***il est clair qu'une note de théorie beaucoup trop basse (du type 5/20 ou moins encore) ne peut que conduire à l'échec de l'AA considérée***.

Cette procédure est d'application tant en 1ère qu'en 2ème session.

2) *Dans le cas où les travaux sont présentés par une équipe de deux étudiants*, chacun d'entre eux doit être capable d'expliquer et de justifier l'intégralité du travail sans de longues recherches dans le code de l'application proposée (pas seulement les parties du travail sur lesquelles il aurait plus particulièrement travaillé).

3) *Dans tous les cas,* tout étudiant doit être capable d'expliquer de manière générale (donc, sans entrer dans les détails) les notions et concepts théoriques qu'il manipule dans ses travaux (par exemple: socket, machine à états de TCP, signature électronique, certificat, etc).

4) En 2ème session, un **report de note** est possible pour **des notes supérieures ou égales à 10/20** en ce qui concerne :

- la note de théorie;
- les notes de laboratoire de l'évaluation 4 (évaluation à l'examen).

Les évaluations de théorie et du laboratoire 4 ayant des **notes inférieures à 10/20** sont donc **à représenter dans leur intégralité** (le refus de représenter une évaluation complète de laboratoire entraîne automatiquement la cote de 0).

Les notes de laboratoire des évaluations 1, 2 et 3 ne sont pas remédiables : comme elles ont pour but de faire acquérir des techniques élémentaires, il n'a plus de sens de tester à nouveau ces acquis en 2ème session et elles resteront donc à la valeur acquise lors de l'évaluation de 1ère session.

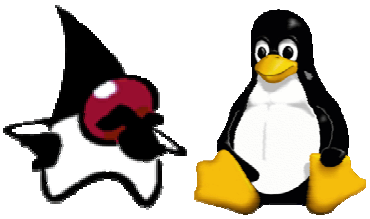
3. Agenda des évaluations

Pour chaque évaluation, le délai est à *respecter* impérativement.

Evaluation	**Evaluation continue : semaine d'évaluation**	**Pondération dans la cote de laboratoire**	**Remédiable en 2ème session**
Evaluation 1 : client-serveur multithread TCP en C/C++	30/9/2019-4/10/2019	20	non
Evaluation 2 : JDBC + client-serveur multithread TCP en Java	21/10/2019-25/10/2019	30	non
Evaluation 3 : programmation Web Java classique	25/11/2019-29/11/2019	30	non
Evaluation 4 : ♦ architecture complète fonctionnelle ♦ client-serveur sécurisé en Java et complément caddie virtuel en Java ♦ communications réseaux C/C++-Java ♦ client-serveur UDP en C/C++ et Java	Examen de laboratoire de janvier 2020	120	oui

Remarque importante : Pour rappel, lors de chaque évaluation, chaque étudiant est sensé connaître les bases théoriques qui lui ont permis de réaliser les développements proposés. Dans le cas contraire, on sera amené à considérer qu'il a développé sans comprendre ce qu'il faisait ou, pire, que le travail proposé a été récupéré ailleurs (plagiat pur et simple) … ce qui conduit automatiquement à une note insuffisante.

4. Avertissements préalables



1) Dans ce qui suivra, un certain nombre de points ont été simplifiés par rapport à la réalité. **Certains points ont également été volontairement laissés dans le vague**: il vous appartient d'élaborer vous-mêmes les éléments qui ne sont pas explicitement décris dans l'énoncé. Cependant, pour vous éviter de vous fourvoyer dans une impasse ou perdre votre temps à des options de peu d'intérêt, il vous est vivement recommandé de prendre conseil au près de l'un de vos professeurs concernés par le projet.

2) Le contenu des évaluations a été déterminé en fonction de l'avancement du cours théorique ET des besoins des autres cours de 3ème bachelier, avec lesquels nous nous sommes synchronisés dans la mesure du possible. Sans ces contraintes, le schéma de développement eût été différent.

3) Autre chose : une condition sine-qua-non de réussite est le traitement systématique et raisonné des **erreurs** courantes (codes de retour, errno, exceptions).

4) Le nom de l'UE et de l'AA comporte le mot "**réseaux**": si travailler au départ en **localhost** est légitime, **il est par contre impérieux de présenter un dossier final qui fonctionne en réseau effectif** (donc client et serveur sur deux machines distinctes).

Une plage de 10 ports TCP-UDP vous est attribuée sur les machines Unix et Linux. Il vous est demandé de la respecter pour des raisons évidentes …

5. Le contexte général

5.1 Présentation générale

La société "**HAppy Ferry Inpres Company**" (**HAFIC**) existe depuis 1992, est basée en zone extra-territoriale franco-belge et exploite des lignes de ferry entre la France et la Belgique d'une part, la Grande-Bretagne, l'Irlande, la Pologne et la Scandinavie d'autre part. Ses concurrents les plus connus sont P&O, North Sea Ferries, Norfolk Line, Titanic Extreme, IcebergForTwo, LesRadeauxDeLaMeduse, …

Outre les navires et leur personnel navigant, la société doit aussi gérer ses terminaux (maritimes, pas informatiques ;-) !) car, à la différence d'autres compagnies, elle est propriétaire de ses installations qui se trouvent près de Calais. C'est à ce niveau que la mise en place d'une structure informatique intervient et c'est le rôle de ce projet "**Happy Ferry Inpres**".

Pour comprendre les besoins de cette structure, considérons le parcours d'un voyageur qui se présente à l'entrée des installations portuaires dans le but d'embarquer sur un ferry HAFIC.

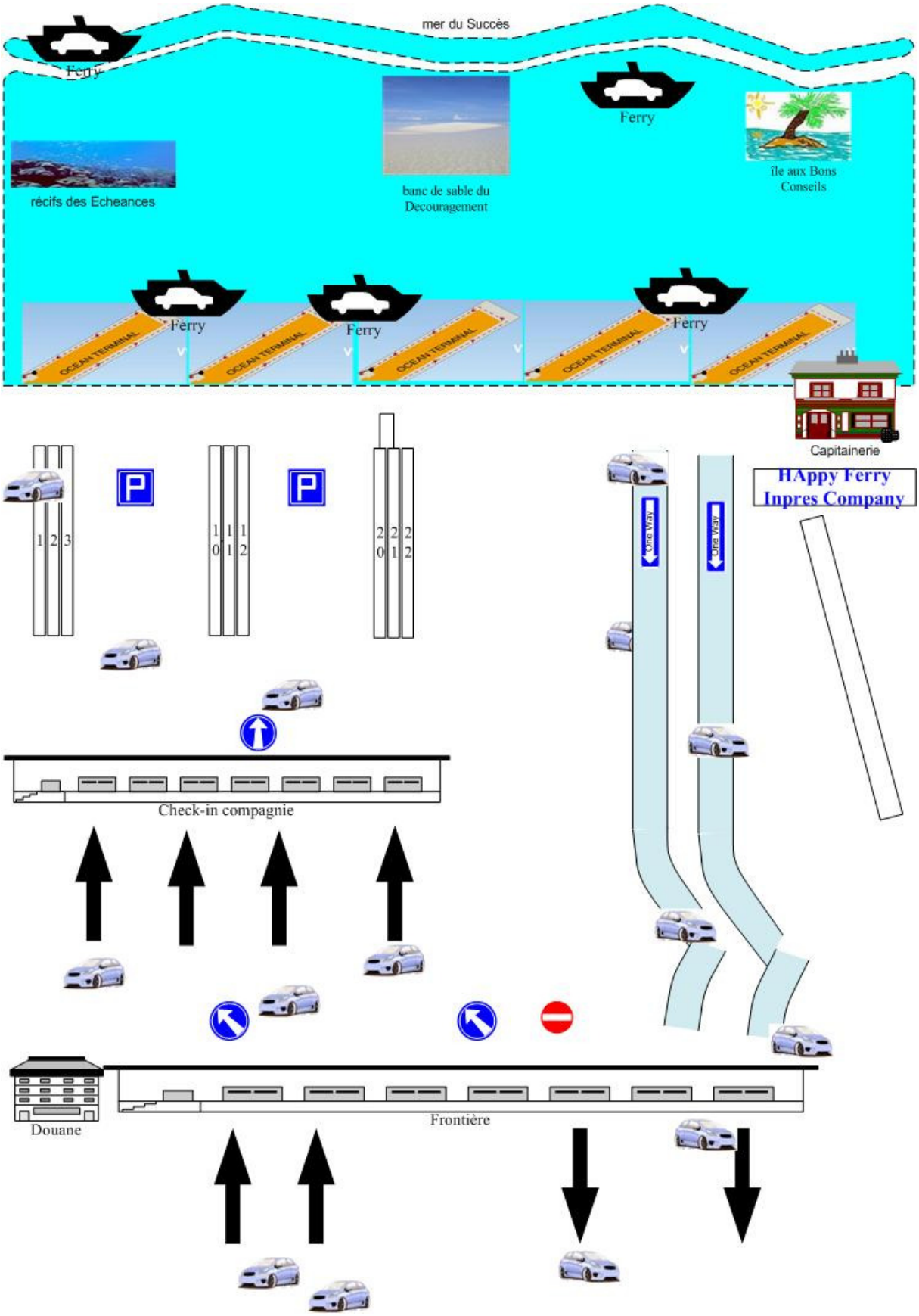
1) La première étape est la frontière et la douane : on quitte un pays pour entrer dans un autre. Il s'agit donc pour le voyageur de produire ses pièces d'identité et celles des autres voyageurs qui sont à bord. Les agents des services des frontières vérifient ces pièces (électroniques ou non) en s'adressant à un serveur **Serveur_Frontières** qui fait relais avec les serveurs du Registre national et les serveurs analogues des autres pays. Inutile de dire que toutes les techniques de sécurité logicielle sont ici nécessaires (cryptages et authentification implémentés spécifiquement ou assurés par SSL). En complément, les éventuels contrôles douaniers sont opérés.

2) La deuxième étape est celle de l'enregistrement des voyageurs par la compagnie maritime (le "check-in") : ces voyageurs peuvent déjà disposer d'un titre de traversée (acheté par Internet) mais ils peuvent aussi acquérir ce titre sur place. Le personnel de la compagnie s'adresse pour ces opérations à un **Serveur_Compagnie** chargé de gérer ces informations au moyen d'une base de données BD_FERRIES. Les voyageurs se voient alors confier un ticket d'embarquement et un panonceau à apposer sur le pare-brise avec un numéro de file d'attente où se placer pour accéder au ferry.
A remarquer que les tickets d'embarquement ne sont confiés aux voyageurs que si on a pu vérifier qu'ils ont passé la frontière sans encombre (le serveur du passage de frontière doit donc correspondre donc avec le serveur de la compagnie).

3) Les véhicules se retrouvent donc ainsi dans une file. Leurs mouvements vers le ferry sont coordonnés par les agents attachés aux terminaux. Ces agents utilisent des postes clients d'un **Serveur_Terminaux** qui gère tous les mouvements (de véhicules et de navires). Il a connaissance des heures de départ et d'arrivée des ferries parce que Serveur_Compagnie lui envoie ces informations journellement.

4) Avant de partir, et pour patienter, les voyageurs peuvent tuer le temps dans une salle d'attente qui est en fait un petit complexe avec cafétaria-bar, jeux et librairie. Ce complexe abrite un serveur **Serveur_Information** dont le seul nom explique le rôle : les postes clients fournissent les informations pratiques, les cours des unités monétaires, la liste des articles achetables en free-tax sur le ferry, …

Schématiquement, l'environnement est donc celui-ci :


mer du Succès
Ferry
Ferry
Ferry
récifs des Echeances
banc de sable du Decouragement
île aux Bons Conseils
Ferry
Ferry
Ferry
OCEAN TERMINAL
OCEAN TERMINAL
OCEAN TERMINAL
OCEAN TERMINAL
OCEAN TERMINAL
Capitainerie
HAppy Ferry Inpres Company
1 2 3
1 0 1 1 2
2 0 1 2 2
One Way
One Way
Check-in compagnie
Douane
Frontière

La gestion correcte d'une telle organisation touristique réclame évidemment une infrastructure informatique importante. Nous allons tout au long de ce projet en mettre en place les acteurs selon des schémas client-serveur.

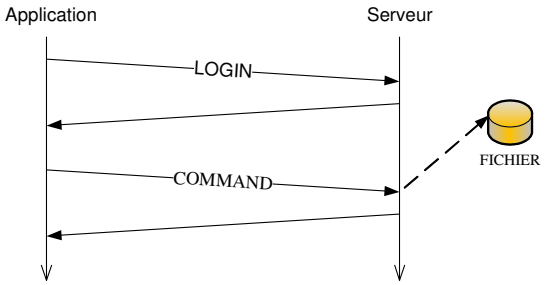
Les travaux de l'évaluation 1 : client-serveur multithread TCP en C/C++

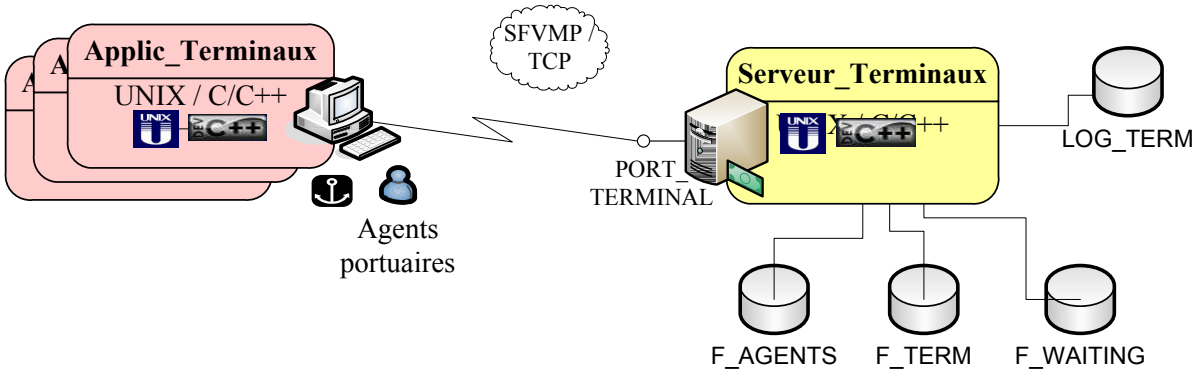
Compétences développées :

- Maîtriser l'implémentation du modèle client-serveur avec sockets TCP en C/C++;
- Concevoir une librairie de fonctions/classes utiles dans un but d'utilisation simplifiée et réutilisable dans le développement;
- Développer des fonctions/classes génériques (dissimulant suffisamment leur fonctionnement interne pour que leur utilisation ne souffre pas d'un changement d'implémentation).

Dossier attendu :

1. tableau des commandes et diagramme du protocole SFVMP - du type ci-contre;
2. code C/C+ du serveur Serveur_Terminaux
3. vues des trames échangées par un sniffer lors du début des connexions (états TCP).





6. Serveur_Terminaux

6.1 Fonctionnalités demandées

Pour rappel, il s'agit d'un **serveur multithread C-C++/ Unix** (en *modèle pool de threads*) qui a pour rôle de répondre, sur le port PORT_TERMINAL, aux requêtes des agents portuaires (qui utilisent **Applic_Terminaux**) chargés de gérer les états et mouvements des ferries. Il utilise un fichier de log LOG_TERM dans lequel sont enregistrées toutes les commandes dont il fait l'objet.

L'application cliente de ce serveur utilise le protocole **SFVMP** (**S**imple **F**erries and **V**ehicles **M**anagement **P**rotocol). Ce ***protocole applicatif*** de conversation entre le serveur et ses clients, est basé **TCP**, est **à états** et possède une liste de commandes prédéfinies que chacune des deux parties sait gérer. Ces commandes sont:

protocole **SFVMP**		
commande	**sémantique de la requête**	**réponse éventuelle**
LOGIN	démarrage de l'application : un agent se fait reconnaître *paramètres* : nom, password, **numéro de terminal**	oui ou non, sur base d'un fichier csv F_AGENTS
ASK-NEXT-DEPARTURE	demande de l'heure du prochain départ ainsi que du nom et de la destination du ferry associé *paramètre* : -	réponses possibles: 1) DEPARTURE-KNOWN: si un ferry se trouve au terminal

		considéré (déterminé en consultant le fichier csv F_TERM) ➔ point **a)** ci-dessous – DEPARTURE-2) UNKNOWN : si pas d'heure trouvée, le ferry est simplement au quai : l'agent reste en attente ou peut aussi fermer la connexion pour gérer un autre terminal (nouveau login) 3) NO-FERRY : si pas de ferry au terminal ➔ point **b)** ci-dessous
a) quand une heure de départ valide a été obtenue, les requêtes suivantes sont utilisables :		
ASK-BEGIN-LOADING	demande d'autorisation du début du chargement du ferry *paramètre* : l'heure (déterminée automatiquement)	ACK ou FAIL applicatif selon que la différence entre l'heure de départ et l'heure envoyée est inférieure à 45 minutes ou pas
NOTIFY-END-LOADING	notifie que le ferry est chargé - demande d'autorisation de départ *paramètres* : l'heure (déterminée automatiquement)	ACK ou FAIL applicatif selon que l'heure envoyée est postérieure d'au moins 15 minutes à l'heure de début de chargement
FERRY-LEAVING	notifie que le ferry quitte le terminal *paramètres* : l'heure (déterminée automatiquement)	le ferry est retiré du fichier F_TERM
b) quand le terminal est libre, les requêtes suivantes sont utilisables :		
ASK-FOR-FERRY	notifie qu'un ferry peut être accepté au terminal	si un ferry est en attente (ces ferries sont référencés dans le fichier F_WAITING), le nom du ferry affecté au terminal dans F_TERM
FERRY-ARRIVING	notifie que le ferry arrive au terminal *paramètres* : les renseignements sur le ferry	ACK ou FAIL applicatif selon qu'un ferry avait bien été affecté au terminal ou pas – si ACK, la commande ASK-NEXT-DEPARTURE peut à présent être envoyée
CLOSE	fermeture de l'application *paramètre* : -	heure de fermeture

On constate donc que le client peut se trouver dans les états "non connecté", "connecté", "départ prévu", terminal libre", …

Trois fichiers interviennent donc dans le fonctionnement du serveur. Ce sont des fichiers textes, plus exactement des fichiers csv : les tuples de données sont représentées comme une ligne de texte avec le point-virgule (ou la virgule) comme séparateur. Ils sont consultables/modifiables avec un éditeur comme JEdit ou un tableur comme Excel.

1) F_AGENTS contient les nom et prénom des agents avec leur mot de passe :

```
Dugland Albert;GrosCerveau
Laplanche Julia;Miammiam
Dugenou Felix;Zezette
```

- ce fichier est **rempli/modifé manuellement**.

2) F_TERM contient l'occupation des terminaux avec pour chacun, s'il est occupé, le nom du ferry, son heure de départ et sa destination

```
term_1;Marie-Galante;19-00;Cork
term2;-;NA;-
term3;-;NA;-
term4;Marie-Martine;NA;Dover
```

où "NA" signifie "Not Available", terminologie classique dans le traitement des Big data.

3) F_WAITING contient la liste des ferries au large en attente d'apontage sur un terminal :

```
Marie-Youplaboum;15h40;Göteborg
Marie-Gluante;NA; Dublin
```

- ce fichier est **rempli/modifé manuellement** avec l'ajout se faisant toujours à la fin.

6.2 Quelques conseils méthodologiques pour le développement de SFVMP

1) Il faut tout d'abord choisir la manière d'implémenter les requêtes et les réponses des protocoles SFVMP et plusieurs possibilités sont envisageables pour écrire les trames;
- uniquement par chaîne de caractères contenant des séparateurs pour isoler les différents paramètres;
- sous forme de structures, avec des champs suffisamment précis pour permettre au serveur d'identifier la requête et de la satisfaire si possible;
- un mélange des deux : une chaîne pour déterminer la requête, une structure pour les données de la requête;
- fragmenter en plusieurs trames chaînées dans le cas d'une liste à transmettre.

On veillera à utiliser des constantes (#DEFINE et fichiers *.h) et pas des nombres ou des caractères explicites.

2) On peut ensuite construire un squelette de serveur multithread et y intégrer les appels de base des primitives des sockets. Mais il faudra très vite (sous peine de réécritures qui feraient perdre du temps) remplacer ces appels par les appels correspondants d'une librairie de

fonctions **SocketsUtilities** facilitant la manipulation des instructions réseaux. Selon ses goûts, il s'agira
- soit d'une bibliothèque **C** de fonctions réseaux TCP/IP;
- soit, mais c'est peut-être un peu moins évident, d'une bibliothèque **C++** implémentant une *hiérarchie de classes* **C++** utiles pour la programmation réseau TCP/IP : par exemple, Socket, SocketClient et SocketServeur;); on évitera cependant la construction de flux réseaux d'entrée et de sortie (genre NetworkStreamBase, ONetworkStream et INetworkStream) car cela devient très(trop) ambitieux pour le temps dont on dispose - des fonctions classiques send(), receive(), etc suffiront.

Dans les deux cas, un mécanisme d'erreur robuste sera mis au point.

3) Quelques remarques s'imposent :
3.1) Une remarque pleine de bon sens ;-) : une fonction (ou une méthode) de bibliothèque ne présente d'intérêt que si elle est plus facile à utiliser que la (les) fonction(s) qu'elle remplace …

3.2) Une deuxième remarque pleine de bon sens ;-) : une fonction (ou une méthode) de bibliothèque ne présente d'intérêt que si elle est indépendante du cas particulier du projet " projet "Happy Ferry Inpres" … Ainsi :

bien ☺	**pas bien ☹**
xxx **send**(void *,int size) /* couche basse : réutilisable dans une autre application */ xxx **AskForDeparture** /* couche haute : propre à cette application */	xxx **send**(Departure *,int size) // et pas de xxx AskForMaterial (...,...) /* une seule couche : la fonction send ne peut être réutilisée dans une autre application */

3.3) Une troisième remarque pleine de bon sens ;-) : multiplier les couches exagérément est certainement le meilleur moyen de compliquer le développement et de ralentir l'exécution !

3.4) Enfin, en tenant compte d'une future administration du serveur, il serait avisé de faire intervenir dans le code du serveur **la notion d'état** de celui-ci (*certaines commandes n'ont de sens que si elles sont précédées d'une autre*).

4) Il est impérieux de surveiller les écoutes, les connexions et les communications réseaux
- au moyen des commandes **ping** et surtout **netstat** (savoir expliquer les états TCP);
- en utilisant un **sniffer** comme Wireshark ou autre encore analysant le trafic réseau (attention au localhost : installer dans ce cas la dernière version de Wireshark). Cette pratique sera demandée lors des évaluations.

5) Il serait aussi intéressant de prévoir un fichier de configuration lu par le serveur à son démarrage. A l'image des fichiers properties de Java, il s'agit d'un simple fichier texte dont chaque ligne comporte une propriété du serveur et la valeur de celle-ci :

serveur_terminaux.conf
Port_Service=70000 Port_Admin=70009 sep-trame=$ fin-trame=# sep-csv=; pwd-master=tusaisquetuesbeautoi

pwd-admin=sebclachri..
…

6.3 L'accès aux données des terminaux

On utilise donc ici des fichiers de données de type csv. Il est bien clair que ce serveur Serveur_Terminaux aura des interactions avec la base de données BD_FERRIES (exposée au point suivant) et il pourra le faire en passant par un autre serveur.

Des librairies d'accès aux bases de données relationnelles existent bien sûr en C/C++, mais elles ne sont pas du tout portables et/ou posent des problèmes de déploiement d'un système à un autre). Nous les éviterons donc.

Comme la base BD_FERRIES n'est pas encore disponible, il convient de prévoir une conception logicielle qui isole les demandes à ce serveur dans une librairie de fonctions dont l'implémentation pourra être modifiée ultérieurement (avec le minimum de réécriture de code). On pense donc ici à des fonctions du type suivant (*ce ne sont que deux exemples - libre à vous d'en concevoir d'autres du même style*) :

librairie **AccessPort**		
fonction (ou méthode)	**sémantique**	**valeur retournée**
yyy getDeparture (char * number)	récupérer un ferry en départ	structure donnant les renseignements sur ce ferry
int setFerry (char * name, char * hour, char *dest)	enregistrement d'un ferry à un terminal	enregistrement effectué ou pas

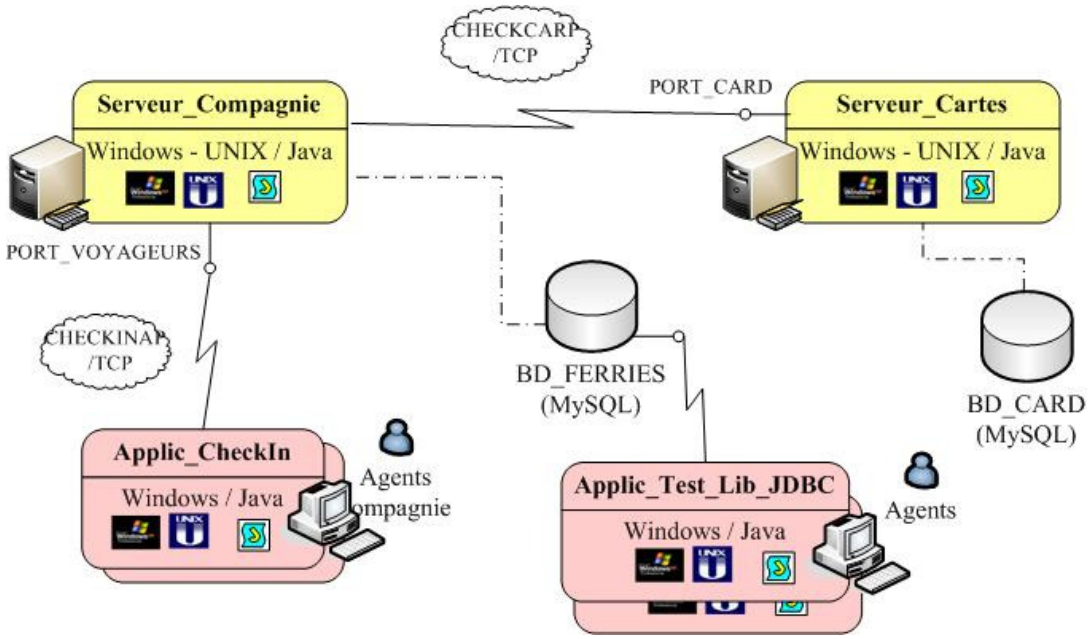
Les travaux de l'évaluation 2 : JDBC et client-serveur multithread TCP en Java

Compétences développées :

- Mettre en œuvre les techniques de threads et JDBC en Java;
- Maîtriser l'implémentation du modèle client-serveur sur base des sockets TCP en Java;
- Mettre en œuvre les techniques de threads et JDBC en Java;

Dossier attendu :

1. schéma relationnel de BD_FERRIES;
2. diagramme de classes UML des classes de database.facility.
3. définition et implémentation des commandes du protocole CHECKINAP;
4. code Java du serveur Serveur_Compagnie;
5. trames échangées et vues par un sniffer.



7. Les accès aux bases de données

7.1 La base de données BD_FERRIES

Cette base doit contenir les informations utiles concernant les réservations et les voyageurs. Ses tables, si on admet un certain nombre d'approximations, de simplifications et d'omissions sans importance pour le projet tel que défini ici, seront en première analyse :

- **navires** : avec comme champs essentiels un matricule, un nom et une capacité en termes de voitures et de véhicules lourds (cars et camions);

- **ports** : nom, pays, nombre de terminaux et nombre de terminaux occupés (information dynamique);

- **traversees** : identifiant (de la forme 20190915-TR01), date et heure de départ, port de départ et de destination, navire utilisé;

- **voyageurs** (voyageurs au nom de qui la réservation est faite = titulaires) : numéro de client, nom, prénom, adresse du domicile, adresse e-mail; un numéro de client est généré au

moyen d'un hashage des autres informations complétées de la date (on utilisera simplement pour cette génération la méthode hashCode() de la classe String).

- **voyageurs-accompagnants** : nom, prénom, adresse du domicile, voyageur titulaire;

- **reservations** : identifiant (de la forme 20110915-RES01), traversée, voyageur-titulaire, payé (O/N), passé au check-in (O/N).

On a toute liberté pour ajouter des champs ou d'autres tables.

7.2 Un outil d'accès aux bases de données

L'accès à la base de données ne devrait pas se faire avec les primitives JDBC utilisées telles quelles, mais plutôt au moyen d'objets métiers encapsulant le travail (typiquement de type Java Beans, mais de toute manière sans utilisation d'un mécanisme d'events).

On demande donc de construire un ensemble de telles classes (package **database.facility**) permettant l'accès (c'est-à-dire à tout le moins la connexion et les sélections de base) le plus simple possible. On souhaite pouvoir accéder, au minimum, à des bases relationnelles de type MySQL et Oracle

Le programme de test **Applic_Test_Lib_JDBC** de la petite librairie ainsi construite proposera un interface graphique de base permettant de se connecter à la base MySQL BD_FERRIES pour y réaliser

- tout d'abord des requêtes élémentaires de type "select * from … where …", "select count(*) from" et "update … set … where" avec affichage des résultats (requêtes adaptées à la table visée – pas de tentative de généricité à ce stade);
- ensuite quelques requêtes qui seront utiles dans la suite (elles seront intégrées dans les applicatifs à développer ultérieurement) comme

1. la liste des voyageurs enregistrés pour une traversée donnée à une date donnée, la liste des voyageurs sur une certaine période regroupés par nationalité,
2. l'âge moyen des voyageurs enregistrés pour une traversée donnée sur une certaine période;
3. l'insertion d'une nouvelle traversée;
4. l'insertion d'une nouvelle réservation.

Rien n'interdit de lancer simultanément plusieurs utilisations de cette application qui est donc "**threadée**": attention donc aux **accès concurrents** !

8. Serveur_Compagnie

Il s'agit d'un serveur multithread Java/Windows-Unix (selon le modèle générique pool de threads) qui est essentiellement chargé de satisfaire les requêtes provenant de l'application **Applic_CheckIn**, utilisée par les agents de la compagnie responsables du check-in des véhicules arrivant de la douane. Toutes les données nécessaires sont stockées dans la base de données BD_FERRIES.

Ce serveur attend, pour cette fonctionnalité, sur le port PORT_VOYAGEURS. Le protocole utilisé par le serveur et l'application cliente **Applic_CheckIn** est **CHECKINAP** (**CHECKIN A**pplication **P**rotocol) : il est basé TCP et comporte les commandes suivantes :

protocole **CHECKINAP**		
commande	**sémantique de la requête**	**réponse éventuelle**
LOGIN	un agent de la compagnie se connecte au serveur *paramètres* : nom, password	oui ou non
VERIF_BOOKING	pour vérifier une réservation *paramètres* : code de la réservation, nombre de passagers (en plus du conducteur)	ACK avec enregistrement du check-in OU FAIL applicatif si réservation non trouvée - dans ce cas, on peut acheter un ticket (commande suivante) ou emprunter la voie de sortie
BUY_TICKET	pour acheter un ticket d'embarquement pour une traversée *paramètres* : nom conducteur, numéro d'immatriculation de la voiture, nombre de passagers (en plus du conducteur) + numéro de carte de crédit	après vérification positive du numéro de carte auprès de **Serveur_Cartes** (voir ci-dessous): ACK avec le code du ticket, le nom du ferry et l'heure de départ (c'est celui qui part le plus tôt qui est choisi si il y a encore de la place) ainsi qu'un numéro de client réutilisable dans la suite OU FAIL applicatif si aucune possibilité trouvée ce jour
CLOSE	fermeture de l'application *paramètre* : -	

A remarquer que la confection des horaires de traversées peut se faire manuellement (en ligne de commande, avec Toad for MySQL ou SQL Workbench ou autre) : la fonctionnalité de gestion de ceux-ci n'est pas demandée dans l'application considérée ici.

Les réservations seront créées par une application Web permettant d'acheter des places pour une traversée par Internet (***voir évaluation 3***). Pour l'instant, les informations de réservations seront insérées manuellement dans les tables concernées.

Une fois le véhicule accepté (soit à cause d'une réservation reconnue par le serveur, soit parce que le ticket vient d'être acheté), l'application cliente choisit un numéro de file d'attente (une file est complètement remplie avant de passer à la suivante) qui est transmis au voyageur. Il peut à présent se diriger vers cette ligne d'attente. Ce type d'information est mémorisé dans une table additionnelle de la base BD_FERRIES.

9. Serveur_Cartes

Ce serveur vérifie que la carte de crédit est une carte valide auprès du serveur **Serveur_Card** qui, dans l'affirmative, réalise le débit du compte. C'est un serveur multithread Java/Windows-Unix (selon le modèle threads à la demande)

Ce serveur est des plus simples: il n'attend sur son port PORT_CARD que ce seul type de requête de vérification/débit (protocole **CHECKCARP** [**CHECK CAR**d **P**rotocol]), le numéro de carte et la somme à débiter. Pour vérifier la validité du numéro de carte, Serveur_Card utilise sa propre base de données BD_CARD, qui contient essentiellement les comptes et les numéros de cartes (un compte peut avoir plusieurs cartes de crédit associées). La seule vérification sur la somme à débiter consiste à vérifier qu'elle ne dépasse pas 5000 EUR.

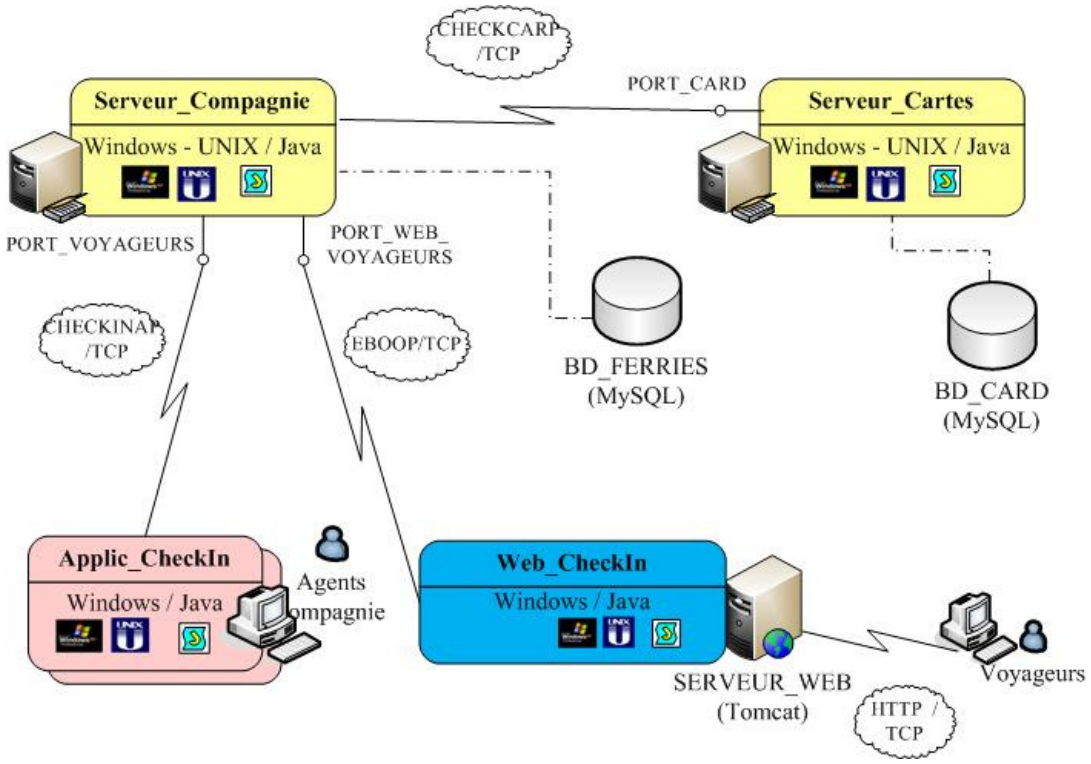
Les travaux de l'évaluation 3 : Programmation Web Java classique

Compétences développées :

- Aborder les techniques de bases du développement Web en Java;
- Mettre en œuvre les techniques de threads et JDBC en Java.

Dossier attendu :

1. diagramme de classes UML des classes de l'application Web;
2. schéma de l'application Web en termes de servlets, JSPs et pages HTML (formalisme libre);
3. description du protocole EBOOP.



10. L'application Web_CheckIn

10.1 Fonctionnalités demandées

Cette application **Web** permet au public d'acheter des places sur un ferry par Internet; elle est donc cliente du serveur Serveur_Compagnie, lequel utilise pour cela le PORT_WEB_VOYAGEURS. C'est par ce moyen que les réservations seront créées dans BD_FERRIES (voir commande VERIF_BOOKING de l'évaluation 2).

a) Les clients doivent tout d'abord se connecter au serveur soit en utilisant leur numéro de client obtenu lors d'une autre réservation (ce qui leur donnera droit à une réduction de 5% sur le montant de leurs réservations), soit en en obtenant un. Ceci se passe en utilisant un formulaire dans une page HTML qui contacte une servlet. Cette servlet communique avec Serveur_Compagnie pour obtenir un numéro de client valide en utilisant une commande MAKE_BOOKING ajoutée au protocole CHECKINAP (commande à définir).

b) Passée la phase d'identification **absolument indispensable** (rien n'est donc possible tant que le client n'a pas effectué cette démarche), l'utilisation du site suit le canevas suivant avec une page HTML proposant les trois services suivants

1. Liste des promotions last-minute à ne rater sous aucun prétexte
2. Achats selon le principe du caddie virtuel
3. Fin de session avec encodage des données complémentaires du client

Les clients effectuent des réservations selon la technique du **"caddie virtuel"** dans un interface graphique qui illustre très schématiquement les traversées disponibles : différentes destinations et différents horaires selon le jour choisi pour la traversée. La démarche consiste à se promener dans les pages catalogues du site et à choisir au fur et à mesure des éléments. On obtient à la fin le montant total des réservations et on réalise alors les opérations de confirmation et de paiement avec introduction des informations

- de réservation : nombre de passagers (en plus du conducteur);
- de contact : adresse e-mail, adresse postale, numéro de téléphone;
- de paiement : numéro de carte de crédit, date de validité.

Comme précédemment, le numéro de carte est vérifié auprès de **Serveur_Cartes** (voir ci-dessus).

Techniquement, ce site est construit selon le modèle MVC, donc avec des java Beans, des Java Server Pages (une par type action) et une (éventuellement des) servlet(s); La base BD_FERRIES sera accédée par les classes développées dans l'évaluation 2: elles appartiennent donc au modèle.

On utilise dans ce contexte un ***protocole applicatif*** basé sur HTTP/TCP et nommé **EBOOP** (**E**asy **BOO**king **P**rotocol) – qu'il convient de définir.

Il vous appartient de définir les commandes du protocole **EBOOP** (et donc de leur donner un nom) et de choisir la manière de les implémenter (objets, chaînes de caractères, etc).

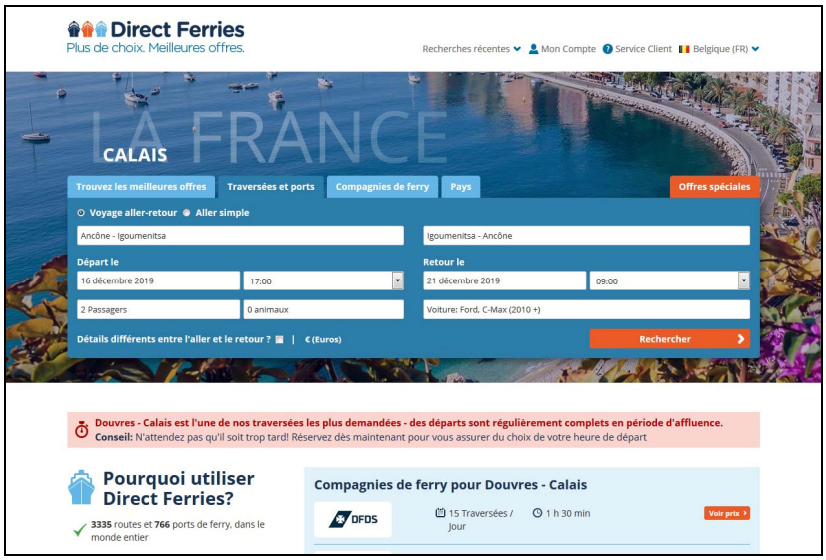
10.2 Quelques conseils méthodologiques pour le développement de Web_CheckIn

Pour rappel, on veillera impérativement à une "gestion de stocks" cohérente en ce qui concerne les réservations :

- évidemment, le nombre de véhicules sur une traversée n'est pas illimité puisque chaque ferry a une contenance maximale (pas question de "commander" : les places sont disponibles ou pas);
- tout ce qui est réservé et placé dans un caddie est considéré comme une promesse ferme : autrement dit **le client est assuré que ce qu'il a réservé ne peut être réservé par un autre client**;
- toute réservation validée (c'est-à-dire payée) doit bien sûr être mémorisée dans la base de données (pratique normale du commerce).
- si un client abandonne sa navigation Web, ce qu'il avait réservé doit redevenir disponible dans un délai d'une demi-heure (par exemple).

En pratique, on testera l'application Web sur PC sous Windows avec un Tomcat distant sur PC.

Attention ! Il se peut que le serveur Serveur_Compagnie, qui fonctionne en permanence, permette déjà d'effectuer des achats de billets pour une traversée proche dans le temps. Ces derniers peuvent donc entrer en concurrence avec les réservations pratiquées par l'application Web !



Les travaux de l'évaluation 4:

- **client-serveur sécurisé en Java**
- **administration d'un serveur et communications réseaux C/C++-Java**
- **client-serveur UDP en C/C++ et Java**

Coming soon …

s: CV, CC & SC

Infos de dernière minute ? Voir l'Ecole virtuelle