Laboratoire de Réseaux et technologie Internet (TCP-UDP/IP, HTTP et sécurité logicielle/cryptologie en C/C++/Java) :

3ème Informatique de gestion
& 3ème Informatique et systèmes
opt. Informatique industrielle et Réseaux-télécommunications
2021-2022

Projet *ScienceAirport*

Claude Vilvens, Christophe Charlet, Sébastien Calmant, Jean-Marc Wagner
(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 géométrique des notes des AA constitutives, sachant que le seul cas de réussite automatique d'une UE est une note de 10/20 minimum.

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 :

- ♦ examen de théorie: un examen écrit en janvier 2022 (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 en évaluation continue: les 2 premières évaluations ("évaluation 1" et "évaluation2" ci-dessous) s'effectuent en 2021 et sont non remédiables, car visant à acquérir des notions de base; chacune est cotée sur 20 et leur moyenne constitue la note d'évaluation continue;
- ♦ examen de laboratoire: un examen oral en janvier 2022 consistant en la présentation de la 3ème partie du laboratoire ("évaluation 3" ci-dessous) et coté sur 20;
- ♦ note finale : **moyenne géométrique de la note de l'examen de théorie (poids de 50%), de la note d'évaluation continue (poids de 15%) et de la note de l'examen de laboratoire (poids de 35%)**.

Dans ces conditions, ***il est clair qu'une note beaucoup trop basse parmi les trois ne peut que conduire à l'échec de l'AA considérée***.

Cette procédure est d'application tant en 1ère qu'en 2ème session (pour celle-ci, les notes de l'évaluation continue sont conservées telles quelles puisque non remédiables).

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;
- ♦ la note de laboratoire de l'évaluation 3 (évaluation à l'examen).

Les évaluations de théorie et du laboratoire 3 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 et 2 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.

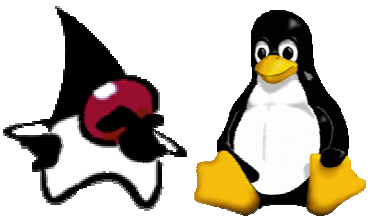
3. Agenda des évaluations

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

Evaluation	**Evaluation continue : semaine d'évaluation**	**Remédiable en 2ème session**
Evaluation 1 : client-serveur multithread TCP en C/C++ + JDBC	4/10/2021-8/10/2021	non
Evaluation 2 : client-serveur multithread TCP en Java + programmation Web Java classique	15/11/2021-19/11/2021	non
Evaluation 3: ♦ architecture complète fonctionnelle ♦ client-serveur sécurisé en Java ♦ communications réseaux C/C++-Java ♦ client-serveur UDP en C/C++ et Java	Examen de laboratoire de janvier 2022	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 …


M48
M50
M49

5. Le contexte général

5.1 Présentation générale

L'aéroport ScienceAirport est un aéroport de taille moyenne, installé dans un cadre bucolique de la région de Seraing. On y gère exclusivement des vols avec passagers (donc, pas d'avions cargos, du moins pour l'instant) qui prennent leur départ à ScienceAirport ou qui utilisent ScienceAirport comme étape pour un vol plus long.

Toutes les opérations habituelles d'un départ de passagers sont prises en charge : check-in, embarquements des passagers et chargement des bagages, départs effectifs des vols. Les billets sont vendus exclusivement soit par le site Web de l'aéroport, soit par l'intermédiaire de tour-operators et agences de voyages.

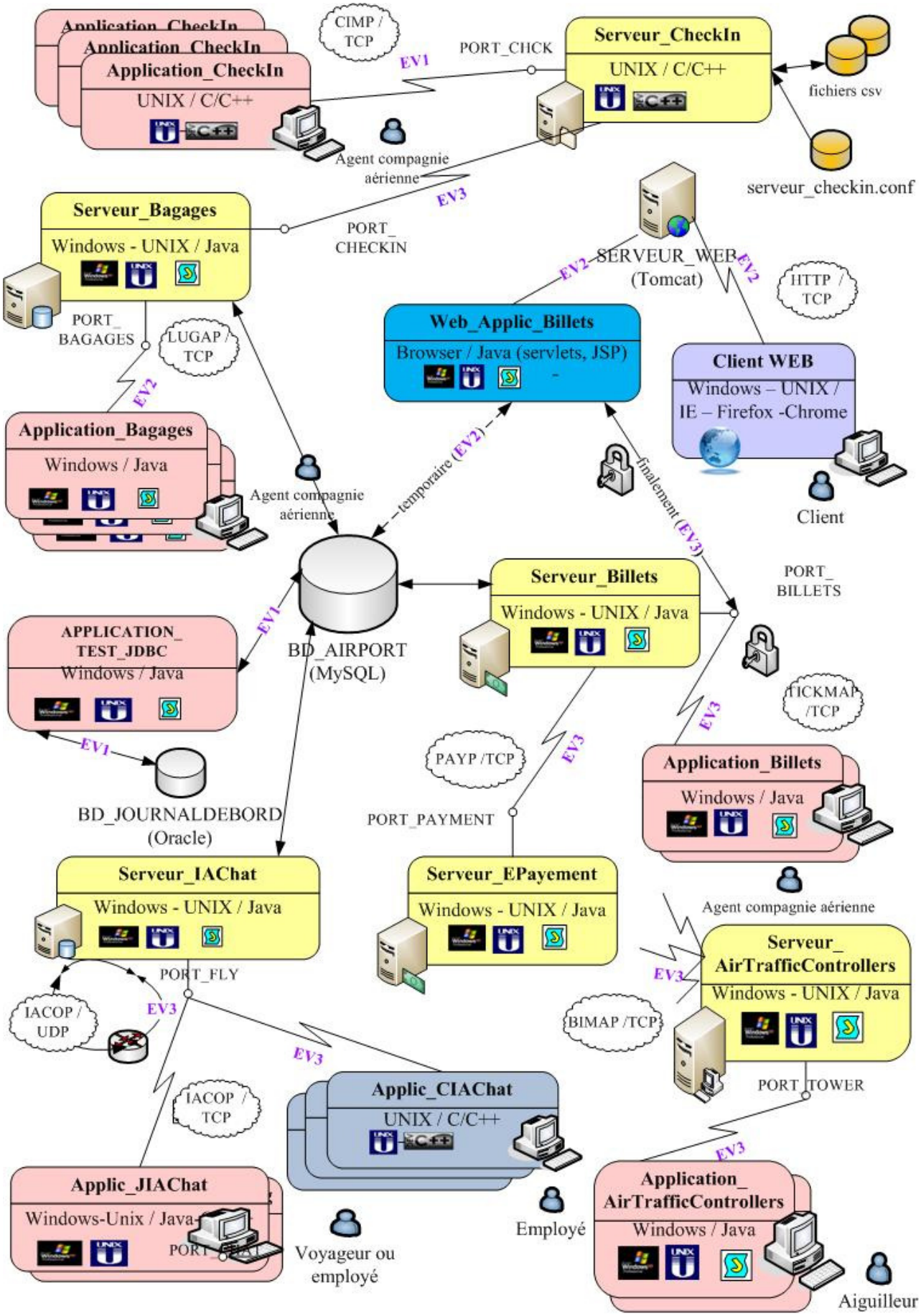
Bien sûr, ce genre d'infrastructure ne peut se gérer sans l'aide d'une structure informatique robuste. Le projet informatique développé ici va y contribuer, mais il ne se préoccupera que des départs, pas des arrivées.

4.2 Fonctionnement d'ScienceAirport

Concrètement, ScienceAirport présente une topologie classique :

- guichets de check-in occupés par les employés des compagnies aériennes selon les vols en partance; les bagages y sont enregistrés;
- portes d'embarquement pour les différents vols avec salle d'attente pour les passagers;
- quais et véhicules d'embarquement des bagages à mettre en soute;
- tour de contrôle avec aiguilleurs du ciel qui coordonnent les atterrissages et les décollages.

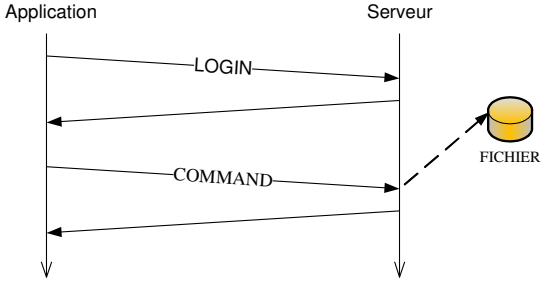
Ces différentes fonctionnalités correspondent à des serveurs informatiques dédiés, dont le rôle respectif sera expliqué en détail ci-dessous. Le schéma suivant illustre globalement ces différents serveurs et clients, ainsi que leurs interactions réseaux et bases de données qui seront implémentées dans ce laboratoire :


Application_CheckIn
UNIX / C/C++
CIMP / TCP
EV1
PORT_CHCK
Serveur_CheckIn
UNIX / C/C++
fichiers csv
serveur_checkin.conf
Agent compagnie aérienne
EV3
Serveur_Bagages
Windows - UNIX / Java
PORT_ CHECKIN
SERVEUR_WEB (Tomcat)
EV2
HTTP / TCP
PORT_ BAGAGES
LUGAP / TCP
Web_Applic_Billets
Browser / Java (servlets, JSP)
Client WEB
Windows – UNIX / IE – Firefox -Chrome
Client
Application_Bagages
Windows / Java
Agent compagnie aérienne
temporaire (EV2)
finalement (EV3)
BD_AIRPORT (MySQL)
Serveur_Billets
Windows - UNIX / Java
PORT_ BILLETS
APPLICATION_ TEST_JDBC
Windows / Java
EV1
TICKMAP /TCP
PAYP /TCP
Application_Billets
Windows / Java
BD_JOURNALDEBORD (Oracle)
PORT_PAYMENT
Serveur_IAChat
Windows - UNIX / Java
Serveur_EPayement
Windows - UNIX / Agent compagnie aérienne
Serveur_ AirTrafficControllers
Windows - UNIX / Java
PORT_FLY
IACOP / UDP
BIMAP /TCP
PORT_TOWER
IACOP / TCP
Applic_CIAChat
UNIX / C/C++
Applic_JIAChat
Windows-Unix / Java
Employé
Application_ AirTrafficControllers
Windows / Java
Voyageur ou employé
Aiguilleur

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

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).
- Mettre en œuvre les techniques de threads et JDBC en Java;



Dossier attendu :

1. tableau des commandes et diagramme du protocole CIMP - du type ci-contre;
2. vues des trames échangées par un sniffer lors du début des opérations.
3. schéma relationnel de BD_AIRPORT;
4. diagramme de classes UML des classes de database.utilities.

1. Serveur_CheckIn et Application_CheckIn

1.1 L'enregistrement des passagers : client-serveur

Le **Serveur_CheckIn** a donc pour mission essentielle de gérer les arrivées des passagers qui sont en possession d'un billet pour un vol donné : il s'agit essentiellement de la vérification des billets, de la validation des billets présentés ainsi que de l'enregistrement des bagages.

Le serveur est un serveur multithread C/Unix en modèle pool de threads. Il est chargé de répondre aux requêtes provenant de **Application_CheckIn** (C/C++) utilisée par les agents des compagnies aériennes qui assurent l'accueil des passagers pour les différents vols programmés. Le serveur attend ce type de requête sur le PORT_CHCK. Il utilise le ***protocole applicatif*** (basé TCP) **CIMP** (**C**heck**I**n **M**anagement **P**rotocol).

exemple : *Mr Walter Charvilrom se présente au check-in pour la porte 4 avec son épouse et ses 9 enfants. Il est en possession du billet 362-22082017-0070 pour le vol 362 de la Powder-Airlines à destination de Peshawar (départ à 6h30). Ses accompagnant(e)s correspondent aux billets 362-22082017-0071 → 362-22082017-0080.*

L'agent de la Powder-Airlines qui les reçoit dispose de l'application **Application_CheckIn** dans laquelle il a du au préalable se faire reconnaître par un classique login-password, ce qui correspond à la requête LOGIN_OFFICER sur base d'un fichier csv avec séparateur ";" (un tel fichier est éditable avec un tableau comme MS-Excel). Un changement d'agent passe par une requête LOGOUT_OFFICER. L'application propose à l'écran :

VOL 362 POWDER-AIRLINES - Peshawar 6h30

Numéro de billet ?

L'agent encode le numéro de billet et le nombre d'accompagnants :

VOL 362 POWDER-AIRLINES - Peshawar 6h30

Numéro de billet ? *362-22082017-0070*
Nombre d'accompagnants ? *10*

Le serveur reçoit une requête CHECK_TICKET et vérifie ces billets sur base d'un 2ème fichier csv (séparateur ";") qui contient les informations nécessaires (dans la suite, il s'adressera à un Serveur_Bagages - voir évaluation ultérieure). En cas de succès, l'agent reçoit une réponse positive et peut encoder les bagages.

exemple : *Leurs bagages, 3 valises et 5 coffres (étiquetés "sucre en poudre - ne pas ouvrir") sont enregistrés sous les identifiants 362-WACHARVILCAL-22082017-0070-001 à 362-WACHARVILCAL-22082017-0070-008.*

L'application :

VOL 362 POWDER-AIRLINES - Peshawar 6h30
Numéro de billet : 362-22082017-0070
Nombre d'accompagnants : 10

Poids du bagage n°1 <Enter si fini> : 27.3
Valise ? O
…
Poids du bagage n°8 <Enter si fini> : 19.95
Valise ? N
Poids du bagage n°9 <Enter si fini> :

Le serveur reçoit une requête CHECK_LUGGAGE et calcule l'éventuel excès de poids (20 kg maximum par bagage) :

VOL 362 POWDER-AIRLINES - Peshawar 6h30
Numéro de billet : 362-22082017-0070
Nombre d'accompagnants : 10

Poids total bagages : 197.5 kg
Excédent poids : 37 kg
Supplément à payer : 109.15 EUR
Paiement effectué ? *Y*

VOL 362 POWDER-AIRLINES - Peshawar 6h30

Numéro de billet ?

Le serveur a reçu la requête PAYMENT_DONE. Les bagages sont provisoirement (voir plus bas - Serveur_Bagages) enregistrés dans le fichier texte 362_22082017_lug.csv qui ressemble donc à ceci :

362_22082017_lug.csv
362-WACHARVILCAL-22082017-0070-001; VALISE *362-WACHARVILCAL-22082017-0070-002; VALISE* … *362- WACHARVILCAL-22082017-0070-007; PASVALISE* *362- WACHARVILCAL-22082017-0070-008; PASVALISE*

Et on peut passer aux passagers suivants.

exemple : Les voyageurs ont à présent accès à la salle d'attente de la porte 4, où les parents se détendre pendant que les enfants peuvent profiter des jeux mis à leur disposition.

1.2 La gestion des bagages : accès à Serveur_Bagages

On utilise donc ici des fichiers de données de type csv. Il est bien clair que ce serveur Serveur_CheckIn devra agir directement sur la base de données (exposée au point suivant) ainsi qu'interagir avec le serveur **Serveur_Bagages** (voir plus loin) qui agit sur cette même base de données BD_AIRPORT.

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_AIRPORT et le Serveur_Bagages ne sont pas encore disponibles, il convient de prévoir une conception logicielle qui isole les demandes à ce serveur dans une librairie de fonctions dont l'implémentation sera modifiée ultérieurement (avec le minimum de réécriture de code). On pense donc ici à des fonctions du type suivant (*ce sont des exemples - libre à vous d'en concevoir d'autres du même style*) :

librairie **AccessBilBag**		
fonction (ou méthode)	**sémantique**	**valeur retournée**
int verifyTicket (char * number, int nbPassengers)	vérification de l'existence d'un billet d'avion avec un certain nombre d'accompagnants	0 ou 1
float addLuggage (char * number, float weight, char suitcase)	enregistrement d'un bagage de poids donné, sous forme de valise ou pas, pour le billet d'avion précisé	poids total actuel pour le billet

1.3 Quelques conseils méthodologiques pour le développement de CIMP

1) Il faut tout d'abord choisir la manière d'implémenter les requêtes et les réponses des protocoles CIMP 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 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.

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 considéré ici … Ainsi :

bien ☺	**pas bien ☹**
xxx **receiveSize**(void * struc, int size) /* couche basse : réutilisable dans une autre application */ xxx **receiveSep**(char *chaine, char *sep) /* couche basse : réutilisable dans une autre application */ *avec* ListePassagers **getPassengers** (...,...) /* couche haute : propre à cette application - *utilise l'une des fonctions ci-dessus* */	xxx receive(ListePassagers *lc, int size) // et pas de xxx getPassengers (...,...) /* une seule couche : la fonction receive ne peut être utilisée 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 de l'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 qui ne permet pas de sniffer simplement). 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_checkin.conf
Port_Service=70000 Port_Admin=70009 sep-trame=$ fin-trame=# sep-csv=; pwd-master=tusaisquetuesbeautoi pwd-admin=jeaclachralf.. …

2. Les accès aux bases de données

2.1 La base de données BD_AIRPORT

Cette base MySQL BD_AIRPORT doit contenir toutes les informations utiles concernant le fonctionnement de l'aéroport (rien que ça !). Ses tables, si on admet un certain nombre d'approximations, de simplifications et d'omissions sans importance pour le projet tel que défini ici (donc, avec le nombre minimum de champs), seront en première analyse celles d'une base de données on ne peut plus classique, que l'on peut définir dans un premier temps sommairement comme contenant les tables

- **Billets** : il s'agit ici de la pierre d'angle de la base : elle permet de repérer tout passager (nom, prénom, numéro de carte d'identité ou de passeport, …) par son billet; ce billet permet d'accéder au vol correspondant ainsi qu'aux bagages qui y sont associés;
- **Vols** : renseignements sur chaque vol : destination, heure arrivée éventuelle, heure de départ, heure prévue d'arrivée à destination, avion utilisé, etc;
- Avions : les appareils utilisés pour les vols, avec notamment une indication "check_OK" pour signifier qu'il est en état de vol (ou pas);
- **Bagages** : cette table remplacera bien sûr le fichier csv évoqué plus haut;
- **Agents** : cette table contient tous les intervenants de l'aéroport (agents de compagnies aériennes, bagagistes, employés agréés de tour-operators, aiguilleurs du ciel, etc).

On a bien sûr toute liberté pour ajouter des tables ou des champs supplémentaires aux tables existantes, voire des vues ou des contraintes, mais de manière limitée et strictement justifiée par le projet à développer ici.

2.2 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, idéalement des Java Beans mais sans utilisation d'un mécanisme d'events.

On demande donc de construire un groupe de telles classes (package **database.utilities**) 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 APPLICATION_TEST_JDBC de la petite librairie ainsi construite proposera un interface graphique de base permettant:

- soit de se connecter à la base MySQL BD_AIRPORT pour y réaliser 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);
- soit de se connecter à une base BD_JOURNALDEBORD, qui est une base Oracle simplissime à deux tables : ACTIVITES (cours, cours-labo et travail labo en équipe, date, description, référence de l'intervenant principal [= prof si cours ou cours-labo, étudiant si activité en équipe de laboratoire) et INTERVENANTS (profs et étudiants avec qui on travaille).

Rien n'interdit de lancer simultanément plusieurs instances de cette application : attention donc aux accès concurrents !

Les travaux de l'évaluation 2 : client-serveur multithread TCP en Java + programmation Web Java classique

Compétences développées :

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

Dossier attendu :

1. définition et implémentation des commandes du protocole LUGAP;
2. code Java du serveur Serveur_Bagages;
3. diagramme de classes UML des classes de l'application Web;
4. schéma de l'application Web en termes de servlets, JSPs et pages HTML (formalisme libre);
5. trames échangées et vues par un sniffer.

3. Le serveur Serveur_Bagages

Nous allons ici nous préoccuper de l'implémentation du modèle client-serveur pour le serveur **Serveur_Bagages** (clients: **Application_Bagages** et **Serveur_CheckIn**).

3.1 Serveur_Bagages

Ce serveur est donc un serveur multithread Java/Windows-Unix (en modèle pool de threads) qui est chargé de gérer tous les accès à la base de données BD_AIRPORT qui relèvent de la gestion des bagages (donc à l'exclusion des opérations portant sur les billets, qui relèvent de Serveur_Billets) :

- soit par les bagagistes des diverses compagnies aériennes;
- soit par les agents de la compagnie aérienne du checkIn, qui agissent par le serveur SerChk interposé.

Le serveur attend ses requêtes sur deux ports différents :

- le port PORT_BAGAGES pour les bagagistes;
- le port PORT_CHECKIN pour les requêtes provenant de Serveur_CheckIn.

3.2 Application_Bagages

Il s'agit donc ici de l'application destinée aux bagagistes. Pour interagir avec eux, le serveur utilise le ***protocole applicatif*** (basé TCP) **LUGAP** (**LUG**gage h**A**ndling **P**rotocol), dont les commandes sont à définir pour satisfaire au scénario exemple suivant :

exemple : *Les bagages de Mr et Mmes Charvilrom ont donc été enregistrés sous les identifiants 362-WACHARVILCAL-22082017-0070-001 à 362- WACHARVILCAL-22082017-0070-008.*

L'application présente donc un GUI qui permet tout d'abord à un bagagiste d'entrer dans l'application sur base d'un login-password (ce password ne passe pas en clair sur le réseau

mais sous la forme d'un **digest salé**). Ce digest sera construit en utilisant la librairie BouncyCastle.

En cas de succès, le bagagiste obtient alors une liste des vols prévus ce jour:

exemple : *Bagagiste : André Ouftinenni*

VOL 714 WALABIES-AIRLINES - Sydney 5h30
VOL 362 POWDER-AIRLINES - Peshawar 6h30
VOL 152 AIR FRANCE CANAILLE - Paris 7h20
…

Un double-clic sur un item de la liste fait apparaître dans une boîte de dialogue un tableau reprenant les bagages enregistrés pour ce vol (données provenant du fichier associé pour les 3 premières colonnes, initialisées avec les valeurs par défaut "N" ou "NEANT" selon le cas):

exemple : *Bagages de : VOL 362 POWDER-AIRLINES - Peshawar 6h30*

Identifiant	*Poids*	*Type*	*Réceptionné (O/N)*	*Chargé en soute (O/N)*	*Vérifié par la douane (O/N)*	*Remarques*
…						
362-WACHARVILCAL-22082017-0070-001	*27.3 kg*	*VALISE*	*N*	*N*	*N*	*NEANT*
…						
362-WACHARVILCAL-22082017-0070-008	*19.95 kg*	*PAS VALISE*	*N*	*N*	*N*	*NEANT*
…						

Le bagagiste, au fur et à mesure de ses activités, va interagir sur ce tableau. Chacune de ses actions va générer une commande spécifique du protocole LUGAP, commande envoyée au serveur Serveur_Bagages sur le port PORT_BAGAGES.

Sur base de l'exemple suivant, il vous appartient de définir les commandes du protocole LUGAP (et donc de leur donner un nom) et de choisir la manière de les implémenter (objets, chaînes de caractères, etc).

exemple : *Bagages de : VOL 362 POWDER-AIRLINES - Peshawar 6h30*

1) le bagagiste a réceptionné un bagage :

Identifiant	*Poids*	*Type*	*Réceptionné (O/N)*	*Chargé en soute (O/N)*	*Vérifié par la douane (O/N)*	*Remarques*

362-WACHARVILCAL-22082017-0070-001	*27.3 kg*	*VALISE*	***O***	*N*	*N*	*NEANT*

2) le bagagiste a réceptionné un 2ème bagage :

Identifiant	*Poids*	*Type*	*Réceptionné (O/N)*	*Chargé en soute (O/N)*	*Vérifié par la douane (O/N)*	*Remarques*
362-WACHARVILCAL-22082017-0070-008	*19.95 kg*	*PAS VALISE*	***O***	*N*	*N*	*NEANT*
…						

3) le bagagiste a alerté un agent des douanes pour un bagage :

Identifiant	*Poids*	*Type*	*Réceptionné (O/N)*	*Chargé en soute (O/N)*	*Vérifié par la douane (O/N)*	*Remarques*
362-WACHARVILCAL-22082017-0070-008	*19.95 kg*	*PAS VALISE*	***O***	*N*	***O***	*pas mauvais ☺*
…						

4) le bagagiste a réalisé le chargement en soute d'un bagage :

Identifiant	*Poids*	*Type*	*Réceptionné (O/N)*	*Chargé en soute (O/N)*	*Vérifié par la douane (O/N)*	*Remarques*
362-WACHARVILCAL-22082017-0070-001	*27.3 kg*	*VALISE*	***O***	***O***	*N*	*NEANT*

Le bagagiste ne peut refermer cette boîte de dialogue que si tous les bagages du vol ont été chargés en soute OU refusés au chargement (ce qui est indiqué par "R" dans la colonne "Chargé en soute") :

Identifiant	*Poids*	*Type*	*Réceptionné (O/N)*	*Chargé en soute (O/N)*	*Vérifié par la douane (O/N)*	*Remarques*

362-WACHARVILCAL-22082017-0070-007	*17.99 kg*	*PAS VALISE*	***O***	***R***	*O*	*Poudre à lessiver de la marque"SchnoufSuper" ???*

5) Quand cette boîte se referme, le bagagiste est automatiquement déconnecté du serveur.

Remarque: Pour la démonstration lors de l'évaluation, prévoir au moins 4 passagers relevant de 2 vols différents afin de pouvoir tester l'aspect multithread et l'aspect transaction.

4. L'application Web_Applic_Billets

Il s'agit donc bien sûr d'acheter en ligne des billets d'avion. On utilisera donc ici **HTTP** avec la technologie des servlets Java et des Java Server Pages.

Dans cette évaluation, il est simplement demandé de réaliser un caddie virtuel classique simple - une amélioration y sera ajoutées par après (évaluation 6). La base BD_AIRPORT sera accédée par les classes développées ci-dessus : elles sont donc le **M**odèle.

Techniquement, ce site est construit

♦ avec une page HTML statique qui propose à l'utilisateur de se connecter à l'application Web en permettant d'y entrer son identifiant et son mot de passe (une case à cocher spécifie si il s'agit d'un nouveau client); ces informations seront vérifiées/enregistrées côté serveur par une servlet qui sert de **C**ontrôleur;

♦ pour la suite de l'utilisation, selon le modèle MVC pour le caddie virtuel qui permet à l'utilisateur de réaliser des achats de billets disponibles (il n'y a évidemment qu'un nombre maximum de places, donc de billets, par vol): ceci consiste à se promener dans les pages du catalogue du site de ScienceAirport pour y choisir des billets au fur et à mesure.

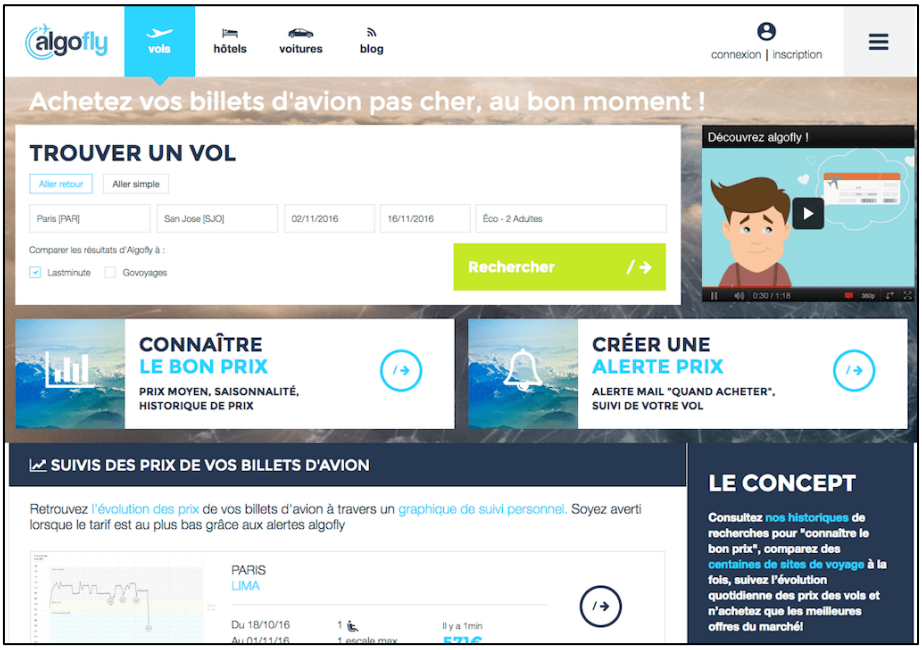
Finalement, on en arrive à réaliser les opérations de paiement. Plus précisément pour le caddie virtuel, on utilisera des Java Server Pages qui seront donc les **V**ues:

- **JSPInit** : Bonjour et initialisation du caddie.
- **JSPCaddie** : Choix de billets et ajouts au caddie.
- **JSPPay** : Confirmation des achats et envoi au paiement par carte de crédit (simple encodage - voir évaluation ultérieure pour un paiement effectif).

La servlet, qui travaille directement sur la base BD_AIRPORT, devrait en fait s'adresser à Serveur_Billets qui gère la vente des billets; mais comme ce serveur n'existe pas encore, la servlet attaquera donc directement la base (voir évaluation 6).

En ce qui concerne ***<u>la politique de gestion des caddies</u>***, on se contentera :

- de vérifier que le(s) demandé(s) est(sont) disponible(s) ;
- tout ce qui est placé dans un caddie est considéré comme une promesse ferme : autrement dit **<u>le client est assuré que ce qu'il a réservé ne peut être pris par un autre client</u>**;
- tout achat validé (c'est-à-dire payé) de billets doit bien sûr être mémorisé dans la base de données (pratique normale du commerce).



Les travaux de l'évaluation 3 : client-serveur sécurisé en Java et complément caddie virtuel en Java, communications réseaux C/C++-Java , client-serveur UDP en C/C++ et Java

Compétences développées :

- Maîtriser les concepts et l'implémentation des techniques cryptographiques classiques en Java;
- Savoir faire communiquer par réseau indifféremment des applications C/C++ et Java;
- Savoir utiliser la programmation UDP et le paramétrage des sockets en C/C++ et Java.

Dossier attendu :

1. définition et implémentation des commandes du protocole TICKMAP;
2. explication du contenu des différents keystores;
3. définition et implémentation des commandes du protocole ACMAP;

diagramme de la machine à états d'un vol.

4. diagramme du handshake pour l'échange des clés publiques;
5. trames échangées et vues par un sniffer lors du début des opérations.

5. Le serveur Serveur_Billets

5.1 Présentation du serveur

Ce serveur multithread **Serveur_Billets** est Java/Windows-Unix (en modèle pool de threads) gère donc tout ce qui touche à la vente des billets pour des vols partant de l'aéroport dans les 7 prochains jours. Il attend pour cela ses requêtes sur le port PORT_BILLETS, requêtes provenant des applications clientes **Application_Billets** (utilisée par les employés des tour-operators et agents des compagnies aérienne) et concernant les achats de billets.

Comme il s'agit de traiter des informations que l'on peut considérer comme sensibles en cette période de vagues terroristes et de protection de la vie privée, il faut envisager que les communications réseaux soient protégées au moyen des outils de cryptographie (clé secrète, clés publique et privée, message digest, signature digitale, HMAC).

Le serveur et ses clients utilisent le ***protocole applicatif*** (basé TCP) **TICKMAP** (**TICK**ests **MA**nagement **P**rotocol), dont les commandes utilisant des techniques cryptologiques correspondent au scénario décrit ci-dessous.

5.2 L'application Application_Billets

Sur base de l'exemple suivant, il vous appartient de définir les commandes du protocole TICKMAP (et donc de leur donner un nom) et de choisir la manière de les implémenter (objets, chaînes de caractères, etc).

1) L'utilisateur de l'application commence par entrer dans l'application sur base d'un login-password : à nouveau, ce password ne passe pas en clair sur le réseau mais sous la forme d'un **digest salé**.

2) En cas de réussite, une procédure de handshake pour le partage des deux clés symétriques est lancée.

On peut, dans un premier temps, se contenter de clés sérialisées dans des fichiers. Mais, dans un deuxième temps, les clés asymétriques et les certificats doivent se trouver dans des **keystores**. Dans un premier temps aussi (et seulement), on peut considérer que les clés

secrètes **symétriques** sont sérialisées dans des fichiers. Mais, dans un deuxième temps, ces clés secrètes doivent être échangées au moyen d'un chiffrement asymétrique : il y a donc alors une phase préalable de **handshake**.

Enfin, la clé symétrique utilisée par un employé pour s'authentifier (HMAC) n'est pas la même que celle qui lui permet de chiffre/déchiffrer : tout employé possède donc **deux clés symétriques** à usage différent.

3) En cas de succès de ce handshake, l'utilisateur obtient dans un tableau la liste des vols prévus sur les 7 jours à venir (destination, jour et heure, prix simple ou aller-retour, …). Il sélectionne le vol qui l'intéresse et introduit dans une boîte de dialogue toutes les informations nécessaires concernant le voyageur de référence et ses accompagnants. La fermeture de cette boîte provoque l'envoi d'une requête d'achat des billets correspondants: cette requête est **cryptée** au moyen de la clé secrète négociée lors du handshake.

4) Le serveur vérifie la disponibilité des places et, en cas de réussite, enregistre les réservations dans la base BD_AIRPORT. Il retourne les numéros de place attribuées sur le vol et le prix à payer, toujours de manière cryptée.

5) Le client doit confirmer son accord avec cette réponse - cette confirmation est accompagnée d'un **HMAC** de l'employé.

6) Mais il faut passer à présent au paiement de ces billets : le serveur attend donc à présent une confirmation du paiement, toujours accompagnée d'un **HMAC** de l'employé.

7) Pour cela, l'employé (et son voyageur) va effectuer le paiement en contactant (selon le protocole **PAYP** - **PAY Protocol** - à définir) un **Serveur_Payment** dont il possède un certificat. L'employé utilise la carte de crédit du voyageur pour envoyer à ce serveur le numéro de carte **crypté asymétriquement**, le nom du propriétaire et le montant de la transaction, le tout accompagné de la **signature** de l'employé (le certificat de celui-ci, plus exactement celui de son tour-operator, est supposé disponible sur Serveur_Payment).

8) Le résultat de la transaction (paiement accepté ou non) est enfin envoyé au Serveur_Billets qui, selon le cas, valide définitivement la réservation des places ou, au contraire, l'annule. Quel que soit le résultat, une trace de cette transaction sera conservée dans une table de la base de données BD_AIRPORT.

5.3 Adaptation de l'application Web

L'application Web_Applic_Billets ne réalise plus les paiements en attaquant la base de données BD_AIRPORT mais en passant par Serveur_Billets, comme si il était un employé de tour operator. Pour ce faire, l'application Web possède un certificat particulier connu de Serveur_Billets.

6. Les communications inter-langages

Cette dernière partie a pour but de finaliser le fonctionnement de notre aéroport.

6.1 L'accès de Serveur_CheckIn à Serveur_Bagages

On se souviendra que le serveur Serveur_CheckIn peut recevoir une requête CHECK_TICKET pour vérifier les billets. A présent, au lieu d'utiliser un fichier csv (évaluation 1), il s'adresse au Serveur_Bagages. Il convient donc de modifier/compléter (selon C ou C++ - polymorphisme) la librairie ***AccessBilBag*** pour permettre de remplacer de la manière la plus souple possible les accès fichiers par des accès réseaux.

6.2 Le grand final : les opérations de décollage

Les aiguilleurs du ciel régissent le trafic aérien, donc notamment les départs des vols. Ils opèrent différentes vérifications avant d'autoriser un avion à décoller et se servent pour cela d'une **Application_AirTrafficControllers** qui accède à un serveur **Serveur_AirTrafficControllers**.

Ce serveur multithread **Serveur_AirTrafficControllers** (SerTow) est Java/Windows-Unix (en modèle threads à la demande - attention au déni de service !) attend ses requêtes sur le port PORT_TOWER, requêtes provenant donc des applications clientes **Application_AirTrafficControllers**.

Le serveur et ses clients utilisent le ***protocole applicatif*** (basé TCP) **ACMAP** (**A**ir **C**ontrol **MA**nagement **P**rotocol), dont les commandes correspondent au scénario décrit ci-dessous.

Sur base de cet exemple suivant, il vous appartient de définir les commandes du protocole ACMAP (et donc de leur donner un nom) et de choisir la manière de les implémenter (objets, chaînes de caractères, etc).

A bien remarquer : dès le choix d'un vol, l'application cliente reste dévolue à ce vol et se trouve dans un état qui ne lui permet que d'envoyer telle ou telle requête au serveur.

1) L'application présente un GUI comportant une liste des vols qui sont prévus au décollage, depuis l'heure courante jusqu'à 3 heures plus tard. La sélection d'un vol fait apparaître une boîte de dialogue permettant de finaliser le décollage de ce vol. Le vol se trouve au départ dans l'état BUSY (les états du vol sont visualisés sur l'interface par un groupe de boutons-radio).

VOL 362 POWDER-AIRLINES - Peshawar 6h30					
BUSY	CHECKIN_OFF	READY	READY_TO_FLY	TAKING_OFF	FLYING
◉	○	○	○	○	○

2) Le première à chose à faire par l'aiguilleur responsable du vol est d'envoyer une requête au serveur Serveur_CheckIn afin de **signaler la fin des opérations de check-in**, si du moins on se trouve à moins de 20 minutes de l'heure de départ prévue. Serveur_CheckIn envoie à tous ses clients un avertissement de fin des opérations de check-in :

VOL 362 POWDER-AIRLINES - Peshawar 6h30

Numéro de billet ?

*** FIN DES OPERATIONS DE CHECK-IN ! ***

ainsi qu'au Serveur_Bagages qui en avise également ses applications clientes (une boîte de dialogue surgit pour cela dans Application_Bagages).
Le vol passe ainsi à l'état CHECKIN_OFF.

VOL 362 POWDER-AIRLINES - Peshawar 6h30					
BUSY	CHECKIN_OFF	READY	READY_TO_FLY	TAKING_OFF	FLYING
○	●	○	○	○	○

3) Il convient ensuite de s'assurer que tous les bagages ont été chargés en soute. L'aiguilleur va donc demander au Serveur_AirTrafficControllers de contacter Serveur_Bagages pour vérifier que tous les bagages sont chargés. Ou bien il reçoit une réponse dans les 10 minutes, ou bien il considère que le chargement est terminé au-delà de ce délai. Il envoie un message radio au pilote de l'avion pour faire fermer les portes et les soutes. Le vol passe ainsi à l'état READY.

VOL 362 POWDER-AIRLINES - Peshawar 6h30					
BUSY	CHECKIN_OFF	READY	READY_TO_FLY	TAKING_OFF	FLYING
○	○	●	○	○	○

4) L'aiguilleur va ensuite demander à son serveur la liste des pistes d'envol libres Ce qu'il reçoit sera affiché dans une boîte combo qui lui permettra de réaliser son choix. Il envoie ce choix au serveur.

5) En cas de réponse positive du serveur, il communique par radio le numéro de piste en question et le vol passe ainsi à l'état READY_TO_FLY - en cas de refus (un autre aiguilleur a pu choisir cette piste pour un autre vol), recommencer le point 4.

VOL 362 POWDER-AIRLINES - Peshawar 6h30					
BUSY	CHECKIN_OFF	READY	READY_TO_FLY	TAKING_OFF	FLYING
○	○	○	●	○	○

6) L'aiguilleur peut enfin donner par radio l'autorisation de décoller - le vol passe à l'état TAKING_OFF.

VOL 362 POWDER-AIRLINES - Peshawar 6h30					
BUSY	CHECKIN_OFF	READY	READY_TO_FLY	TAKING_OFF	FLYING
○	○	○	○	●	○

Quand il est avisé du décollage effectif, le vol passe dans l'état FLYING et l'avion est pris en charge par les écrans radars.

VOL 362 POWDER-AIRLINES - Peshawar 6h30					
BUSY	CHECKIN_OFF	READY	READY_TO_FLY	TAKING_OFF	FLYING

○	○	○	○	○	●

L'aiguilleur peut passer à un autre vol.

7. ScienceAirportChat

7.1 Fonctionnalités demandées

Les clients de l'aéroport (c'est-à-dire tous ceux qui ont acheté au moins un billet d'avion pour un vol d'ScienceAirport) ainsi que les employés des différentes compagnies aériennes peuvent discuter par chat pour s'informer sur le temps qu'il fait, savoir si il y a des promotions en vue, obtenir la liste des emplois vacants ou accéder à d'autres autres informations plus (f)utiles. C'est le "ScienceAirportChat", nom que nous abrégerons en "IAChat".

On est admis dans ce chat sur base de ses informations d'identification sur le serveur dont on dépend, donc un numéro de billet d'avion pour les voyageurs, son nom et mot de passe pour un employé.

7.2 Client UDP en Java

Au moyen d'une application **Application_JIAChat** écrite en Java, on se joint au groupe en UDP sur une adresse de classe D précise et un port PORT_CHAT précis qui aura été obtenu en s'adressant en TCP à un serveur **Serveur_IAChat** (SerIAC) qui vérifie l'état de voyageur ou l'appartenance à une compagnie aérienne dans la base de données BD_AIRPORT contenant les informations nécessaires à l'identification.

Les agents utilisent pour cela le protocole **IACOP** (**ScienceAirport C**ommunity c**O**nversation **P**rotocol). Basé principalement UDP, ce ***protocole applicatif*** utilise donc cependant au préalable une connexion TCP à SerIAC écrit en Java sous Windows; celui-ci n'attend sur le port PORT_FLY qu'une seule commande permettant de valider le voyageur ou l'employé :

protocole **IACOP (partie TCP)** - PORT_FLY		
application cliente : **Application_JIAChat**		
commande	**sémantique**	**réponse éventuelle**

LOGIN_GROUP	quelqu'un veut se joindre au groupe *paramètres* : nom et **digest "salé"** du mot de passe	oui + envoi de l'adresse et du port PORT_CHAT à utiliser ce jour; ou non

Le digest sera construit en utilisant le provider Bouncy Castle.
En cas de succès, l'agent pourra alors réellement participer au chat :

protocole **IACOP (partie UDP)** - PORT_CHAT		
application cliente : **Application_JIAChat**		
commande	**sémantique**	**réponse éventuelle**
POST_QUESTION	Pose question : un utilisateur pose une question et espère des réponses *paramètres* : un tag d'identification de question à utiliser dans la réponse (généré), la question sous forme de chaîne de caractères accompagnée d'un **digest** pour contrôler l'intégrité	une réponse à la question précédée du tag
ANSWER_QUESTION	Réponse à une question après vérification de son digest. *paramètres* : le tag d'identification de question et le texte de la réponse	une réponse à la question précédée du tag
POST_EVENT	Un utilisateur signale un fait, donne une information mais n'attend pas de réponse (un deuxième type de tag est cependant généré pour identifier l'événement)	-

Il s'agira évidemment de mettre d'abord en place le Serveur_IAChat, qui est un serveur Java qui tourne sur un PC et qui se révèle, du point de vue TCP, des plus simples puisque monothread et mono commande. A noter tout de même qu'il appartient aussi au groupe multicast UDP. Le multicast sera ensuite implémenté, tout d'abord dans le sous-réseau local.

7.3 Client UDP en C/C++

On élargira ensuite le chat en ajoutant une application **Application_CIAChat** écrite en C/C++ (pour les employés des compagnies aériennes qui utilisent plutôt des applications écrites dans ces langages).

En guise de conclusion

Les principaux développements évoqués dans ce dossier de laboratoire ont été définis avec le plus de précision possible.

Certains points ont cependant été laissés suffisamment vagues pour vous permettre d'envisager différents scénarii pour satisfaire à la demande. De plus, certaines pistes sont à peine entr'ouvertes - à vous de voir, si vous en avez le temps, ce que vous pouvez ajouter à vos développements pour leur apporter un plus valorisant. Comme toujours, **prudence** : l'avis de l'un des professeurs concernés est sans doute (un peu-beaucoup-très fort) utile ;-).

Soyez créatifs et imaginatifs … mais restez rationnels et raisonnables …

s: CV, CC, SC & JMW

Infos de dernière minute ? Voir l'Ecole virtuelle