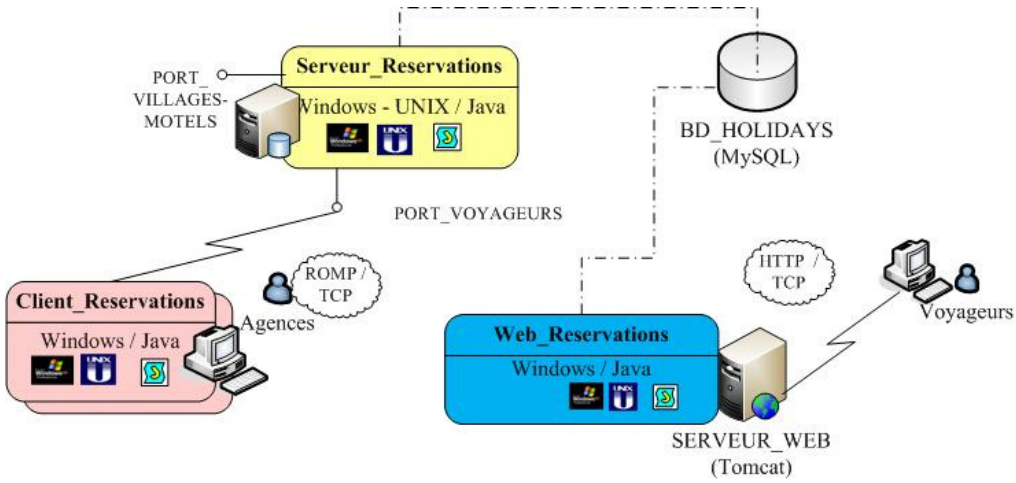
PROOM	Pay Room : accord et paiement de la réservation *paramètres* : n° de chambre, nom du client de référence et numéro de carte de crédit	
CROOM	Cancel Room : suppression d'une réservation de chambre *paramètres* : n° de chambre, nom du client	oui ou non + date dépassée
LROOMS	List of Rooms : liste des chambres d'hôtel réservées ce jour *paramètres* : -	numéros de chambres + noms des clients correspondant

Inutile sans doute de le préciser : comme ce genre d'informations concerne directement les voyageurs (et leur facture !), toutes les données nécessaires sont stockées dans la base de données BD_HOLIDAYS.



5. L'application Web_Reservations

Il s'agit donc bien sûr d'effectuer en ligne des réservations pour les motels et les villages. 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 3). La base

BD_HOLIDAYS 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 réservations de séjours: ceci consiste à se promener dans les pages du catalogue du site de IEYH pour y choisir une ou des chambres dans les villages et/ou les motels.

Une fois ces réservations effectuées, 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 chambres et ajouts au caddie.
- **JSPPay** : Confirmation des réservations et envoi du paiement par carte de crédit (simple encodage - voir évaluation ultérieure pour un paiement effectif).

La servlet travaille directement sur la base BD_HOLIDAYS en utilisant le package database.facility développé pour l'évaluation 2.

En ce qui concerne ***la politique de gestion des caddies***, on veillera à :

♦ vérifier la disponibilité des chambres choisies (sur base de la capacité de chaque village ou motel); également, on ne perdra pas de vue qu'un même voyageur ne peut pas être à deux endroits différents au même moment;

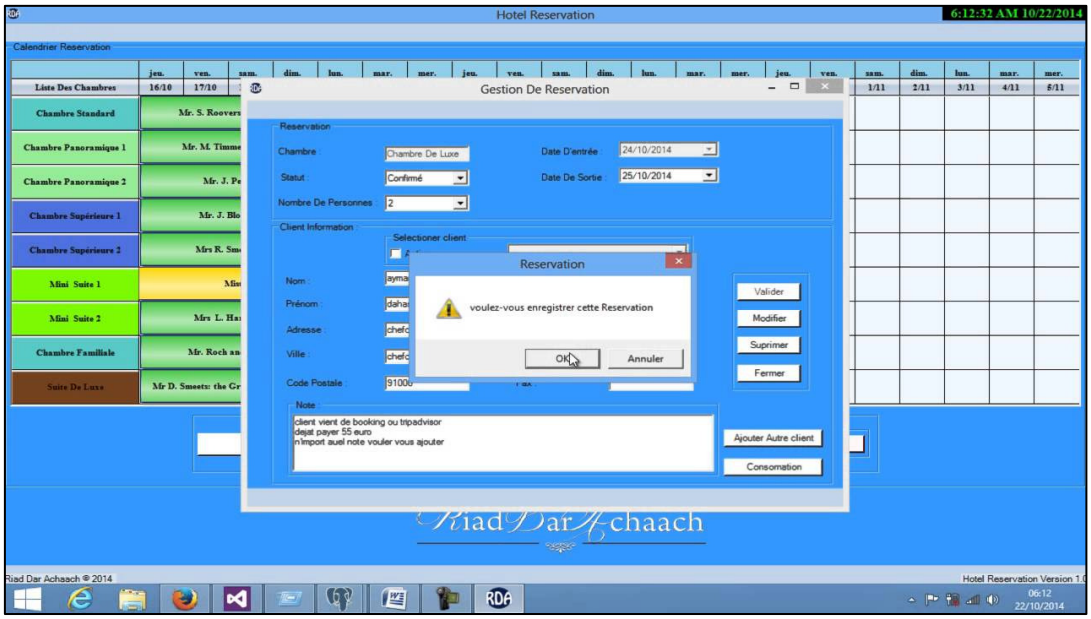
♦ tout ce qui est réservé et placé dans un caddie est considéré comme une promesse ferme : autrement dit **le client est assuré que ce qu'il a réservé ne peut être réservé par un autre client**;

♦ 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 qui a effectué des réservations (caddie non vide) ne clôture pas sa session correctement (il ne passe pas par la phase de paiement), il faut annuler ces réservations "pendantes", autrement dit reverser le contenu du caddie dans la table chambres disponibles.

Attention ! Le serveur Serveur_Reservations fonctionne en parallèle et permet donc toujours d'effectuer des réservations. Ces dernières peuvent donc entrer en concurrence avec les réservations pratiquées par l'application Web !

(*petit exemple …*)



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.
- Maîtriser les concepts d'administration distante d'un serveur;
- Savoir utiliser la programmation UDP et le paramétrage des sockets en C/C++ et Java.

Dossier attendu :

1. définition et implémentation des commandes des protocoles SPAYMAP et CCAPP;
2. explication du contenu des différents keystores;
3. diagramme du handshake pour l'échange des clés publiques;
4. trames échangées et vues par un sniffer lors du début des opérations.
5. définition et implémentation des commandes du protocole HSA;

6. Le serveur Serveur_Paiements

6.1 Présentation du Serveur_Paiements

Ce serveur multithread **Serveur_Paiements** est Java/Windows-Unix (en modèle pool de threads) gère donc tout ce qui touche aux paiements des réservations. Il attend pour cela ses requêtes sur le port PORT_PAY, requêtes provenant

- des applications clientes **Application_Paiements,** utilisée par les employés des agences de voyages pour leurs clients qui n'ont pas payé immédiatement l'intégralité de leur réservation (voir ci-dessous);
- du **Serveur_Reservations** (lui-même sollicité par les applications **Client_Reservations** et **Web_Reservations**).

Comme il s'agit de traiter des informations que l'on peut considérer comme sensibles en cette période de vagues cyber-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) **SPAYMAP** (**S**ecure **PAY**ment **MA**nagement **P**rotocol), dont les commandes utilisant des techniques cryptologiques correspondent au scénario décrit ci-dessous.

La nouveauté par rapport aux étapes de développement précédentes est donc qu'une réservation effectuée en agence n'est pas forcément payée immédiatement.

Si c'est néanmoins le cas, la commande PROOM du protocole ROMP a pour effet que Serveur_Reservations s'adresse alors au Serveur_Paiements. Il en est d'ailleurs de même pour Web_Reservations, car dans ce contexte Web, une réservation doit être payée immédiatement.

Sinon, ce sera l'application Application_Paiements qui sera ensuite utilisée pour les paiements (partiels ou non) effectués par le client par l'intermédiaire de l'agence de voyages.

6.2 L'application Application_Paiements et le serveur Serveur_Card

On peut donc imaginer que le client d'agence revient quelques jours plus tard payer une partie ou la totalité de sa (ses) réservation(s) : c'est donc dans ce contexte qu'intervient l'application **Application_Paiements**.

Sur base de l'exemple suivant, il vous appartient de définir les commandes du protocole SPAYMAP (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 (clé secrète) 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 réservations non encore payées. Il sélectionne la réservation qui l'intéresse et introduit dans une boîte de dialogue toutes les informations nécessaires de paiement (en particulier, la somme versée qui n'est pas forcément le montant total et le numéro de carte de crédit du client). La fermeture de cette boîte provoque l'envoi d'une requête de paiement: cette requête est **cryptée** au moyen de la clé secrète négociée lors du handshake et accompagnée de la **signature** de l'employé (le certificat de celui-ci, plus exactement celui de son tour-operator, est supposé disponible sur Serveur_Paiements).

4) Le 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.

Ce serveur est des plus simples: il n'attend sur son port PORT_CARD que ce seul type de requête de vérification/débit (protocole **CCAPP** [**C**heck **CA**rd and **P**ayment **P**rotocol]), le numéro de carte et la somme à débiter étant **cryptés asymétriquement** accompagné de la **signature** du Serveur_Paiements (celui-ci dispose de 7 couples de clés, une par jour). Ce 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).

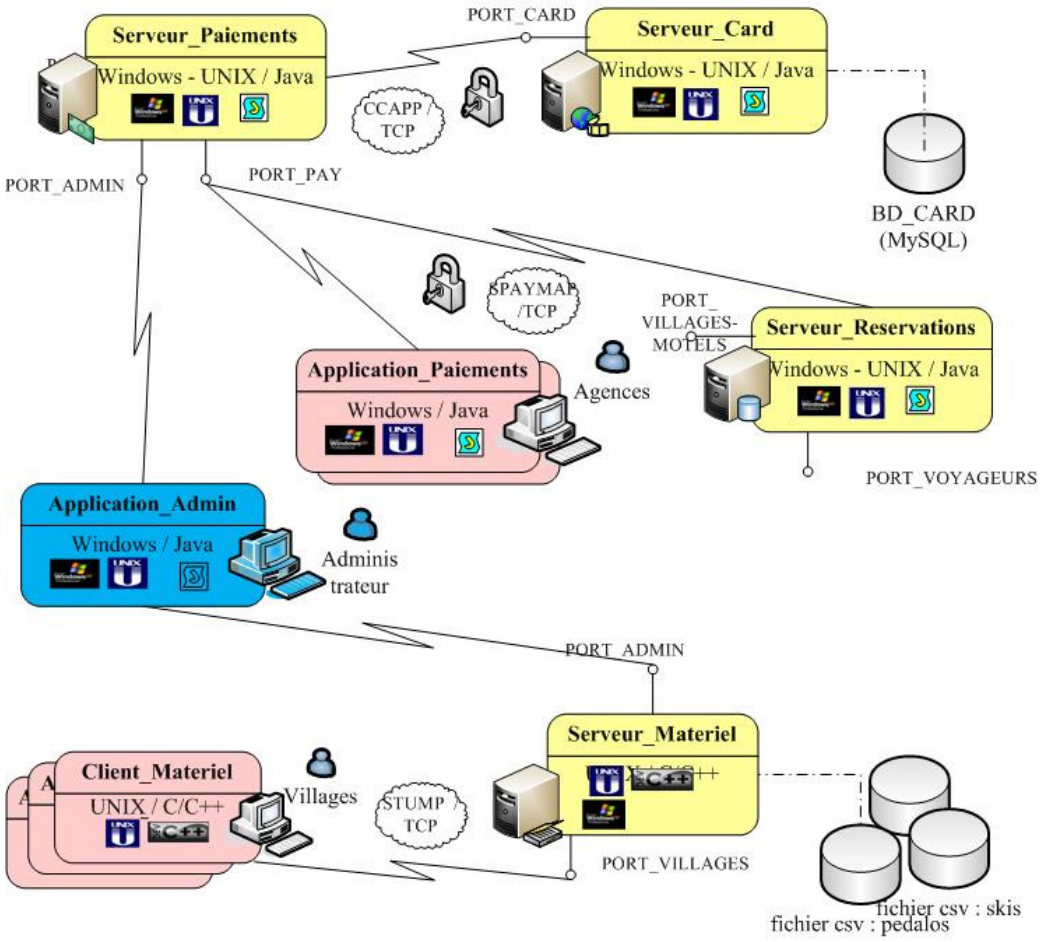
En cas de succès, Serveur_Paiements effectue et enregistre les modifications sur les réservations dans la base BD_HOLIDAYS. Il retourne le solde restant du, toujours de manière **cryptée**. Il retourne un numéro de transaction financière au Serveur_Paiements, lequel le retourne au client.

5) Le client confirme la réception de ce numéro de transaction au moyen d'un **HMAC** de ce numéro et du login de l'employé.

6) Si le solde est payé entièrement après ce paiement, la facture finale de la réservation peut être générée et envoyée, accompagnée de la **signature** du Serveur_Paiements.

6.3 Adaptation de l'application Web

L'application **Web_Reservations** ne traite plus les paiements en attaquant la base de données BD_HOLIDAYS directement mais en passant par Serveur_Paiements, comme si il était un employé d'agence (donc comme applications **Client_Reservations**).



7. L'administration des serveurs Serveur_Paiements et Serveur_Materiel

Les serveurs **Serveur_Paiements** (**Java**) et **Serveur_Materiel** (**C/C++**) attendent chacun sur un PORT_ADMIN les requêtes d'une application **Application_Admin** pour permettre aux responsables informatiques d'administrer ces serveurs. Un client de ce type peut donc administrer ces serveurs à distance du point de vue technique (arrêt, reprise, liste des clients connectés). L'application **Application_Admin** est programmée en **Java** et permet donc de choisir le serveur à administrer. Elle fonctionne sur un PC Windows ou sur une machine UNIX (machine Jupiter); elle utilise le protocole **HSA** (**H**olidays **S**ervers **A**dministration). Ce ***protocole applicatif***, (basé TCP), a pour commandes :

protocole **HSA** - PORT_ADMIN		
application cliente : **Application_Admin**		
Commande	**sémantique de la requête**	**réponse éventuelle**
LOGINA	un administrateur autorisé se connecte *paramètres* : nom, password	oui ou non (par exemple : un autre administrateur est déjà connecté et il ne peut y en avoir qu'un seul à la fois)
LCLIENTS	List CLIENTS : liste des préposés connectés paramètres : -	pour chaque client : adresse IP
PAUSE	PAUSE Server : le serveur est mis en pause temporaire; les clients sont prévenus; les nouveaux clients sont refusés. paramètres : -	oui – serveur suspendu (si une commande est en cours, elle est arrêtée) ou erreur
STOP	STOP Server: on réalise un shutdown du serveur en prévenant les clients de l'imminence de l'arrêt *paramètres* : nombre de secondes avant arrêt	oui ou erreur

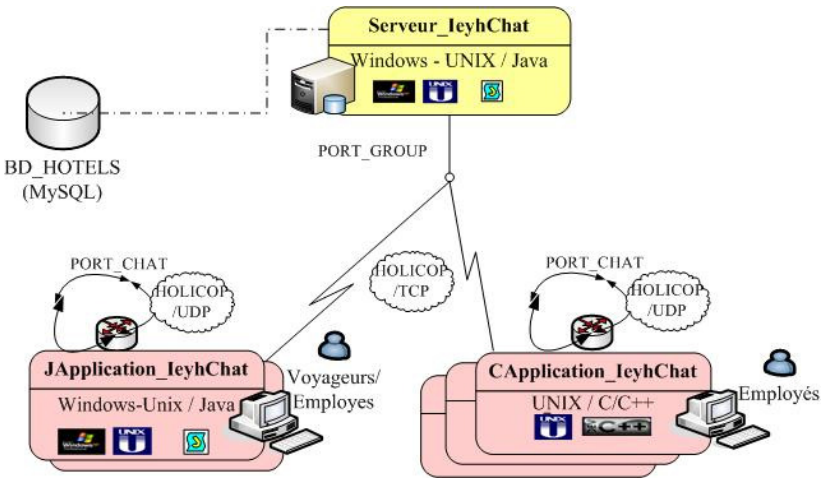
Pour que les clients connectés de ces deux serveurs soient avisés, *de manière asynchrone*, d'un arrêt temporaire ou d'un shutdown du serveur considéré (dans ce dernier cas, afin de se terminer en douceur), il convient de choisir une stratégie :
- ou chaque client normal s'est connecté sur un deuxième port (PORT_URGENCE) et attend des messages urgent par cette voie;
- ou le serveur considéré se connecte sur chaque client en cas de nécessité (le client aura envoyé son port d'écoute lors de sa connexion sur PORT_PAY ou PORT_VILLAGES).

8. IEYHChat

8.1 Fonctionnalités demandées

Les voyageurs qui logent dans les villages (c'est-à-dire tous ceux qui ont effectué et payé une réservation de ce type) ainsi que les employés de IEYH peuvent discuter par chat pour s'informer sur le temps qu'il fait, savoir si il y a des promotions en vue sur les activités, connaître le nom de la nouvelle animatrice/du nouvel animateur, échanger leurs impressions/sensations ou simplement accéder à d'autres autres informations plus (f)utiles. C'est le "IEYHChat".

On est admis dans ce chat sur base de ses informations d'identification sur le serveur dont on dépend, donc un numéro de réservation pour les voyageurs, son nom et mot de passe pour un employé de IEYH ou encore pour un employé d'une agence de voyages agréée par IEYH.



8.2 Client UDP en Java

Au moyen d'une application **JApplication_IeyhChat** 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_IeyhChat** qui vérifie l'état de voyageur, l'appartenance à IEHY ou à une agence dans la base de données BD_HOLIDAYS (contenant donc les informations nécessaires à l'identification).

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

protocole **HOLICOP (partie TCP)** - PORT_ GROUP		
application cliente : **JApplication_IeyhChat**		
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 **HOLICOP (partie UDP)** - PORT_CHAT		
application cliente : **JApplication_IeyhChat**		
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_IeyhChat, 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 en localhost, puis dans le sous-réseau local. En pratique, on utilisera pour cela wifietudiantprotect. De plus, pour rappel, il faut sélectionner l'interface réseau correctement pour l'adresse multicast avec setNetworkInterface(java.net.NetworkInterface netIf).

8.3 Client UDP en C/C++

On élargira ensuite le chat en ajoutant une application **CApplication_IeyhChat** écrite en C/C++ (pour les employés de IEYH qui gèrent le matériel des activités et 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, JMW & SC

Infos de dernière minute ? Voir l'Ecole virtuelle