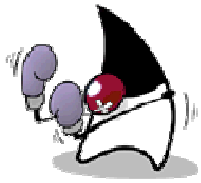
Laboratoire de Réseaux et technologie Internet (TCP-UDP/IP & HTTP en C/C++/Java) : projet "Inpres-Holidays"

3ème Informatique de gestion & 3ème Informatique et systèmes (opt. Informatique industrielle et Réseaux-télécommunications) 2012-2013

Claude Vilvens, Christophe Charlet, Mounawar Madani, Jean-Marc Wagner
(Network programming team)




1. Préambule

Les travaux de programmation réseaux présentés ici constituent la description technique des travaux de laboratoire du cours de "**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 cours de "**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 titres associés portent une astérisque).

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, Dev-C++ ou Code::Blocks sur les PCs peuvent vous permettre de développer du code en dehors de ces machines. Les développements Java sont à réaliser avec **NetBeans 6/7.***. Le serveur Web avec moteur à servlets/JSP utilisé est **Tomcat 6/7.*** sous Windows (PC) et/ou Unix (machines U2 ou Inxs).

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).

2. Règles d'évaluation du laboratoire

Afin d'éviter tout problème lié à l'évaluation du cours de Réseaux et technologie Internet, il a été établi un document pédagogique définissant les règles de cotation utilisées par les enseignants de l'équipe responsable du cours.

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

♦ théorie : un examen écrit en janvier 2013 (sur base d'une liste de questions fournies en novembre et à préparer) et coté sur 20;
♦ laboratoire : 3 évaluations (aux dates précisées ci-dessous), chacune cotée sur 20; la moyenne de ces 3 cotes fournit une note de laboratoire sur 20;
♦ note finale : **moyenne pondérée de la note de théorie (poids de 4/10) et de la note de laboratoire (poids de 6/10)**.

Cette procédure est d'application tant en 1ère qu'en 2ème session, ainsi que lors d'une éventuelle prolongation de 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 (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, signature électronique, certificat, etc).

4) En 2ème session, un **report de note** est possible pour chacune des trois notes de laboratoire ainsi que pour la note de théorie **pour des notes supérieures ou égales à 10/20**.
Toutes les évaluations (théorie ou laboratoire) ayant des **notes inférieures à 10/20** sont **à représenter dans leur intégralité**.

5) En prolongation de session, un **report de note** est possible **pour des notes supérieures ou égales à 10/20** :
- pour la note de laboratoire mais **seulement pour sa totalité** (donc pour la somme des différentes évaluations, pas pour l'une ou l'autre partie);
- pour la note de théorie.

Les évaluations (théorie ou laboratoire dans sa totalité) ayant des **notes inférieures à 10/20** sont donc **à représenter dans leur intégralité**. Mais de plus :
- pour l'examen théorique : les réponses écrites seront présentées et explicitées oralement à deux professeurs responsables du cours;
- pour l'examen de laboratoire : on gardera le même contexte, mais des fonctionnalités différentes pourront être demandées en lieu et place d'anciennes.

La première partie des travaux de programmation réseaux (applets, threads, JDBC, servlets, JSP) sera **évaluée** par l'un des professeurs du laboratoire **à partir du 22 octobre 2012** (avec rentrée ce 22 octobre à 9h au plus tard d'un dossier papier limité au code de l'application Web - le délai est à *respecter* impérativement).

La deuxième partie de ces travaux sera **évaluée** par l'un des professeurs du laboratoire **à partir du 26 novembre 2012** (avec rentrée ce 26 novembre à 9h d'un dossier papier limité au code des serveurs - le délai est toujours à *respecter* impérativement).

La troisième partie sera évaluée lors de l'examen de laboratoire en janvier-février 2013 (le dossier papier n'est plus nécessaire).

3. Le contexte général

Nous sortons de la période des vacances d'été. Afin de prolonger cette délicieuse tranche de vie, nous vous proposons de mettre en place une plate-forme de gestion d'hôtels et de villages de vacances appartenant à la célèbre chaîne de loisirs Inpres-Holidays.

Cette infrastructure se compose de :

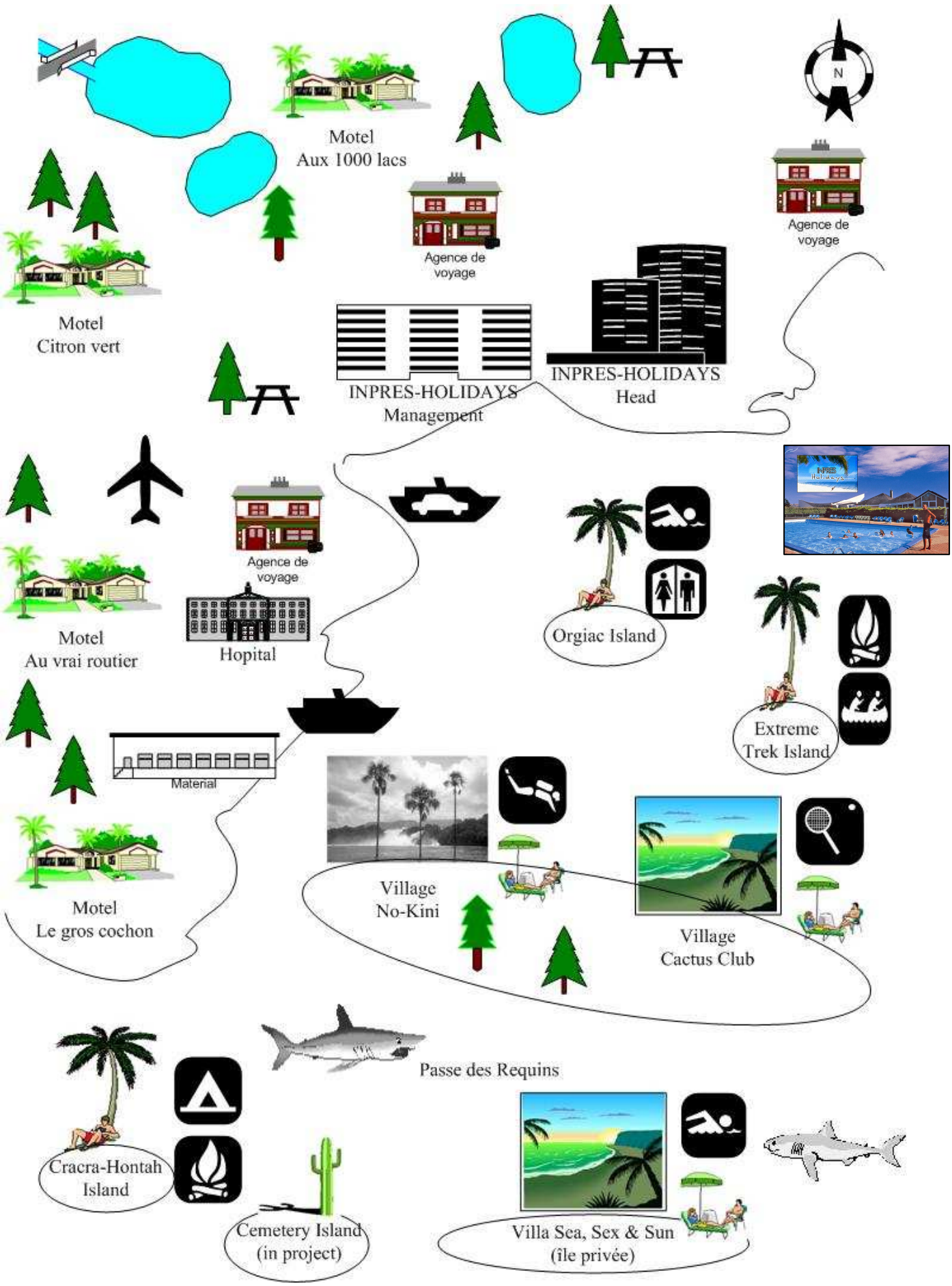
♦ plusieurs motels (comme " Motel Au vrai routier ", "Motel Citron vert", etc), situés sur le continent (le plus souvent à proximité des autoroutes et des zonings industriels) et qui se limitent à la location de chambres pour routiers, voyageurs de passage, etc; seul le petit déjeuner peut être obtenu;

♦ deux villages de vacances ("Village No-Kini" et "Village Cactus Club") installés sur une île paradisiaque et qui proposent, outre des mangeailles monstrueuses, de nombreuses activités pour passer un séjour agréable : plages, pédalos, court de tennis, cinéma (pas de réservation nécessaire) ou cours et formation diverses (ski nautique, vie dans la nature, etc) mais encore aussi participation à des "excursions" à "cracra-hontah" (durée : 5 jours), "extreme trek" (durée : 7 jours) ou "orgiac island" (durée : 3 jours – faut pas exagérer); ces excursions sont organisées le 1er et le 15 de chaque mois;

♦ un centre de gestion des réservations de séjours, un centre de gestion des activités et un centre de maintenance de matériel nécessaire, tous situés sur le continent et en relation réseau avec les motels (seulement pour les réservations de séjours) et les villages;

♦ le siège central de la société Inpres-Holidays et le siège du management, avec ses bureaux d'études (notamment en gestion-marketing).

Schématiquement, l'environnement est donc celui-ci :


N
Motel
Aux 1000 lacs
Agence de
voyage
Agence de
voyage
Motel
Citron vert
INPRES-HOLIDAYS
Management
INPRES-HOLIDAYS
Head
Agence de
voyage
Motel
Au vrai routier
Hopital
Orgiac Island
Extreme
Trek Island
Material
Motel
Le gros cochon
Village
No-Kini
Village
Cactus Club
Passe des Requins
Cracra-Hontah
Island
Cemetery Island
(in project)
Villa Sea, Sex & Sun
(île privée)

4. L'infrastructure informatique générale

La gestion de la plate-forme évoquée ci-dessus a nécessité la mise en place de plusieurs serveurs que nous allons décrire brièvement ici.

1) un serveur multithread **Serveur_Reservations** Java/Windows-Unix qui est essentiellement dévolu à la gestion des chambres des motels et des villages. Il est chargé de satisfaire les requêtes provenant

- d'une application installée classique **Applic_Reservations** utilisée par les agences de voyages qui effectuent des réservations pour les motels et les villages (ou les annulent) ou encore par une application Web **Web_Applic_Reservations** qui permet au public d'effectuer des réservations pour les motels et les villages par Internet; le serveur attend ce type de requête sur le PORT_VOYAGEURS; : les agences de voyages possèdent un mot de passe, signent électroniquement leurs demandes, cryptent les données sensibles comme les noms ou les numéros de cartes, etc;
- d'une application **Applic_Occupations** utilisée par les responsables des motels et des villages; ils visualisent les séjours réservés chez eux; le serveur attend ce type de requête sur le PORT_VILLAGES-MOTELS;

Le serveur et toutes ces applications sont programmés en Java. La base de données BD_HOTELS utilisée par le serveur est une base MySQL.

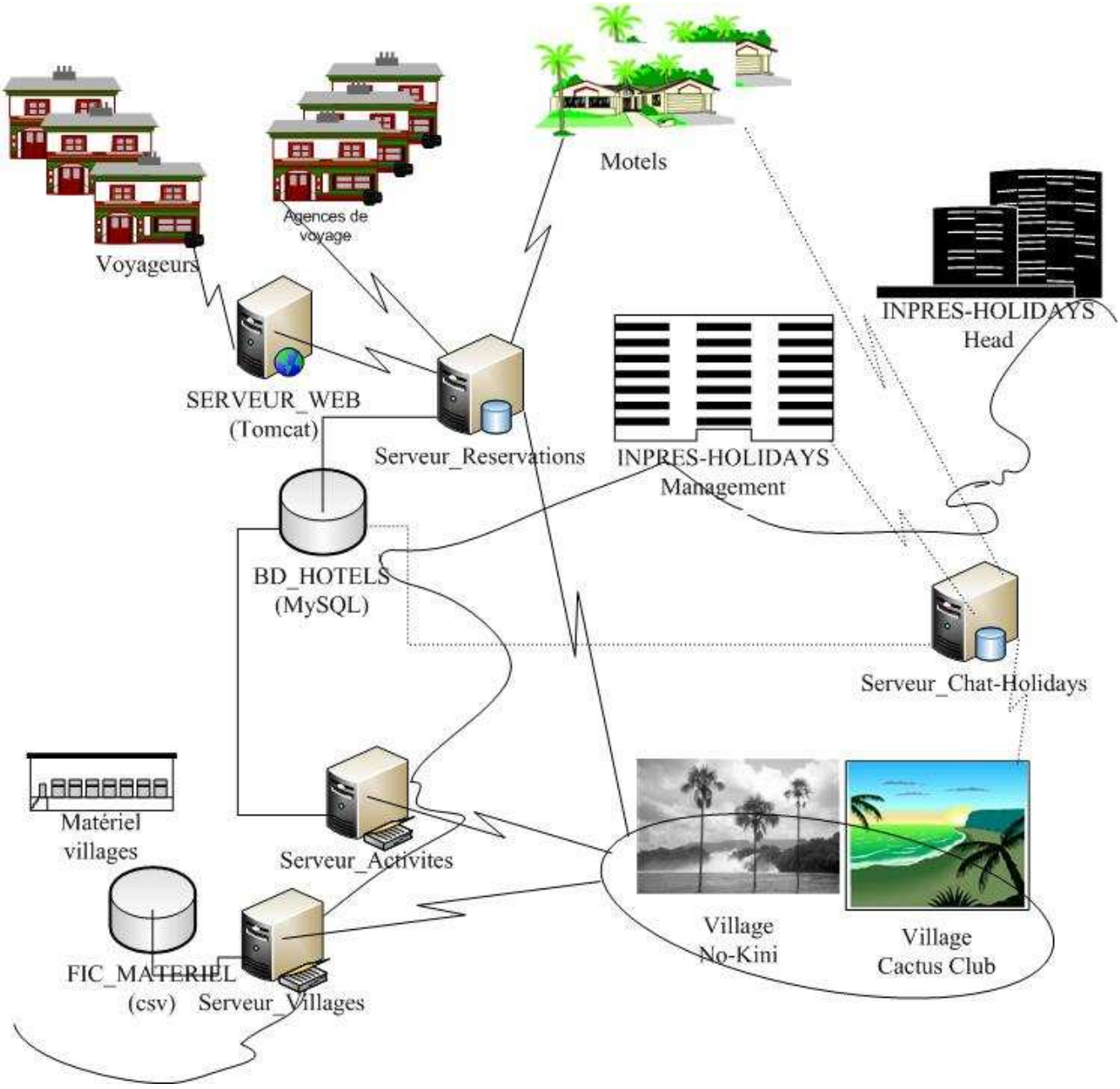
2) un serveur multithread **Serveur_Villages** C-C++/ Unix qui est essentiellement dévolu à la gestion du matériel nécessaire aux activités spécifiques des villages de vacances (fournitures, maintenance, etc). Il est chargé de satisfaire les requêtes provenant

- d'une application **Applic_Materiel** (C/C++) utilisée par les villages de vacances; le serveur attend ce type de requête sur le PORT_VILLAGES;
- d'une application Java **Admin_Terminaux** (Java) pour permettre aux responsables informatiques d'administrer ce serveur; le serveur attend ce type de requête sur le PORT_ADMIN.

3) un serveur multithread **Serveur_Activites** Java/Windows-Unix qui est essentiellement dévolu à la gestion des activités spécifiques des villages de vacances (inscription aux activités qui nécessitent une réservation). Il est chargé de satisfaire les requêtes provenant d'une application **Applic_Activites** (Java) utilisée par les responsables seuls habilités au travail de gestion des activités village; le serveur attend ce type de requête sur le PORT_ACTIVITES.

4) un serveur **Serveur_Chat-Holidays** Java/Windows-Unix qui vérifie l'appartenance à la société Inpres-Holidays et qui permet aux utilisateurs d'une application **Applic_Chat-Holidays** (Java) ou **Applic_C-H** (C/C++) de se joindre au groupe de discussion de la société; le serveur donne le sésame du chat en attendant les requêtes sur le port PORT_GROUP.

Schématiquement, l'environnement des serveurs est donc celui-ci :



5. Avertissement préalable

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 ou l'obscur. Dans tous les cas, l'imagination est la bienvenue pour donner à vos solutions un cachet plus personnel, plus réaliste, plus professionnel. 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.

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

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

Les travaux de l'évaluation 1 : programmation Web avec threads et beans

Nous allons ici nous préoccuper de l'application Web **Web_Applic_Reservations** qui permet au public d'effectuer par Internet des réservations pour les motels et les villages.

Les technologies utilisées ici sont les applets, les threads, JDBC, les servlets et les Java Server Pages.

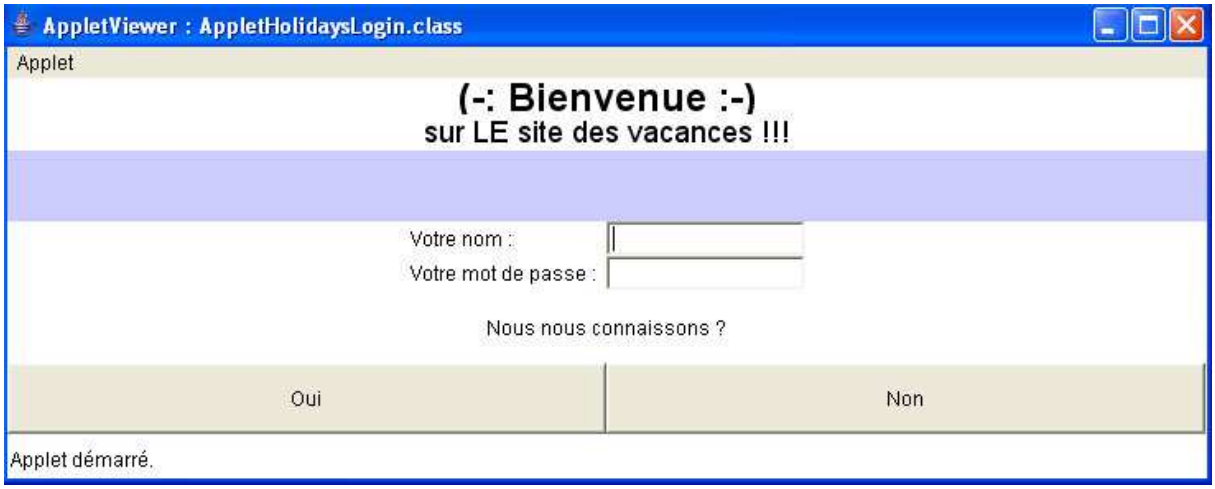
1. Le login sur l'application Web : les applets

1.1 Deux applets Java

Il s'agit de confectionner un interface de type GUI qui permettra aux clients utilisant l'application Web **Web_Applic_Reservations** (voir plus loin) d'accéder au site Web pour effectuer des réservations de séjour dans un motel ou un village. Bien sûr, à ce premier stade, il ne saurait y avoir de réaction côté serveur (ce sera le rôle de servlets et de Java Server Pages évoquées plus loin).

L'interface est constituée d'une page HTML contenant deux applets :

♦ l'applet de login se présente sous la forme d'un GUI demandant l'introduction d'un nom et prénom (séparés par un espace – comme sur l'EV ;-)) et d'un mot de passe, donc selon le canevas suivant :



"Oui" correspond à un client qui possède déjà un mot de passe (donc qui s'est déjà identifié par le passé), "Non" à un nouveau client qui n'en possède pas encore (ou à un client qui l'a oublié) Dans ce dernier cas, il peut créer un mot de passe au moyen de la 2ème applet :

♦ l'applet Password se présente sous la forme du canevas suivant (elle récupère le nom-prénom grâce à la première applet – il y a donc communication entre les deux applets) :

L'appui sur le bouton "Valider" provoquant la recopie du mot de passe dans la zone "Votre mot de passe" de la première applet Login.

L'appui sur le bouton "Oui" enverra provisoirement, en cas de vérification réussie, sur une page statique de bienvenue. Par après, ce sera une servlet qui réagira (voir encore une fois point 13)

La page HTML et ses deux applets seront déployées sur un serveur Tomcat non local tournant sur un PC.

1.2 Version avec threads

L'interface de l'applet de login va se compléter de 3 affichages supplémentaires :

il s'agit donc des affichages des heures locales de Bruxelles, New York et Tokyo (oui, vous avez raison : ce sera plus joli quand vous aurez tenu compte des formats d'affichage local des dates ;-)). Le plus raisonnable est de d'abord créer 3 threads (une par heure locale – le thread travaille pour un fuseau donné) et les tester dans une application :

pour ensuite jouer du "copier-coller" dans l'applet.

1.3 Version avec moniteur

Les trois threads génèrent un nombre aléatoire qui est additionné dans un objet totalisateur. Un quatrième thread va y rechercher périodiquement la somme et, si elle est multiple d'un nombre "de chance" bien précis (par exemple 4 ou 77 – **ultérieurement, il sera fourni par une servlet**), tout numéro de carte de crédit se terminant par le nombre affiché aura une réduction de 10% sur le prix à payer (pour l'instant, ceci se limite à un simple affichage, puisque notre client n'est pas encore en relation avec un serveur en ce point de l'énoncé) :

2. Les accès aux bases de données

2.1 La base de données BD_HOTELS

Cette base MySQL BD_HOTELS 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 :

- **voyageurs** (voyageurs au nom de qui la réservation est faite = titulaires) : numéro de client (celui dont il est question dans les applets ci-dessus), nom, prénom, adresse du domicile, adresse e-mail;
- **voyageurs-accompagnants** : nom, prénom, adresse du domicile, voyageur titulaire identifié par son numéro de client);
- **reservations** : identifiant (de la forme 20110915-RES01), chambre, voyageur-titulaire, payé (O/N), …
- **chambres** : numéro (le 1er chiffre identifie un village ou un motel), douche/baignoire/cuvette, nombre d'occupants, prix HTVA;
- **activités** : identifiant, type (choisi dans une liste), nombre maximum de participants, prix HTVA, …

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

2.2 Un bean 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. On demande donc de construire un groupe de classes (**BeanBDAccess***xxx* …) permettant l'accès (c'est-à-dire à tout le moins la connexion et les sélections de base) le plus simple possible.

Outre l'accès aux bases relationnelles de type MySQL ou Oracle, on souhaite aussi pouvoir accéder à des documents CSV, c'est-à-dire des fichiers textes correspondant à une table SQL dont les tuples sont des lignes de texte avec des champs séparés par un séparateur convenu (ces fichiers sont très en vogue à cause de leur simplicité par rapport à XML – de

plus, un tableur comme Excel sait les reconnaître ou les générer). Un exemple d'un tel fichier pourrait être celui qui comporte les champs nom, prénom, type (1=privé, 2=entreprise, password;

F_AGENTS.csv
nom, prénom, niveau, password Lendormi, Julien, 2, Arglxx007 Lajolie, Julie, 1, UnJourMonPrinceViendra Lenaif, Albert, 1, azerty Sculpturale, Valérie, 2, Johnny69 …

Le programme de test de la petite librairie ainsi construite

- se connectera au choix à une base MySQL ou CSV;
- enverra une requête de type "select count(*) from" et affichera le résultat;
- enverra une requête de type "select * from where" (adaptée à la table visée – pas de tentative de généricité à ce stade) et affichera le résultat.

3. L'application Web_Applic_Reservations

On utilise donc ici **HTTP** avec la technologie servlets/Java Server Page (modèle MVC) et applet.

En cas de succès du login du client, la page HTML obtenue propose les trois services suivants

1. Valeur du numéro de chance (à l'heure GMT)
2. Locations et réservations (caddie virtuel)
3. Fin de session avec tirage au sort d'un cadeau si il y a eu achat

Techniquement, ce site est construit

- selon le modèle applets-servlet pour le login; il s'agit donc d'adapter le point précédent pour que l'applet, au lieu d'enregistrer et/ou de vérifier le mot de passe du client, contacte une servlet de login pour réaliser ces opérations;
- selon le modèle applet-servlet par tunnel http pour obtenir la valeur courante du numéro de chance (comme évoqué ci-dessus dans la description des premières applets);
- selon le modèle MVC pour le caddie virtuel qui permet à l'utilisateur de réaliser des réservations : ceci consiste à se promener dans les pages catalogues du site de Inpres-Holidays, à choisir au fur et à mesure des réservations et finalement à réaliser alors les opérations de paiement. Les réservations sont gérées

* au niveau de l'évaluation 1 : par la servlet elle-même qui accède à la base de données BD_HOTELS au moyen d'un Java Beans de la librairie développée au point précédent; mais il faut programmer ceci de manière **modulaire** car …
* au niveau de l'évaluation 3 : par le serveur **Serveur_Reservation** qui est contacté par la servlet comme le fait l'application installée **Applic_Reservations**, donc selon le protocole RMP (voir évaluation 3 pour plus de détails).

Plus précisément pour le caddie virtuel, on utilisera des Java Server Pages :

- **JSP 1** : Bonjour et initialisation du caddie
- **JSP 2** : Choix et ajouts au caddie

♦ **JSP 3** : Confirmation des achats et paiement par carte de crédit.

En ce qui concerne ***la politique de gestion des caddies***, les principes suivants seront d'application :

♦ tout ce qui est placé dans un caddie est considéré comme une promesse ferme : autrement dit, pour peu que le client termine son marché **dans un délai raisonnable, il est assuré que ce qu'il a choisi ne peut être pris par un autre client**;

♦ si un client tarde plus d'une demi-heure (par exemple) ou encore si il ferme son navigateur sans avoir conclu, ce qui se trouvait dans son caddie est remis sur le marché

♦ tout achat validé doit être mémorisé dans la base de données (pratique normale du commerce).

Les travaux de l'évaluation 2 : client-serveur TCP en C/C++ et Java

Nous allons ici nous préoccuper de l'implémentation du modèle client-serveur pour les serveurs **Serveur_Villages** (client: **Applic_Materiel**) et **Serveur_Activites** (client: **Applic_Activites**).

Les technologies utilisées ici sont les sockets TCP (en C et en Java), les librairies réseaux génériques et dédiées, les threads (en C et en Java), le modèle des serveurs génériques.

1. Serveur_Villages

1.1 Le service matériel

Pour rappel, il 'agit d'un serveur multithread C-C++/ Unix (en modèle pool de threads) qui est essentiellement dévolu à la gestion (livraison, maintenance, déclassement, etc) du matériel nécessaire aux activités spécifiques des villages de vacances (pédalos, hors-bord, …). Le matériel disponible est mémorisé de manière simple (voire simpliste) dans des fichiers .csv (consultables avec un tableur comme Excel).

Le serveur est tout d'abord chargé de satisfaire les requêtes provenant d'une application **Applic_Materiel** (C/C++) utilisée par les responsables des villages de vacances; le serveur attend ce type de requête sur le PORT_VILLAGE. Il utilise le ***protocole applicatif*** (basé TCP) **FHMP** (**F**unny **H**olidays **M**anagement **P**rotocol) dont les commandes sont

protocole **FHMP**		
commande	**sémantique**	**réponse éventuelle**
LOGIN	un gestionnaire de matériel veut se connecter *paramètres* : nom et mot de passe	oui ou non + cause
BMAT	Booking Material : demande d'une action (livraison, réparation ou déclassement) portant sur du matériel existant (pédalo, jet-ski, barque, …) *paramètres* : action, matériel, date souhaitée	oui + identifiant donné à l'action ou non + pourquoi (par exemple : livraison impossible car produit commandé pas encore arrivé)
CMAT	Cancel Material: suppression d'une action commandée antérieurement *paramètres* : identifiant de l'action, nom	oui ou non + date dépassée
ASKMAT	Asking for Material : commande de matériel d'un type existant déjà ou pas (1 seul item avec accessoires éventuels) *paramètres* : type existant ou pas, libellé, marque, prix approximatif, liste accessoires	oui + identifiant donné à l'action à utiliser pour solliciter la livraison + délai ou non + pourquoi

1.2 Quelques conseils méthodologiques pour le développement de FHMP

1) Il faut tout d'abord choisir la manière d'implémenter les requêtes et les réponses des protocoles FHMP 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 **bibliothèque** (**librairie**) facilitant la manipulation des instructions réseaux. Selon ses goûts, il s'agira
- soit d'une bibliothèque **C** de fonctions réseaux TCP/IP;
- soit d'une bibliothèque **C++** implémentant une *hiérarchie de classes* **C++** utiles pour la programmation réseau TCP/IP : par exemple, Socket, SocketClient et SocketServeur, éventuellement (mais cela devient très(trop) ambitieux pour le temps dont on dispose), si cela est estimé utile, flux réseaux d'entrée et de sortie NetworkStreamBase, ONetworkStream et INetworkStream).

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 "Inpres-Ferries" … Ainsi :

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

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. 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_villages.conf
Port_Village=70000 Port_Admin=70009 sep-trame=$ fin-trame=# sep-csv=, pwd-admin=maitrevcmw …

1.3 L'administration du serveur

Le serveur attend sur le PORT_ADMIN les requêtes d'une application Java **Admin_Terminaux** pour permettre aux responsables informatiques d'administrer ce serveur. Un client de ce type peut donc administrer le serveur à distance du point de vue technique (arrêt, reprise, liste des machines du réseau). L'application **Admin_Terminaux** est programmée en Java et fonctionne sur un PC Windows ou sur une machine UNIX (machine Sunray); elle utilise le protocole **FHMPA** (**FHMPA**dministration). Ce ***protocole applicatif***, (basé TCP), a pour commandes :

protocole **FHMPA**		
commande	**sémantique de la requête**	**réponse éventuelle**
LOGINA	un administrateur autorisé se connecte *paramètres* : nom, password	oui ou non
LCLIENTS	List CLIENTS : liste des agents 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

1.4 Quelques conseils méthodologiques pour le développement de FHMPA

Le client administrateur se connecte sur **le port d'administration** PORT_ADMIN et se fait reconnaître comme administrateur en introduisant un mot de passe propre à l'administration. Pour que les clients connectés soient avisés, *de manière asynchrone*, d'un arrêt temporaire ou d'un shutdown du serveur (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 se connecte sur chaque client en cas de nécessité (le client aura envoyé son port d'écoute lors de sa connexion sur PORT_ADMIN).

2. Serveur_Activites

2.1 Le service activités

Pour rappel, il 'agit d'un serveur multithread Java/Windows-Unix (en modèle pool de threads) qui est essentiellement dévolu à la gestion des activités spécifiques des villages de vacances (principalement l'inscription aux activités qui nécessitent une réservation). Comme

ce genre d'informations concerne directement les voyageurs (et leur facture !), toutes les données nécessaires sont stockées dans la base de données BD_HOTELS.

Il est chargé de satisfaire les requêtes provenant d'une application **Applic_Activites** (Java) utilisée par les responsables seuls habilités au travail de gestion des activités; le serveur attend ce type de requête sur le PORT_ACTIVITES. Il utilise le ***protocole applicatif*** (basé TCP) **FOAMP** (**F**unny and **O**rgiac **A**ctivities **M**anagement **P**rotocol) dont les commandes sont

protocole **FOAMP**		
commande	**sémantique**	**réponse éventuelle**
LOGIN	un gestionnaire d'activités veut se connecter *paramètres* : nom et mot de passe	oui ou non + cause
BACTFUN	Booking Funny Activities : inscription à une activité d'un jour (ski nautique, cours de survie, initiation aux orgies romaines, etc) *paramètres* : type d'activité, date souhaitée, nom du client	oui + date proposée + prix ou non + pourquoi
ACKACTFUN	Acknowledgement Funny Activities : acceptation de la date proposée (si la date ne convient pas ou si le prix est trop élevé, le client ne donne simplement pas suite) *paramètres* : type d'activité, date acceptée, nom du client	oui + n° d'inscription ou non + pourquoi
BTREKFUN	Booking Funny Trek : inscription à une activité d'une certaine durée ("cracra-hontah" (5 jours), "extreme trek" (7 jours) ou "orgiac island" (3 jours)) *paramètres* : type d'activité, session du 1er ou du 15 du mois, nom	oui + n° d'inscription ou non + pourquoi

Le serveur sera conçu selon un **mode générique**, plus ou moins inspiré de celui proposé au cours théorique avec les adaptations nécessaires.

Les travaux de l'évaluation 3 : client-serveur sécurisé en Java et UDP

Nous allons ici nous préoccuper de la sécurisation d'un modèle client-serveur au moyen des outils cryptologiques, ceci pour le serveur **Serveur_Reservations** (client: **Applic_Reservations** et **Applic_Occupations**) ainsi que de l'intégration de l'application installée/application Web. Nous aborderons également l'utilisation de la programmation UDP pour un "chat".

Les technologies utilisées ici sont les sockets TCP en Java, la cryptologie et les librairies cryptographiques, les servlets, les sockets UDP en C et en Java, le paramétrage des sockets.

1. Serveur_Reservations

1.1 Les réservations par agence

Pour rappel, il 'agit d'un serveur multithread Java/Windows-Unix (en modèle pool de threads) qui est essentiellement dévolu à la gestion des chambres des motels et des villages.

Il est tout d'abord chargé de satisfaire les requêtes provenant d'une application (Java) **Applic_Reservations** utilisée par les agences de voyages qui effectuent des réservations pour les motels et les villages (ou les annulent); le serveur attend ce type de requête sur le PORT_VOYAGEURS. Comme il faut éviter que n'importe qui connaisse les noms des clients qui ont réservés (spécialement pour les villages aux activités spéciales) ou, pire, les sabote par des inscriptions fictives (par exemple), il faut envisager que ces communications réseaux soient protégées au moyen des outils de cryptographie (clés publique et privée, message digest, signature digitale). Le serveur utilise le ***protocole applicatif*** (basé TCP) **RMP** (**R**oom **M**anagement **P**rotocol), dont les commandes utilisant des techniques cryptologiques, sont

protocole **RMP**		
commande	**sémantique**	**réponse éventuelle**
LOGIN	un gestionnaire de chambres veut se connecter *paramètres* : nom et **digest "salé"** du mot de passe	oui ou non
BROOM	Booking Room : réservation d'une chambre *paramètres* : catégorie (Motel ou Village), type de chambre (Simple, Double, Familiale), date d'arrivée, nombre de nuitées, nom du client de référence **chiffré symétriquement** + **signature** du responsable	oui + n° de chambre proposée + prix ou non + pourquoi
PROOM	Pay Room : accord et paiement de la réservation *paramètres* : n° de chambre, nom du client de référence et numéro de carte de crédit **chiffrés symétriquement** + **signature** du responsable	
CROOM	Cancel Room : suppression d'une réservation de chambre *paramètres* : n° de chambre, nom du client + **signature** du responsable	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, le tout **chiffré symétriquement**

Les mécanismes de cryptographie **asymétrique** utilisent des clés publiques et privées. On peut, dans un premier temps, se contenter de clés sérialisées dans des fichiers. Mais, dans un deuxième temps, les clés asymétriques doivent se trouver dans des **keystores**.

Dans un premier temps, 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.

Le serveur utilise la base de données BD_HOTELS est une base MySQL. Les numéros de chambre sont supposés uniques dans tout Inpres-Holidays.

1.2 Les réservations par Internet

Le serveur doit aussi satisfaire les requêtes provenant de l'application Web **Web_Applic_Reservations** qui permet au public d'effectuer des réservations pour les motels et les villages par Internet. Plus précisément, l'une des servlets de cette application Web doit donc être capable d'utiliser le protocole RMP et il faut donc implémenter celui-ci en tenant compte du contexte : c'est la servlet qui fait office d'agence de voyage. Ainsi, par exemple, la signature utilisée sera construite au moyen d'une clé privée attribuée à la servlet. A remarquer cependant que la requête LROOMS est refusée pour le ce type de client "grand public".

1.3 La consultation des réservations

Ici, une application **Applic_Occupations** utilisée par les responsables des motels et des villages, leur permet de visualiser les séjours réservés chez eux. Le serveur (qui, pour cette partie, est multithread mais peut être en modèle threads à la demande) attend ce type de requête sur le PORT_VILLAGES-MOTELS. Il utilise le ***protocole applicatif*** (basé TCP) **RLP** (**R**oom **L**isting **P**rotocol), dont les commandes utilisant des techniques cryptologiques, sont

protocole **RLP**		
commande	**sémantique**	**réponse éventuelle**
LOGIN	un gestionnaire de motel/village veut se connecter *paramètres* : nom, identifiant du motel/village et **digest "salé"** du mot de passe	oui ou non
BDROOM	BookeD Room : liste des réservations pour le motel/village pour un jour donné *paramètres* : date demandée + **signature** du responsable	oui + liste des réservations ou non + pourquoi
ARRROOM	ARRival Room : les titulaires d'une réservation sont arrivés *paramètres* : n° de chambre, nom du client de référence + **signature** du responsable	ok ou non – cette réservation n'existe pas
MISROOM	MISsing person for Room : signale que les titulaires d'une réservation ne sont pas arrivés au jour prévu *paramètres* : n° de chambre, nom du client + **signature** du responsable	ok ou non - cette réservation n'existe pas

2. Holidays-Chat

2.1 Fonctionnalités demandées

Les responsables des villages et motels, ainsi que les gestionnaires du matériel, peuvent discuter par chat pour s'informer sur le temps qu'il fait, signaler l'arrivée de jolies filles ou de beaux garçons ou autres informations futiles. C'est le "Holidays-Chat".

On est admis dans ce chat sur base de ses informations d'identification sur le serveur dont on dépend, donc son nom et mot de passe, complété d'une phrase connue de lui seul.

2.2 Client UDP en Java

Au moyen d'une application **Applic_Chat-Holidays** 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_Chat-Holidays** qui vérifie l'appartenance à la Inpres-Holidays dans la base donnée BD_HOTELS contenant les informations nécessaires à l'identification.

Les clients utilisent pour cela le protocole **FECOP** (**FE**rries **C**ommunity c**O**nversation **P**rotocol). Basé principalement UDP, ce ***protocole applicatif*** utilise donc cependant au préalable une connexion TCP à **Serveur_Chat-Holidays** écrit en Java sous Windows; celui-ci n'attend sur le port PORT_GROUP qu'une seule commande permettant de fournir le nom et le mot de passe de l'agent :

protocole **FECOP (partie TCP)**		
commande	**sémantique**	**réponse éventuelle**
LOGIN_GROUP	un agent veut se joindre au groupe *paramètres* : nom, password	oui + envoi de l'adresse et du port PORT_CHAT à utiliser ce jour; ou non

En cas de succès, le client pourra alors réellement participer au chat :

protocole **FECOP (partie UDP)**		
commande	**sémantique**	**réponse éventuelle**
POST_QUESTION	Pose question : un agent pose une question et espère des réponses *paramètres* : la question sous forme de chaîne de caractères (un tag d'identification de question à utiliser dans la réponse est généré)	une réponse à la question précédée du tag
ANSWER_QUESTION	Réponse à une question *paramètres* : le tag d'identification de question et le texte de la réponse	une réponse à la question précédée du tag
POST_EVENT	Un agent 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_Chat, 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.

2.3 Client UDP en C/C++

On élargira ensuite le chat en ajoutant une application **Applic_C-H** écrite en C/C++.

En guise de conclusion

Les principaux développements évoqués dans ce dossier de laboratoire ont été définis avec le plus de précision possible. Le schéma de la page suivante vous en propose un résumé avec le nom des applications, des protocoles et des ports.

Certains points ont cependant été laissés suffisamment vagues pour vous permettre d'envisager différents scénarios (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,MM, JMW

Infos de dernière minute ? Voir l'Ecole virtuelle

Web_Applic_Reservations
Windows / Java
Voyageurs
Applic_Reservations
Windows / Java
Agences
HTTP / TCP
RMP /TCP
SERVEUR_WEB
(Tomcat)
PORT_ VOYAGEURS
Serveur_Reservations
Windows - UNIX / Java
BD_HOTELS
(MySQL)
PORT_VILLAGES-MOTELS
RLP /TCP
Serveur_Activites
Windows - UNIX / Java
PORT_ACTIVITES
Applic_Activites
Windows / Java
Villages
FOAMP TCP
Applic_Occupations
Windows / Java
Motels et villages
Admin_Terminaux
Windows – UNIX / Java
Adminis-trateur
Serveur_Chat-Holidays
Windows - UNIX / Java
FIC_MATERIEL
(csv)
FHMPA / TCP
PORT_ADMIN
PORT_GROUP
FECOP / TCP
PORT_CHAT
FECOP / UDP
Serveur_Villages
Applic_Chat-Holidays
Windows-Unix / Java-C/C++
Agents
PORT_VILLAGES
FHMP / TCP
Applic_Materiel
UNIX / C/C++
Villages
Applic_C-H
UNIX / C/C++
Agents