Archenault Sylvain GM4
Houpert Yves 2006

Rapport de Mini-Projet C++

Nous avons décidé de développer un client de messagerie instantanée. Celui-ci s'intègre dans le projet réalisé pour l'option Java. Le projet Java consiste à développer complètement un système de messagerie instantanée, à savoir un programme serveur et un programme client.

Chaque utilisateur doit par le biais de l'application cliente, pouvoir se connecter au service, effectuer des opérations de base sur la gestion de ses contacts et bien entendu pouvoir discuter avec eux. Il est important de noter que l'on peut discuter qu'avec une seule personne au sein d'une discussion, mais plusieurs discussions sont possibles simultanément. Nous avons donc développé un logiciel client en C++, celui-ci est destiné à une plateforme GNU/Linux et compatible.

Lors de ce projet, nous avons abordé plusieurs notions développées en cours : Héritage (polymorphisme), surcharge d'opérateurs, patrons, libraire Qt, librairie STL, ... ainsi que d'autres notions : programmation réseau (socket), traitement de documents XML.

1. Héritage, surcharge d'opérateurs, exception et patrons.

Ce sont des notions de bases du langage C++.

1.1 Héritage, polymorphisme

Nous avons bien entendu utilisé l'héritage et le polymorphisme afin de rendre la programmation plus facile. Il y a en effet une hiérarchie entre les différents type de messages :

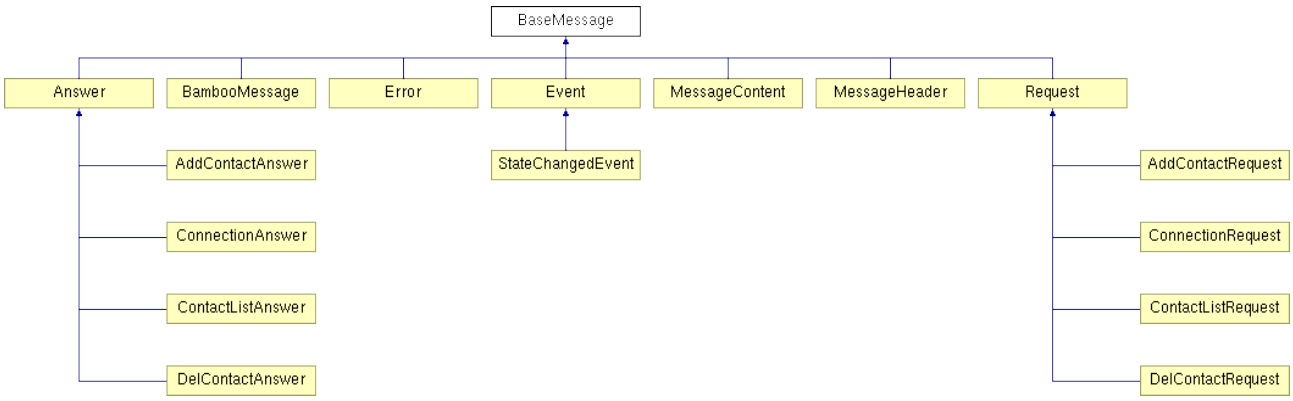

Illustration 1: Diagramme de classe

1.2 Surcharge d'opérateurs

Nous avons également surchargé quelques opérateurs pour faciliter l'utilisation de certaines méthodes :

- ✗ Opérateur « == » (classe contact) : Pour comparer deux objets contacts, nous avons donc surchargé cette opérateur. Deux contacts sont égaux s'ils possèdent le même identifiant, celui-ci est unique.
- ✗ Opérateur « << » (classe Socket) : Permet l'écriture de données dans le socket.

1.3 Constructeur par recopie, opérateur d'affectation, const

Dans le but de développer un code propre, nous nous sommes attachés à écrire les constructeurs par recopie lorsque cela était nécessaire. Cela s'est révélé utile uniquement pour la classe BambooMessage, c'est en effet la seule classe possédant des membres alloués dynamiquement.

D'un autre côté, nous voulions empêcher la copie de certains objets (KernelMessenger, ConnectionWidget, Handler XML), nous avons choisi d'utiliser une technique classique. Celle-ci consiste à déclarer le constructeur par recopie et l'opérateur d'affectation en tant que méthodes privées. De plus, elles ne sont pas implémentées. De cette façon, il est impossible de les utiliser.

Toujours en gardant cette idée de clarté, nous avons essayé d'utiliser le mot clé *const* à chaque fois que cela était possible.

Enfin afin de rendre le programme plus efficace, nous avons en général opté pour un passage des arguments par référence constante plutôt que par valeur. Cette règle s'applique pour les arguments qui ne sont pas des types primaires (int, bool, ...)

1.4 Patrons

Nous n'avons pas créé de patrons car nous n'en avons pas eu l'utilité. Cependant, nous en avons utilisé, essentiellement lors de l'utilisation de la librairie STL. Les patrons se révèlent très puissant, et assez facile à utiliser. Ils permettent de réduire la duplication du code. Cependant, dans ce projet, il n'y a pas de partie se prêtant à cette généricité.

1.5 Exceptions

Afin de gérer les possibles erreurs lors du parsing des documents XML, nous avons utilisé les exceptions. En effet, le parsing peut parfois se révéler complexe, et par conséquent le traitement de erreurs difficiles. Les exceptions facilitent ce traitement. Nous avons également attaché de l'importance à ce que la levée d'exceptions n'entraîne pas un plantage du programme. Il est nécessaire de restaurer le programme dans l'état ou il était avant l'appel à la fonction ayant générée l'exception.

2. Librairie Qt et STL

2.1 Qt

Nous utilisé la librairie Qt pour créer l'interface graphique (IHM). Cette librairie s'est révélée assez facile d'utilisation. Grâce au logiciel *Qt Designer*[1], nous avons pu créer facilement les fenêtres. Afin de développer proprement dit l'application, nous avons utilisé l'environnement de programmation *Kdevelop*[2] qui s'est révélé efficace.

Nous avons également utilisé cette librairie pour la programmation réseau.

2.2 STL

Nous avons utilisé à plusieurs reprises les possibilités offertes par la STL, notamment ses contenaires (vecteur, liste, map).

Nous avons notamment utilisé une « map » pour sauvegarder la liste des discussions en cours. On a associé à l'identifiant du contact, un pointeur vers la fenêtre affichant la discussion. De cette manière, on peut facilement et rapidement accéder à la bonne fenêtre lors de la réception d'un nouveau message. Dans le cas ou l'association n'existe pas, un nouvelle fenêtre de discussion est créée.

Nous avons utilisé une liste pour stocker la liste des contacts de l'utilisateur.

Nous avons également utilisé un objet très pratique fourni par la STL connu sous le noms des « Smart Pointer », plus précisément la classe *std::auto_ptr<T>*[3]. Ces objets permettent de gérer plus facilement les pointeurs et notamment leur destruction.

3. Programmation réseau

Afin de communiquer avec le serveur, nous avons utilisé une connexion socket basée sur le protocole TCP/IP. Nous avons utilisé les classes fournies pas la librairie Qt, notamment la classe QSocket.

4. Traitement de documents XML (Xerces)

Afin de communiquer entre le serveur et le client, nous nous sommes appuyés sur le langage XML. C'est un langage qui est aujourd'hui très utilisé dans les applications web (Web Services), mais également dans les applications « locales » pour stocker des informations notamment sur la configuration du logiciel.

Pour traiter ces documents, nous avons utilisé la librairie Xerces[4] développée par la fondation Apache[5]. Nous avons plus précisément utilisé un parseur SAX qui a pour intérêt d'être rapide et peu gourmand en mémoire.

Dans l'application, « Tout est objet ». Cependant, nous ne pouvons faire transiter par le réseau des objets C++, et encore moins les convertir en objet Java (pour le serveur). Pour

1 http://www.trolltech.com/trolltech/products/qt/features/designer
2 http://www.kdevelop.org/
3 http://gcc.gnu.org/onlinedocs/libstdc++/latest-doxygen/classstd_1_1auto__ptr.html
4 http://xerces.apache.org
5 http://www.apache.org

résoudre ce problème, avant d'envoyer un objet sur le réseau, nous le transformons en XML, il s'agit en fait d'une représentation de l'état de l'objet. Du côté de la réception, nous analysons le document XML reçu et on recrée l'objet en attribuant à ses attributs leur valeurs.

En conclusion, ce projet fût très intéressant à réaliser. Il nous a permis de mettre en application les connaissances vues en cours. De plus, nous nous sommes efforcés de programmer d'une manière stricte (utilisation de *const,* utilisation propre de la mémoire). Nous avons également utilisé au maximum les possibilités de la programmation objet comme le polymorphisme ou l'encapsulation afin de rendre la programmation plus aisée, ainsi que sa maintenance. Nous avons également attaché un effort particulier à la documentation du code, nous avons généré une documentation grâce à l'outil doxygen[6]. Cette documentation, les sources du programme, ainsi que ce rapport sont disponibles à l'adresse suivante : http://gm.insa-rouen.fr/~archenas/projets/messenger/qtclient/.

6 http://www.stack.nl/~dimitri/doxygen/