Le retour du développement natif en C++

Évoquer le développement natif, cela implique qu'une application appelle en direct les primitives (ou API) du système d'exploitation sur lequel il s'exécute. Or, les systèmes d'exploitation sont majoritairement faits en C/C++. Le développement natif se traduit naturellement en développement C/C++ car les primitives des systèmes sont soit des fonctions, des classes ou des interfaces exposées pour être utilisées par un compilateur C++.

C++ est dans l'O.S et pas seulement dans Windows, il est dans le navigateur web et pas seulement dans Internet Explorer. La communauté C++ ne se recrute pas seulement au travers des universités et des entreprises, c'est la plateforme la plus populaire (25%) devant Java (17%), C# (7%), PHP (6%) et Objective-C (6%) – cf. l'indice TIOBE disponible sur http://www.tiobe.com. C++ veut dire développement natif. C++ permet d'accéder à toutes les fonctionnalités sous-jacentes du système d'exploitation. C'est une caractéristique unique.

Ce n'est pas le cas des langages managés comme C# ou Java qui sont dépendants d'un framework et d'un runtime d'exécution. Et lorsqu'une fonctionnalité n'est pas exposée au travers du couple framework/runtime, un langage managé doit faire un appel à des API natives au travers d'un système de traversée de couches (marshalling) souvent exotique à coder (P/Invoke en C#, JNI en Java).

La nouvelle génération d'application appelle le développement natif. La volonté de faire des interfaces utilisateur comme dans le film « Minority Report » appelées aussi « Native UI » en est le meilleur exemple. L'autre argument est la puissance et la performance qu'apporte C++ et il n'y a pas d'équivalent. Un très haut degré d'abstraction mais sans compromis sur la performance.

Aujourd'hui, l'essor des plateformes mobiles et des tablettes et des périphériques portables nécessite l'utilisation d'outils natifs pour travailler avec le socle hardware et le système d'exploitation du périphérique. Les batteries sont limitées et les préserver est une priorité.

Comment cela se traduit-il ? Le programme doit gérer tout de A à Z. Et pour cela, C++ est le bon langage car rien n'est caché, on a accès à tout.

> *« C++ permet d'accéder à toutes les fonctionnalités sous-jacentes du système d'exploitation. »*

> Les évolutions de C++

C++ a évolué au travers de différentes étapes de standardisation, C++98, C++03, TR1, C++0X et maintenant C++11, la dernière version sortie en mars 2011. L'exemple le plus intéressant dans ce nouveau C++ est le rapport à la gestion du cycle de vie des objets. Avant, le code C++ utilisait des appels à new et le code pouvait produire des fuites mémoires difficiles à reproduire et donc à corriger. Avec les templates comme shared_ptr<T> et weak_ptr<T>, il est possible de s'affranchir de certains problèmes liés à la restitution de la mémoire au travers d'un mécanisme de compteurs de références sophistiqué et d'une implémentation qui ressemble au pattern Observateur. Bjarne Stroustrup, inventeur du C++ déclare entre autres, que *« C++ is the best language for garbage collection, principally because it creates less garbage »*. C++ est le meilleur langage pour le garbage collector car il ne crée rien à nettoyer. La notion de GC n'existe pas en C++. Concernant la notion de garbage collector disponible dans les autres langages, cela ne marche pas en C++. C++ supporte la diversité. Il y a plusieurs systèmes de gestion mémoire et pas un moyen unique. Les allocations peuvent se faire en tirant parti soit du langage (new(), delete()), soit de la librairie standard STL (utilisation des templates shared_ptr<T>) ou bien de l'OS (exemple VirtualAlloc() en Win32). Le problème n'est pas d'avoir une application qui libère automatiquement la mémoire mais plutôt d'avoir une application qui démarre vite et qui soit rapide. Dans l'industrie du jeu vidéo, les applications pré-allouent un immense bloc de mémoire, le partitionnent et fournissent plusieurs mécanismes de caches. Dans le milieu de l'embarqué, certains programmes doivent allouer la mémoire au début et doivent être en consommation constante et ne plus allouer de mémoire supplémentaire. Seul C++ peut garantir cela. Les GC sont fantastiques pour les langages artificiels et les langages managés, pas pour C++. En C++, les pointeurs partagés permettent de gérer le cycle de vie des objets efficacement. Dans le monde .NET, il n'existe qu'une seule méthode pour allouer de la mémoire. Pour être rapide et efficace, utiliser un GC n'est pas une bonne option.

Le responsable de Visual C++ (Ale Contenti) préfère cette maxime *« C++ brings Power and Performance without comprise»*. C++ ne sacrifie jamais le pouvoir donné au programmeur en faisant des compromis sur les performances.

> C++ Renaissance

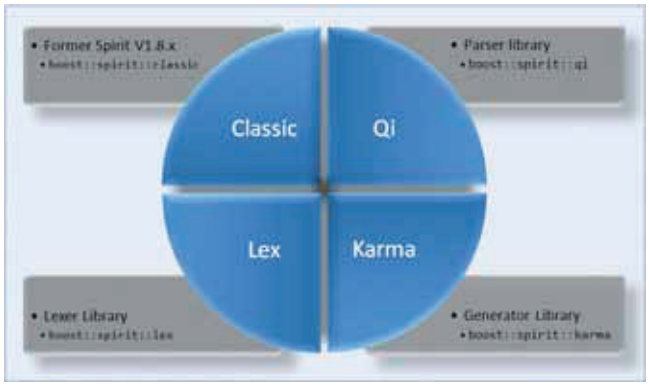
Les conditions sont réunies en 2011 pour que C++ revienne au centre de l'actualité. Le nouveau C++11 vient d'être standardisé. Il y a de plus en plus de personnes qui posent des questions et qui veulent revenir à des fondamentaux comme les applications rapides et fluides. C++ est utilisé dans différentes lignes métier comme les mobiles, les serveurs, les jeux, partout. La communauté n'est pas la plus grande mais elle continue de grossir.

Même chez Microsoft, là ou l'engouement pour le code managé est très grand, tous les produits majeurs sont faits en C/C++ : Windows, Office, SQL Server. C++ est à la fois un langage de très bas niveau mais qui

peut aussi être très abstrait. Microsoft a commencé par faire du C avec des classes et maintenant pratique le C++ moderne. Comme l'électricité, son utilisation est naturelle et il est construit de telle manière que le pouvoir et la performance sont fournis par défaut. Il permet de faire des choses rapides comme en C mais aussi des opérations typées, de la programmation objet puissante. La performance est impressionnante. Il suffit de jeter un coup d'œil au code assembleur généré qui est fantastique. Exemple : les compilateurs et optimiseurs de code C++ utilisent les registres CPU. En C++, la performance n'est pas négociable. De plus, les optimisateurs de code savent parler aux architectures processeurs pour injecter les bonnes instructions et les bonnes extensions CPU. En termes d'avantage marketing, le support aux dernières innovations ou le support des derniers matériels est presque toujours fourni sous forme d'accès natif. C++ permet cela. Concernant la productivité, celle-ci est différente en natif et en code managé. Pour la conception logicielle, ce n'est pas le dernier framework ou la dernière librairie à la mode qui va orienter la conception d'une application. En C++, vous n'êtes pas obligés de faire de l'orienté objet ou des classes. On peut utiliser directement des fonctions. De plus, C++ permet de faire de la programmation à base de meta templates (templates meta programming). La syntaxe est un peu exotique mais c'est toujours du C++, et lui seul permetra cela. La librairie Boost::Spirit est souvent citée en exemple. Faites un tour sur www.boost.org.

> Les nouveaux challenges parallèles

Maintenant, le challenge qui se profile à l'horizon est le passage à la programmation multi-core et l'exploitation pleine et entière des nouvelles architectures processeurs. Dans ce domaine, Intel travaille en assembleur mais son principal support est le langage C++. Dans les prochaines années, nous allons avoir des librairies construites à base de templates avec un très haut degré d'abstraction mais une implémentation qui va directement tirer parti des dernières extensions des processeurs et aussi des cartes graphiques (GPU). Microsoft fournit dans Visual C++ la librairie PPL (via la runtime C) qui est construite à base de classes, d'interfaces et de templates. Intel propose TBB. Dans son prochain Visual Studio,



Microsoft va fournir une librairie AMP pour tirer parti des GPU en utilisant les templates, DirectX et les pixels shaders.

C++ est plus qu'un langage. Il est indissociable des librairies, la runtime C, la runtime C++ nommée STL.

Une librairie peut être générique et portable comme une librairie de calcul. Une interface utilisateur peut être portable. Les développeurs C++ sont assez sophistiqués pour comprendre ce qui peut être portable et ce qui doit être dépendant d'une plateforme spécifique. Le « one size fits all in C++ » n'existe pas. Le langage C++ a tellement de fonctionnalités élégantes comme les templates et des moins élégantes comme les macros. Le langage possède des attributs uniques et C++ est associé à l'excellence dans le domaine du développement logiciel. Si vous êtes un développeur C++, vous êtes forcément un peu plus curieux. C++ est natif et les systèmes d'exploitation aussi, donc il est facile de faire des applications qui exploitent plus et sont plus performantes. C++ est connu comme un langage dangereux et non sécurisé : c'est un mythe et en plus, c'est faux. Il s'agit d'un argument marketing pour vous détourner vers d'autres langages ! Les outils de développement comme Visual Studio vont contenir de plus en plus de librairies et spécificités dédiées uniquement à C++. Rendez-vous avec Visual Studio 2011.

> Pourquoi utiliser C++ ?

Pourquoi utilisons-nous C++ demande Herb Sutter, qui préside le comité de standardisation C++. Il y a 3 éléments de réponse :

- la performance par rapport au coût
- la performance par rapport au nombre de transistors disponibles (CPU, GPU)
- la performance par rapport aux cycles CPU

Dans les années 90, C++ est devenu un langage standard. Dans les années 2000, la productivité est le sujet majeur et ce fût l'âge java, C#, les langages managés. Mais la question est : peut-on tout faire avec un langage managé ? La réponse est non. Lorsque Herb Sutter a rejoint Microsoft, le futur système d'exploitation, nom de code Longhorn étant en phase de développement, les nouvelles API seraient managées. C'était l'idée.

Microsoft a essayé d'introduire du code managé dans le système d'exploitation et ce fût un challenge intéressant. Mais le résultat n'était pas bon et loin d'être parfait, que ce soit sur la robustesse des interfaces et du code développé, les performances et le mode de déploiement. C'est la raison pour laquelle Windows est toujours en C et C++. C# est fait pour la productivité. Depuis la dernière décennie, les développeurs se sont tournés vers les langages managés. Par contre, les utilisateurs demandent des interfaces fluides, des tablettes, des périphériques portables évolués, des caméras temps réel, des fonctionnalités d'OCR temps réel et des expériences utilisateurs qui répondent vite. Aujourd'hui, seul le langage C++ permet de répondre à cela. Tout est histoire de performance. Les langages qui sont construits autour de l'optimisation de code auront une nouvelle vie, disait Herb Sutter en 2004. Les conditions sont aujourd'hui réunies en 2011.

Christophe **Pichaud**
.NET Rangers by Sogeti
Consultant sur les technologies Microsoft
christophepichaud@hotmail.com
www.windowscpp.net

Vidéos Channel9 :

http://channel9.msdn.com/Shows/C9-GoingNative/GoingNative-0-Help-us-fly-this-plane-Some-modern-C-Meet-Ale-Contenti
http://channel9.msdn.com/Shows/Going+Deep/Craig-Symonds-and-Mohsen-Agsen-C-Renaissance
http://channel9.msdn.com/posts/C-and-Beyond-2011-Herb-Sutter-Why-C
http://channel9.msdn.com/Shows/Going+Deep/Mohsen-Agsen-C-Today-and-Tomorrow
http://channel9.msdn.com/tags/STL/