Le futur du compilateur Delphi

par Nick Hodges (Blog de Nick Hodges) Eric GASPARD (traduction)

Date de publication : 16 Janvier 2009

Dernière mise à jour :

Nick Hodges, responsable des produits Delphi et C++ Builder chez CodeGear, partage avec nous les dernières nouvelles du front concernant la direction prise pour l'avenir des compilateurs Delphi et C++ Builder. Traduction de l'article en anglais que vous pouvez trouver sur le **site de CodeGear**.

I - Le futur du compilateur Delphi

I-A - Introduction

Cet article a pour but de donner à tous les utilisateurs de Delphi quelques aperçus de ce sur quoi nous sommes en train de travailler et imaginer pour le compilateur Delphi. Nous savons tous au QG d'Embarcadero World-Wide combien vous aimez votre compilateur Delphi, et comme il existe quelques technologies sympa pour le langage et le compilateur qui mijotent dans les labos de R&D, nous avons voulu être sûr que vous étiez à jour concernant le futur de votre compilateur favori.

I-B - Un peu d'histoire

Mais tout d'abord, laissez-moi vous donner une petite leçon d'histoire sur le compilateur. La plupart de ces choses ne seront pas nouvelles pour vous, mais c'est important de les rappeler. Premièrement, le compilateur Delphi existe depuis fort longtemps. Il a prouvé de nombreuses fois qu'il était un outil rapide et moderne, et vous a fournit de nombreuses fonctionnalités modernes et sympathiques comme les génériques, les méthodes anonymes, les procédures inline et bien d'autres. Toutefois, à cause de ce riche héritage, il a réellement encore du code à l'intérieur datant du temps du Turbo Pascal, et ces temps furent, ma foi, il y a bien longtemps dans une galaxie très très lointaine. Il a toujours des fonctionnalités du langage et constructions de code qui trainent et dont beaucoup d'entre vous ne connaissent même pas leur existence. Par exemple, combien parmi vous savent que vous pouvez créer des commentaires avec (*...*) ? Ou encore combien d'entre vous savent que ceci compile toujours :

```
var
   TableauBizarre: array(.1..10.) of string;
```

Allons allons, soyez honnêtes. Peu d'entre vous connaissaient ceci.

Bien sûr, le compilateur maintient ces constructions, et bien d'autres encore, pour compatibilité ascendante. La compatibilité ascendante est vraiment, vraiment importante pour vous et vraiment, vraiment importante pour nous. Laissez-moi répéter cela : *La compatibilité ascendante est vraiment, vraiment importante pour vous et vraiment, vraiment importante pour nous.* (Espérons que l'italique permet d'insister sur ce point. OK, maintenant nous pouvons continuer). Néanmoins, la plupart de ces fonctionnalités ne sont plus utilisées par la majorité d'entre vous. Quoi qu'il en soit, nous continuons de les maintenir, de les tester et de les valider à chaque sortie du compilateur. Cela représente beaucoup de travail. Mais à partir du moment où vous n'utilisez probablement pas toutes ces choses (sérieusement, n'essayez même pas de me dire que vous connaissiez le « parenthèse-point »...), il en résulte que cela ne vous importe pas plus que ça. Assurer le support de ces fonctionnalités peu usitées coûte du temps que nous pourrions utiliser à travailler sur des nouvelles possibilités.

Les exemples précédents ne sont pas plus que ça, des exemples. Mais beaucoup de ces sémantiques de code sont des fonctionnalités qui peuvent réellement nous empêcher de réaliser des changements et de moderniser le langage. Laissez-moi être clair avant de continuer plus avant : je ne suis pas en train de parler d'éliminer des fonctionnalités; je discute simplement d'un problème que le langage a actuellement. Nous n'avons aucune intention de supprimer des fonctionnalités du langage ou briser votre code. Une nouvelle fois : nous n'avons aucune intention de faire cela. Nous avons toutefois de nouveaux plans intéressants.

I-C - Pour cette raison...

Pour cette raison, nous avons longuement et durement réfléchi à la question. En particulier : comment faire évoluer notre compilateur pour qu'il puisse continuer à faire toutes ces choses que les compilateurs modernes et puissants font ? Comment évoluer notre compilateur en lui ajoutant plein de trucs cool tout en préservant votre code existant ?

Et bien, la réponse à cette question est loin d'être courte ou rapide. Ce n'est pas quelque chose que nous prenons à la légère. Nous pourrions juste réécrire complètement un compilateur des pieds à la tête et ainsi créer un nouveau

langage Delphi qui fait ce que voudrions et qui laisserait sur le carreau nombre de code en votre possession. Mais cela serait mauvais. Vraiment très mauvais. Nous n'avons véritablement aucune envie de faire cela. Laissez-moi le répéter : nous n'avons aucuns plans, aucunes intentions et aucunes raisons de faire en sorte que votre code existant ne fonctionne plus.

Ce que nous voulons faire, néanmoins, c'est être capable de continuer à faire évoluer Delphi dans le but de le garder à la pointe du développement natif (et du développement en général). Mais parfois, évoluer devient difficile à cause de ces choses du passé. Delphi est un langage très ***type-safe*** (avec une grande sécurité de type) mais pas totalement *type-safe*. Vous pouvez toujours faire des trucs genre appeler GetMem ou créer un tableau ouvert (array[0..0] of integer) et faire toutes sortes de choses dingues avec en manipulant la mémoire librement par ces biais. C'est une chose vraiment puissante de pouvoir être capable de le faire, mais cela implique que vous êtes beaucoup moins descriptifs dans ce que vous faites, de fait le compilateur vous laisse aller où bon vous semble. Dans la plupart des cas, cela fera exactement ce que vous vouliez mais c'est le premier exemple des ouvertures qui permettent de vous tirer une balle dans le pied. Delphi vous laisses rouler sans ceinture de sécurité, mais le fait que vous puissiez faire cela implique qu'il y a des choses que Delphi lui-même ne peut pas faire. Ne serait-ce pas merveilleux si vous pouviez conduire votre rutilante voiture mais avec un solide harnais trois points et pleins d'airbags. Du coup, que fait un compilateur alors ?

I-D - La Partie Avant du compilateur

Les compilateurs ont en général ce qui est couramment appelé « Partie Avant ». Cette Partie Avant prend le code, l'analyse et créé un arbre qui représente le but sémantiques du programme. C'est la partie qui décode la syntaxe du langage et au final détermine ce qu'est le langage. Dans Delphi, la Partie Avant sait comment lire les unités, les boucles for, les paires de begin...end, les déclarations de variables, les classes génériques, tout les « Trucs Delphi ». C'est également la Partie Avant qui crée les fichiers DCU qui sont actuellement spécifiques à un compilateur. C'est également le côté qui se charge de toute la partie syntaxe historique vue précédemment. Donc la question devient : comment modifier la Partie Avant de Delphi pour lui permettre de supporter toutes les nouvelles et modernes fonctionnalités du langage tout en continuant de garantir que votre précieux code continue de tourner ?

Et si nous créions une nouvelle Partie Avant au compilateur qui vous permettrait de choisir entre une nouvelle syntaxe, sans compatibilité ascendante, et l'autre façon de coder ? (Laissez-moi être clair à nouveau : nous ne parlons pas de supprimer des fonctionnalités. Si vous vous ruez sur les commentaires pour demander « Alors, quelles constructions de code vont devenir dépréciées ? » je peux déjà vous répondre par : « Aucune, tout votre code continuera de tourner. ». Je penses que nous avons été assez clair sur le sujet maintenant, n'est-ce pas ?) Ou plus exactement, pourquoi pas une nouvelle façon de voir le code existant et le « nouveau » code de sorte qu'ils puissent fonctionner ensemble. Vous permettant de mettre du nouveau code dans de nouveaux modules tandis que votre ancien code continue de fonctionner comme il l'a fait jusqu'à présent ? Il sera toujours possible d'injecter le code existant dans la nouvelle façon de faire les choses, mais si vous voulez utiliser les choses « nouvelles et améliorées », vous deviez les faire dans un nouveau type de module de code ? Et maintenant pourquoi pas, tant que nous y sommes, finir par créer une DCU (ou quelque chose s'approchant de ce que sont les DCU aujourd'hui) dont le format ne serait pas spécifique à une version du compilateur ? Comment cela sonne-t-il à vos oreilles ?

Cela sonne vraiment sympa pour moi mais, bien sûr, faire quelque chose comme ça est loin d'être trivial et requerra beaucoup de temps et d'effort. Mais quelque chose comme ça en vaut la peine, non ?

Mais la Partie Avant n'est que la moitié de la bataille. Il a aussi la Partie Arrière du compilateur.

I-E - La Partie Arrière du compilateur

Pour ceux qui ne sont pas des gourous du compilateur, la « Partie Arrière » d'un compilateur est la partie qui lit ce que la Partie Avant produit et écrit le véritable code machine produisant des binaires utilisables ou bien quelque chose qui est utile à un autre type d'exécution ailleurs. La Partie Arrière de nos compilateurs Delphi et C++ produisent des binaires 32-bit pour Windows. Mais bien sûr, le Parties Arrières n'ont plus lieu à se limiter à faire du 32-bit ou du Windows de nos jours, n'est-ce pas.

L'une des nouvelles choses que nous voudrions que notre Partie Arrière produise des binaires 64-bit. Par conséquent, produire des binaires 64-bit va exiger une nouvelle Partie Arrière. Nous pourrions juste prendre notre Partie Arrière 32-bit existante et bidouiller pour qu'elle devienne 64-bit et vous la balancer tel quel. Mais ce ne serait pas la bonne façon de procéder, notamment en prenant en compte le fait que nos compilateurs C++ et Delphi ont deux Parties Arrières différentes. Pourquoi faire le travail deux fois alors que nous pourrions le faire qu'une seule fois ? Et bien sûr, un projet 64-bit couvre bien plus que le compilateur. Nous voulons être sûr que le débogueur, l'éditeur, le système de fichiers, la refactorisation, la modélisation, la documentation tout soit fait correctement et complètement capable de vous permettre de développer des applications 64-bit sans aucune limitation. Cela demandera un peu de temps.

À la place, la meilleure façon de le faire est d'écrire une nouvelle Partie Arrière. Et si vous voulez écrire une nouvelle Partie Arrière, la bonne façon de faire c'est de faire une unique Partie Arrière qui puisse être utilisé à la fois par Delphi et C++ Builder. Et si vous voulez faire comme ça, vous voudriez une Partie Arrière qui soit architecturée de telle sorte qu'elle soit assez souple vis-à-vis du respect de l'architecture qu'elle vise en réalité. Intéressant tout cela, mmh ?

Donc cela devient logique de créer une toute nouvelle Partie Arrière, unifiée, qui fonctionnera tant pour Delphi que pour C++Builder. Mais faire cela prend du temps. Bien plus de temps que nous voudrions, honnêtement, mais ces choses ne peuvent être bâclées. Le résultat, en le faisant correctement, le faire fonctionner comme il doit fonctionner, et créer une véritable architecture de compilateur dont bénéficieront Delphi et C++ Builder, ne sera pas un effort trivial. Le faire correctement est important pour nous, et je suspecte que c'est aussi important pour vous.

Chose intéressante, le temps nécessaire à créer cette nouvelle Partie Arrière et faire un produit 64-bit, crée une opportunité pour nous de créer la nouvelle Partie Avant de Delphi. Nous avons donc pris la décision de prendre ce temps de faire tous les travaux nécessaires à l'arrière, ce qui nous permet de réaliser toutes ces choses intéressantes à l'avant. Nous n'allons pas bidouiller quelque chose et vous le cracher tel quel. Nous voulons créer un nouveau compilateur qui sera vraiment cool. Nous supposons que vous également.

Le compilateur Delphi est sur un point d'inflexion. Et il semble que ce point d'inflexion nous (et, à la fin, vous) donne l'opportunité de faire en sorte que ce point d'inflexion aura une réelle signification.

I-F - Résultat des courses

Maintenant, quel est le résultat des courses ? Et bien, pour en venir au fait (enfin !) avec des travaux majeurs réalisés sur la Partie Avant et le compilateur 64-bit, il semble que ce soit le bon moment pour faire « un pas en avant » et faire la transition vers cette nouvelle architecture de compilateur. Nous espérons que la version 64-bit de Delphi sera prêtre mi-2010. Du fait que le compilateur est la première étape vers un produit complètement 64-bit, nous avons prévu de sortir un avant-goût du compilateur 64-bit mi-2009, comme cela ceux qui sont extrêmement impatient de voir un Delphi 64-bit pourront commencer en avance.

Nous sommes conscients qu'il s'agit d'un changement, mais nous croyons et nous espérons que vous êtes d'accord que cela vaut le coup de prendre le temps qu'il faut pour faire ça bien et faire en sorte que les changements les plus cool permettront de garder Delphi à la pointe de la technologie des langages. Et vous aimez les choses nouvelles et cool n'est-ce pas ?

I-G - Conclusion

Delphi doit aller de l'avant. C++ Builder doit aller de l'avant. Notre compilateur doit aller de l'avant. Pour aller de l'avant, les compilateurs doivent intégrer les nouvelles fonctionnalités que les développeurs modernes demandent. Tous les langages doivent fournir les solutions que vous recherchez maintenant et dans le futur. Afin de mettre en place ces nouvelles fonctionnalités, nous allons devoir casser quelques ufs pour pour faire une omelette, comme dit le dicton.

Mais vous autres avez des myriades de lignes de code que vous voulez continuer à voir fonctionner. Nous avons très bien compris cela. Nous sommes donc en train de créer un « nouveau Delphi » et une nouvelle architecture de compilateur pour faire que votre code continue à tourner, pour générer des binaires 64-bit en utilisant tant Delphi que C++ Builder, et peut-être d'autres types de binaires tant que nous y sommes. Et tout doit être fait dans les règles de l'art pour que cela fonctionne correctement pour vous.

Et vous savez quoi ? C'est exactement ce que nous prévoyons de faire.