Institut Supérieur d'Informatique de
Modélisation et de leurs Applications

Campus des Cézeaux
24 avenue des Landais
BP 10125
63173 AUBIÈRE Cedex

Rapport d'Ingénierie Dirigée par les Modèles
Filière Génie Logiciel et Systèmes Informatiques

Conception et Implémentation du Modèle Objet : 1ére partie

Présenté par : **Nicolas PRUGNE & Antoine COLMARD**

Table des matières

Table des figures et illustrations

Introduction

L'ingénierie des modèles (IDM) est une discipline relativement récente en informatique et qui trouve sa source dans un problème récurrent du génie logiciel. En effet, bien souvent le code produit par les ingénieurs en charge d'un programme informatique satisfait pleinement les spécifications de ce dernier. Cependant si ces spécifications viennent à évoluer, comme c'est le cas lors de mises à jour ou lors de sorties de nouvelles versions d'un produit par exemple, il est fort probable que ce code soit assez difficile à faire évoluer. Ce manque d'évolutivité peut alors conduire les personnes responsables du développement à repasser par une phase de conception très couteuse pour l'entreprise qui finance le produit.

En réponse à ce problème l'Object Management Group (OMG) a décidé d'introduire, à la fin du XXème siècle, une technique de conception baptisée Model Driven Architecture (MDA) qui n'est rien d'autre qu'une variante particulière de l'IDM. Cette discipline propose de recentrer la production de logiciels autour de la conception de modèles.

Dans un processus de développement classique, les ingénieurs élaborent des modèles à partir des spécifications du soft qu'ils doivent produire. Ensuite, il leur est éventuellement possible de générer le squelette du code correspondant grâce à des outils de génie logiciel. De là, s'ensuit alors une phase d'écriture et de maintenance du code avec tous les problèmes que cela comporte, notamment lorsque la modélisation du problème devient obsolète.

Dans un processus de développement dirigé par les modèles, l'objectif est d'arriver à produire des outils qui vont permettre aux ingénieurs de se concentrer uniquement sur la conception et sur la maintenance de modèles. A partir des modèles conçus, ces outils vont ensuite être capables de générer automatiquement le code correspondant pour aboutir à un programme fonctionnel. L'objectif étant, d'une part de faciliter le travail des développeurs et d'autre part de réduire fortement les coûts liés à l'étape de modélisation.

Pour comprendre où se situe le lien entre ce TP et l'IDM, il faut voir les concepts du modèle objet comme les spécifications d'un projet à réaliser. D'un point de vue théorique, l'objectif est alors de développer un outil capable de transcrire un modèle objet en programme exécutable. La première chose qu'il est alors nécessaire de définir, pour l'utilisateur, est un moyen de décrire son modèle à l'outil, afin que ce dernier soit en mesure de le transformer en exécutable.

Dans ce TP il a été choisi que le moyen pour décrire un modèle objet serait le langage C++. Ainsi, l'outil qu'il faut développer est un compilateur. Reste à savoir comment faire pour passer du C++ à un programme exécutable. Pour répondre à ce problème, il a été choisi que le langage C servirait de passerelle entre un programme écrit en C++ et un exécutable. Ainsi, le compilateur qu'il faut réaliser devra produire en sortie un code C qui pourra ensuite être compilé en exécutable. Cette technique est inspirée des premiers compilateurs C++ qui étaient des « *cfront* » et qui généraient du C.

Cependant la création d'un compilateur de A à Z nécessite un travail poussé et ne peut se faire sur quelques séances de TP. Pour rappel la compilation d'un programme fait intervenir plusieurs composants en amont du processus qui va réellement générer le code exécutable. Les trois principaux sont l'analyseur lexical, l'analyseur syntaxique et l'analyseur sémantique (Figure 1).

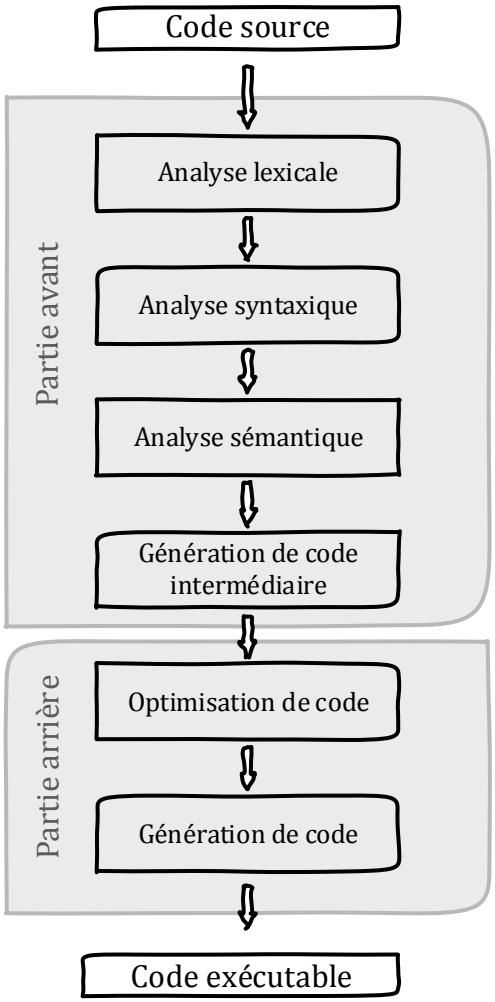

Figure 1 - Schéma d'une chaîne de compilation classique

L'analyse lexicale est un processus qui permet de découper le code source en unités atomiques appelées lexème, il peut s'agir de mots clefs du langage, d'identifiants ou de symboles. Cet ensemble de lexèmes va ensuite être traité par l'analyseur syntaxique (ou *parser* en anglais) qui va vérifier que cette suite de jetons est conforme à la grammaire du langage considéré, ici le C++. Si tel est le cas, c'est ensuite l'analyseur sémantique qui prendra la relève et qui aura pour rôle de vérifier que le code en cours de compilation ait une certaine logique, un certain sens. Il sera question, par exemple, de vérifier si les noms des fonctions utilisés dans le code existent bien, ou encore, de contrôler si deux variables impliquées dans une affectation ont des types cohérents.

Bien que ces trois étapes fassent partie du schéma de compilation classique, elles ne sont pas directement liées à de l'IDM, c'est pourquoi, elles ne sont abordées que brièvement dans ce rapport. En effet, ce dernier traite surtout de l'étape de génération de code intermédiaire. Il s'agit d'une phase pendant laquelle le compilateur va retranscrire le code source vers un autre langage, appelé langage intermédiaire, plus adapté pour d'éventuels optimisations par exemple. De nombreux langages de programmation utilisent le C comme langage intermédiaire et c'est le cas du compilateur dont il est question dans ce TP.

Cependant, l'objectif ici est non pas de créer un outil capable de transformer du C++ en C, mais plutôt de voir comment ce code C pourrait être structuré si réellement il fallait réécrire un compilateur « *cfront* ». Effectivement, bien que le C offre un niveau d'abstraction plus

élevé que des langages comme l'assembleur par exemple, il n'en reste pas moins dépourvu des concepts objets. Il faut donc trouver un moyen d'organiser ce code C afin qu'il puisse émuler les mécanismes d'un langage objet comme le C++. C'est précisément de cette organisation dont il est question dans ce rapport.

Ainsi, le travail réalisé a plutôt consisté, par une approche fonctionnelle, à implémenter un modèle objet en C. Cette tâche a donc nécessité d'étudier et de modéliser les principaux concepts objets pour pouvoir les retranscrire en C. Si l'on prend un peu de recul sur ce travail, on remarque qu'il consiste à concevoir un modèle qui décrit les concepts du modèle objet, on parle alors de méta-modèle. Les méta-modèles sont un des outils clefs de l'IDM puisqu'ils permettent de passer d'une modélisation à l'autre facilement. Ici la transformation serait celle d'un code C++ décrivant un modèle objet que l'on changerait en un code C retranscrivant exactement les mêmes fonctionnalités que le code d'origine. Tout le problème réside dans la conception du méta-modèle qui permet de passer d'une représentation C++ à son équivalent C.

1 L’encapsulation

2 L’héritage

3 Le polymorphisme

Conclusion

Références webographiques