Proposta de implementação de método para tradução/conversão de linguagens de alto nível

Eliakim Zacarias[1], Maurício Sulzbach[1]

[1]Departamento de Engenharias e Ciência da Computação – Universidade Regional Integrada do Alto Uruguai e das Missões (URI)
Caixa Postal: 184 – 98.400-000 – Frederico Westphalen – RS – Brasil

Abstract. *This article describes the process of translation between programming languages, principally between high level languages that, if automatized, can assist in the translation of large projects and allow a globalization and great compatibility between the leading available programming languages.*

Resumo. *Este artigo descreve o processo de tradução entre linguagens de programação, principalmente entre as linguagens de alto nível que, se automatizado, pode auxiliar na tradução de grandes projetos e permitir uma globalização e grande compatibilidade entre as principais linguagens de programação existentes.*

1. Introdução

A tradução de linguagens de programação é um assunto bastante complexo que sempre têm despertado interesse na área da programação, principalmente quando se faz necessária a migração de um sistema completo para outra linguagem. O termo "tradução" define a ação de interpretar dados presentes em uma linguagem de programação de origem (um código-fonte) e transformá-lo em outro código, baseado em uma linguagem de destino (um código-alvo), muitas vezes, em código-objeto (MUCHNICK, 1997).

É bastante comum encontrar tradução de linguagens de alto nível para linguagens de mais baixo nível. Este é o processo padrão conhecido como "compilação", aplicado por todos os compiladores, conforme explica Louden (2004):

> Compiladores são programas de computador que traduzem de uma linguagem para outra. Um compilador recebe como entrada um programa escrito na linguagem-fonte e produz um programa equivalente na linguagem-alvo. Geralmente, a linguagem-fonte é uma linguagem de alto nível, como C ou C++, e a linguagem-alvo é um código-objeto (algumas vezes também chamado de código de máquina [...]).
> (LOUDEN, 2004)

Em contrapartida, a tradução de uma linguagem de alto nível para outra linguagem de alto nível é bem menos comum e muito mais complexa. Geralmente os desenvolvedores são encorajados a fazer o processo de forma manual (SALIH, TONCHEV, 2008, p. 7).

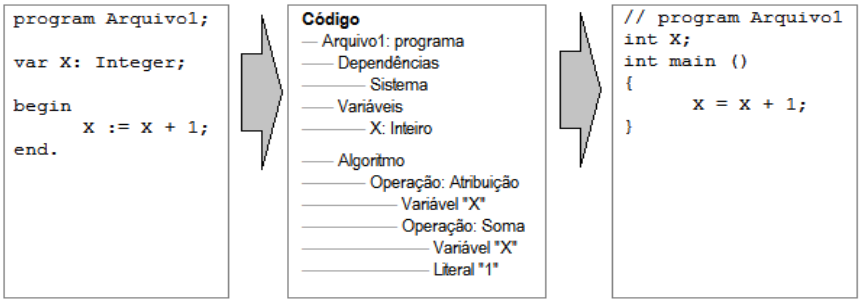
Atualmente, existem aplicativos capazes de traduzir entre determinadas linguagens. O tipo mais comum de aplicativos capazes de automatizar o processo de tradução é baseado em dicionários, pegando pequenos trechos de código e substituindo-os pelos códigos correspondentes. Como exemplo, pode-se citar o aplicativo *OpenC2Pas*, capaz de traduzir de C para Pascal (*SourceForge*, 2009). Existe também o projeto PtoC, o qual

traduz de Pascal para C e C++. Os resultados aparentam ser satisfatórios, mas a tradução limita-se a estas linguagens. (KNIZHNIK, 2008).
A principal vantagem da tradução de linguagem de alto nível para outras linguagens de alto nível está na possibilidade de traduzir grandes bibliotecas, além de permitir escolher uma linguagem de programação para trabalhar, apenas adaptando-se às características de *frameworks* e bibliotecas presentes na linguagem de destino.
Este artigo busca analisar um método de tradução baseado em uma linguagem capaz de definir um compilador e os processos necessários para automatizar a tradução.

2. Desmontando e remontando o código: o princípio da tradução

Um processo de tradução eficiente consiste em desmontar o código escrito em uma linguagem de origem e armazená-lo em uma estrutura interna para, em seguida, reescrevê-lo na linguagem de destino. Para tanto, é necessária uma estrutura do tipo árvore que armazenará todas as informações dispostas no código de origem, com a finalidade de replicá-las utilizando a gramática de destino.
O processo de desmontagem de código requer, por parte do aplicativo responsável pelo processo, a interpretação completa de todo o código presente nos arquivos, observando as regras da linguagem de origem. Nesta fase são apontados todos os possíveis problemas de compilação. Em caso de erro, deve-se procurar manualmente uma solução para o mesmo. Caso contrário, a tradução não poderá ser efetuada. Durante a interpretação, o código deverá ser armazenado em uma estrutura do tipo árvore, respeitando os níveis, conforme o exemplificado na figura 1.

Figura 1. Fluxo da tradução



Conforme a figura 1 exemplifica, recebe-se o código na linguagem de origem (neste caso, Pascal) para em seguida desmontá-lo em uma estrutura mais genérica. Ainda no exemplo da figura 1, a estrutura armazena um único arquivo, nomeado “Arquivo1”, o qual possui apenas uma dependência que se refere ao sistema, isto é, o próprio compilador. O mesmo arquivo possui uma variável declarada com escopo global para todo o arquivo, nomeada X e do tipo inteiro. Um algoritmo principal que será chamado ao executar o arquivo. Este algoritmo inicia com uma atribuição, onde se atribui à variável X o valor resultante da soma entre X e o valor literal “1”.
A partir da árvore, o código pode ser diretamente interpretado, caso exista interesse nesta ação, ou pode ser otimizado e traduzido para outra linguagem com funcionalidades semelhantes. Observa-se na terceira parte da figura 1, o mesmo código escrito na linguagem de destino (neste caso, C).

Para auxiliar na tradução, deve ser criada uma linguagem cuja gramática e recursos permitam personalizar a tradução. Para tornar esta linguagem eficiente, deve-se iniciar com a análise sintática, chamando o analisador léxico sempre que necessário avançar pelo código. A linguagem ainda deve considerar alguns fatores, como a análise léxica de valores literais, palavras-chave e identificadores, além de considerar a estrutura das declarações e o fluxo dos algoritmos e operações.

3. Definindo as linguagens de origem

É função dos arquivos da linguagem de origem interpretar todo o código e, com ele, criar uma árvore intermediária. A árvore intermediária é a estrutura que armazenará todo o programa dividido em nodos. É importante que a árvore possua um padrão único que possa abranger as diversas linguagens de programação, a fim de guardar em seus nodos todas as declarações, algoritmos e operações.

A definição da linguagem de origem deve possuir um conjunto de formatos de arquivos, os quais compõe a análise léxica, sintática e semântica. Para criar um novo interpretador, deverão ser criados primeiramente todos os algoritmos de análise léxica que identificarão, e retornarão sempre que requerido, as palavras-chaves, símbolos e valores literais presentes na linguagem de origem. Logo após, devem ser definidos os analisadores sintáticos para todos os formatos presentes na linguagem (ex.: C++ possui cabeçalhos e códigos-fonte). No final, através de outro formato de definição da linguagem de origem, deve ser definido o analisador semântico, que identificará possíveis problemas na estrutura do aplicativo.

É importante que a linguagem de origem monte a árvore intermediária seguindo um padrão pré-determinado para todas as linguagens, de forma que a tradução não precise se preocupar com diferenças na árvore. O padrão da árvore intermediária deve incluir a possibilidade de definir recursos presentes nas mais novas linguagens (ex.: definição de classes para Orientação a Objetos). Este padrão deverá ser atualizado sempre que um novo recurso aparecer, mas é importante sempre manter a compatibilidade com as versões anteriores, de forma que não seja necessário revisar as linguagens de origem já definidas.

Ao definir os recursos e montar a árvore intermediária, é necessário concluir a tradução. Para tanto, utiliza-se as definições de uma linguagem de destino.

4. Realizando a tradução: a linguagem de destino

A definição da linguagem de destino, tal como a definição da linguagem de origem, deve ser feita através de uma linguagem de programação especial para o aplicativo tradutor. A definição pode incluir um ou mais arquivos, cujo principal objetivo é, através dos dados padronizados na árvore intermediária, criar um código-fonte legível na linguagem de destino. A tradução não deve considerar itens fora dos padrões da árvore intermediária, de forma que a tradução possa ser executada a partir de qualquer linguagem de origem. No entanto, pode existir casos em que é impossível ou inviável traduzir o código disposto na árvore intermediária.

Isto pode ocorrer, por exemplo, em função das diferenças entre linguagens tipadas e não-tipadas, bem como, devido às diferenças entre linguagens orientadas a eventos e linguagens orientadas a aspectos. Esta viabilidade deve ser observada durante a definição da linguagem de destino. Além disso, a tradução não visa alterar a estrutura, orientação ou lógica do código, mas apenas traduzi-lo. No entanto, em casos mais simples, pode-se definir, por exemplo, um padrão de como proceder para traduzir um código orientado a objetos para uma linguagem que não dê suporte a tal recurso.

Também se podem definir normas a fim de padronizar a tradução a partir de linguagens de alto nível para as linguagens de mais baixo nível, além de definir normas que padronizem o procedimento da tradução quando exposta a linguagens de funcionalidades muito diferentes, tal como o caso citado sobre as diferenças de linguagens fortemente tipadas e não-tipadas.

5. Resultados esperados

Este projeto ainda está em fase inicial, de forma que o objetivo do mesmo é apenas apresentar a teoria geral sobre a possibilidade de tradução entre linguagens, com a finalidade de "generalizar" as linguagens de programação de forma a torna-las compreensíveis ao maior número de compiladores possível, além de gerar uma tradução amigável e auxiliar em processos de migração de sistemas, a partir de linguagens descontinuadas ou que não atendem ao desejado, para linguagens que possuam novos recursos.

Em uma segunda etapa, serão definidos os padrões das linguagens que são capazes de satisfazer a criação de um interpretador para as linguagens de origem, bem como gerar uma linguagem tradutora para as linguagens de destino. Poderão ser definidas normas de compatibilidade, a fim de evitar a criação de padrões próprios para cada linguagem, gerando incompatibilidades graves entre as linguagens de origem e destino. Ainda em uma terceira etapa, poderá ser implementada a aplicação.

Referências

KNIZHNIK, K. (2008) "Pascal to C/C++ Converter". Disponível em: <http://www.garret.ru/ptoc/Readme.htm>. Acesso em: setembro, 2012.

LOUDEN, K. C. (2004) "Compiladores: princípios e práticas". Cengage Learning Editores.

MUCHNICK, S. S. (1997) "Advanced Compiler Design and Implementation". Academic Press.

SALIH, M., TONCHEV, O. (2008) "High-level programming languages translator". Blekinge Institute of Technology.

SourceForge. (2009) "OpenC2Pas". Disponível em: <http://sourceforge.net/projects/c2pas/>. Acesso em: setembro, 2012.