Trabalho de Conclusão de Curso

Implementação de um compilador para uma linguagem de programação com geração de código Microsoft .NET Intermediate Language

Giancarlo Tomazelli
Orientador: José Roque V. da Silva

Roteiro

- **Introdução**
- **Objetivos do Trabalho**
- **Linguagens de Programação**
- **Compiladores**
- **Máquinas Virtuais**
- **Plataforma Microsoft .NET**
- **Especificação da Linguagem Proposta**
- **Especificação do Compilador**
- **Implementação do Compilador**
- **Ambiente de Desenvolvimento**
- **Conclusões, Limitações e Extensões**

Introdução

- Em **Julho/2000**, a Microsoft lançou a plataforma Microsoft .NET.
- Esta plataforma é baseada em:
    - Uma nova infra-estrutura de *software* (.NET *Framework*);
    - Um novo ambiente de desenvolvimento (Visual Studio .NET);
    - Diversas linguagens de programação.
- O trabalho apresenta a definição de uma linguagem de programação simplificada e em português, gerando código para ser executado na plataforma Microsoft .NET.

Objetivos do Trabalho

- Criar uma nova linguagem de programação.
- Construir um compilador para a nova linguagem, gerando código Microsoft *Intermediate Language* (MSIL).
- Construir um ambiente de desenvolvimento para a linguagem proposta.
- Executar os programas gerados pelo compilador na plataforma Microsoft .NET.

Linguagens de Programação

Compiladores

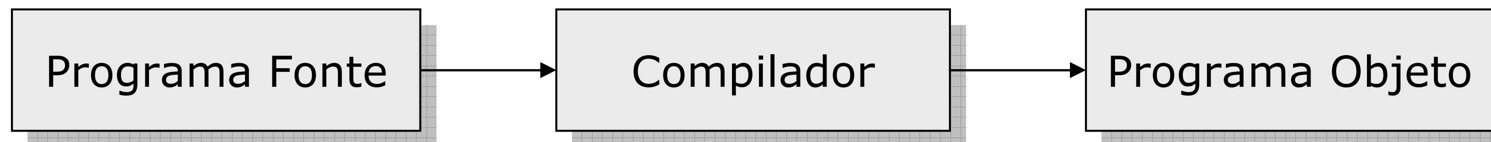
Máquinas Virtuais

Linguagens de Programação

- Segundo Price e Toscani (2001, p. 1) na programação de computadores utilizam-se linguagens de programação como meio de comunicação para com o computador escolhido.
- As linguagens mais utilizadas são as linguagens de alto nível, entendidas pelos seres humanos, mas totalmente irreconhecíveis aos computadores, que compreendem somente a linguagem de baixo nível (linguagem de máquina).
- Os programas escritos nas linguagens de alto nível precisam ser traduzidos para programas em linguagens de baixo nível.
- Esta tradução é realizada por sistemas especializados denominados compiladores.

Compiladores

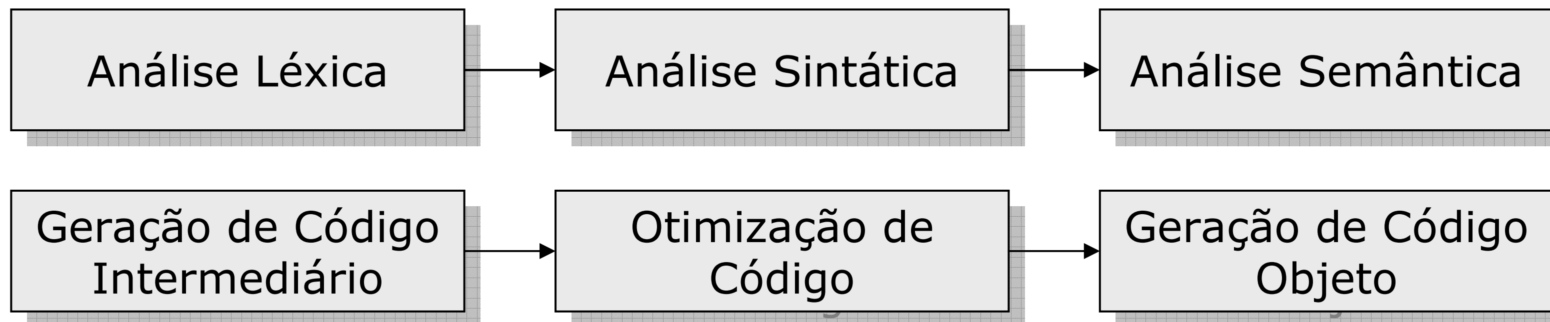
- Compilador: “um programa que aceita como entrada um texto de programa em uma certa linguagem e produz como saída um texto de programa em outra linguagem, preservando o significado deste texto” (GRUNE et al., 2001, p. 1).
- Mapeia um programa escrito em uma linguagem de alto nível (programa fonte) para programas equivalentes em linguagens de baixo nível (simbólica ou de máquina) viabilizando a sua execução em determinada arquitetura.



Compiladores: Fases de Compilação

- Os compiladores são compostos por funções padronizadas, onde a saída de uma função constitui a entrada de outra.

- O processo de compilação é dividido em duas grandes fases, que compreendem a **análise** do programa fonte e a posterior **síntese** para a derivação do código objeto. (PRICE; TOSCANI, 2001, p. 7).



- O código objeto pode ser a linguagem assembler de uma determinada arquitetura ou uma linguagem intermediária a ser interpretada por uma máquina virtual.

Máquinas Virtuais

- Usualmente, as máquinas virtuais são um conjunto de instruções que operam em uma máquina de pilhas abstrata.
- Segundo Gough (2002, p. 2), o principal objetivo no uso de máquinas virtuais é promover a portabilidade.
- As instruções de uma máquina virtual baseada em pilhas retiram e inserem operandos que estão em uma pilha de avaliação. As instruções são basicamente de 3 tipos:
    - Instruções para inserir operandos;
    - Instruções para trabalhar com operandos;
    - Instruções para retirar operandos e armazená-los em áreas de memória.

Máquinas Virtuais

- Criação da linguagem Java: os programas Java são compilados para *bytecodes* compreendidos pela *Java Virtual Machine* (JVM).
- A plataforma Microsoft .NET também é baseada em uma máquina virtual: o *Common Language Runtime* (CLR).
- Características em comum:
    - São baseadas em pilhas (*stack-based*);
    - Possuem um mecanismo para gerenciamento de memória (*garbage collector*);
    - Instruções primitivas que dão suporte para a Orientação a Objetos.

Plataforma Microsoft .NET

Plataforma Microsoft .NET

- É uma plataforma de desenvolvimento para promover a integração de aplicativos e serviços na *internet*.
- É semelhante ao Java2 *Enterprise Edition*.
- O maior é objetivo é promover a interoperabilidade das aplicações na *internet*.
- A plataforma é composta basicamente de:
    - *Common Language Runtime* (CLR);
    - .NET *Framework Class Library* (FCL);
    - Microsoft *Intermediate Language* (MSIL).

Plataforma Microsoft .NET: CLR

- É o mecanismo de execução dos aplicativos, composto por um conjunto de rotinas que carregam os aplicativos compilados para a plataforma Microsoft .NET.
- Durante a carga dos aplicativos, é realizada a verificação de todas as referências para as classes, métodos, permissões de segurança além do gerenciamento de memória do aplicativo.
- Objetivos do CLR:
    - Melhorar a confiabilidade e segurança dos aplicativos;
    - Reduzir o volume de código de baixo nível nos fontes dos programas.

Plataforma Microsoft .NET: CLR

- Permite ao desenvolvedor utilizar qualquer linguagem de programação, desde que os compiladores para as linguagens escolhidas gerem código MSIL.
- O resultado do processo de compilação é um módulo gerenciado (*managed module*). Um módulo gerenciado é um arquivo executável, que requer o CLR para ser executado.
- Vários módulos gerenciados formam um *assembly*.
- Um *assembly* pode conter diversas classes.

Plataforma Microsoft .NET: FCL

- O *Framework Class Library* é um conjunto de *assemblies* com milhares de classes que implementam diversas funcionalidades para os desenvolvedores.
- O uso da FCL permite aos desenvolvedores projetarem os seguintes tipos de aplicativos:
    - *Web Services*;
    - *Web Forms*;
    - *Windows Forms*;
    - *Windows Console Applications*;
    - *Windows Services*;
    - *Component Libraries*;
- As classes do FCL são agrupadas conforme a sua funcionalidade em *namespaces* (RICHTER, 2002, p. 22).

Plataforma Microsoft .NET: MSIL

- É uma linguagem de máquina de alto nível, independente de CPU.
- Todas as instruções trabalham com valores inseridos em uma pilha de avaliação.
- Os desenvolvedores de compiladores não precisam se preocupar com o gerenciamento de registradores, pois o MSIL não possui instruções pra manipulação de registradores.
- O MSIL contém cerca de 220 instruções. Dois terços destas instruções são instruções básicas; o restante são instruções para manipulação de objetos.

Plataforma Microsoft .NET: JIT

- As CPUs atuais não são capazes de executar as instruções do MSIL diretamente.
- Para ser executado, o MSIL precisa ser traduzido para instruções nativas da CPU.
- Essa tradução é realizada pelo compilador *Just-In-Time*.
- Existem três tipos de compiladores JIT:
  - *DefaultJIT*;
  - *EconoJIT*;
  - *PreJIT*.

Especificação da Linguagem Proposta

Especificação da Linguagem Proposta

- A nova linguagem desenvolvida constitui uma linguagem de programação sequencial imperativa.
- Gera aplicações do tipo *console* a ser executada na plataforma Microsoft .NET.
- Características:
  - Comandos em língua portuguesa;
  - Estrutura semelhante a linguagem C;
  - *Case-sensitive*.

Especificação da Linguagem Proposta: Comandos

- Os comandos da linguagem proposta incluem:
  - Declaração de variáveis locais;
  - Comando de seleção;
  - Comando de repetição;
  - Comando para entrada de dados;
  - Comando para saída de dados;
  - Definição e uso de módulos;

Especificação da Linguagem Proposta: Tokens

- Comentários:
  - Comentário de linha: //
  - Comentário em bloco: /* */
- Identificadores:

```
identificador ::= {L}+{L|D|_}
L ::= a..zA..Z
D ::= 0..9
```

- Palavras reservadas:

```
programa   logico   byte  caractere inteiro real     cadeia vazio
retorne    escreva  leia  se        senao   enquanto
verdadeiro falso
```

Especificação da Linguagem Proposta: Tokens

- Constantes númericas inteiras e reais:

```
integer ::= {D}+
float   ::= {D}[.{D}+]|.{D}+
```

- Constantes cadeias:

```
string ::= "{C}"
```

- Constantes caracteres:

```
character ::= 'C'
C ::= \32..\255
```

- Constantes lógicas:

```
boolean_literal ::= verdadeiro | falso
```

Especificação da Linguagem Proposta: Operadores

- Operadores relacionais, aritméticos e lógicos que podem ser utilizados na linguagem:

Operador	**Descrição**
==	Igual a
!=	Não igual a
<	Menor que
>	Maior que
<=	Menor ou igual a
>=	Maior ou igual a
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão
&&	E lógico
||	Ou lógico
!	Negação lógica

Especificação da Linguagem Proposta: Sintaxe

- Um programa consiste em uma sequência de declarações de módulos inserido em um programa principal.
- A recursão é permitida.
- Parâmetros dos módulos são passados por valor.
- Módulos podem ou não retornar um valor.
- Nos módulos são inseridos os comandos do programa.
- Todo programa deve possuir um módulo denominado `principal()`, que define o ponto de entrada para a execução.

Especificação da Linguagem Proposta: Sintaxe (BNF)

<compilation_unit> ::= <program_declaration>

<program_declaration> ::=
  programa <identifier>
  { [<method_declarations>] **}**

<method_declarations> ::=
  { <method_declaration> }

<method_declaration> ::=
  <type_specifier>
  <identifier>
  ([<parameter_list>] **)**
  <statement_block>

<type_specifier> ::=
  logico | **byte** | **caractere** | **inteiro** | **real** | **cadeia** | **vazio**

<parameter_list> ::=
  <parameter> [, <parameter_list>]

Especificação da Linguagem Proposta: Sintaxe (BNF)

```
<parameter> ::=
  <type_specifier>
  <identifier>

<statement_block> ::=
  { [{ <statement> }] }

<statement> ::=
    <statement_block>
  | <variable_declaration>
  | <if_statement>
  | <while_statement>
  | <return_statement>
  | <write_statement>
  | <read_statement>
  | <assign_statement>
  | <call_statement>

<variable_declaration> ::=
  <type_specifier>
  <identifier>
  [= <variable_initializer>] ;
```

Especificação da Linguagem Proposta: Sintaxe (BNF)

```
<variable_initializer> ::=
  <expression> ;

<if_statement> ::=
  se
  ( <expression> )
  <statement>
  [ senao <statement> ]

<while_statement> ::=
  enquanto
  ( <expression> )
  <statement>

<return_statement> ::=
  retorne [ <expression> ] ;

<write_statement> ::=
  escreva ( <argument_list> ) ;

<argument_list> ::=
  <expression> [, <argument_list>]
```

Especificação da Linguagem Proposta: Sintaxe (BNF)

```
<read_statement> ::=
  leia ( <identifier_list> ) ;

<identifier_list> ::=
  <identifier> [, <identifier_list>]

<assign_statement> ::=
  <identifier> <assign_op> <expression> ;

<assign_op> ::= = | *= | /= | %= | += | -=

<call_statement> ::=
  <identifier> ( [argument_list] ) ;

<expression> ::= <or_term>
<or_term>    ::= <and_term> [<or_level_op> <or_term>]
<and_term>   ::= <cmp_term> [<and_level_op> <and_term>]
<cmp_term>   ::= <add_term> [<cmp_level_op> <cmp_term>]
<add_term>   ::= <mult_term> [<add_level_op> <add_term>]
<mult_term>  ::= <factor> [<mult_level_op> <mult_term>]
```

Especificação da Linguagem Proposta: Sintaxe (BNF)

```
<factor> ::=
    <reference_term>
  | <literal_expression>
  | <casting_expression>
  | ( <expression> )
  | <negation_op> <factor>
  | <unary_op> <factor>

<or_level_op>   ::= ||
<and_level_op>  ::= &&
<cmp_level_op>  ::= == | != | < | > | >= | <=
<add_level_op>  ::= + | -
<mult_level_op> ::= / | * | %
<unary_op>      ::= - | +
<negation_op>   ::= !

<reference_term> ::=
  <identifier> [ ( [<argument_list>] ) ]

<casting_expression> ::=
  ( <type_specifier> ) <expression>
```

Especificação da Linguagem Proposta: Sintaxe (BNF)

<literal_expression> ::=
<numeric_literal> | <boolean_literal> | <string> | <character>

<numeric_literal> ::=
<integer> | <float>

<boolean_literal> ::= **verdadeiro** | **falso**
<integer> ::= {D}+
<float> ::= {D}[.{D}+]|.{D}+
<identifier> ::= {L}+{L|D|**_**}
<string> ::= "{C}"
<character> ::= 'C'

D ::= **0..9**
L ::= **a..zA..Z**
C ::= **\32..\255**

Especificação da Linguagem Proposta: Semântica

```
programa t01
{
  vazio principal()
  {
    escreva("olá");
  }
}
```

```
.assembly extern mscorlib { }
.assembly t01 { }
.module t01.exe

.class public t01
{
  .method public static void principal()
  {
    .entrypoint
    ldstr "olá\n"
    call  void [mscorlib]System.Console::Write(string)
    ret
  }
}
```

Especificação da Linguagem Proposta: Semântica

```
vazio principal()
{
  inteiro iIdade;
  iIdade = 10;
}
```

```
.method public static void principal()
{
  .entrypoint
  .locals (int32 V_0)
  ldc.i4 10
  stloc  V_0
  ret
}
```

Especificação da Linguagem Proposta: Semântica

```
escreva(media(8, 7));

real media(inteiro iN1, inteiro iN2)
{
  retorne (iN1 + iN2) / 2.0;
}
```

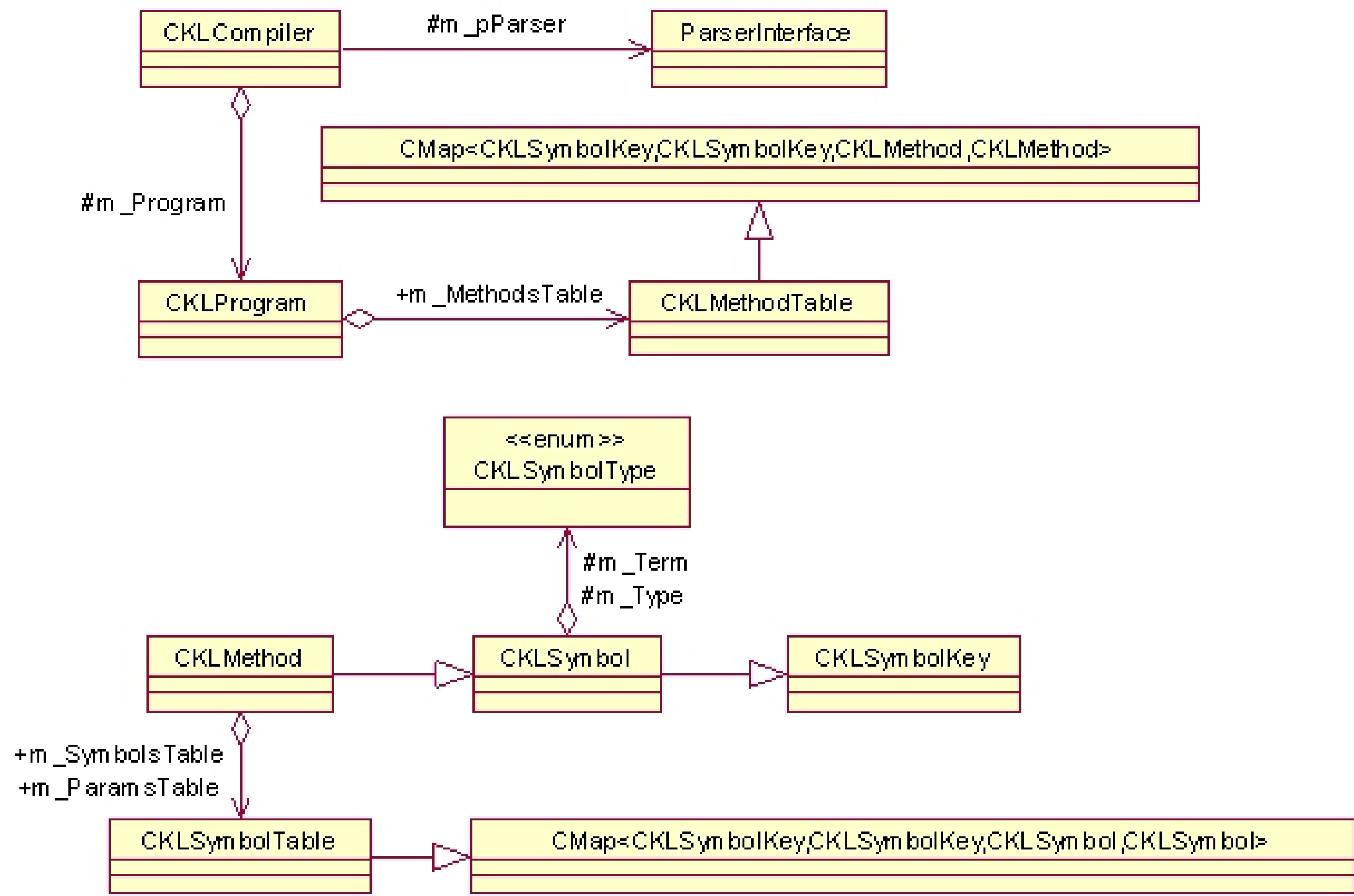
```
ldc.i4 8
ldc.i4 7
call float64 t01::media(int32,int32)

.method public static float64 media(int32 A_0,int32 A_1)
{
  ldarg A_0
  conv.r8
  ldarg A_1
  conv.r8
  add
  ldc.r8 2.0
  div
  ret
}
```

Especificação do Compilador

Especificação do Compilador

- Foi especificado utilizando a *Unified Modeling Language* (UML).
- A ferramenta *ProGrammar* foi utilizada para proceder a análise léxica e sintática do programa fonte.
- A análise semântica e a geração de código são realizadas através da varredura da árvore de *parsing* gerada pela ferramenta.
- Deve ser capaz de informar erros léxicos, sintáticos e semânticos no fonte.
- É um compilador de duas passagens.
- A BNF deve ser adaptada para uma forma que seja compreendida pela ferramenta *ProGrammar*.

Especificação do Compilador: Classes



Implementação do Compilador

Implementação do Compilador

- No construtor da classe CKLCompiler, a gramática da linguagem proposta é atribuída para a interface ParserInterface.
- O processo de compilação é iniciado através do método Compile() da classe CKLCompiler.
- O arquivo a ser compilado é informado para a interface ParserInterface.
- Através do método Parse na interface ParserInterface, é iniciado o processo de análise léxica e sintática do texto.
- Processamento da árvore gramatical gerada no caso de sucesso ou relatório de erros de compilação.
- Processamento das ações semânticas e geração de código IL a partir da árvore gramatical.

Implementação do Compilador: BNF

```
compilation_unit ::= [program_declaration] ;

 program_declaration ::=
  "programa" program_name "{" [method_declarations] "}" ;

 method_declarations ::=
  {method_declaration} ;

 method_declaration ::=
  type_specifier method_name "("[parameter_list]")"
    statement_block ;

 type_specifier<TERMINAL> ::=
    "logico" | "byte"   | "caractere" | "inteiro"
  | "real"   | "cadeia" | "vazio" ;

 method_name<TERMINAL>  ::= ident ;
 program_name<TERMINAL> ::= ident ;
```

Implementação do Compilador: Árvore Gramatical

```
programa t02
{
  vazio principal()
  {
  }
}
```

```
compilation_unit
  program_declaration
    program_name = "t02"
    method_declarations
      method_declaration
          type_specifier = "vazio"
          method_name = "principal"
          statement_block = "{...}"
```

- Para cada regra de produção é definida uma ação semântica a ser executada.

Implementação do Compilador: Ações Semânticas

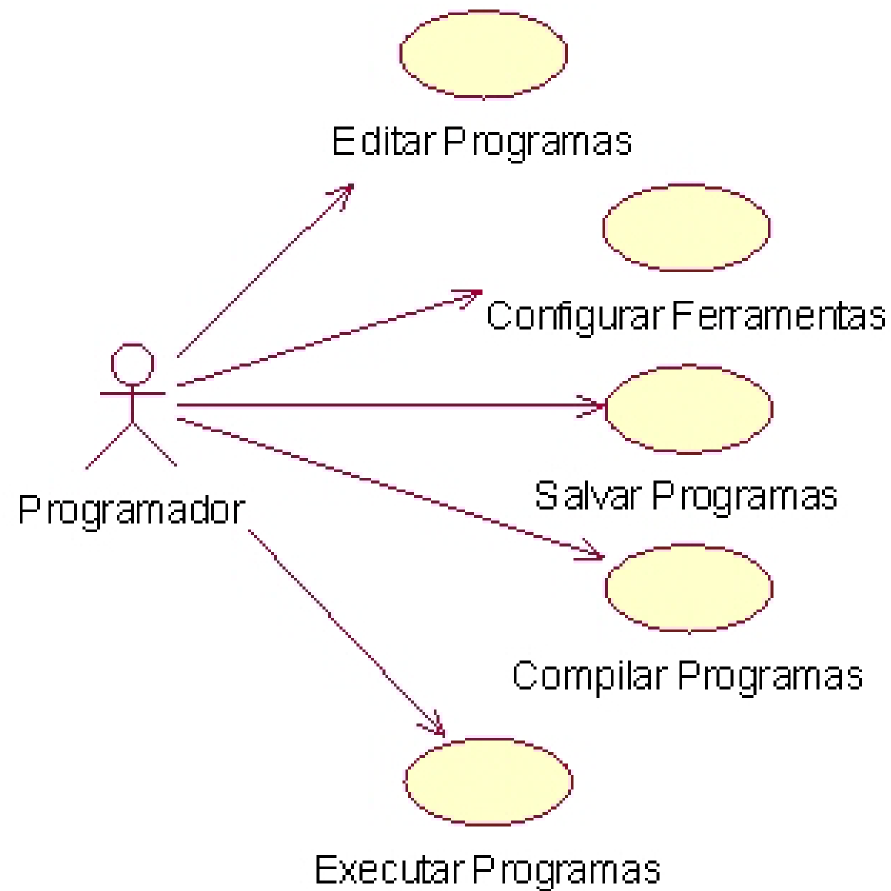
```
void CKLCompiler::ParseProgramDecl(const long& lNodeID)
{
  // nome programa
  m_Program.SetName(m_pParser->GetValue(m_pParser->GetChild(lNodeID, 0)));

  CString sCode;
  // gera codigo do cabecalho
  sCode.Format("\n.assembly extern mscorlib { }\n.assembly %s { }" \
               "\n.module %s.exe\n\n.class public %s\n{\n", \
         m_Program.GetName(), m_Program.GetName(), m_Program.GetName());

  m_FileOut.WriteString(sCode);

  // metodos programa
  for (long lM=0; lM<m_pParser->GetNumChildren(
    m_pParser->GetChild(lNodeID, 1)); lM++)
  {
     ParseMethodDecl(
       m_pParser->GetChild(m_pParser->GetChild(lNodeID, 1), lM));
  }

  m_FileOut.WriteString("}\n");
}
```

Ambiente de Desenvolvimento

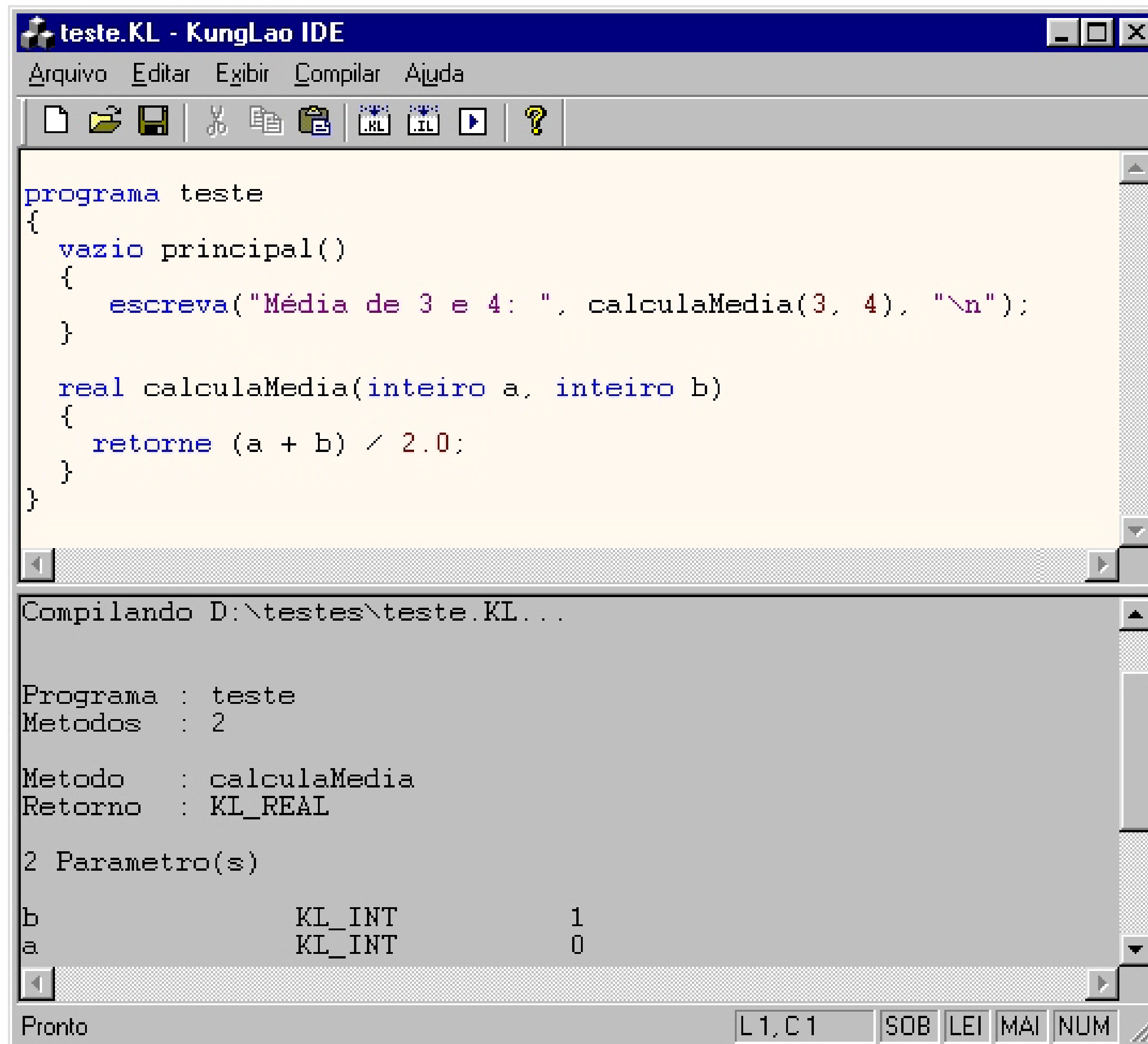
Ambiente de Desenvolvimento

- Foi especificado utilizando a *Unified Modeling Language* (UML).
- Objetivo do ambiente é facilitar o desenvolvimento de programas na linguagem proposta.
- Possui um editor de textos, opções para configurar e executar ferramentas de compilação (compilador e montador).
- Permite a execução dos programas criados na linguagem proposta.

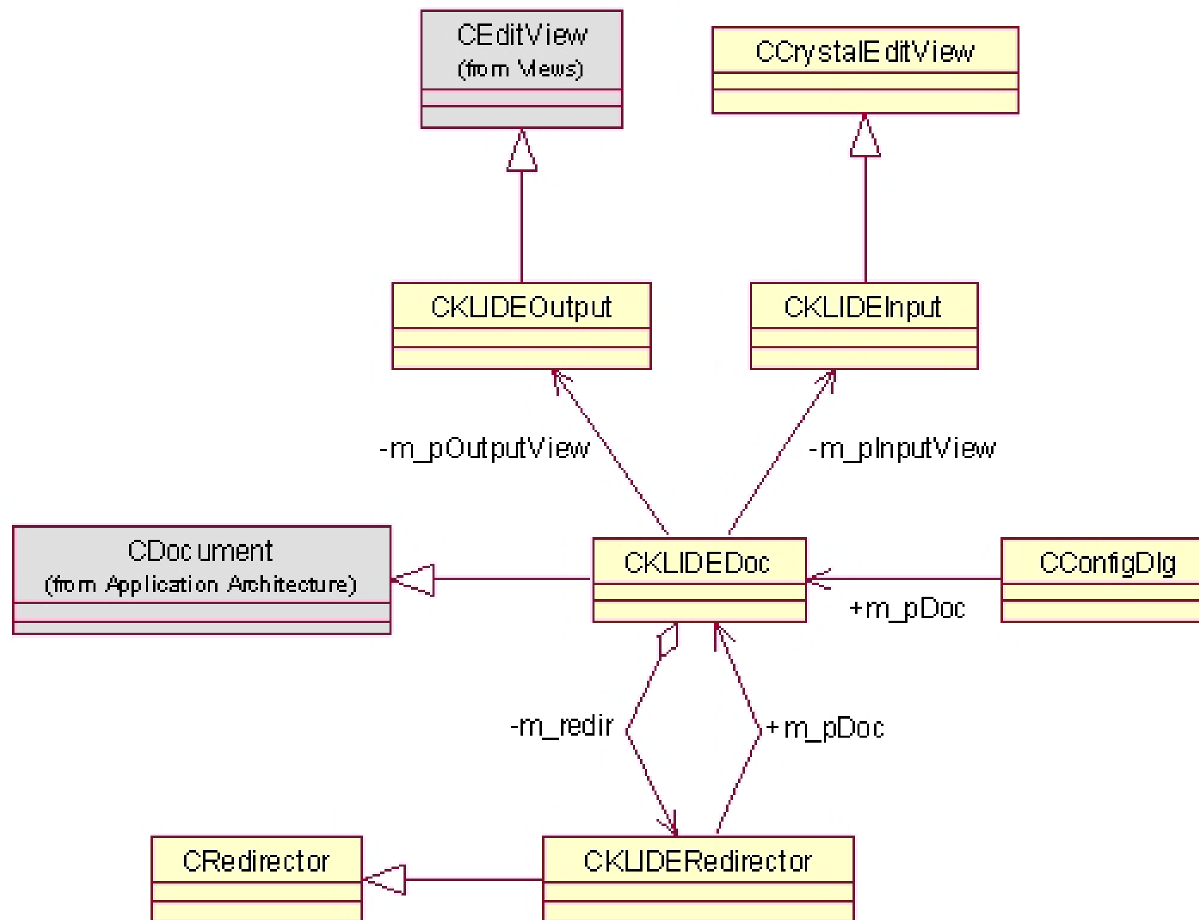
Ambiente de Desenvolvimento: Casos de Uso



Ambiente de Desenvolvimento: Interface



Ambiente de Desenvolvimento: Classes



Conclusões

- O objetivo principal, construir um compilador que traduz o código da linguagem proposta para código MSIL foi alcançado.
- A pesquisa e o uso de ferramentas adequadas promoveram a simplificação e rapidez no processo de desenvolvimento.
- Contribuição: o estudo das instruções do MSIL e do funcionamento da plataforma Microsoft .NET.

Conclusões: Limitações/Possíveis Extensões

- Uso de vetores e matrizes.
- Passagem de parâmetros por referência nos módulos.
- Alocação dinâmica de memória.
- Suporte a orientação a objetos.