Trabalho de Conclusão de Curso

FURBOL.NET

Autor: Leonardo Gesser
Orientador: José Roque Voltolini da Silva

Roteiro

- **Introdução**
  - **Objetivos do trabalho**
- **Fundamentação teórica**
- **Desenvolvimento**
  - **Requisitos do sistema**
  - **Especificação**
  - **Implementação**
  - **Operacionalidade**
- **Conclusões**

Introdução

Introdução

- Linguagens de programação
- Compiladores
- Microsoft .NET
- FURBOL

Objetivos do trabalho

Especificar uma nova linguagem orientada a objetos, tendo como base a sintaxe da linguagem FURBOL

- Criar uma Integrated Development Environment (IDE)
- Gerar código para a linguagem intermediária CIL
- Permitir a utilização da Base Class Library (BCL) e de outros assemblys escritos em linguagens .NET
- Permitir a utilização de recursos da orientação a objetos como herança e polimorfismo.

Fundamentação teórica

Linguagens de programação

- Descrevem operações computacionais para serem interpretadas por pessoas e máquinas

- Classificação por geração

    - Primeira geração - linguagens de máquina

    - Segunda geração - linguagens assembly

    - Terceira geração - linguagens de alto nível: Fortran, C# e Java

    - Quarta geração - desenhadas especificamente para aplicações: Structured Query Language (SQL), PostScript

    - Quinta geração - linguagens baseadas em lógica: Prolog e OPS5.

Linguagens de programação

Outras classificações

- Linguagens imperativas
- Linguagens declarativas
- Linguagens orientadas a procedimentos
- Linguagens orientadas a objetos

Linguagens orientadas a objetos

Conceitos

- Abstração - habilidade e capacidade de modelar conceitos, entidades, elementos, problemas e características do mundo real

- Encapsulamento - habilidade de um objeto em esconder seus dados da visão exterior

- Herança - permite criar novas classes a partir de classes já existentes, aproveitando as características da classe a ser estendida

- Polimorfismo - denota uma situação na qual um objeto pode se comportar de maneiras diferentes ao receber uma mensagem, dependendo de como foi criado

Linguagens orientadas a objetos

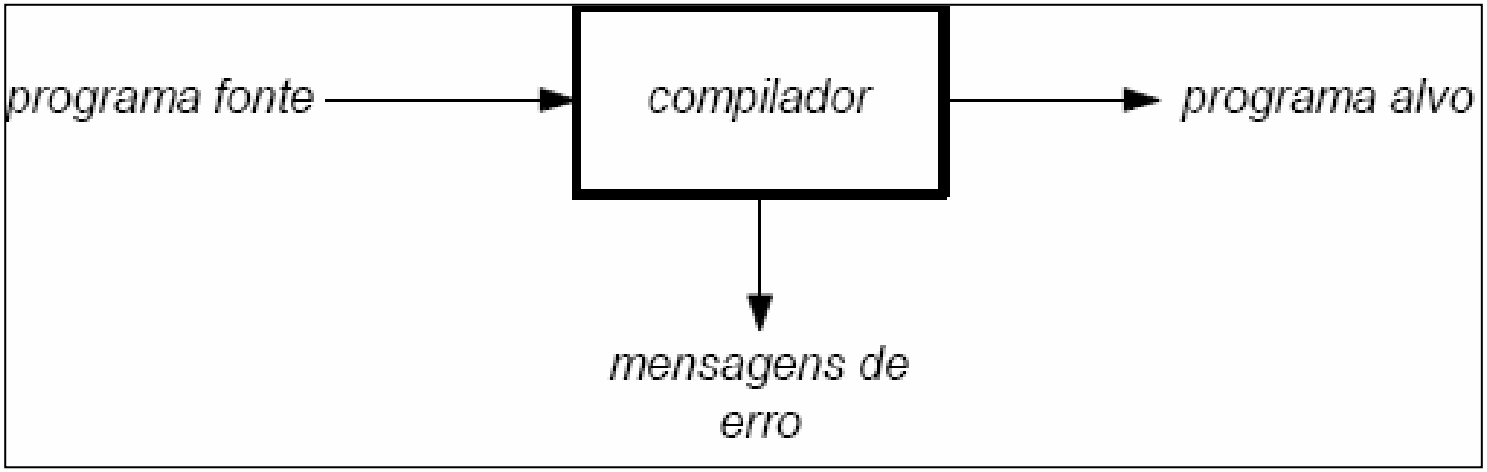
Vantagens

- aumento de produtividade
- reutilização de código
- redução das linhas de código programadas
- separação de responsabilidades
- componentização
- maior flexibilidade do sistema
- escalabilidade
- facilidade na manutenção

Compiladores

Efetua automaticamente a tradução de textos, redigidos em uma determinada linguagem de programação para alguma outra forma que viabilize sua execução

Em geral esta forma é a de uma linguagem de máquina



Fonte: Aho, Sethi e Ullman (1995, p. 1).

Compiladores

Fases da compilação

- Análise léxica
- Análise sintática
- Análise semântica
- Geração de código intermediário
- Otimização de código
- Geração de código objeto

MICROSOFT .NET

É um componente integrante do Windows que tem suporte a execução e geração de aplicativos e serviços Web XML

Objetivos

- prover a programação em um ambiente orientado a objetos
- prover um ambiente de execução de código seguro
- eliminar problemas de desempenho com scripts ou ambientes interpretados
- permitir desenvolvimento desktop, Web, móvel, etc…

.NET framework

Gerencia todos os aspectos da execução da aplicação

Componentes

- Common Language Runtime (CLR)
- Base Class Library (BCL)
- Common Language Specification (CLS)
- Common Type System (CTS)
- Common Intermadiate Language (CIL)

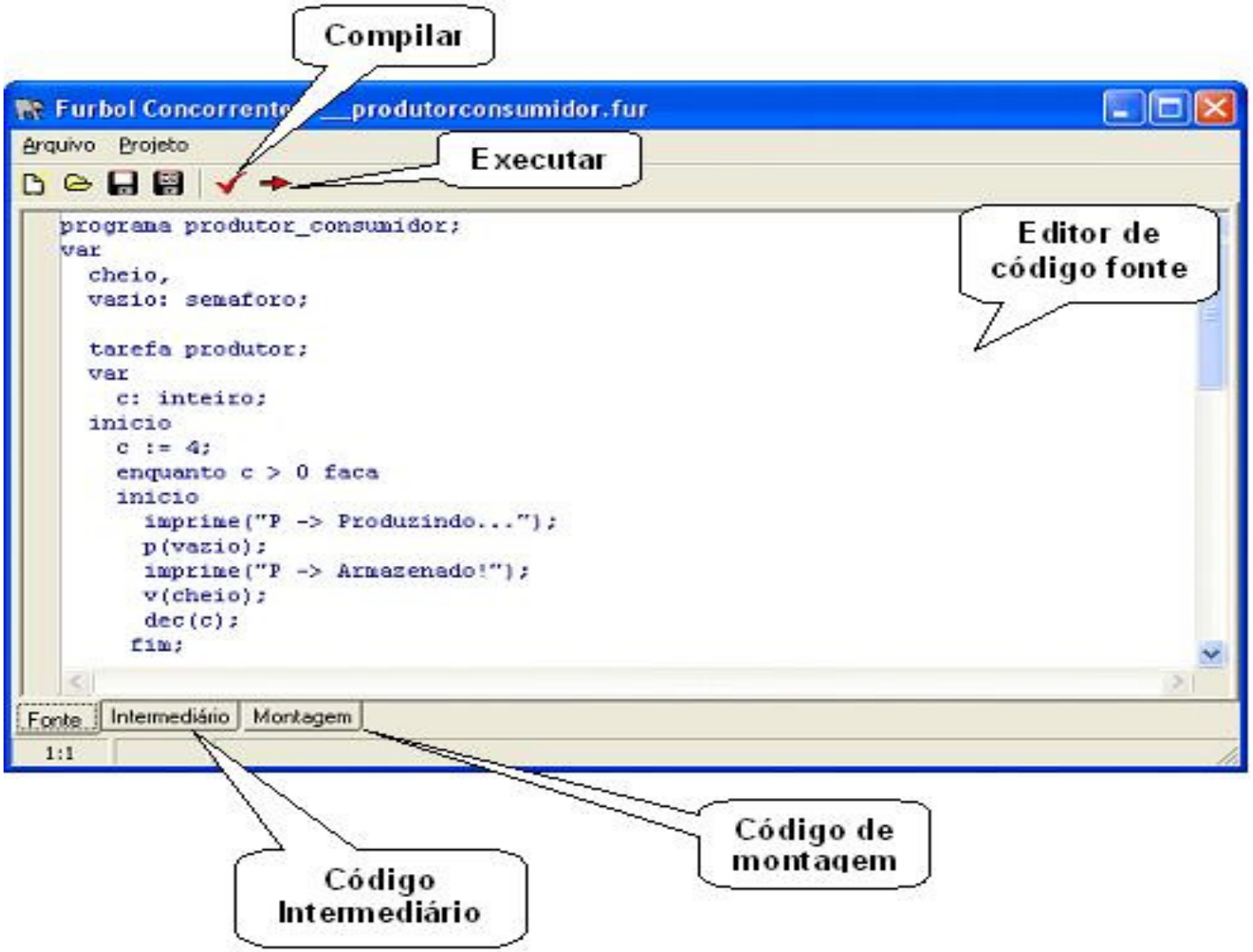
FURBOL

Vem sendo desenvolvido por vários integrantes do curso de Ciências da Computação da FURB

Características

- Comandos em língua portuguesa
- Comando de condição e repetição
- Unidades de programas concorrentes
- Mecanismo de sincronização da comunicação entre unidades concorrentes do tipo semáforo
- Tipos abstratos de dados
- Gera código de montagem (assembly) compatível com microprocessadores 8088

Interface do FURBOL



Desenvolvimento

Requisitos do Sistema

Requisitos da linguagem FURBOL.NET

- A linguagem deve dar suporte a programação orientada a objetos e na medida do possível preservar a sintaxe do FURBOL (Requisito Funcional - RF)

- A linguagem deve permitir utilizar a BCL (RF)

- Ser implementada em C# utilizando o Microsoft Visual Studio 2005 (Requisito Não Funcional - RNF)

Requisitos do Sistema

Requisitos da interface

- Deve prover mecanismos para edição, compilação e execução (RF)

- Deve ser independente do gerador de código, permitindo a implementação futura de geradores de códigos para outras máquinas (RNF)

- Ser implementado em C# utilizando o Microsoft Visual Studio 2005 (RNF)

Requisitos do Sistema

Requisitos do compilador

- Reportar os erros léxicos, sintáticos e semânticos (RF)
- Gerar código CIL correspondente ao código fonte compilado (RF)
- Gerar um assembly a partir do código CIL gerado (RF)
- Ser implementado em C# utilizando o Microsoft Visual Studio 2005 (RNF)

Especificação da linguagem

Características do FURBOL.NET

- Comandos em língua portuguesa
- Comandos de controle de execução semelhante ao FURBOL
- Estrutura de classes semelhante ao Java ou C#
- Case-insensitive

Especificação léxica da linguagem

Comentários

```
comentario_linha: // ([^(\n | \r)])* (\n | \r | \r\n)
comentario_bloco: /\* ([^\*/])* \*/
```

Númericos

```
digito : [0-9]
inteiro : (digito)+ (“l” | “L”)?

real : (digito)* (“.” (digito)+)? expoente? sufixo?
expoente : (“e” | “E”) (“+” | “-“)? digito+
sufixo : “f” | “F” | “d” | “D” | “m” | “M”
```

Caracteres

```
char : [^[\\,\n,\r]] | \\ (n | " | ' | \\
                    | 0 | a | b | f | r | t | v)
caractere : ’ char ‘
cadeia_regular: “\”” (char)* “\””
cadeia_verbatim: “@\”” ([^”\””] | “\”\””)* “\””
```

Especificação léxica da linguagem

Identificador

```
letra  : [a-zA-Z]
identificador : (letra | _) (letra | digito | _) *
```

Palavras reservadas

abstrato	entao	inc	padrao
ate	enumeracao	inicio	para
atribui	esse	inteiro	privado
busca	estatico	inteiro16	procedimento
byte	estende	interface	propriedade
cada	estrutura	interno	protegido
caracter	faca	interrompa	publico
caso	falso	levante	retorna
classe	fim	literal	se
construtor	final	logico	senao
continue	finalize	longo	sincronize
dec	flutuante	mod	sobrepor
decimal	flutuante64	nao	super
destrutor	for	novo	trate
e	funcao	nulo	var
eh	implementa	ou	verdadeiro
em	importar	pacote	virtual
enquanto			

Exemplo de Tokens

```
10 – inteiro
10L – longo
10l – longo
10.0 – flutuante
10.0D – flutuante64
10.0M – decimal
10.0e2 – flutuante
‘a’ – caractere
‘\n’  – caractere
“abc” – literal
“abc\n” – literal
@”abc\n” – literal
_id – identificador
identificador – identificador
abstrato – palavra reservada
@abstrato – identificador
```

Exemplo sintático de TADs

```
Importar System;

pacote Furb.Estruturas.Exemplo
inicio
  classe publico ClasseExemplo estende Object implementa InterfaceExemplo
  inicio
  fim

  interface InterfaceExemplo
  inicio
  fim

  estrutura EstruturaExemplo
  inicio
  fim

  enumeracao EnumeracaoExemplo
  inicio
  fim
fim
```

Especificação semântica de TADs

- TADs podem conter modificadores de acesso publico e interno
- Classes herdam implicitamente de System.Object
- Estruturas herdam implicitamente de System.ValueType
- Enumeracoes herdam implicitamente de System.Enum
- Classes podem assumir modificadores de definição
    - Abstrato: A classe não pode ser instanciada e pode possuir membros abstratos
    - Final: A classe não poderá ser estendida por outra classe

Exemplo sintático de membros de classe

```
classe publico ClasseExemplo
  inicio
    _valorCalculado : inteiro;

    procedimento Calcular()
    inicio
    fim

    funcao publico CalculaValor(): inteiro
    inicio
      retorna _valorCalculado + 10;
    fim

    propriedade ValorCalculado : Inteiro
    inicio
      busca inicio
        retorna _valorCalculado;
      fim

      atribui inicio
        valorCalculado := valor;
      fim
    fim

    construtor (valor : inteiro)
    inicio
      _valorCalculado := valor;
    fim

    destrutor
    inicio
    fim
  fim
```

Especificação semântica de membros da classe

- Métodos e propriedades, além de estáticos, podem ser
    - Virtual
    - Abstrato
    - Sobrepor
- Propriedades possuem métodos chamados de acessores
    - Busca
    - Atribui
- O acessor `atribui` possui um parâmetro oculto de nome `valor`

Especificação semântica de membros da classe

```
classe abstrato ClasseAbstrata
inicio
  funcao publico virtual CalcularValor() : inteiro
  inicio
  fim

  propriedade publico abstrato Valor : Inteiro
  inicio
    busca;
  fim
fim

classe abstrato ClasseFilha estende ClasseAbstrata
inicio
  funcao publico abstrato sobrepor CalcularValor(): Inteiro;
fim

classe final ClasseNeta estende ClasseFilha
inicio
  funcao publico sobrepor CalcularValor(): Inteiro
  inicio
  fim

  propriedade publico sobrepor Valor : Inteiro
  inicio
    busca inicio
      retorna 10;
    fim
  fim
fim
```

Especificação semântica de membros da classe

Construtores não são auto-herdados, sempre referenciando outro construtor

```
classe publico Pai
inicio
  valor : inteiro;

  construtor (parametro: inteiro)
  inicio
    valor := parametro;
  fim

  construtor ()
  inicio
  fim
fim

classe publico ClasseExemplo estende Pai
inicio
  construtor (parametro: Inteiro)
  inicio
  fim

  construtor ()
    :esse(0)
  inicio
  fim

  construtor (parametro: inteiro; parametro2: inteiro)
     :super(parametro2)
  inicio
  fim
fim
```

Código CIL de classes

```
pacote PacoteExemplos
inicio
  classe publico ClasseExemplo
  inicio
    privado _atributo : inteiro;

    propriedade publico Atributo :
                            Inteiro
    inicio
      busca inicio
        retorna _atributo;
      fim
      atribui inicio
        _atributo := valor;
      fim
    fim

    procedimento publico
        Principal(args: literal[])
    inicio
    fim
  fim
fim
```

```
.assembly ExemploDefinicoes
{
}
.assembly extern mscorlib
{
    .publickeytoken = (B7 7A 5C 56 19 34 E0 89)
}
.class public auto ansi beforefieldinit PacoteExemplos.ClasseExemplo
 extends [mscorlib]System.Object
{
    .field  private   int32 _atributo
    .method  public  newslot hidebysig  void Principal (string[] args)  cil
managed
    {}
    .method public hidebysig specialname instance int32
            get_Atributo() cil managed
    {}
    .method public hidebysig specialname instance void
            set_Atributo(int32 'valor') cil managed
    {}
    .method public hidebysig specialname rtspecialname
            instance void  .ctor() cil managed
    {}
    .method  public  specialname rtspecialname  void .ctor ()  cil managed
    {}
    .property instance int32 Atributo()
    {
      .get instance int32 PacoteExemplos.ClasseExemplo::get_Atributo()
      .set instance void PacoteExemplos.ClasseExemplo::set_Atributo(int32)
    }
}
```

Expressões

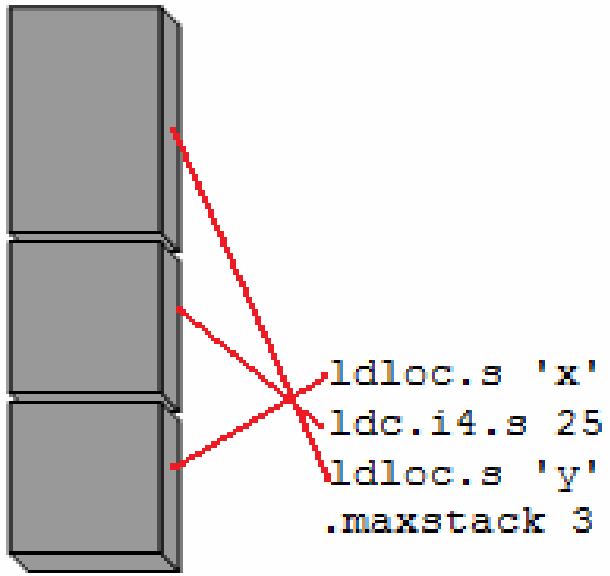
- Expressões unárias
- Expressões binárias
- Atribuição
- Chamada de métodos
- Criação de objetos
- Carregar variáveis
- Carregar atributos
- Carregar constantes
- Conversão
- Acessar arrays

Exemplo de expressões

```
// Expressão binária
// Atribuição
// Nova expressão
x := (10 - 3) * 5

// Constante
Console.WriteLine(10.ToString());

// Novo objeto
novo Object();

// Qualificador
esse.atributo = 10;
super.metodo();

// Expressão unária
x = - x;

// Conversão
var y : inteiro := inteiro(x);
var z : Classe := Classe(k);
```

Código CIL de expressões

- Expressões somente geram código dentro de métodos
- Métodos possuem variaveis locais e uma pilha de trabalho
- Todas as operações com variáveis e atributos passam pela pilha
- A pilha de um método tem um tamanho limite



Exemplo de expressões

Código FURBOL.NET

```
var a : ClasseX := novo ClasseX();
metodoEstatico();
a.metodoNaoEstatico();
var b : EstruturaX := novo EstruturaX();
b.metodoNaoEstatico(a, 10);
```

Código CIL

```
newobj instance instance void PacoteX.ClasseX::'.ctor'()
stloc.s 'a'
call void PacoteX.ClasseX::'metodoEstatico'()
ldloc.s 'a'
call instance void PacoteX.ClasseX::'metodoNaoEstatico'()
ldloca.s 'b'
initobj PacoteX.EstruturaX
ldloca.s 'b'
ldloc.s 'a'
ldc.i4 0xA
call instance void PacoteX.EstruturaX::'metodoNaoEstatico'(object, int32)
```

Arrays

Arrays Retangulares

Código FURBOL.NET

```
var ArrayRet : Inteiro[,] := novo inteiro[10, 20]; // 200 elementos
ArrayRet[2, 3] := 10;    // acessando array retangular
Console.WriteLine(ArrayRet[2, 3]);
```

Código CIL

```
ldc.i4 0xA
ldc.i4 0x14
newobj instance void int32[0...,0...]::.ctor(int32, int32)
stloc.s 'ArrayRet'
ldloc.s 'ArrayRet'
ldc.i4 0x2
ldc.i4 0x3
ldc.i4 0xA
call instance void int32[0...,0...]::Set(int32, int32,int32)
ldloc.s 'ArrayRet'
ldc.i4 0x2
ldc.i4 0x3
call instance int32 int32[0...,0...]::Get(int32, int32)
call void [mscorlib]System.Console::'WriteLine'(int32)
LB_0001: ret
```

Arrays

Arrays de Arrays

Código FURBOL.NET

```
var ArrayRetArray : Inteiro[,][] := novo Inteiro[10, 10][];
ArrayRetArray[3, 9] := novo Inteiro[15];
ArrayRetArray[3, 9][4] := novo inteiro();
```

Código CIL

```
ldc.i4 0xA
ldc.i4 0xA
newobj instance void int32[][0...,0...]::.ctor(int32,int32)
stloc.s 'ArrayRetArray'
ldloc.s 'ArrayRetArray'
ldc.i4 0x3
ldc.i4 0x9
ldc.i4 0xF
newarr [mscorlib]System.Int32
call instance void int32[][0...,0...]::Set(int32,int32,int32[])
ldloc.s 'ArrayRetArray'
ldc.i4 0x3
ldc.i4 0x9
call instance int32[] int32[][0...,0...]::Get(int32,int32)
ldc.i4 0x4
ldelema [mscorlib]System.Int32
initobj [mscorlib]System.Int32
LB_0001: ret
```

Especificação semântica dos comandos

- Comando `retorna`

```
funcao publico estatico MinhaFuncao(): inteiro
inicio
  retorna 10;
fim
```

```
.method  public  newslot hidebysig  static  int32 'MinhaFuncao' ()  cil managed
{
    ldc.i4 0xA
    br 'LB_0001'
    LB_0001: ret
    .maxstack 1
}
```

- Comando `levante`

```
funcao publico estatico MinhaFuncao(): inteiro
inicio
  levante novo Exception();
Fim
```

```
.method  public  newslot hidebysig  static  int32 'MinhaFuncao' ()  cil managed
{
    newobj instance instance void [mscorlib]System.Exception::'.ctor'()
    throw
    br 'LB_0001'
    LB_0001:
    ret
    .maxstack 1
}
```

Especificação semântica dos comandos

- Declaração de variavel e bloco de comandos interno

```
var x : inteiro;
inicio
  x := 10;
fim
trate (f : Exception)
inicio
fim
finalize
inicio
fim
```

```
.locals init
(
    [0] int32 'x'
    ,[1] class [mscorlib]System.Exception 'f'
)
.try
{
    .try
    {
        ldc.i4 0xA
        stloc.s 'x'
        leave.s 'LB_0003'
    }
    catch [mscorlib]System.Exception
    {
        stloc.s 'f'
        leave.s 'LB_0003'
    }
    LB_0003:
    leave.s 'LB_0002'
}
finally
{
    endfinally
}
LB_0002:
LB_0001:
ret
.maxstack 1
```

Especificação semântica dos comandos

- Comando `Se`

```
Se valor > 10 entao
Inicio
  Console.WriteLine(verdadeiro);
fim
senao
inicio
  Console.WriteLine(falso);
fim
```

```
ldloc.s 'valor'
ldc.i4 0xA
cgt
brfalse 'LB_0002'
ldc.i4.1
call void [mscorlib]System.Console::'WriteLine'(bool)
br 'LB_0003'
LB_0002:
ldc.i4.0
call void [mscorlib]System.Console::'WriteLine'(bool)
LB_0003:
LB_0001:
ret
```

Especificação semântica dos comandos

- Comando `Caso`

```
caso valor for
inicio
  1 inicio
       Console.WriteLine(“opção 1”);
     fim
  2 inicio
       Console.WriteLine(“opção 2”);
     fim
  padrao
     inicio
       // diferente de 1 e 2
     fim
fim
```

```
ldloc.s 'valor'
stloc.s 'FB$00001'
ldloc.s 'FB$00001'
ldc.i4 0x1
ceq
brtrue 'LB_0003'
ldloc.s 'FB$00001'
ldc.i4 0x2
ceq
brtrue 'LB_0004'
br 'LB_0002'
LB_0003:
ldstr "opção 1"
call void [mscorlib]System.Console::'WriteLine'(string)
br 'LB_0002'
LB_0004:
ldstr "opção 2"
call void [mscorlib]System.Console::'WriteLine'(string)
br 'LB_0002'
LB_0002:
LB_0001:
```

Especificação semântica dos comandos

- Comandos `Enquanto` e `Para`

```
enquanto  valor < 10 faca
inicio
  // bloco de repetição
fim

para var i : inteiro := 0
 ate i >= 10
 faca i := i + 1
inicio
  // bloco de repetição
fim
```

```
LB_0002:
ldloc.s 'valor'
ldc.i4 0xA
clt
brfalse 'LB_0003'
br 'LB_0002'
LB_0003:
ldc.i4 0x0
stloc.s 'i'
LB_0005:
ldloc.s 'i'
ldc.i4 0xA
clt
ldc.i4.0
ceq
brtrue 'LB_0004'
LB_0006:
ldloc.s 'i'
ldc.i4 0x1
add.ovf
stloc.s 'i'
br 'LB_0005'
LB_0004:
LB_0001:
Ret
```

Especificação semântica dos comandos

- Comandos `Para cada, interrompa, continue, inc` e `dec`

```
para cada j : inteiro em colecao faca
inicio
  se(j > 0) entao
  inicio
    interrompa;
  fim
  senao
  inicio
    continue;
  fim
Fim
Inc(valor);
Dec(valor);
```

```
ldloc.s 'colecao'
callvirt instance class [mscorlib]System.Collections.IEnumerator
[mscorlib]System.Collections.IEnumerable::'GetEnumerator'()
stloc.s 'FB$00001'
br 'LB_0003'
LB_0004:
ldloc.s 'FB$00001'
callvirt instance object
[mscorlib]System.Collections.IEnumerator::'get_Current'()
unbox.any [mscorlib]System.Int32
stloc.s 'j'
ldloc.s 'j'
ldc.i4 0x0
cgt
brfalse 'LB_0005'
br 'LB_0002'
br 'LB_0006'
LB_0005:
br 'LB_0003'
LB_0006:
LB_0003:
ldloc.s 'FB$00001'
callvirt instance bool
[mscorlib]System.Collections.IEnumerator::'MoveNext'()
brtrue 'LB_0004'
LB_0002:
ldloc.s 'valor'
ldc.i4 0x1
add.ovf
stloc.s 'valor'
ldloc.s 'valor'
ldc.i4 0x1
sub.ovf
stloc.s 'valor'
LB_0001:
ret
```

Especificação semântica dos comandos

- Comando `Sincronize`

`sincronize objetoLock`
`inicio`
`  // Área crítica`
`fim`
	`ldsfld object Pacotao.minha::objectLock`
	`dup`
	`stloc.s 'FB$00001'`
	`call void [mscorlib]System.Threading.Monitor::'Enter'(object)`
	`.try`
	`{`
	`    leave.s 'LB_0002'`
	`}`
	`finally`
	`{`
	`    ldloc.s 'FB$00001'`
	`    call void [mscorlib]System.Threading.Monitor::'Exit'(object)`
	`    endfinally`
	`}`
	`LB_0002:`
	`LB_0001:`
	`ret`

Casos de usos da interface

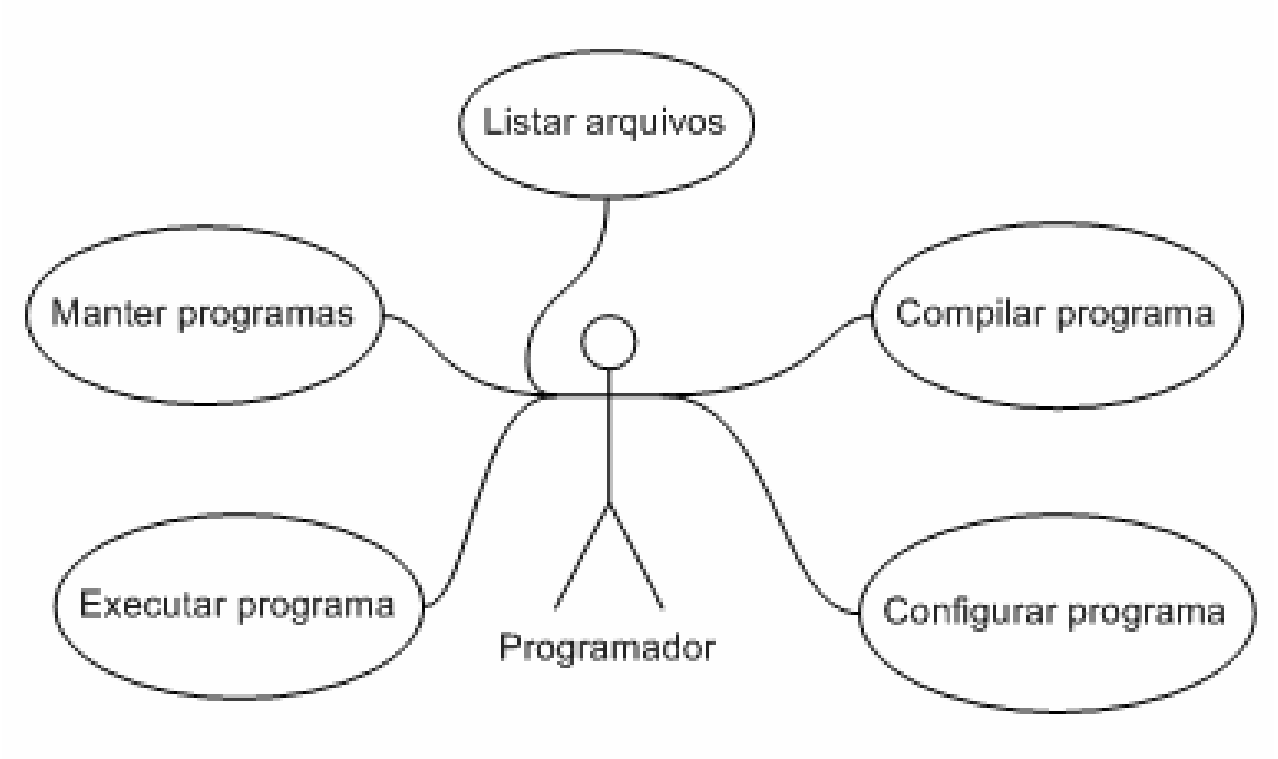


Diagrama de classes da interface

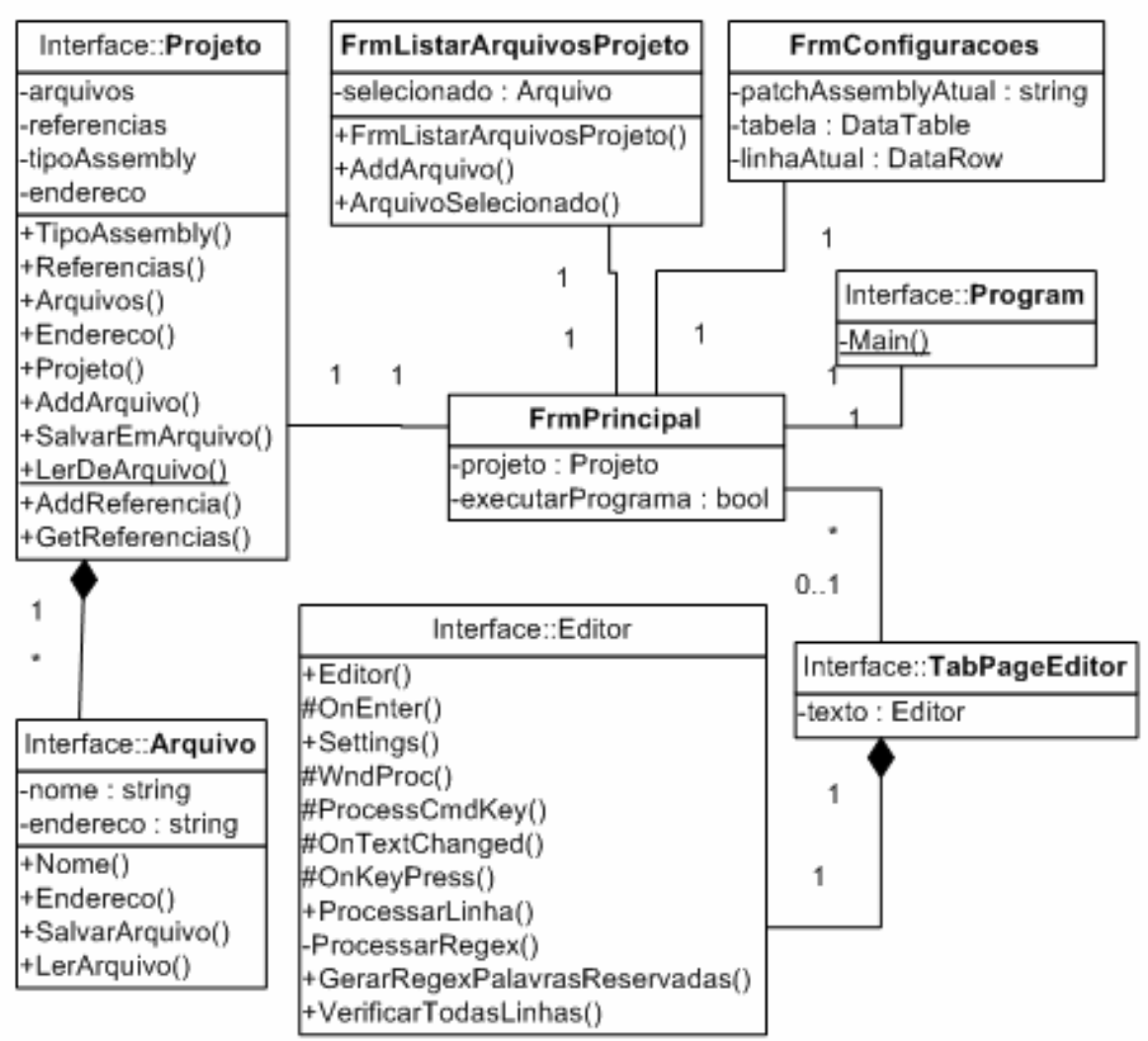


Diagrama de classes do Compilador

Pacotes

- Furb.Furbol.Compilador
- Furb.Furbol.Compilador.Lexico:
- Furb.Furbol.Compilador.Sintatico
- Furb.Furbol.Compilador.Sintatico.ArvoreSintatica
- Furb.Furbol.Compilador.Semantico
- Furb.Furbol.Compilador.Codigo

Diagrama de classes: CompiladorFurbol

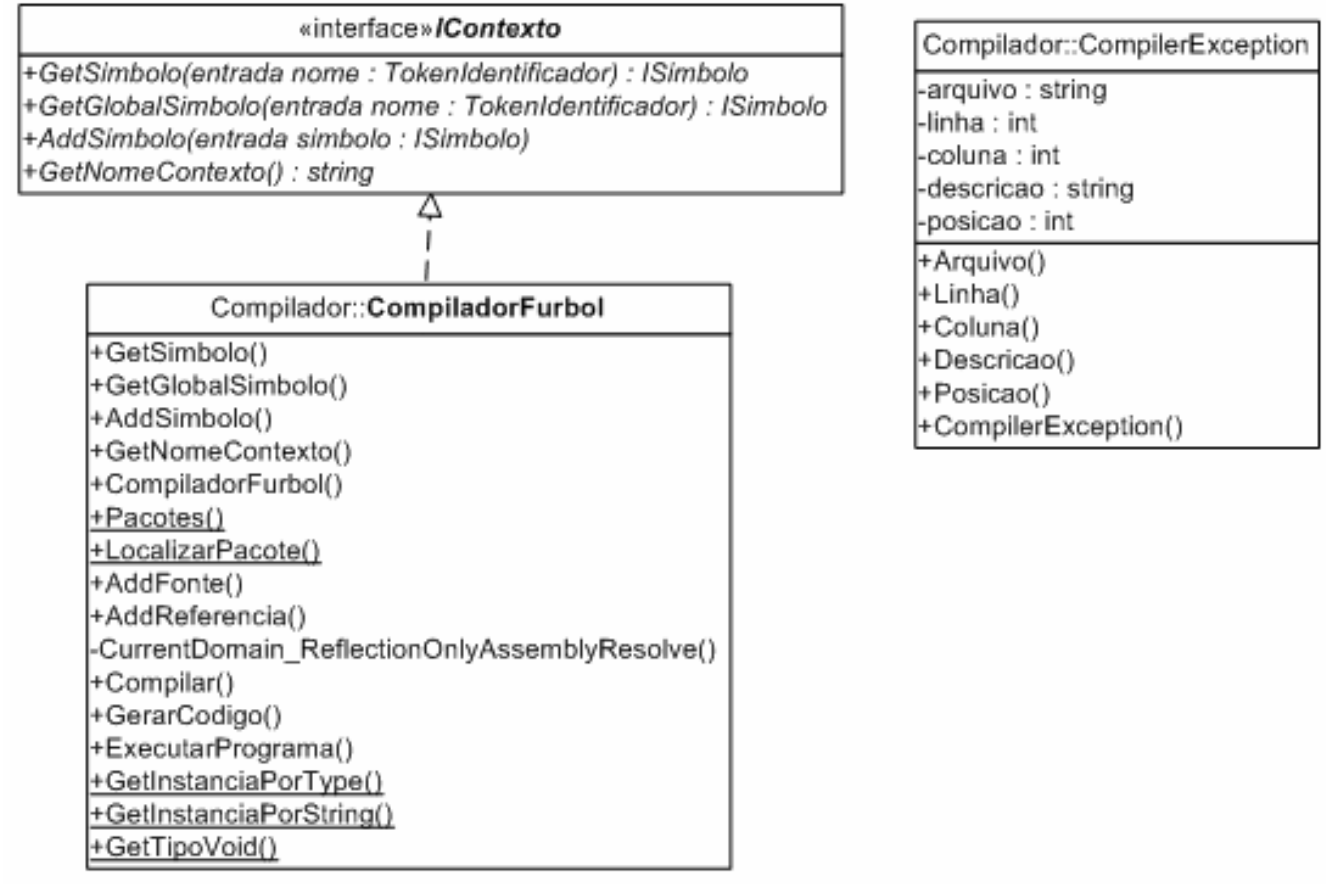


Diagrama de classes: Léxico

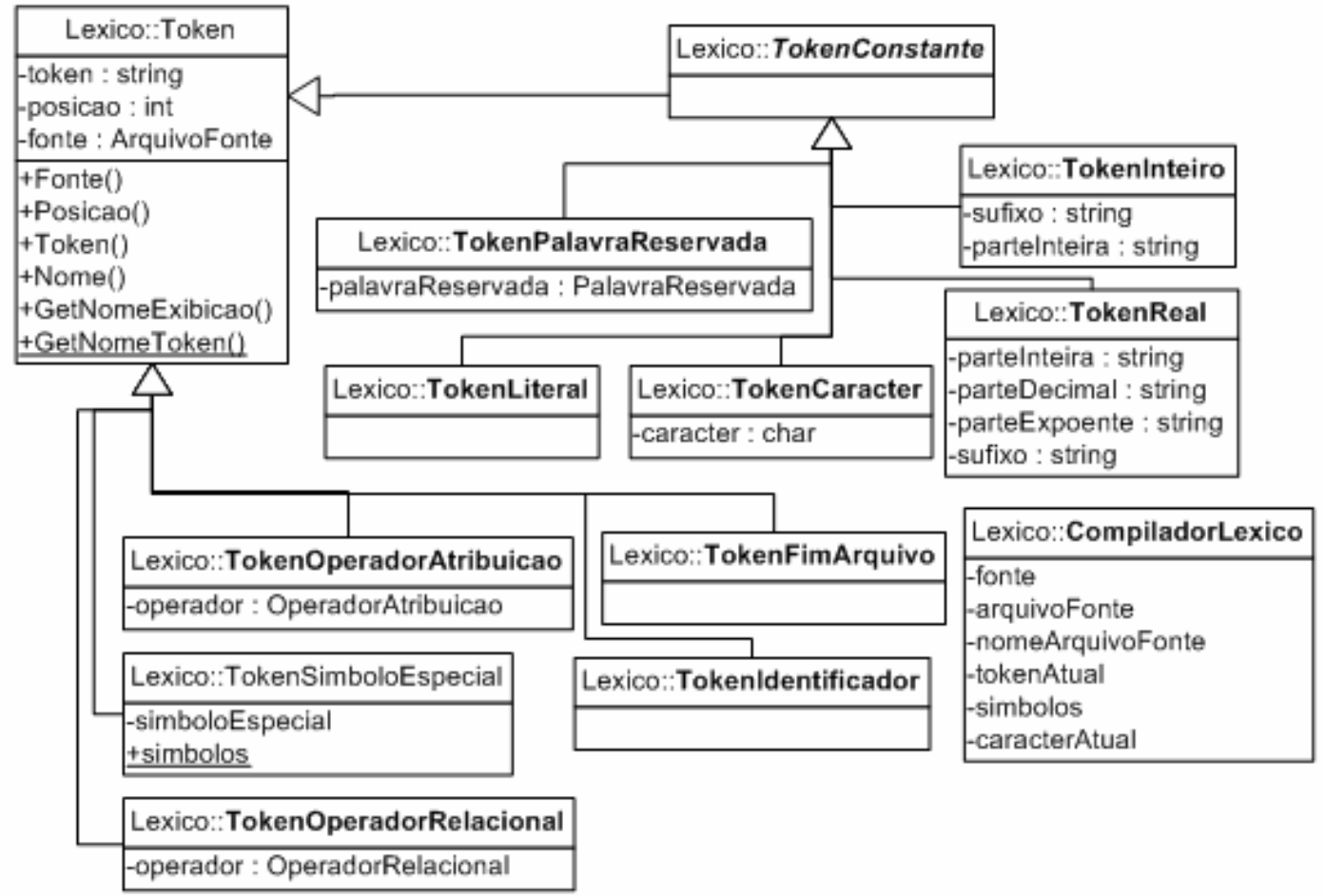


Diagrama de classes: Sintático

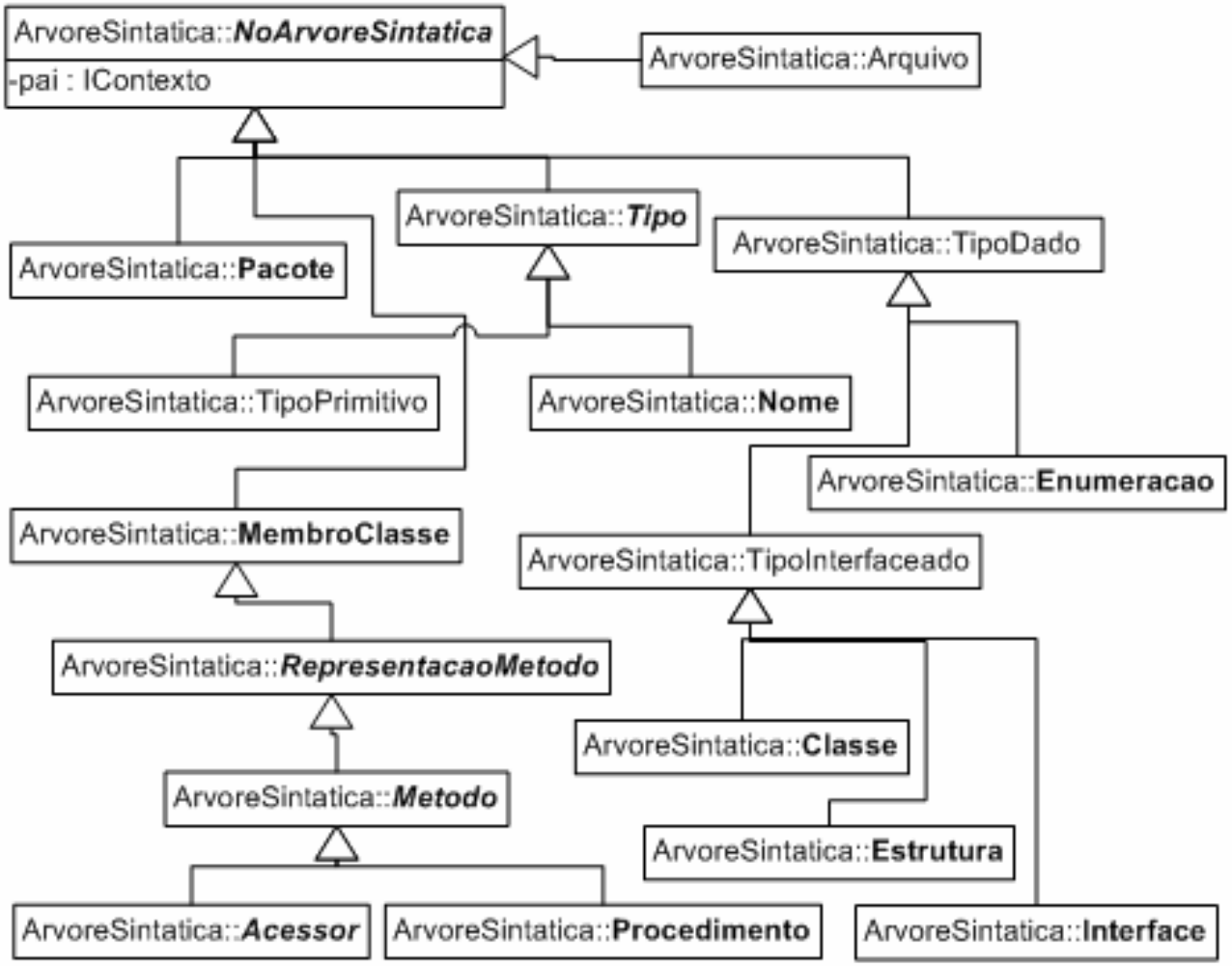


Diagrama de classes: Semântico

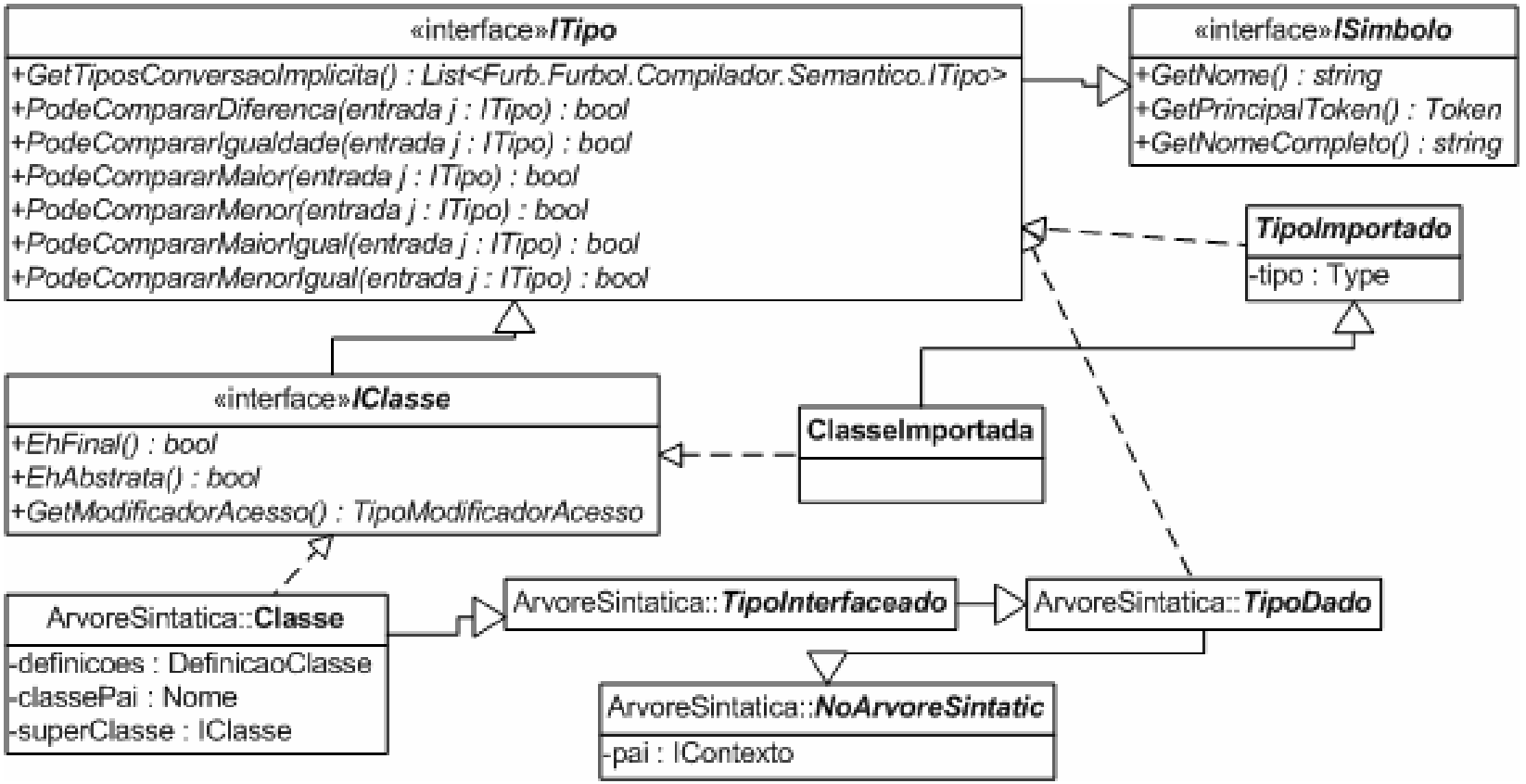


Diagrama de classes: Codigo

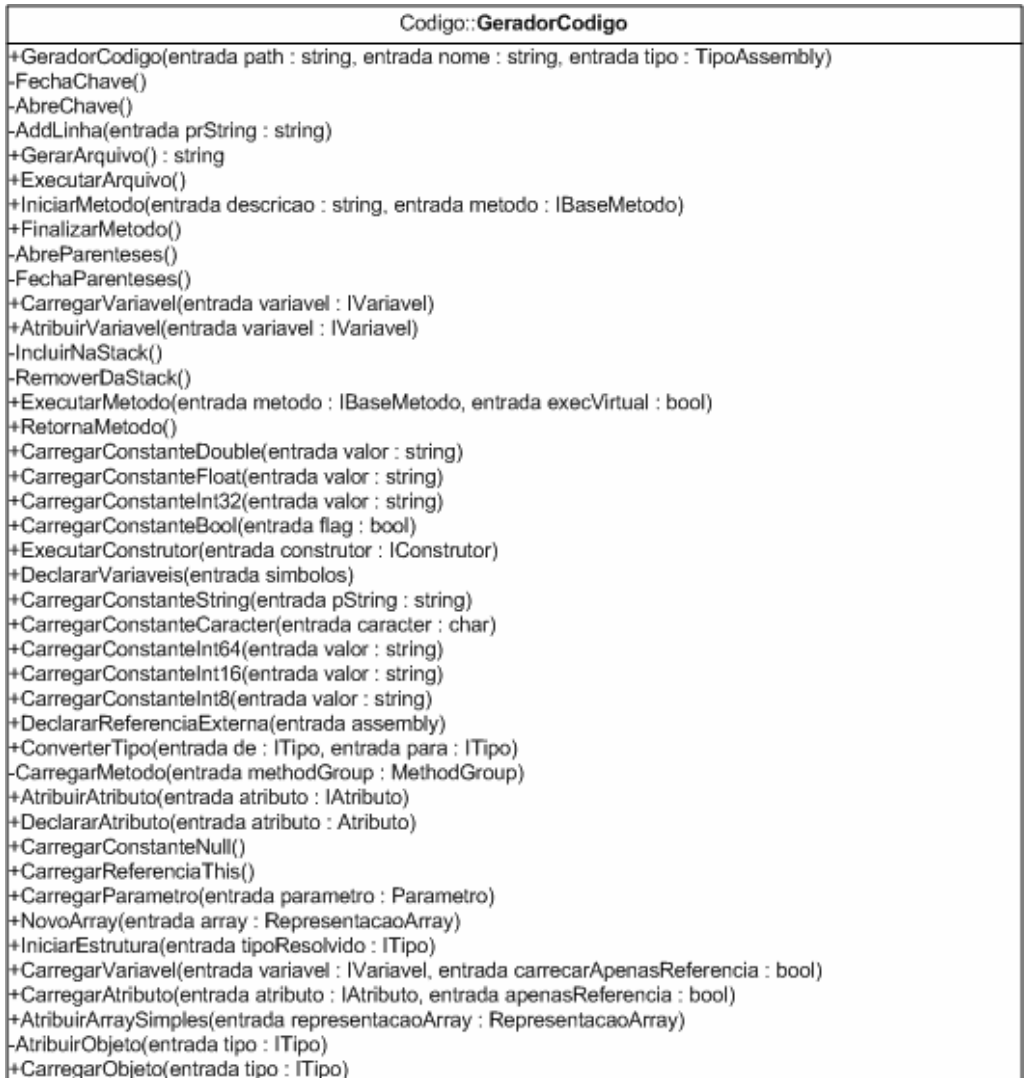
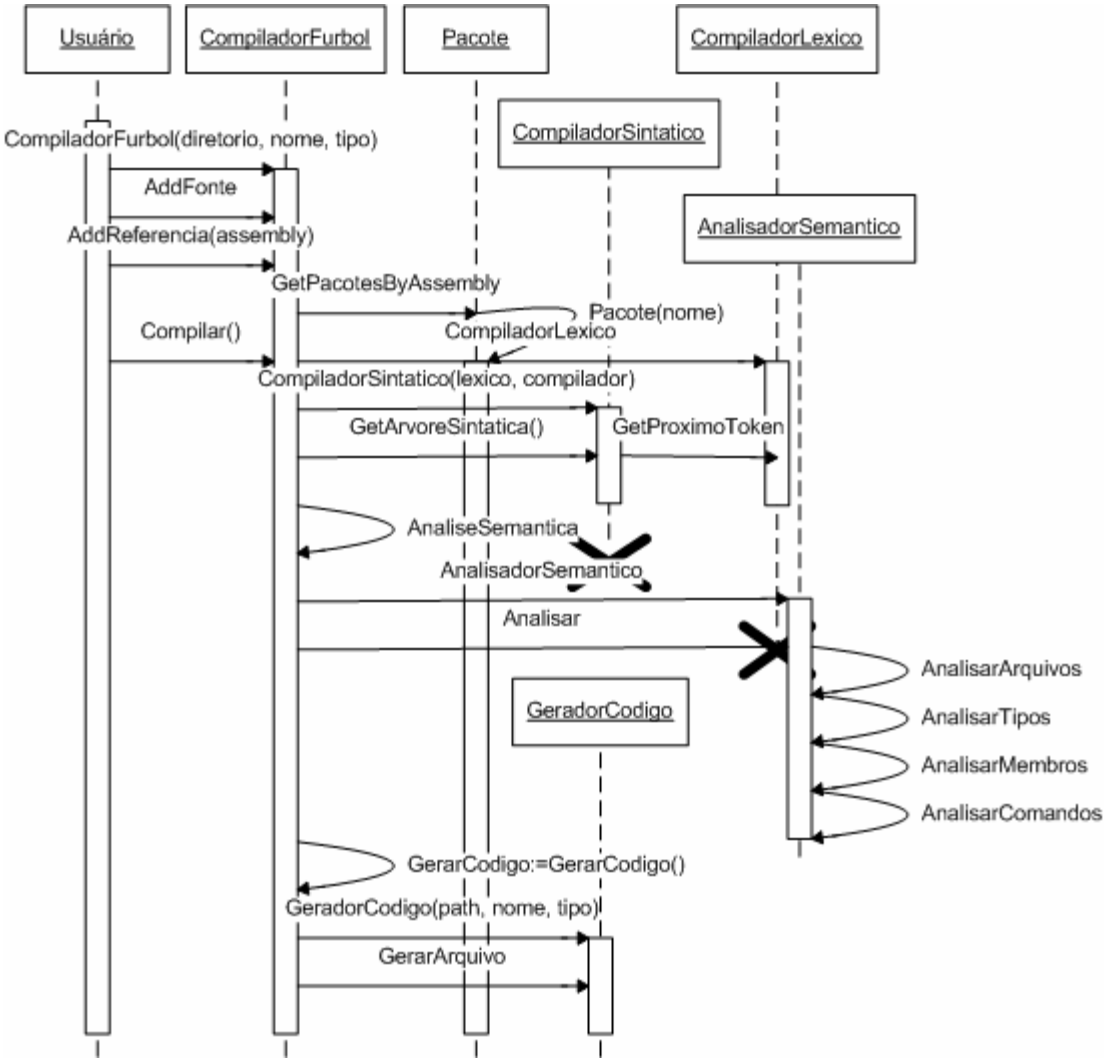


Diagrama de sequência do Compilador



Implementação

Técnicas e ferramentas utilizadas

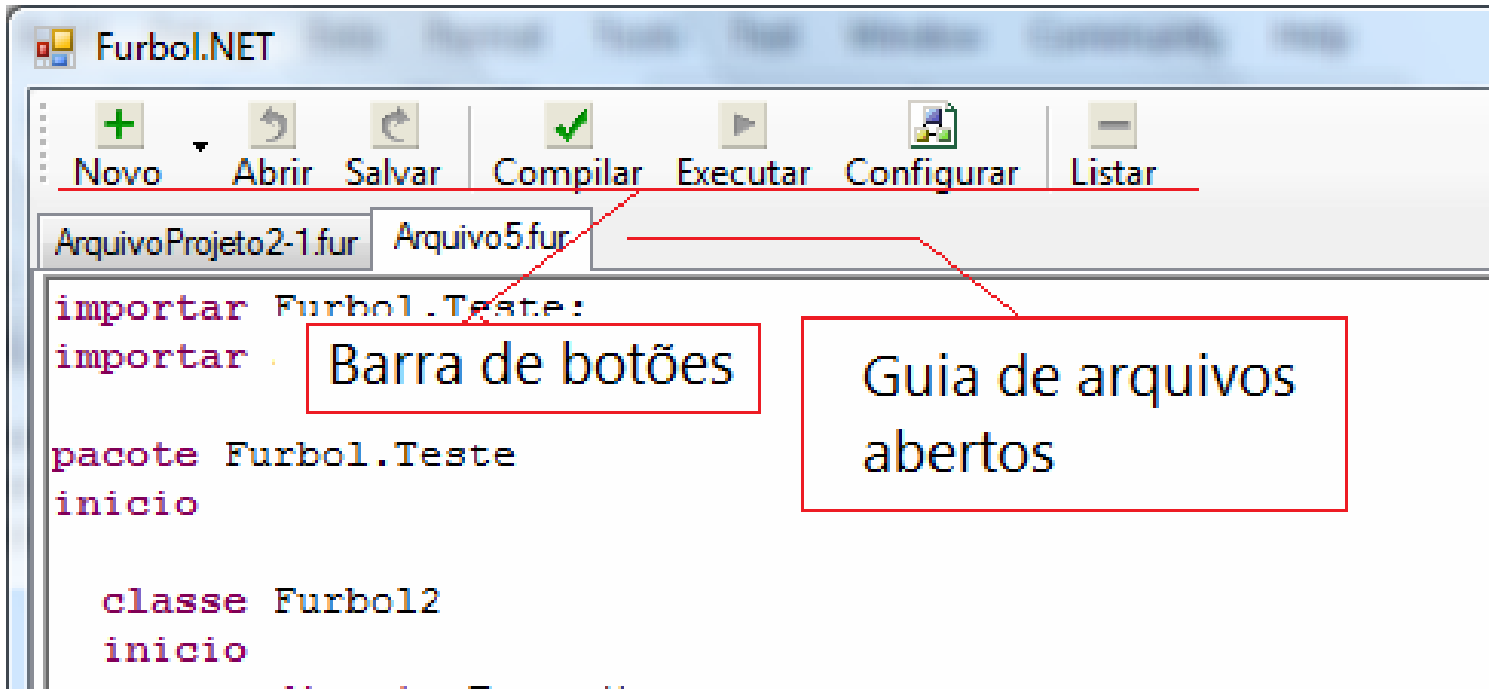
Microsoft Visual Studio 2005

- Integração com Microsoft Visio
- Utilização de diagramas não UML para manutenção de classes

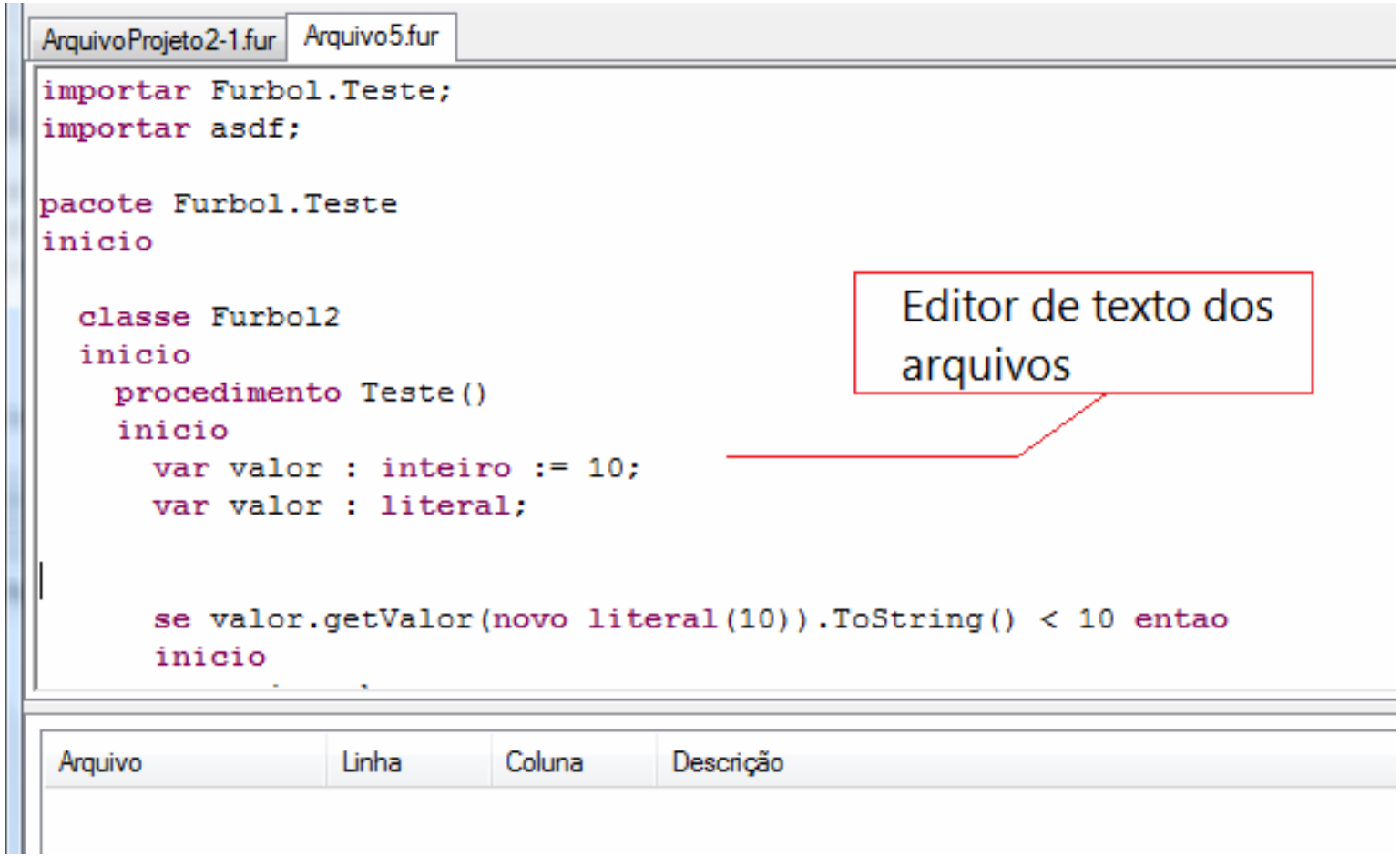
Classe Atributo >> NoArvoreSintatica

```
public override void AnaliseSemantica()
{
        base.AnaliseSemantica();

        tipoResolvido = tipo.ResolverTipo(GetTipoDeclarante());

        if (ContemModificadorDefinicao(TipoModificadorMetodo.Abstrato))
            throw new ModificadorMetodoInvalidoException(GetModificadorDefinicao(TipoModificadorMetodo.Abstrato).TokenPalavraReservada);
        if (ContemModificadorDefinicao(TipoModificadorMetodo.Sobrepor))
            throw new ModificadorMetodoInvalidoException(GetModificadorDefinicao(TipoModificadorMetodo.Sobrepor).TokenPalavraReservada);
        if (ContemModificadorDefinicao(TipoModificadorMetodo.Virtual))
            throw new ModificadorMetodoInvalidoException(GetModificadorDefinicao(TipoModificadorMetodo.Virtual).TokenPalavraReservada);
        if (ContemModificadorDefinicao(TipoModificadorMetodo.Final))
            throw new ModificadorMetodoInvalidoException(GetModificadorDefinicao(TipoModificadorMetodo.Final).TokenPalavraReservada);

        if (TipoDeclarante.EhTipoValor() && !Estatico)
            if (inicializacao != null)
                throw new AtributoEstruturaInicializadoException(Token, this, TipoDeclarante);

}
```

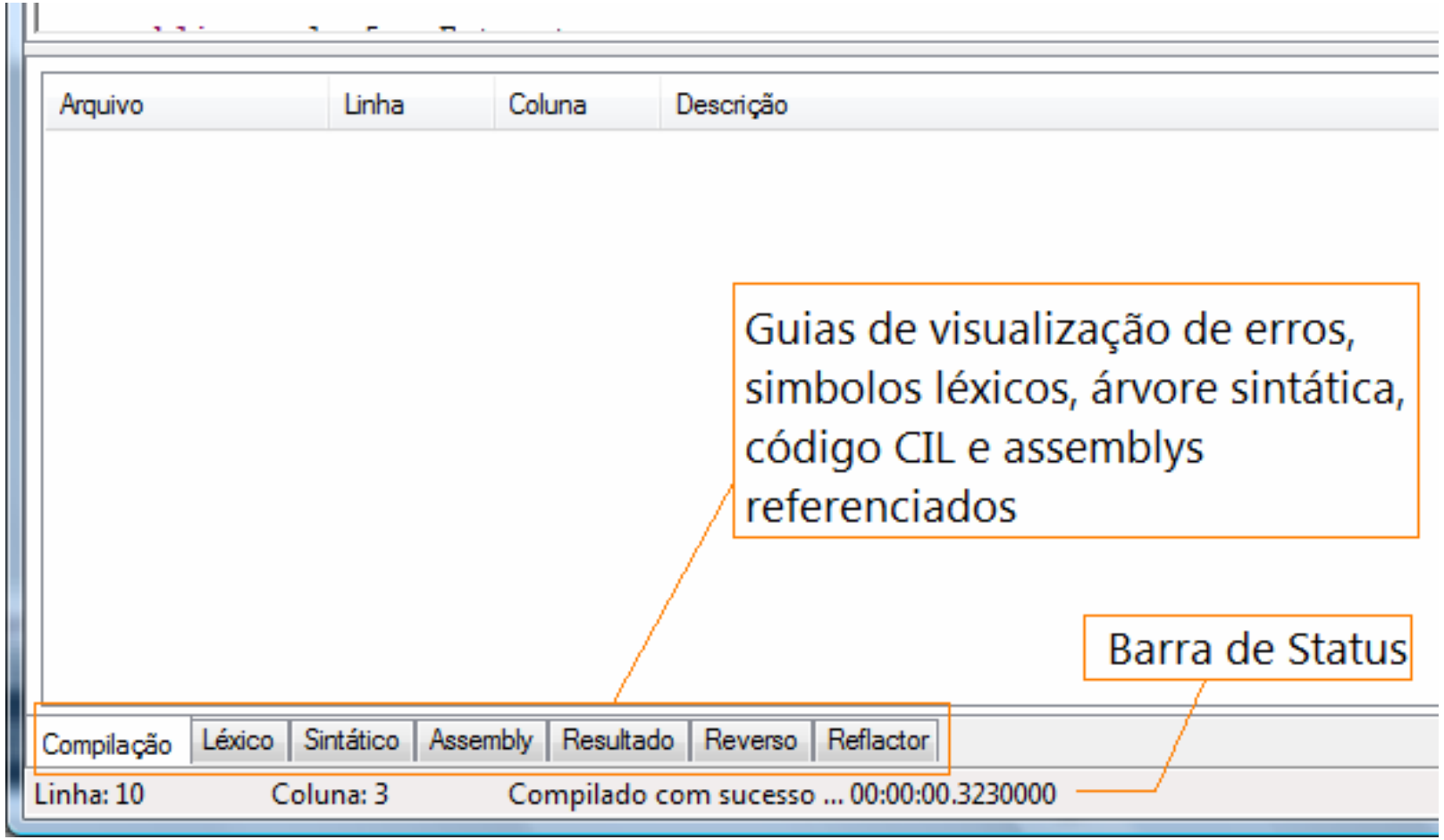
Operacionalidade da implementação



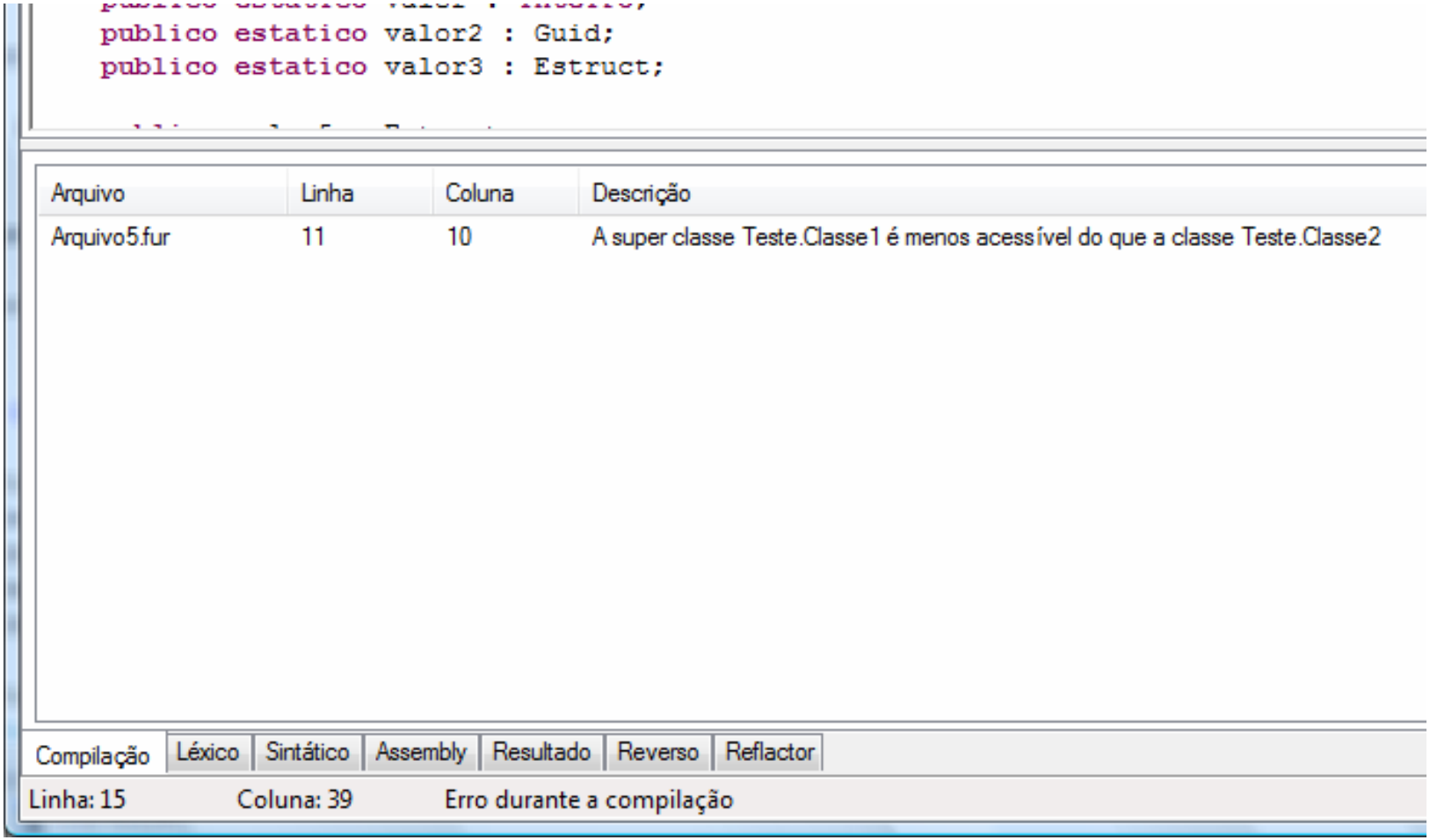
Operacionalidade da implementação



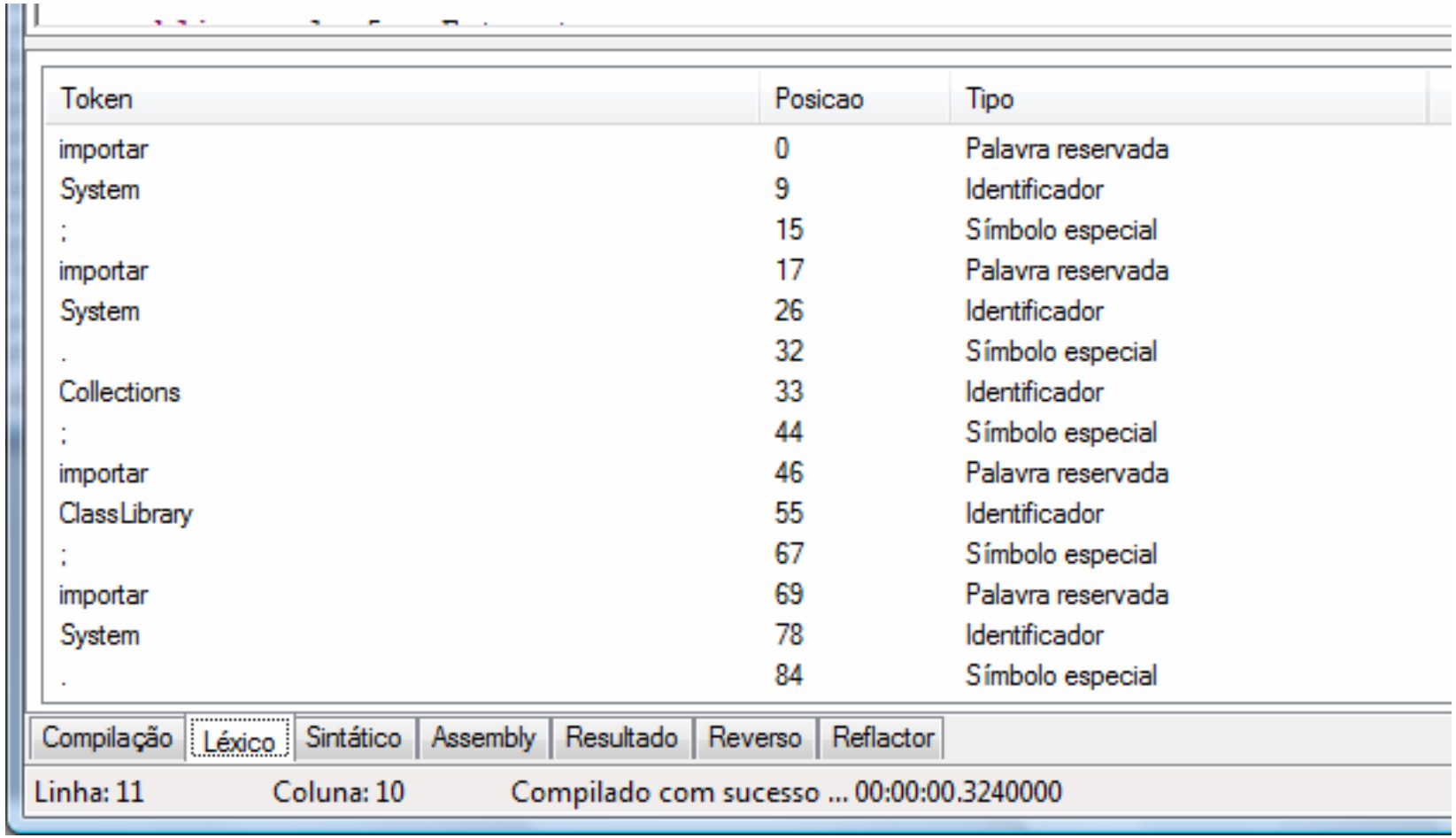
Operacionalidade da implementação



Operacionalidade da implementação

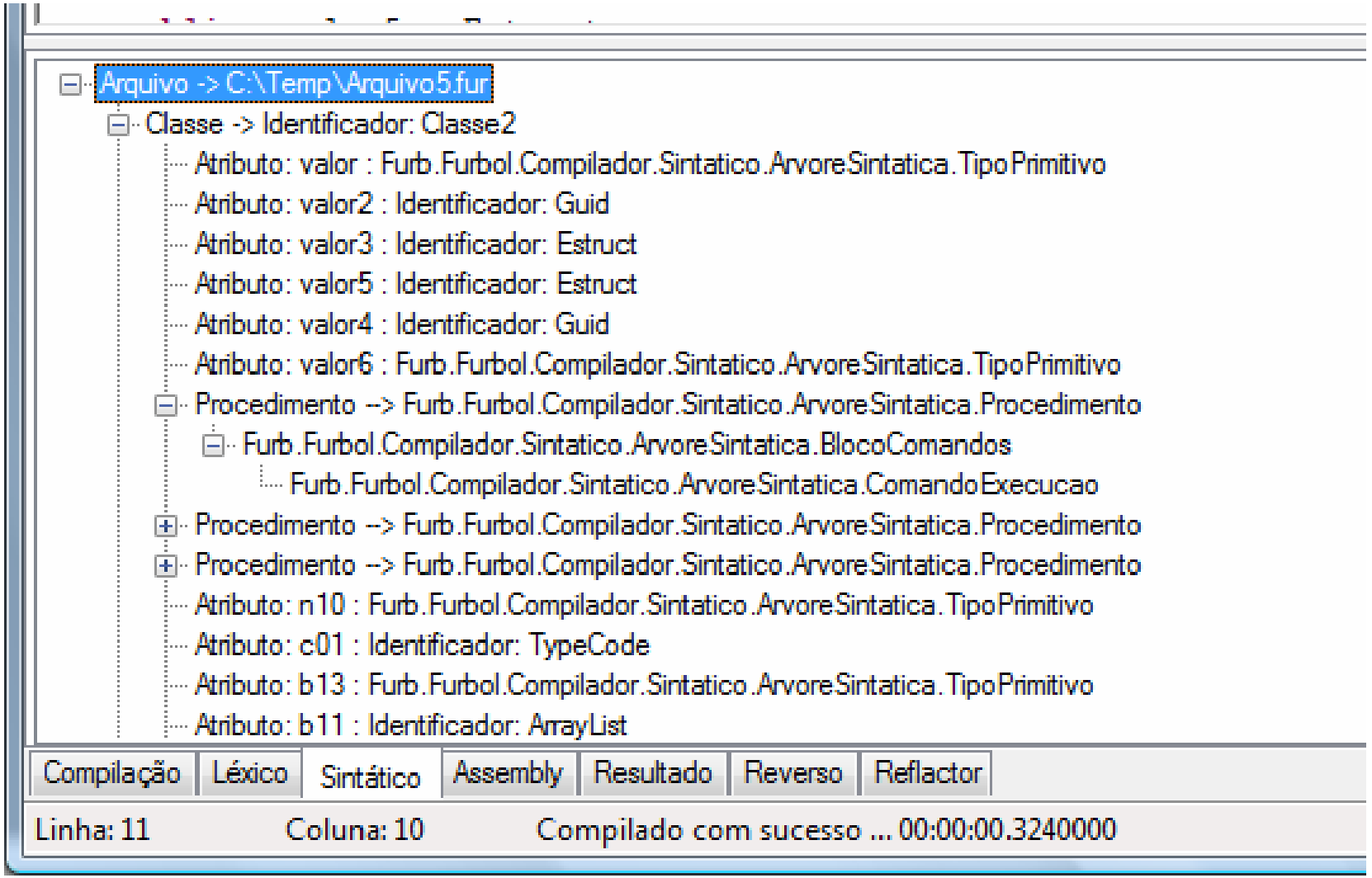


Operacionalidade da implementação

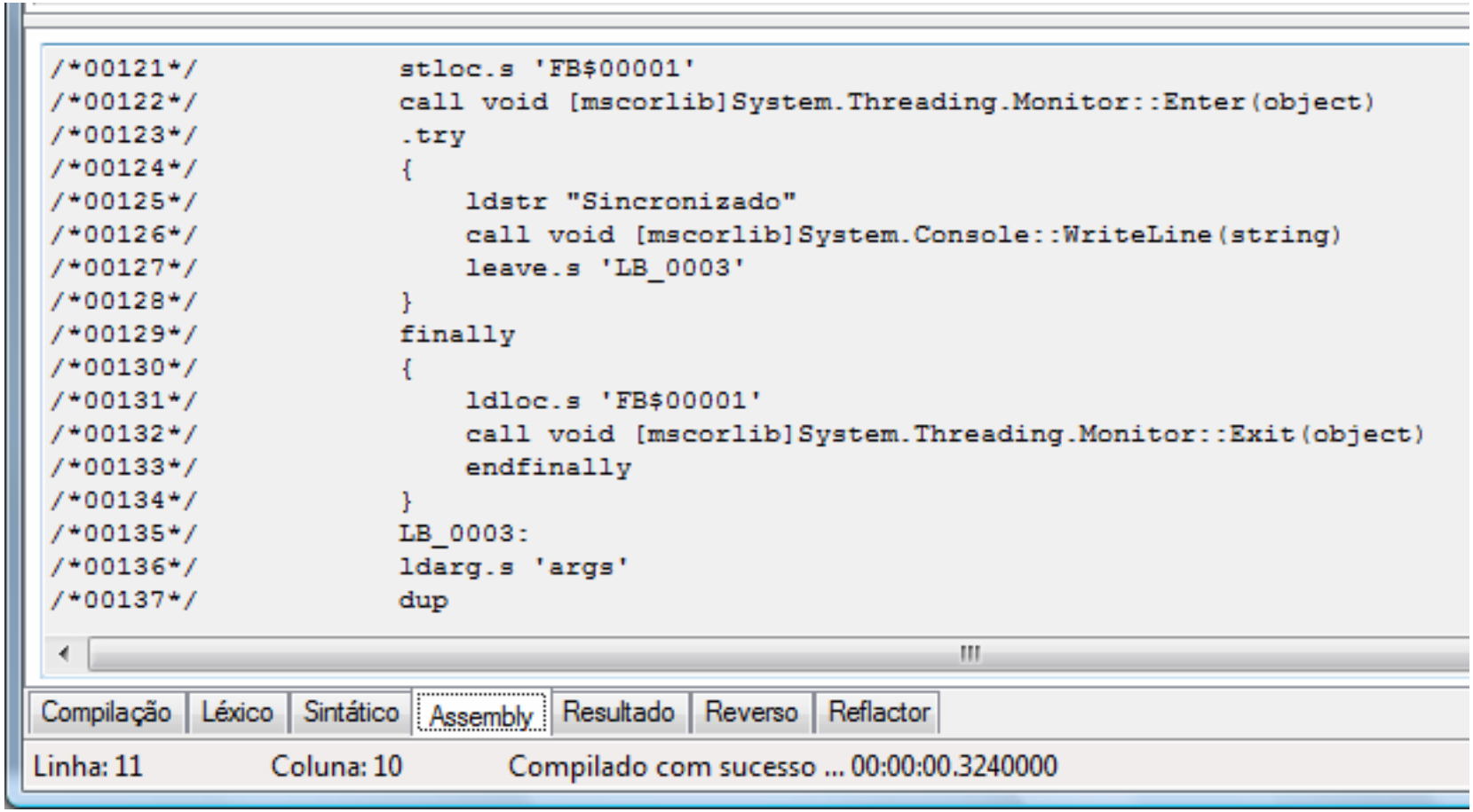


Token	Posicao	Tipo
importar	0	Palavra reservada
System	9	Identificador
;	15	Símbolo especial
importar	17	Palavra reservada
System	26	Identificador
.	32	Símbolo especial
Collections	33	Identificador
;	44	Símbolo especial
importar	46	Palavra reservada
ClassLibrary	55	Identificador
;	67	Símbolo especial
importar	69	Palavra reservada
System	78	Identificador
.	84	Símbolo especial

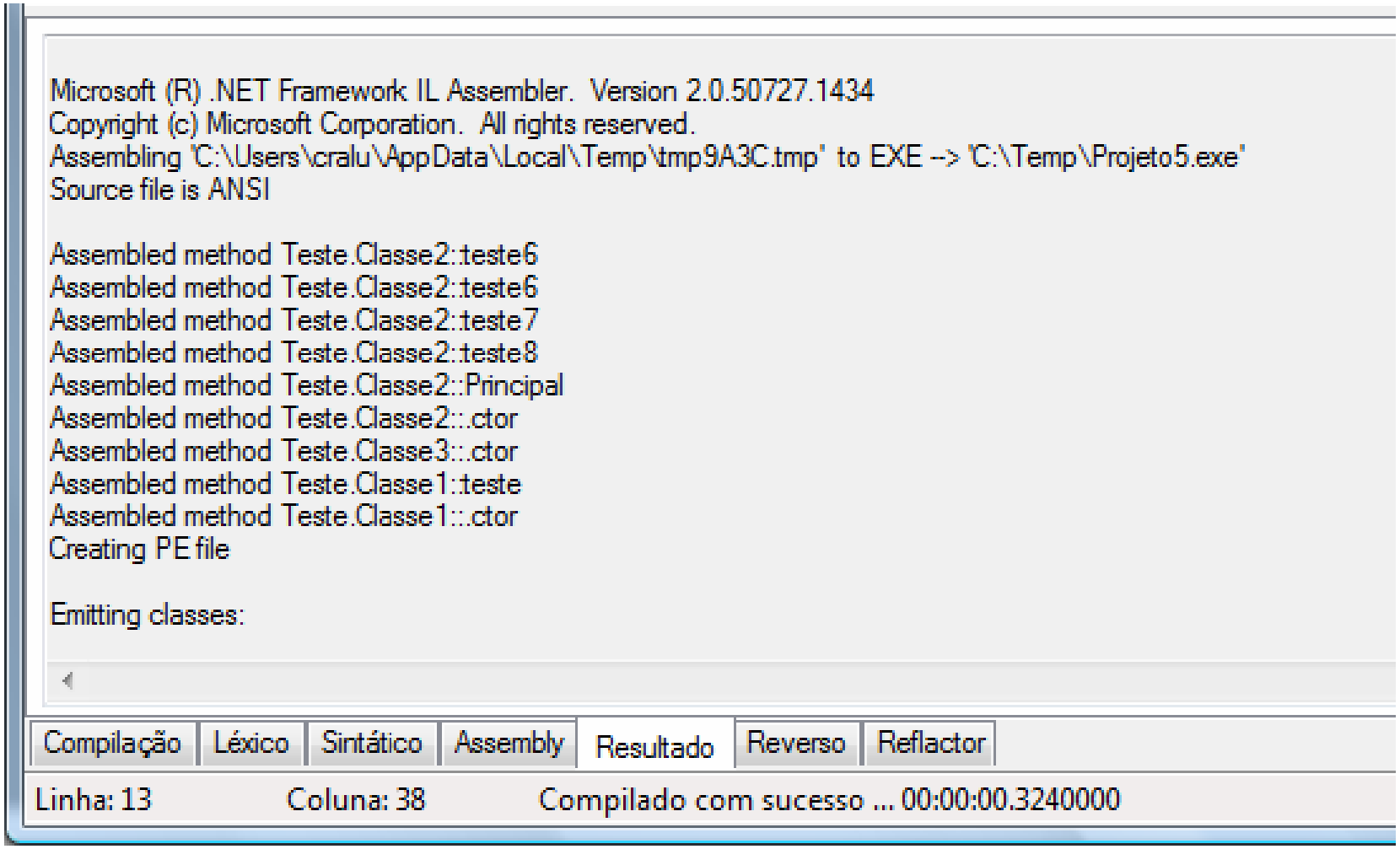
Operacionalidade da implementação



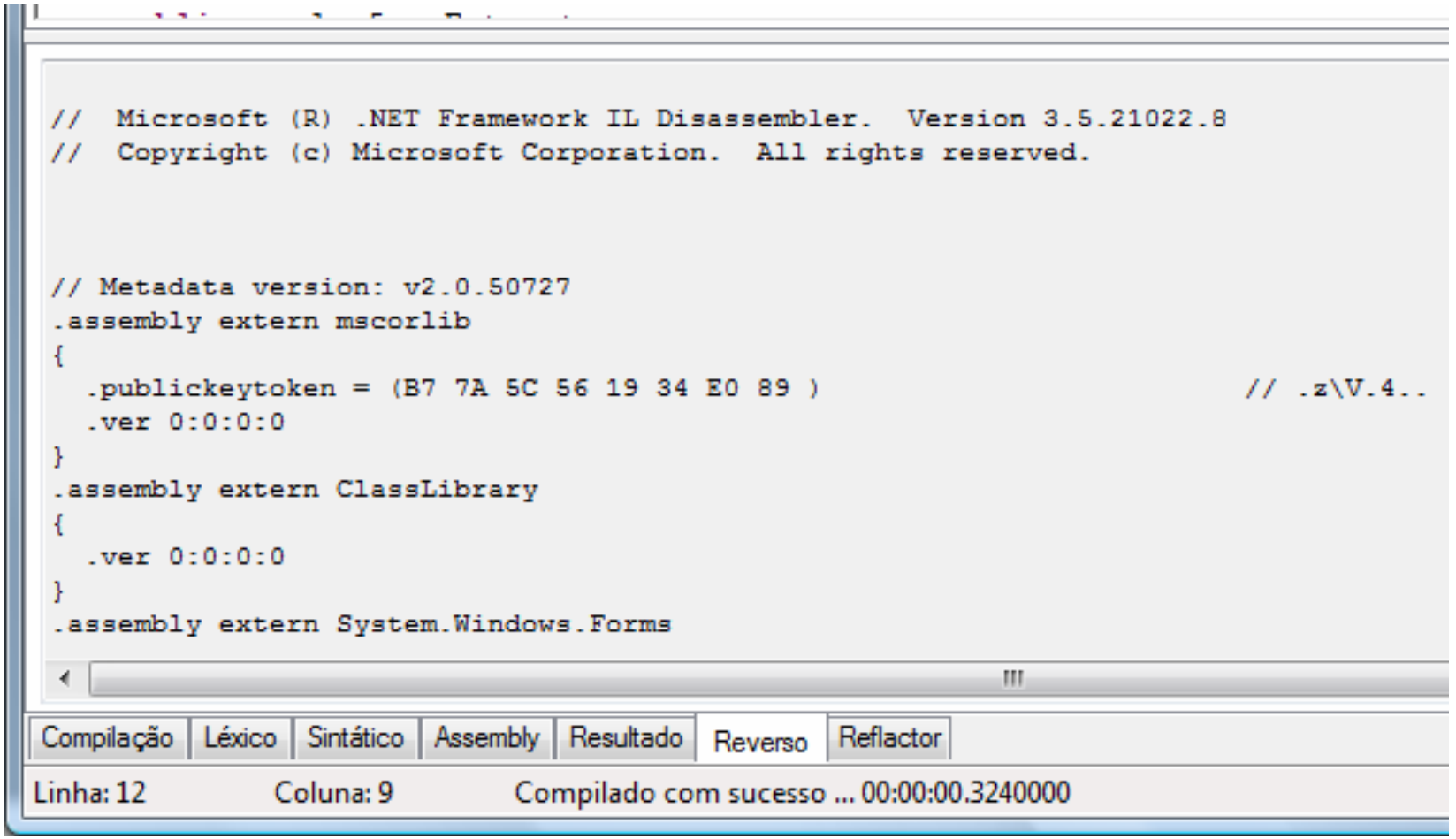
Operacionalidade da implementação



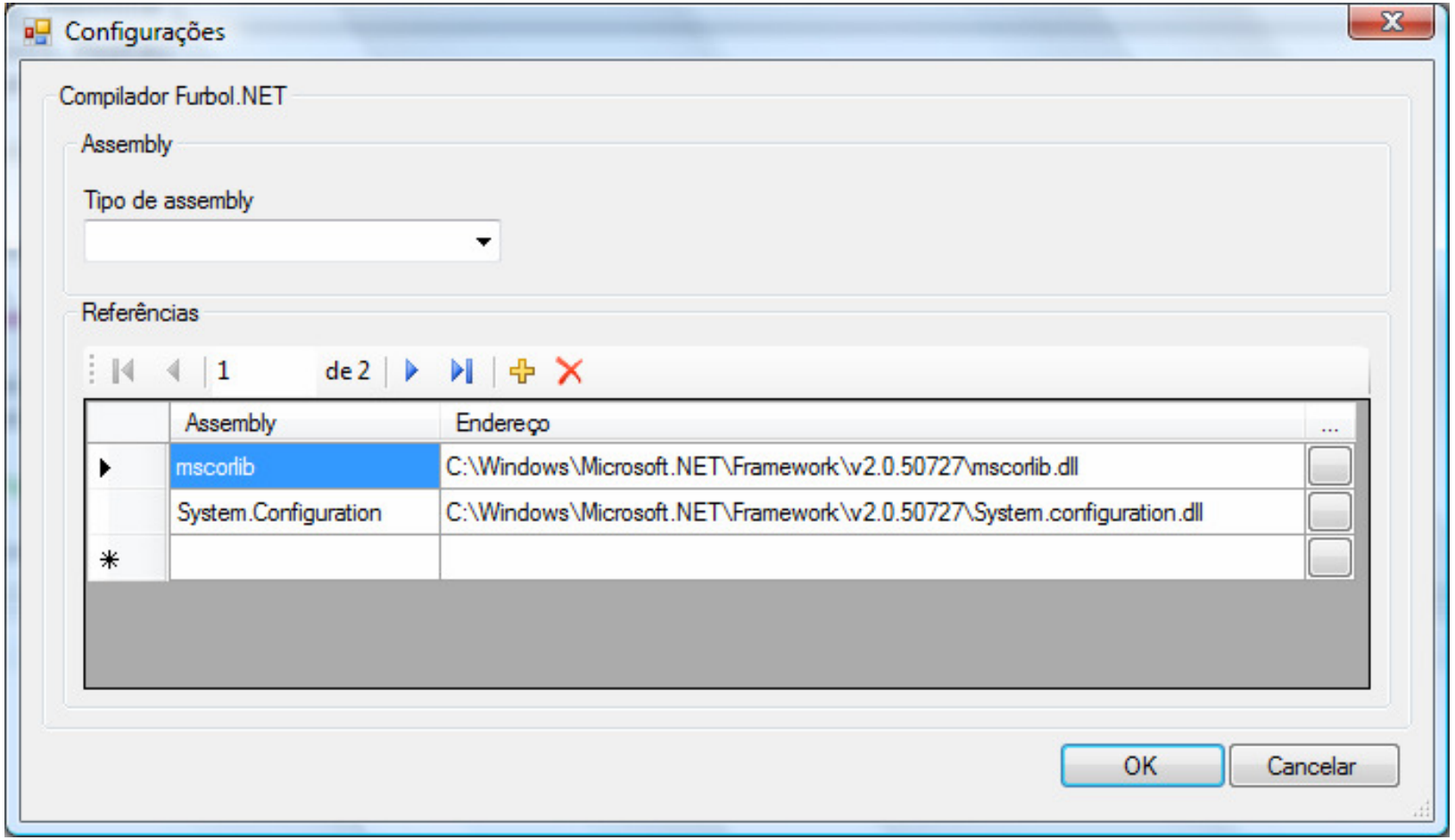
Operacionalidade da implementação



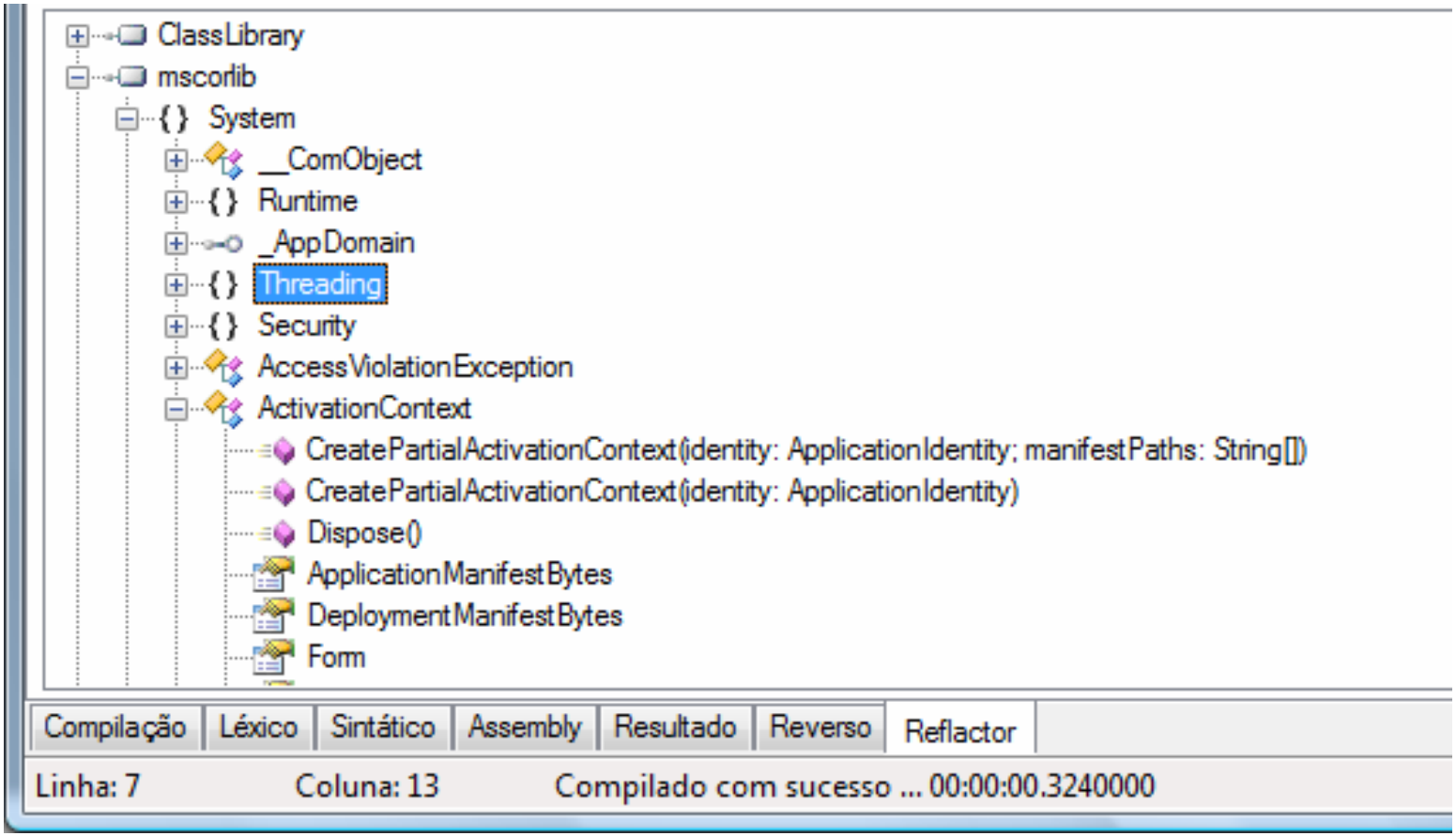
Operacionalidade da implementação



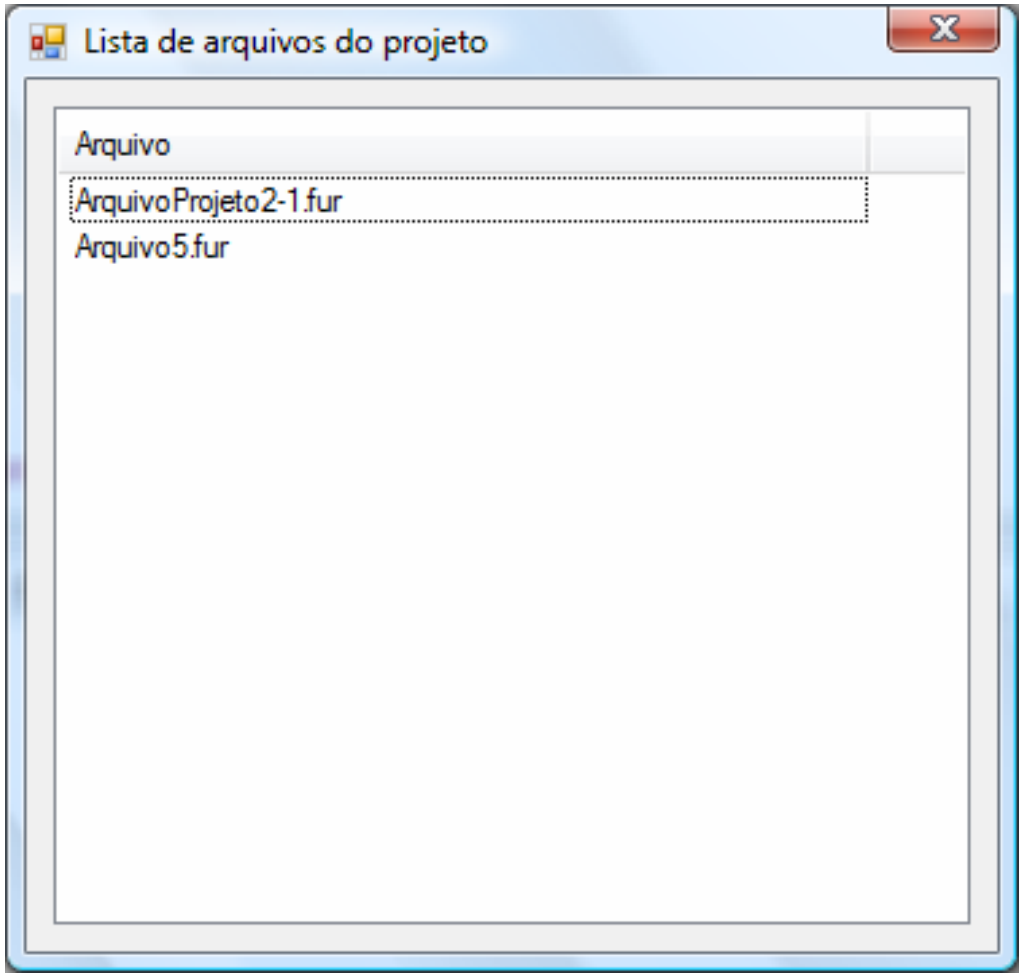
Operacionalidade da implementação



Operacionalidade da implementação



Operacionalidade da implementação



Conclusões

Conclusões

- O objetivo do trabalho foi alcançado
  - criar uma Integrated Development Environment
  - gerar código para a linguagem intermediária CIL
  - permitir a utilização da Base Class Library (BCL) e de outros assemblys escritos em linguagens .NET
  - permitir a utilização de recursos básicos da orientação a objetos, como herança e polimorfismo

- Relevância do trabalho
  - Contribuir com um trabalho de pesquisa realizado pela FURB
  - Nasce uma linguagem .NET em português com suporte a várias construções

Extensões

- Permitir a definição de delegates e eventos
- Permitir a definição de sobrecarga de operadores
- Implementar a recuperação de erros de compilação
- *Generics*: implementar *Generics*, recurso para a completa utilização da BCL do .NET versão 2.0

FIM