UNIVERSIDADE DO VALE DO ITAJAÍ
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ADAPTAÇÃO DO PORTUGOL CORE PARA PERMITIR A INTEGRAÇÃO COM OUTRAS FERRAMENTAS

por

Luiz Fernando Noschang

Itajaí (SC), junho de 2012

UNIVERSIDADE DO VALE DO ITAJAÍ
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ADAPTAÇÃO DO PORTUGOL CORE PARA PERMITIR A INTEGRAÇÃO COM OUTRAS FERRAMENTAS

Área de Informática na Educação

por

Luiz Fernando Noschang

Relatório apresentado à Banca Examinadora do Trabalho Técnico-científico de Conclusão do Curso de Ciência da Computação para análise e aprovação.
Orientador: André Luís Alice Raabe, Dr.

Itajaí (SC), junho de 2012

RESUMO

NOSCHANG, Luiz Fernando. **Adaptação do Portugol Core para integração com outras ferramentas.** Itajaí, 2012. 74. Trabalho Técnico-científico de Conclusão de Curso (Graduação em Ciência da Computação) – Centro de Ciências Tecnológicas da Terra e do Mar, Universidade do Vale do Itajaí, Itajaí, 2012.

A disciplina de Algoritmos e Programação, presente nos cursos de Ciência da Computação, Engenharia da Computação e Sistemas para Internet da UNIVALI (Universidade do Vale do Itajaí), vêm sendo fonte comum de problemas de aprendizagem. Vários destes problemas já foram discutidos em trabalhos do Grupo de Informática na Educação da UNIVALI, os quais geraram trabalhos com propostas para reduzir e atender adequadamente a estes problemas. Dentre estas contribuições estão: (i) o PortugolCore: um conjunto de ferramentas (analisador sintático, analisador semântico e interpretador) para a linguagem de programação Portugol criada na UNIVALI e; (ii) o PortugolStudio: uma IDE (Integrated Development Environment) de programação para a linguagem Portugol que utiliza as funcionalidades do PortugolCore. Estas duas ferramentas atuam em conjunto proporcionando um ambiente onde o aluno pode desenvolver e testar algoritmos para solucionar problemas em sala de aula. Ambas as ferramentas foram desenvolvidas utilizando a linguagem Java e, portanto, puderam ser integradas de forma natural, sem a necessidade de softwares intermediários (middlewares). No entanto, ainda não era possível integrar o PortugolCore com nenhum software escrito em outras linguagens de programação. Desta forma, buscando aproveitar o potencial oferecido pelo PortugolCore e também disponibilizá-lo a outros grupos de pesquisa, foi realizada uma pesquisa para identificar as possibilidades de integração do mesmo com ferramentas escritas em outras linguagens. Durante o desenvolvimento do trabalho foi realizada uma pesquisa a fim de identificar as técnicas/tecnologias existentes para a integração entre sistemas. Ao final da pesquisa, decidiu-se utilizar a tecnologia CORBA por se tratar de uma tecnologia madura e que atendia a todos os requisitos de integração especificados. Utilizando o CORBA, foi criado um módulo de integração do PortugolCore com a linguagem C#, o qual foi validado através de um estudo de caso realizado com o software BIPIDE, desenvolvido na UNIVALI. Nos testes realizados durante o estudo de caso, foi possível acessar e utilizar no BIPIDE, todos os serviços disponibilizados pelo PortugolCore, cumprindo assim o objetivo geral deste trabalho. O estudo de caso está documentado no capítulo 3.3 ESTUDO DE CASO – FERRAMENTA BIPIDE e apresenta a metodologia utilizada, bem como as dificuldades encontradas e os resultados obtidos. Para permitir a difusão e utilização do PortugolCore, foram criados três artefatos de documentação. O primeiro é voltado aos alunos e professores, está em formato HTML (HyperText Markup Language) e aborda toda a sintaxe da linguagem Portugol 2.0. O segundo é voltado aos desenvolvedores e colaboradores do projeto, está em formato Javadoc e documenta as classes e pacotes do PortugolCore. O terceiro é voltado aos utilizadores, está em formato PDF e documenta os passos necessários para a integração com a linguagem C# utilizando o módulo desenvolvido. Estes documentos estão disponíveis juntamente com o código fonte em um repositório de projetos OpenSource no endereço: https://github.com/UNIVALI-L2S/Portugol. O projeto também possui uma página de apresentação (em fase inicial de construção) no endereço: http://univali-l2s.github.com/Portugol.

Palavras-chave: Portugol. Integração. CORBA.

ABSTRACT

The discipline of Algorithms and Programming, present in the courses of Computer Science, Computer Engineering and Systems for Internet at UNIVALI (Universidade do Vale do Itajaí) have been common source of learning problems. Several of these problems were discussed by the Computers in Education Group at UNIVALI, which generated jobs with proposals to reduce and adequately address these problems. Among these contributions are: (i) PortugolCore: a set of tools (parser, semantic analyzer and interpreter) for the programming language Portugol created at UNIVALI and; (ii) PortugolStudio: an IDE (Integrated Development Environment) for the Portugol language that uses the features of PortugolCore. These tools work together providing an environment where students can develop and test algorithms to solve problems in the classroom. Both tools were developed using the Java language and therefore could be integrated in a natural way, without the need for intermediate software (Middleware). However, it was not yet possible to integrate PortugolCore with any software written in other programming languages. Thus, aiming to take advantage of the potential offered by PortugolCore and also make it available to other research groups, a research was made in order to identify the possibilities of its integration with tools written in other languages. During the development of this work a research was performed in order to identify the existing techniques and technologies for the systems integration. At the end of the study, it was decided to use CORBA technology because it was a mature technology and met all the integration requirements specified. Using CORBA, an integration module was created for PortugolCore, allowing it to interoperate with C#. This module was validated through a case study conducted with BIPIDE software, developed in UNIVALI. In tests conducted during the case study, it was possible to access and use in BIPIDE, all services provided by PortugolCore, thus fulfilling the main goal of this work. The case study is documented in Chapter 3.3 ESTUDO DE CASO – FERRAMENTA BIPIDE and presents the methodology used, the difficulties encountered and the results obtained. To allow the dissemination and use of PortugolCore, three documentation artifacts were created. The first is aimed at students and teachers, it is in HTML (HyperText Markup Language) format and covers all the Portugol 2.0 language syntax. The second is aimed at the project's developers and contributors, it is in Javadoc format and documents all classes and packages within PortugolCore. The third is aimed at users, it is in PDF format and documents the steps required for integration with C # using the module developed. These documents are available along with the source code into a repository of open source projects at: https://github.com/UNIVALI-L2S/Portugol. The project also includes a presentation page (in the initial phase of construction) at: http://univali-l2s.github.com/Portugol.

***Keywords**: Portugol. Integration. CORBA.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE EQUAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ANTLR	Another Tool for Language Recognition
API	Application Programming Interface
ASA	Árvore Sintática Abstrata
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
HTTP	HiperText Transfer Protocol
HTML	HiperText Markup Language
IDE	Integrated Development Environment
IDL	Interface Definition Language
IOR	Interoperable Object Reference
JDK	Java Development Kit
JNI	Java Native Interface
JRE	Java Runtime Environment
JVM	Java Virtual Machine
MSDN	Microsoft Developer Network
OMG	Object Management Group
ORB	Object Request Broker
OSI	Open Source Initiative
REST	REpresentational State Transfer
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
TCC	Trabalho de Conclusão de Curso
TI	Tecnologia da Informação
UNIVALI	Universidade do Vale do Itajaí
URL	Uniform Resource Locator
XML	Extensible Markup Language
W3C	World Wide Web Consortium
WSDL	WebService Description Language

“Fracassos? Não sei do que falas, em cada experiência descubro um dos motivos pelo qual a lâmpada não funciona. Agora sei mais de mil maneiras de como não fazer a lâmpada”. Thomas A. Edison.

SUMÁRIO

1. INTRODUÇÃO

Atualmente na UNIVALI (Universidade do Vale do Itajaí), dentre os vários cursos oferecidos, existem três cursos relacionados à área de TI (Tecnologia da Informação): Ciência da Computação, Engenharia de Computação e Sistemas para Internet.

Embora cada um destes cursos ofereça formação em um ramo/segmento mais específico da TI, existem algumas disciplinas que são comuns entre eles, pois abordam temas considerados básicos ou essenciais para a formação acadêmica nesta área.

Dentre estas disciplinas está a disciplina de Algoritmos e Programação, a qual é ministrada nos primeiros semestres dos três cursos.

A disciplina é fonte comum de problemas de aprendizagem, já discutidos em diversos trabalhos na área da computação (RAABE e SILVA, 2005). Neste contexto, o grupo de pesquisa em Informática na Educação da UNIVALI, vem buscando encontrar alternativas para reduzir os problemas de aprendizagem dos alunos e consequentemente melhorar a qualidade da mesma.

Deste esforço de pesquisa foram criadas várias ferramentas, dentre elas o WebPortugol que é um ambiente Internet desenvolvido por Hostins e Raabe (2007), cujo objetivo é auxiliar no desenvolvimento de algoritmos em pseudolinguagem (português estruturado).

Ele foi desenvolvido como uma solução própria, ao invés de adotar uma existente, para colocar a prova uma ideia simples explorada por Miranda (2004) e que possibilita ao aluno realizar a verificação de conformidade do algoritmo desenvolvido com base em valores pré-estabelecidos de entrada e saída. O WebPortugol foi criado em 2007 e vem sendo utilizado desde então na disciplina para auxiliar no aprendizado dos alunos.

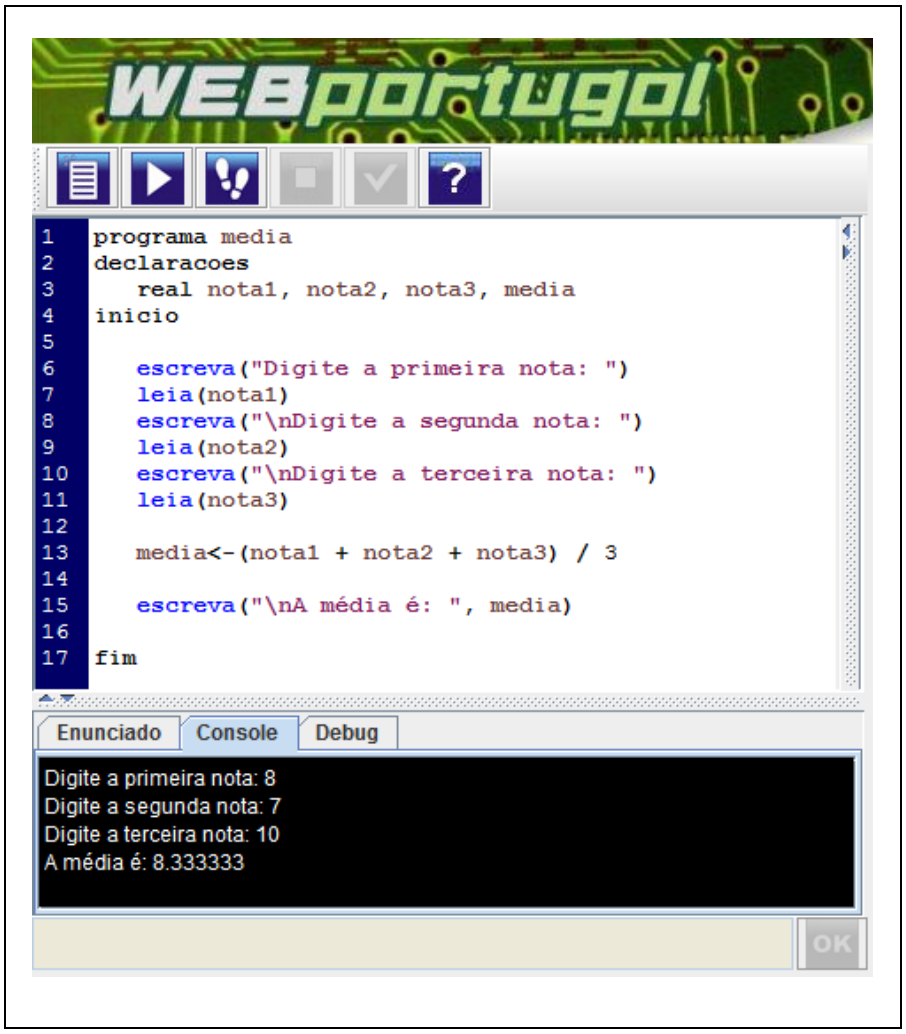

Figura 1. Ferramenta WebPortugol

Durante o primeiro semestre de 2009, um novo projeto foi iniciado tendo como base o WebPortugol. A proposta deste projeto era alterar a sintaxe utilizada pelo ambiente WebPortugol (denominada Portugol 1.1) tornando-a mais próxima à sintaxe das linguagens C e PHP. O objetivo era reduzir o impacto sentido pelos alunos na transição do Portugol para estas linguagens.

A nova sintaxe foi criada e batizada de Portugol 2.0 e, em decorrência das mudanças na sintaxe, optou-se por descontinuar o WebPortugol e construir um novo ambiente para dar suporte à linguagem. Desta forma surgiram duas novas ferramentas: o PortugolCore e o PortugolStudio.

O PortugolCore compreende o núcleo da linguagem Portugol 2.0 e está internamente dividido em três partes: analisador sintático, analisador semântico e interpretador. Uma das

principais vantagens deste núcleo é o fato de ele ser totalmente independente do ambiente de desenvolvimento, permitindo a sua utilização com outras ferramentas desenvolvidas em Java.

O PortugolStudio, por sua vez, constitui o ambiente de desenvolvimento construído para permitir a criação e a execução dos programas escritos em Portugol 2.0. O PortugolStudio possui todos os recursos básicos de uma IDE (Integrated Development Environment): manipulação de arquivos de código-fonte (abrir, salvar, desfazer, etc.), execução e interrupção de programas, console para entrada e saída de dado, console para exibição dos erros de compilação e de execução, Syntax Highlight e Code Completion (em fase experimental).

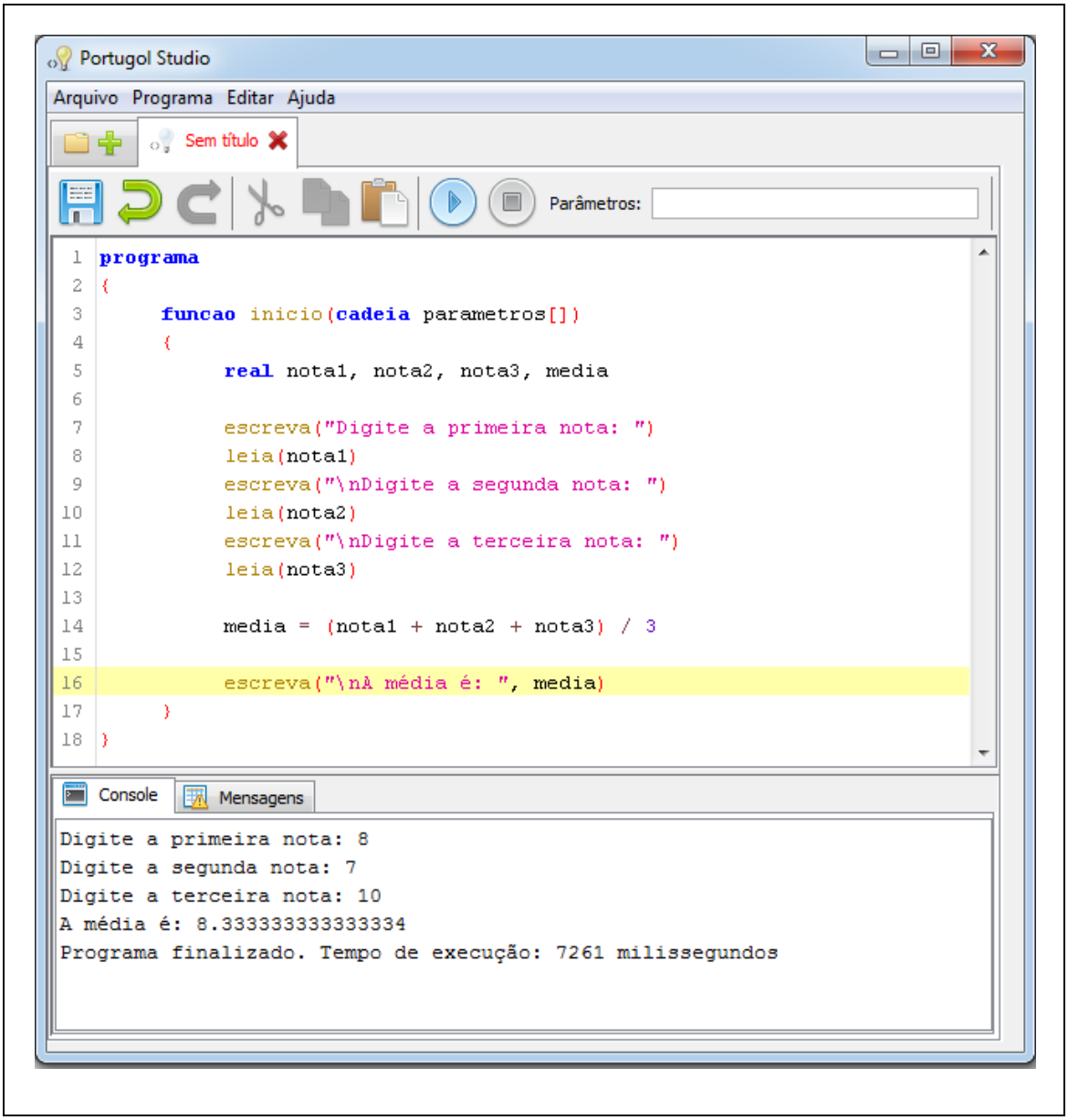

Figura 2. Ferramenta PortugolStudio

1.1 PROBLEMATIZAÇÃO

1.1.1 Formulação do Problema

A forma como o PortugolCore foi desenvolvido permite que ele seja facilmente integrado com outras ferramentas escritas em Java. No entanto ainda é impossível integrá-lo a ferramentas que tenham sido desenvolvidas em outras linguagens de programação. Esta limitação diminui o potencial do PortugolCore em contribuir com outros trabalhos de pesquisa.

Este é o caso, por exemplo, da ferramenta Bipide (VIEIRA, ZEFERINO e RAABE, 2009) desenvolvida na UNIVALI utilizando a linguagem C# e na qual existe o interesse do grupo de pesquisa em permitir que a ferramenta reconheça a sintaxe do Portugol 2.0 utilizando as funcionalidades disponibilizadas pelo PortugolCore.

Além disto, o PortugolCore não possui uma documentação que detalhe sua organização interna e as funcionalidades que ele oferece, dificultando desta forma, até mesmo a integração com outras aplicações escritas em Java.

Por fim, as falhas (bugs) ainda existentes no PortugolCore, tanto na execução de programas quanto na detecção de erros sintáticos e semânticos, ainda inviabilizam o seu uso efetivo.

1.1.2 Solução Proposta

Primeiramente serão corrigidas o máximo possível de falhas existentes no PortugolCore de modo a obter-se uma versão estável do mesmo.

Logo após, ele será aprimorado para permitir a integração com outras linguagens. Este aprimoramento será realizado utilizando uma das tecnologias/técnicas a serem pesquisadas no decorrer do trabalho.

Por último será gerada uma documentação detalhando os aspectos discutidos anteriormente e a mesma será disponibilizada juntamente com o código-fonte à comunidade científica através de um repositório de projetos OpenSource.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Adaptar o PortugolCore provendo mecanismos de integração com ferramentas desenvolvidas em outras linguagens de programação.

1.2.2 Objetivos Específicos

- Concluir a implementação do PortugolCore, corrigindo as falhas encontradas;
- Pesquisar os métodos de integração entre a linguagem Java e outras linguagens;
- Realizar estudos de caso que evidenciem as possibilidades de integração com outras ferramentas;
- Documentar o PortugolCore; e
- Disponibilizar o PortugolCore e sua documentação em um servidor de projetos OpenSource.

1.3 METODOLOGIA

Durante deste trabalho foi realizada uma pesquisa bibliográfica buscando identificar as principais tecnologias existentes para a integração entre sistemas. A pesquisa foi realizada através da leitura de livros relacionados ao tema e através de buscas na Internet, na qual foram analisados artigos científicos e trabalhos similares também relacionados ao tema.

As tecnologias pesquisadas foram analisadas com intuito de identificar a melhor alternativa de integração para ser utilizada com o PortugolCore. A escolha foi feita analisando para cada tecnologia pesquisada, se a mesma atendia os requisitos de integração exigidos pelo PortugolCore.

A tecnologia escolhida para o projeto do sistema foi a arquitetura CORBA, por atender a todos os requisitos de integração estabelecidos (Interoperabilidade, Reusabilidade, Licença e Documentação). O projeto do sistema foi realizado utilizando descrição textual e figuras ilustrativas.

Logo após, seguindo o modelo do projeto desenvolvido, foi realizada a implementação de um módulo de integração com a linguagem C#. O módulo foi criado utilizando a biblioteca IIOP.NET, uma implementação CORBA para C# com licença livre.

Para validar o módulo de integração criado, foi realizado um estudo de caso utilizando o software BIPIDE, desenvolvido na UNIVALI. No estudo de caso, cada um dos serviços disponibilizados pelo PortugolCore foi chamado pelo BIPIDE através do módulo criado e os resultados da execução dos serviços foram apresentados nas telas do software.

Por fim, após validada a implementação, foi elaborada uma documentação para permitir a difusão e utilização do PortugolCore e do módulo desenvolvido:

1. Documentação da linguagem Portugol 2.0: esta documentação foi criada com foco nos alunos e professores. Nela foram abordados todos os aspectos da linguagem Portugol 2.0, tais como: diferenças com o Portugol 1.1, sintaxe, tipos de dados, declarações de variáveis, laços de repetição, desvios condicionais, funções, compatibilidade entre tipos de dados e exemplos. Esta documentação foi desenvolvida no formato HTML;

2. Documentação do código fonte do PortugolCore: esta documentação foi criada com foco nos desenvolvedores e colaboradores do projeto. Nela foi abordada toda a organização interna do PortugolCore, especificando o propósito e o funcionamento de cada classe e pacote do código fonte através de textos descritivos e exemplos. Esta documentação foi desenvolvida diretamente no código fonte do PortugolCore no formato Javadoc (podendo ser exportada em HTML); e

3. Documentação do módulo de integração para C#: esta documentação foi criada com foco nos utilizadores do PortugolCore. Consiste em um passo a passo demonstrando como instalar e utilizar o módulo desenvolvido em uma aplicação C#. Esta documentação foi desenvolvida em formato DOC (documento do Word) e exportada para PDF.

1.4 ESTRUTURA DO TRABALHO

Este documento está estruturado em quatro capítulos. O Capítulo 1, Introdução, apresentou uma visão geral do trabalho.

O Capítulo 2, Fundamentação Teórica, na primeira parte, apresenta uma visão geral das funcionalidades do PortugolCore, juntamente com um exemplo de integração em Java. Na segunda parte, é feito um levantamento dos requisitos de integração que se espera obter das tecnologias a serem pesquisadas. Na terceira parte é realizada a pesquisa bibliográfica de algumas tecnologias de integração entre sistemas, descrevendo suas principais características. Por último, é feita uma análise comparativa entre estas tecnologias, aonde uma delas é escolhida para ser utilizada no projeto.

O Capítulo 3, Desenvolvimento, na primeira parte, apresenta uma pesquisa mais aprofundada da arquitetura CORBA, a qual foi realizada com o intuito de obter o conhecimento necessário para a modelagem do projeto. A segunda parte apresenta o projeto do mecanismo de integração desenvolvido. A terceira parte apresenta a metodologia utilizada na implementação do projeto. Por fim, é feito um estudo de caso para validar a implementação realizada, no qual são apresentados os testes executados, os problemas encontrados e os resultados obtidos.

Concluindo, no Capítulo 4, apresentam-se as conclusões, onde é discutida a metodologia empregada; as soluções utilizadas; as técnicas e ferramentas aplicadas; os problemas encontrados; os resultados obtidos e as contribuições do trabalho; e os projetos futuros. O texto ainda inclui dois apêndices que complementam as informações apresentadas.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os temas relevantes a este trabalho. Na primeira parte é apresentada uma visão geral das funcionalidades do PortugolCore, juntamente com um exemplo de integração em Java. Na segunda parte, é feito um levantamento dos requisitos de integração que se espera obter das tecnologias a serem pesquisadas. Na terceira parte, é realizada a pesquisa bibliográfica de algumas tecnologias de integração entre sistemas, descrevendo suas principais características. Por último, é feita uma análise comparativa entre estas tecnologias, aonde uma delas é escolhida para ser utilizada no projeto.

2.1 VISÃO GERAL DO PORTUGOL CORE

O PortugolCore surgiu durante o primeiro semestre de 2009 a partir de um projeto de pesquisa na UNIVALI cuja proposta era alterar a sintaxe da linguagem utilizada pelo ambiente WebPortugol (denominada Portugol 1.1) tornando-a mais próxima à sintaxe das linguagens C e PHP. O objetivo era reduzir o impacto sentido pelos alunos na transição do Portugol para estas linguagens.

A nova sintaxe foi criada e batizada de Portugol 2.0 e em decorrência das mudanças na sintaxe, optou-se por descontinuar o WebPortugol e construir um novo ambiente para dar suporte à linguagem. Desta forma surgiu o PortugolCore.

O PortugolCore constitui o núcleo do Portugol 2.0 e foi construído utilizando técnicas tradicionais de construção de linguagens de programação que incluem especificação de expressões regulares, gramáticas livres de contexto e ações semânticas. Para facilitar seu desenvolvimento foi utilizada uma ferramenta para geração de compiladores chamada ANTLR (Another Tool for Language Recognition). Maiores detalhes sobre o desenvolvimento do núcleo serão produzidos na documentação do PortugolCore que está prevista para a segunda etapa deste trabalho, por este motivo serão descritas a seguir apenas as informações importantes para que se possa avaliar os mecanismos de integração existentes.

No momento de sua concepção, decidiu-se que o PortugolCore deveria ser desenvolvido como um projeto independente mas de modo que pudesse ser integrado a diferentes ambientes de desenvolvimento (IDEs). Tal decisão foi tomada tendo em vista os

diferentes projetos em desenvolvimento na UNIVALI e que poderiam se aproveitar desta integração.

Assim sendo, o PortugolCore disponibiliza três funcionalidades básicas: análise e tratamento de erros, geração de código intermediário e execução de programas, as quais são expostas através de um conjunto de classes que abstraem os detalhes de implementação.

Para integrar o PortugolCore a uma aplicação Java, primeiramente deve-se incluir o código compilado (arquivo JAR) no Classpath da aplicação. Uma vez que isto tenha sido feito as funcionalidades poderão ser acessadas a partir de uma classe especial chamada "Portugol". A seguir (nas seções 2.1.1, 2.1.2 e 2.1.3) cada uma das funcionalidades será descrita e serão demonstrados exemplos de utilização.

2.1.1 Análise e tratamento de erros

De acordo com Aho *et al* (2007, p. 3),

> A parte de análise de um compilador subdivide o programa fonte em partes constituintes e impõe uma estrutura gramatical sobre elas. Depois, usa essa estrutura para criar uma representação intermediária do programa fonte. Se a parte de análise detectar que o programa fonte está sintaticamente mal formado ou semanticamente incorreto, então ele precisa oferecer mensagens esclarecedoras, de modo que o usuário possa tomar a ação corretiva.

Seguindo esta ideia, o PortugolCore provê um mecanismo de análise e tratamento de erros para garantir a corretude dos algoritmos escritos em Portugol. Esta análise está internamente dividida em duas partes distintas: análise sintática e análise semântica e, ocorre nesta ordem.

A análise sintática do PortugolCore é responsável por identificar os erros sintáticos do programa. Segundo Sebesta (2000, p. 112), "a sintaxe de uma linguagem de programação é a forma de suas expressões, de suas instruções e de suas unidades de programa". Assim sendo, podemos entender como erros sintáticos, os erros relacionados à má formação destas construções gramaticais, como por exemplo, o não fechamento de um parêntese em uma expressão.

A análise semântica por sua vez, identifica os erros semânticos, ou seja, as construções gramaticais que não possuem um significado válido dentro do contexto da linguagem, como por exemplo, a soma entre tipos de dados incompatíveis. Além dos erros semânticos, são detectadas conversões entre tipos e arredondamentos, os quais são tratados pelo PortugolCore não como erros, mas como avisos. Por último, a análise semântica do PortugolCore só pode ser realizada em programas que estejam sintaticamente corretos, criando uma dependência com a análise sintática.

O processo de análise de erros do PortugolCore envolve um conjunto extenso de classes e interfaces que comunicam-se entre si. No entanto, a aplicação que utiliza o PortugolCore não precisa conhecer nem interagir diretamente com estas classes. O PortugolCore provê uma interface de integração que abstrai estes detalhes de implementação através de um conjunto reduzido de classes que são expostas à aplicação, conforme ilustrado no diagrama de classes da Figura 3.

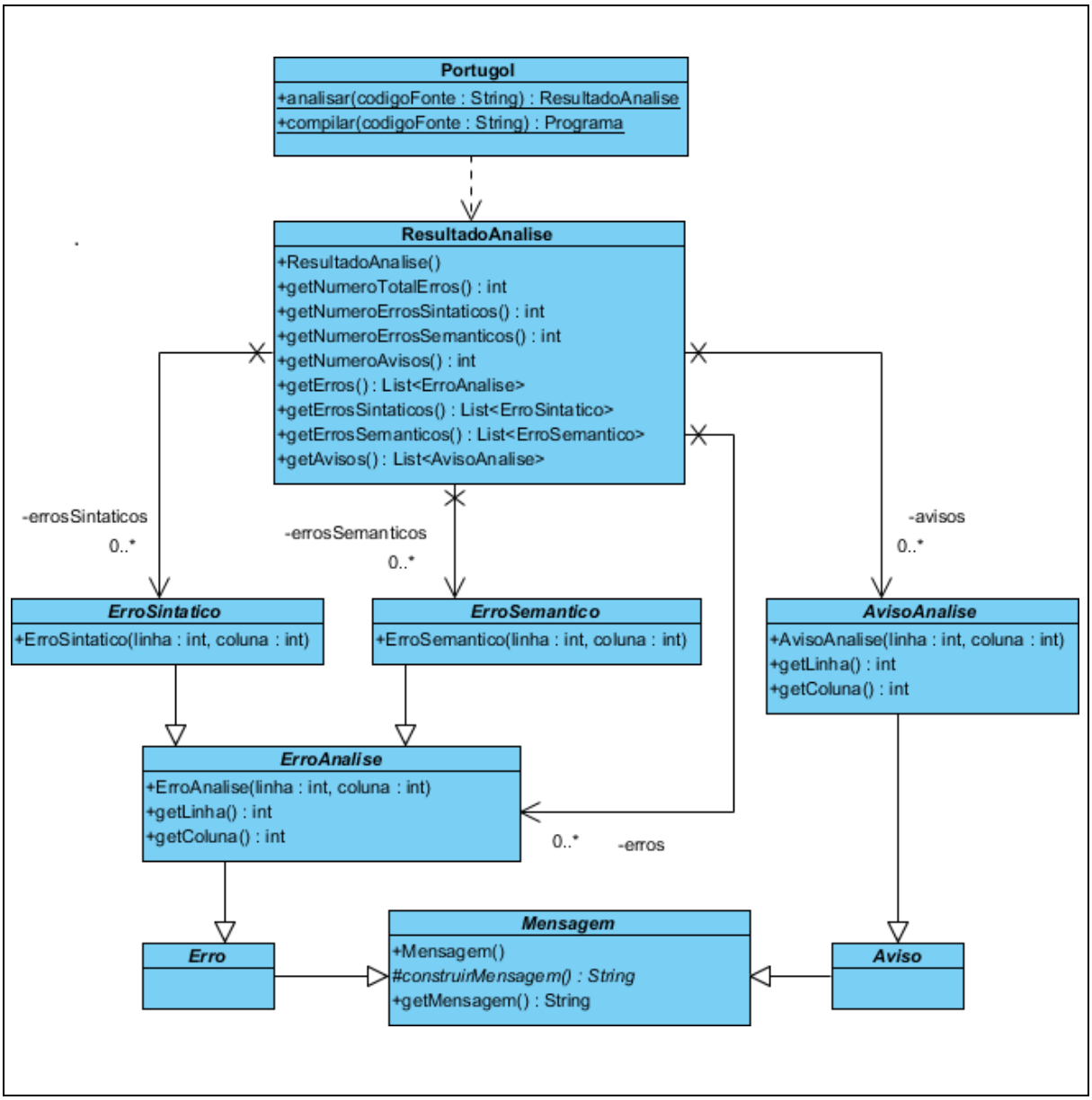

Figura 3. Interface de integração da análise de erros do PortugolCore

Para utilizar a análise de erros do PortugolCore, a aplicação deve chamar o método estático “analisar” da classe “Portugol” passando por parâmetro o código fonte a ser analisado. No final da análise, um objeto da classe “ResultadoAnalise” é retornado, o qual contém os erros e avisos encontrados durante a análise. A aplicação pode então percorrer os erros tomando as ações adequadas, como por exemplo, exibi-los ao usuário. O Quadro 1 apresenta um trecho de código que demonstra a utilização da análise de erros do PortugolCore.

```
package [...]

public final class IDE
{
      public static void main(String[] args)
      {
            File arquivo = new File("/home/noschang/algoritmo.por");
            String algoritmo = CodigoFonte.carregar(arquivo);
            ResultadoAnalise resultadoAnalise =
Portugol.analisar(algoritmo);

            if (resultadoAnalise.getNumeroTotalErros() > 0)
            {
                  System.out.println
                  (
                      "Seu algoritmo contém " +
                       resultadoAnalise.getNumeroTotalErros() +
                       " erros:"
                  );

                  for (ErroAnalise erro: resultadoAnalise.getErros())
                        System.out.println(erro.getMensagem());
            }
      }
}
```

Quadro 1. Exemplo de utilização da análise de erros do PortugolCore

2.1.2 Geração de código intermediário

De acordo com Aho *et al* (2007, p. 6), "no processo de traduzir um programa fonte para um código objeto, um compilador pode produzir uma ou mais representações intermediárias, as quais podem ter diversas formas". Seguindo esta definição, o PortugolCore permite gerar o código intermediário de qualquer algoritmo em Portugol.

No caso do PortugolCore este código intermediário é representado por uma estrutura em árvore chamada ASA (Árvore Sintática Abstrata) onde cada instrução do código-fonte corresponde a um nó da árvore e é representada por um objeto. O diagrama da Figura 4 ilustra o código intermediário gerado a partir do algoritmo do Quadro 2.

```
Programa
{
      funcao inicio()
      {
            inteiro a = 20
            escreva("A variável 'a' vale: " + a)
      }
}
```

Quadro 2. Exemplo de algoritmo do Portugol 2.0

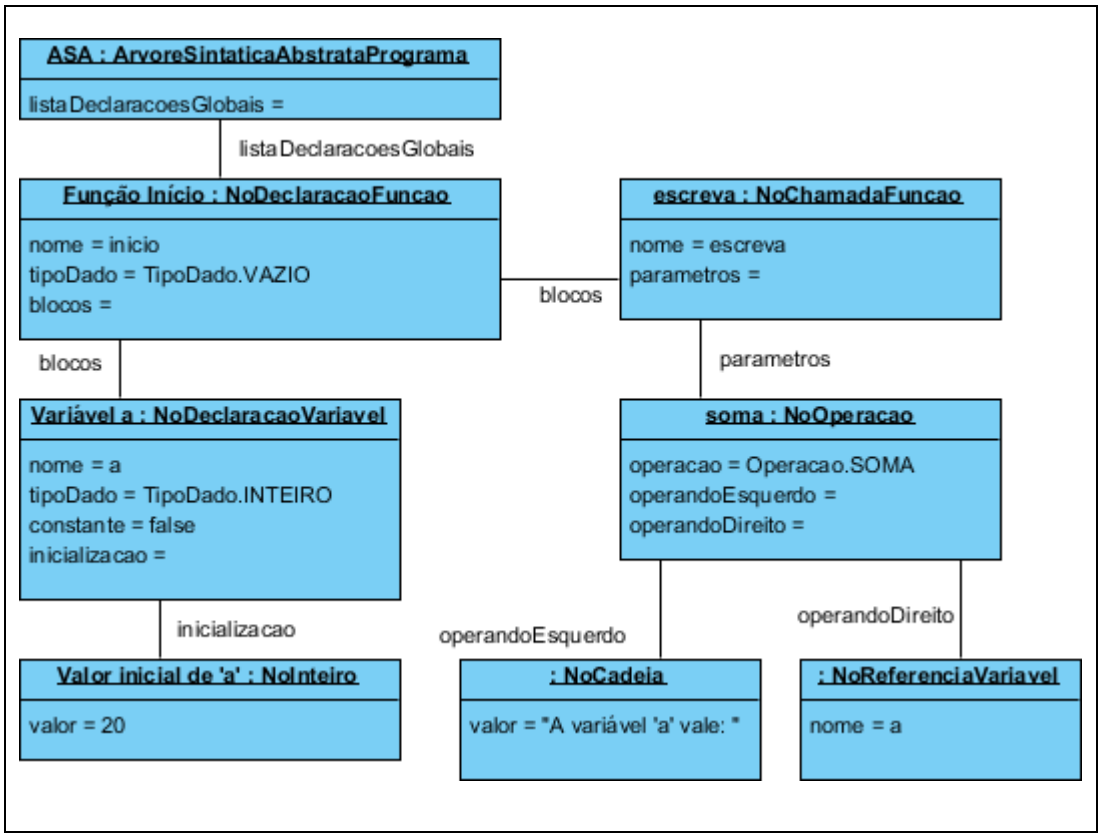

Figura 4. Código intermediário do PortugolCore

Neste momento, não serão detalhados todos nós presentes na ASA, pois isto será feito na segunda etapa deste trabalho durante a documentação do PortugolCore.

Ao utilizar a abordagem da ASA, o PortugolCore abstrai a gramática, tornando a linguagem mais dinâmica. Não importa, por exemplo, se na gramática uma multiplicação é representada pelo operador '*' ou 'x', pois o que o PortugolCore "enxergará" no final das contas será um objeto da classe "NoOperacao".

Internamente o PortugolCore utiliza a ASA para a execução dos programas. Porém, ela pode ser manipulada para obter outros resultados, como por exemplo: otimização de código, conversão do código-fonte em Portugol para outras linguagens de programação, análise de fluxo (ex.: detectar chamadas recursivas infinitas), detecção de código morto (ex.: atribuir uma variável a si mesma) entre outros. Para facilitar a criação de novas funcionalidades, a ASA do PortugolCore dá suporte ao padrão de projeto Visitor através da interface "VisitanteASA".

Várias classes estão envolvidas no processo de geração de código intermediário do PortugolCore. Mas assim como na análise de erros, esta implementação está abstraída através de um conjunto de classes reduzido, conforme ilustrado no diagrama de classes da Figura 5.

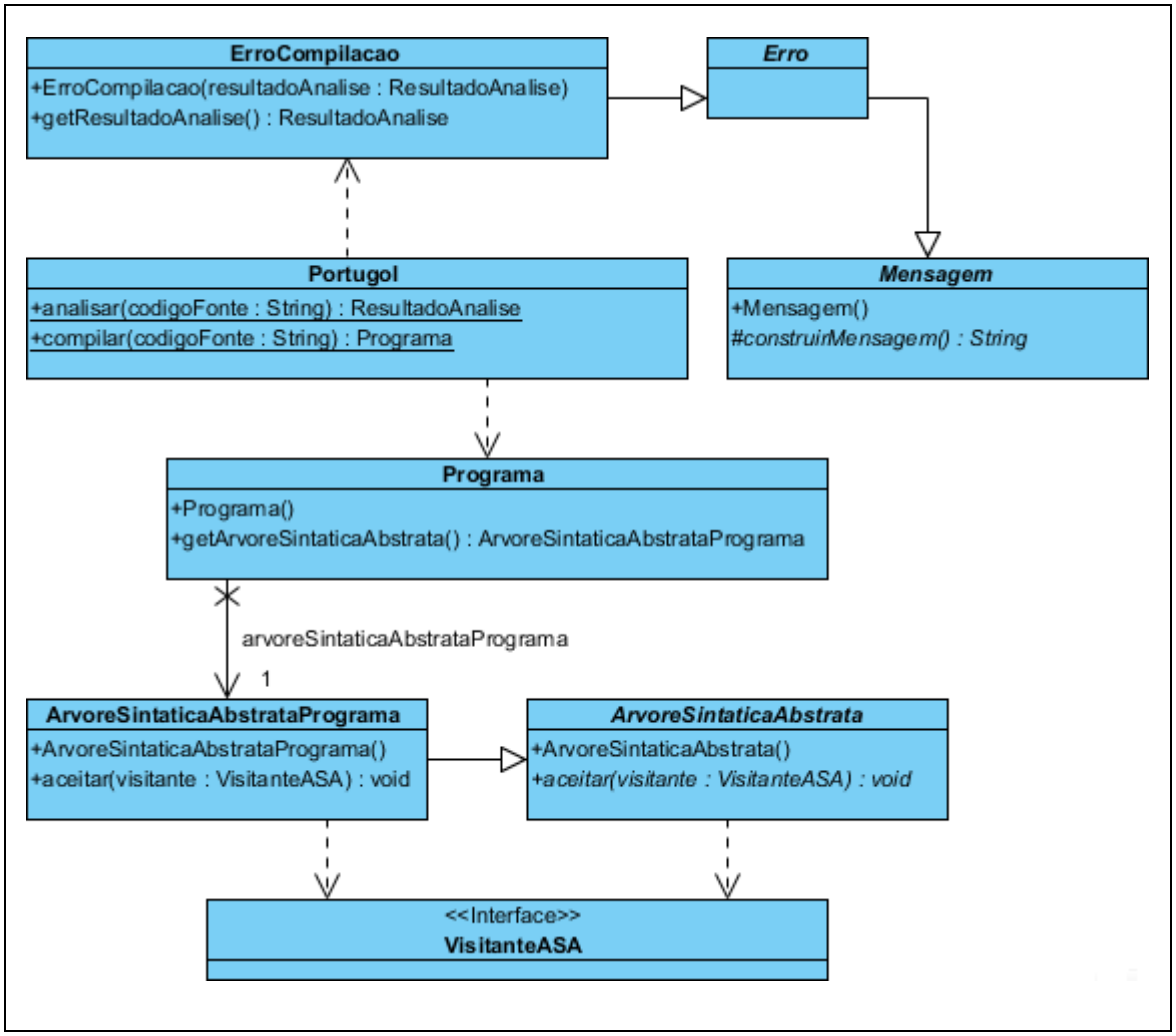

Figura 5. Interface de integração da geração de código intermediário do PortugolCore

Para utilizar a geração de código intermediário do PortugolCore, a aplicação deve chamar o método estático "compilar" da classe "Portugol" passando o código fonte a ser compilado. No final da compilação um objeto da classe "Programa" é retornado e, o código intermediário pode então ser obtido através do método "getArvoreSintaticaAbstrata".

A geração de código intermediário só pode ser realizada em algoritmos que não possuem erros sintáticos ou semânticos. Desta forma, ao chamar o método "compilar" o PortugolCore automaticamente realiza uma análise de erros. Caso o algoritmo contenha erros, será lançada uma exceção do tipo "ErroCompilacao", os erros poderão então ser obtidos através do método "getResultadoAnalise" e tratados da forma demonstrada anteriormente. O

Quadro 3 apresenta um trecho de código que demonstra a utilização da geração de código intermediário do PortugolCore.

```
package [...]

public final class IDE
{
	public static void main(String[] args)
	{
		File arquivo = new File("/home/noschang/algoritmo.por");
		String algoritmo = CodigoFonte.carregar(arquivo);

		try
		{
			Programa programa = Portugol.compilar(algoritmo);

			ArvoreSintaticaAbstrata asa =
			programa.getArvoreSintaticaAbstrata();

			for (NoDeclaracao
declaracao:asa.getListaDeclaracoesGlobais())
			{
				if (declaracao instanceof NoDeclaracaoVariavel)
				{
					System.out.println
					(
						"Seu algoritmo contém uma variável " +
							"chamada '" + declaracao.getNome()
+ "'."
					);
				}
			}
		}
		catch (ErroCompilacao e)
		{
			exibirErros(e.getResultadoAnalise());
		}
	}

	private static void exibirErros(ResultadoAnalise resultadoAnalise)
	{
		if (resultadoAnalise.getNumeroTotalErros() > 0)
		{
			System.out.println
			(
				"Seu algoritmo contém " +
					resultadoAnalise.getNumeroTotalErros() +
				" erros:"
			);

			for (ErroAnalise erro: resultadoAnalise.getErros())
				System.out.println(erro.getMensagem());
		}

	}
}
```

Quadro 3: Exemplo de utilização da geração de código intermediário do PortugolCore

2.1.3 Execução de programas

Segundo Sebesta (2007, p. 45),

> Alguns sistemas de implementação de linguagem são um meio-termo entre os compiladores e os interpretadores puros; eles traduzem programas em linguagem de alto nível para uma linguagem intermediária projetada para permitir fácil interpretação.[...] Essas implementações são chamadas de sistemas de implementação híbridos.

Partindo desta afirmação o PortugolCore pode ser classificado como um sistema de implementação híbrido, pois além de gerar código intermediário a partir dos algoritmos, ele executa os programas percorrendo a ASA e interpretando cada nó no momento em que é encontrado. A Figura 6 demonstra um programa sendo executado pelo PortugolCore no PortugolStudio.

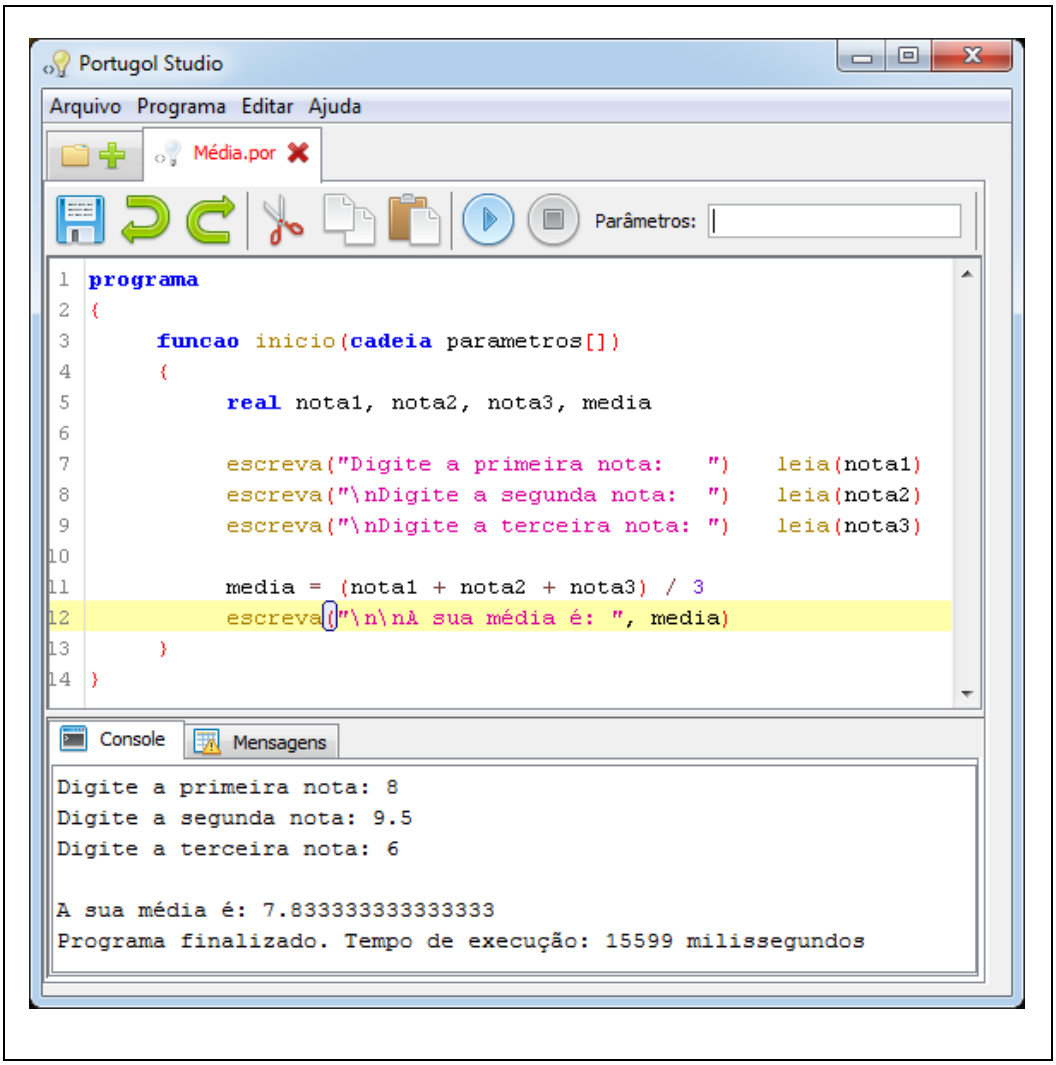

Figura 6. Programa sendo executado pelo PortugolCore no PortugolStudio

No PortugolCore, cada programa é executado em sua própria thread, permitindo desta forma que vários programas diferentes sejam executados ao mesmo tempo. Da mesma forma que as demais funcionalidades do PortugolCore, a execução de programas é abstraída por um conjunto de classes e interfaces, conforme é ilustrado no diagrama de classes da Figura 7.

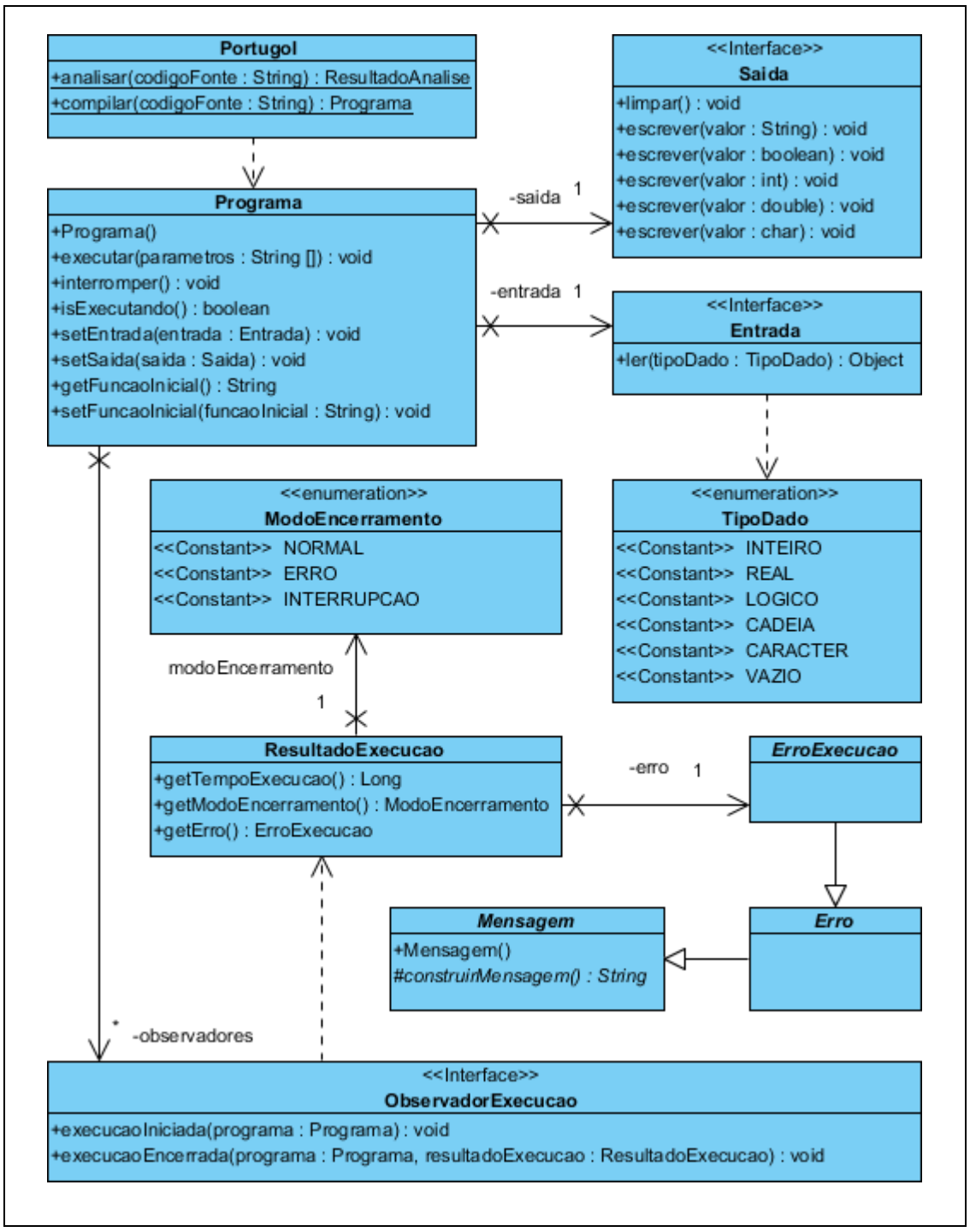

Figura 7. Interface de integração da execução de programas do PortugolCore

Para que uma aplicação possa utilizar a funcionalidade de execução de programas do PortugolCore, as interfaces “Entrada” e “Saída”, representadas no diagrama, devem ser obrigatoriamente implementadas.

A interface “Entrada” é responsável pela leitura dos valores de entrada do programa utilizando o comando “leia”. Para realizar esta leitura, a interface provê o método “ler” passando como argumento o tipo de dado esperado. A aplicação ao implementar esta interface deve retornar um objeto correspondente ao tipo de dado solicitado. A entrada dos dados pode ser feita de forma interativa com o usuário ou de forma automática. O Quadro 4 apresenta um trecho de código que demonstra uma implementação interativa:

```
@Override
public Object ler(TipoDado tipoDado) throws Exception
{
	String entrada = JOptionPane.showInputDialog
	(
		"Digite um valor do tipo " +
		tipoDado.toString() + ":"
	);

	switch (tipoDado)
	{
		case INTEIRO: return Integer.parseInt(entrada);
		case REAL: return Double.parseDouble(entrada);
		case CARACTER: return entrada.charAt(0);
		case CADEIA: return entrada;
		case LOGICO:
		{
			if (entrada.equals("verdadeiro")) return true;
			else
			if (entrada.equals("falso")) return false;
		}

		default: return null;
	}
}
```

Quadro 4. Entrada de dados do PortugolCore

A interface “Saída” é responsável pela escrita dos valores de saída do programa utilizando o comando “escreva”. Para realizar esta escrita, a interface provê cinco sobrecargas do método “escrever”, uma para cada tipo de dado, passando como argumento o valor a ser escrito. Esta interface provê ainda o método “limpar” utilizado para limpar a saída de dados.

A implementação da saída de dados pode ser feita de qualquer forma, ficando a critério da aplicação. É possível, por exemplo, redirecioná-la para alimentar um banco de

dados ou a entrada de outros algoritmos. O Quadro 5 apresenta um trecho de código que demonstra uma saída de dados simples onde os valores são impressos no console da JVM (Java Virtual Machine).

```
@Override
public void escrever(String cadeia) throws Exception
{
      System.out.print(cadeia);
}

@Override
public void escrever(boolean logico) throws Exception
{
      System.out.print(logico? "verdadeiro" : "falso");
}

@Override
public void escrever(int inteiro) throws Exception
{
      System.out.print(inteiro);
}

@Override
public void escrever(double real) throws Exception
{
      System.out.print(real);
}

@Override
public void escrever(char caracter) throws Exception
{
      System.out.print(caracter);
}

@Override
public void limpar() throws Exception
{
      for (int i = 0; i < 20; i++)
            System.out.println();
}
```

Quadro 5. Saída de dados do PortugolCore

Ao implementar as interfaces “Entrada” e “Saída” deve-se tomar muito cuidado com a utilização de threads, pois a entrada e saída de dados do PortugolCore ocorre de forma síncrona. Em outras palavras, quando uma instrução de entrada ou saída é encontrada, o PortugolCore fica aguardando até que a leitura/escrita termine de ser processada para então continuar a execução do programa. Caso sejam utilizadas threads, cabe à aplicação garantir a sincronia das operações, caso contrário elas podem resultar em inconsistências na execução.

Existe ainda uma terceira interface no mecanismo de integração chamada “ObservadorExecucao”. Ela não é obrigatória, mas pode ser utilizada para monitorar o estado da execução de um programa. Na verdade, podem existir vários observadores de execução para um mesmo programa, mas geralmente isto não é necessário. Para permitir este monitoramento, esta interface provê dois métodos: “execucaoIniciada” e “execucaoEncerrada”.

O método “execucaoIniciada” é chamado pelo interpretador do PortugolCore antes de iniciar a execução de um programa e recebe por parâmetro a instância do programa em questão. Este método pode ser utilizado para realizar ações de pré-inicialização. O Quadro 6 ilustra um trecho de código que demonstra a utilização deste método imprimindo uma mensagem na saída de dados:

```
@Override
public void execucaoIniciada(Programa programa)
{
	System.out.println("Iniciando a execução do programa...");
}
```

Quadro 6. Monitorando o início da execução de um programa

O método “execucaoEncerrada” é chamado pelo interpretador do PortugolCore após o encerramento de um programa. Existem três modos diferentes de encerramento de um programa: encerramento normal, encerramento por interrupção e encerramento ocasionado por erro. O encerramento normal ocorre quando todas as instruções foram interpretadas e o programa chegou ao seu fim. O encerramento por interrupção ocorre quando o usuário interrompe manualmente o programa por algum motivo. O encerramento ocasionado por erro ocorre quando o programa encontra um erro em tempo de execução (ex.: divisão por zero; acesso a um índice de vetor inválido).

O método “execucaoEncerrada” ao ser chamado, recebe por parâmetro o programa que estava sendo executado e um objeto do tipo “ResultadoExecucao”, a partir do qual é possível obter três informações importantes: o tempo de execução do programa, o modo de encerramento do programa e, caso tenha ocorrido, o erro que ocasionou o encerramento. O Quadro 7 ilustra um trecho de código que demonstra a utilização deste método.

```
@Override
public void execucaoEncerrada(Programa programa, ResultadoExecucao
resultado)
{
  Long tempo = resultado.getTempoExecucao();
  ModoEncerramento modoEncerramento = resultado.getModoEncerramento();

  switch (modoEncerramento)
  {
     case NORMAL: System.out.println("Programa finalizado."); break;
     case INTERRUPCAO: System.out.println("Programa interrompido."); break;
     case ERRO: System.out.println(resultado.getErro().getMensagem());
break;
  }

  System.out.println("Tempo de execução: " + tempo + " milissegundos.");
}
```

Quadro 7. Monitorando o encerramento de um programa

Uma vez que estas interfaces tenham sido implementadas, é necessário instruir o PortugolCore a utilizá-las. Isto é feito através dos métodos “adicionarObservadorExecucao”, “setEntrada” e “setSaida” do objeto “Programa”. Por fim, a execução pode ser iniciada realizando-se uma chamada ao método “executar” do objeto “Programa”. O Quadro 8 ilustra um trecho de código que demonstra a execução de um programa.

```
private void iniciar(String[] parametros)
{
      File arquivo = new File("/home/noschang/algoritmo.por");
      String algoritmo = CodigoFonte.carregar(arquivo);

      try
      {
            Programa programa = Portugol.compilar(algoritmo);

            programa.setEntrada(this);
            programa.setSaida(this);
            programa.adicionarObservadorExecucao(this);
            programa.executar(parametros);

      }
      catch (ErroCompilacao e)
      {
            exibirErros(e.getResultadoAnalise());
      }
}
```

Quadro 8. Exemplo de utilização da execução de programas do PortugolCore

A execução do programa pode ser interrompida a qualquer momento chamando-se o método “interromper” do objeto “Programa”. Além disso, é possível verificar se um

programa está em execução através do método “isExecutando”. O PortugolCore também permite a passagem de parâmetros para o programa através do método “executar”.

Por último, o interpretador do PortugolCore necessita que seja definido um ponto de partida para a execução. Isto é feito através do método “setFuncaoInicial” do objeto “Programa”, passando-se o nome de uma função. O programa a ser executado deverá obrigatoriamente conter esta função, caso contrário o interpretador lançará um erro e a execução não ocorrerá. Se a função inicial não for informada, o PortugolCore assume por padrão a função “inicio”.

2.2 REQUISITOS DE INTEGRAÇÃO

O objetivo geral deste trabalho é adaptar o PortugolCore de forma que ele possa ser integrado com ferramentas escritas em outras linguagens de programação. Esta integração será feita utilizando uma das tecnologias/técnicas de integração a serem pesquisadas. A seguir, é apresentada uma relação dos requisitos que se espera obter destas tecnologias. Os requisitos serão utilizados como critério de avaliação para determinar qual das tecnologias pesquisadas representa a melhor solução a ser adotada.

Serão determinados pesos para cada requisito indicando sua relevância dentro do trabalho. A soma do peso de todos os requisitos totalizará 100.00 pontos. Para cada requisito também serão criadas classificações que determinarão os possíveis valores para cada requisito e o peso de cada valor.

Interoperabilidade

Tendo em vista o cumprimento do objetivo geral deste trabalho, a solução a ser adotada deve permitir a integração com o maior número possível de linguagens de programação. Este requisito terá peso 25.00 e será classificado da seguinte forma:

Interoperabilidade (Peso: 25.00)			
Quantidade de linguagens suportadas	Uma	Duas a quatro	Cinco ou mais
Classificação	Baixa	Média	Alta
Peso	25.00	50.00	100.00

Reusabilidade

A principal motivação em possibilitar a integração do PortugolCore com outras ferramentas está em permitir com que elas reutilizem as funcionalidades oferecidas pelo PortugolCore ao invés de implementá-las. A solução a ser adotada deve garantir a reutilização das implementações existentes. Este requisito terá peso 30.0 e será classificado da seguinte forma:

Reusabilidade (Peso: 30.00)		
Classificação	Não contempla	Contempla
Peso	00.00	100.00

Licença

Um dos objetivos específicos do trabalho é disponibilizar o PortugolCore e sua documentação em um repositório de projetos OpenSource para que ele venha contribuir com a comunidade científica. Assim sendo, a solução a ser adotada deve possuir licença livre ou OpenSource. Este requisito terá peso 30.00 e será classificado da seguinte forma:

Licença (Peso: 30.00)		
Classificação	Proprietária / Indisponível / Outra	OpenSource / Livre / Não possui
Peso	00.00	100.00

Para auxiliar na classificação das licenças, foi elaborada a Tabela 1, a qual contém uma relação das principais licenças OpenSource existentes. A tabela foi elaborada a partir de uma pesquisa no site da OSI (Open Source Initiative).

Tabela 1. Principais licenças OpenSource

Nome	**Abreviatura**
Apache License, 2.0	Apache-2.0
BSD 3-Clause "New" or "Revised" license	BSD-3-Clause
BSD 3-Clause "Simplified" or "FreeBSD" license	BSD-2-Clause
Common Development and Distribution License	CDDL-1.0
Eclipse Public License	EPL-1.0
GNU General Public License	GPL
GNU Library or "Lesser" General Public License	LGPL
MIT license	MIT
Mozilla Public License 1.1	MPL-1.1

Documentação

A solução a ser adotada deve possuir uma documentação ampla, que auxilie na elaboração e implementação do projeto do sistema. A documentação deve conter descrições claras que possibilitem sua compreensão, exemplos de utilização, códigos-fonte, diagramas, figuras, etc. Este requisito terá peso 15.00 e será classificado da seguinte forma:

Documentação (Peso: 15.00)			
Classificação	Não possui	Pobre	Rica
Peso	00.00	50.00	100.00

O cálculo final da relevância será feito utilizando uma média ponderada entre os requisitos, conforme mostra a Equação 1.

$$R = \frac{(VI * PI) + (VR * PR) + (VL * PL) + (VD * PD)}{PI + PR + PL + PD} \qquad \text{Equação 1}$$

Onde: R = Relevância; VI = Valor Interoperabilidade; PI = Peso Interoperabilidade; VR = Valor Reusabilidade; PR = Peso Reusabilidade; VL = Valor Licença; PL = Peso Licença; VD = Valor Documentação; PD = Peso Documentação.

A Equação 2 ilustra o cálculo realizado para uma determinada tecnologia que possui média interoperabilidade, contempla reusabilidade, possui licença proprietária e cuja documentação é pobre. Neste exemplo, a tecnologia possui relevância 50.00.

$$R = \frac{(50 * 25) + (100 * 30) + (0 * 30) + (50 * 15)}{25 + 30 + 30 + 15} = 50 \qquad \text{Equação 2}$$

Será eleita como melhor tecnologia a ser utilizada no desenvolvimento do trabalho, aquela que apresentar a maior relevância. Caso haja mais de uma tecnologia com a mesma relevância, uma delas será escolhida mediante uma análise mais detalhada.

2.3 TECNOLOGIAS DE INTEGRAÇÃO

Existem atualmente diversas tecnologias para possibilitar a integração de sistemas. Algumas mais antigas e simples como a comunicação baseada em arquivos textuais. Outras mais robustas e elaboradas como os WebServices. A seguir é feita uma análise de/ algumas das tecnologias de integração existentes, descrevendo características que permitam avalia-las em relação aos requisitos estabelecidos.

2.3.1 Arquivos textuais

Uma das técnicas mais simples para a integração entre sistemas é através de arquivos textuais. Neste tipo de integração, as aplicações leem e escrevem arquivos de texto no disco, contendo as informações a serem compartilhadas. Para que este tipo de integração funcione, as aplicações devem definir uma estrutura de arquivo que será conhecida por ambas.

A maioria das linguagens de programação (se não todas) implementa rotinas de leitura e escrita de arquivos. Desta forma, é possível a integração entre diversas linguagens de

programação. No entanto, a simplicidade dos arquivos de texto faz com seja difícil trocar informações complexas entre essas aplicações.

2.3.2 XML

A XML (Extensible Markup Language) é uma linguagem textual que possibilita a troca de informações entre aplicativos. Uma das principais vantagens da XML é que esta permite representar de forma textual, estruturas de dados complexas e hierárquicas, como por exemplo, uma coleção de objetos e seus atributos (que podem ser outros objetos) em Java. Isto a torna mais robusta do que a abordagem com arquivos textuais simples.

Cummins (2002, p. 245), cita oito características que popularizaram a linguagem XML: (i) é uma tecnologia não patenteada (portanto não possui licença ou restrições de uso); (ii) é independente de plataforma (permite representar dados em qualquer linguagem e sistema operacional); (iii) é compatível com o protocolo HTTP; (iv) suporta internacionalização; (v) é extensível; (vi) é autodefinível; (vii) possui diversas ferramentas de apoio; e (viii) permite transformação.

A maioria das linguagens possui processadores de XML, permitindo com que seja possível trocar informações via XML entre várias linguagens de programação. Por sua flexibilidade, a XML é utilizada em conjunto com outras tecnologias para permitir troca de informações e integração, um exemplo disso é a plataforma .NET.

A linguagem XML possui uma excelente documentação no site da W3Schools que inclui diversos exemplos. Lá é possível encontrar não só a documentação da linguagem XML, mas de várias outras tecnologias relacionadas, como por exemplo, o XML Schema.

2.3.3 Sockets

Sockets consistem em um conjunto de interfaces que permitem a comunicação entre aplicativos e/ou dispositivos em uma rede através do protocolo TCP. Segundo Santos Junior (2007, p. 80), com sockets uma aplicação pode conversar com aplicações em outras linguagens como C++, Delphi, etc.

As interfaces padronizadas dos sockets trabalham com dados em baixo nível, os bytes. Desta forma, é necessário estabelecer um formato de dados para a comunicação entre as aplicações. Em outras palavras, uma aplicação deve “saber ler” o que a “outra escreve”

através dos sockets. Este baixo nível de abstração faz com que a comunicação entre as aplicações, embora possível, seja difícil.

Os sockets geralmente são implementados e disponibilizados pelo sistema operacional, desta forma, não possuem licença nem restrição de uso e funcionam em qualquer sistema operacional que os implementem. Na Internet é possível encontrar boa documentação com exemplos.

2.3.4 RPC

RPC (Remote Procedure Call) é um protocolo de comunicação entre processos que permite que um procedimento de determinado processo seja chamado por outro processo em um espaço de memória diferente. Diversas das tecnologias de integração existentes utilizam o protocolo RPC para implementar suas funcionalidades.

O RPC é mais robusto do que uma comunicação via sockets porque, ao contrário destes, a comunicação ocorre de forma mais transparente. A aplicação que chama o procedimento remoto geralmente não sabe que ele está sendo executando remotamente.

No entanto, segundo Togni (2005), "RPC é baseado no modelo estruturado, portanto, um servidor pode ter apenas uma instância de uma dada função, e os parâmetros que podem ser passados são tipos simples como números e sequências de caracteres".

2.3.5 JNI

A JNI (Java Native Interface) é uma API (Application Programming Interface) da plataforma Java que permite a uma aplicação Java chamar ou ser chamado por rotinas implementadas em outras linguagens de programação (geralmente em C e C++). Por se tratar de uma extensão do Java, esta tecnologia possui licença livre.

A JNI surgiu como uma solução para as aplicações que não podem ser desenvolvidas totalmente em Java. A JNI é utilizada pela própria plataforma Java para disponibilizar recursos de baixo nível do sistema operacional como leitura e escrita de arquivos.

Bishop (2003) apresenta um exemplo de integração entre Java e C# utilizando JNI. Para realizar a integração foi necessário criar uma ponte entre as duas linguagens utilizando C++. Bishop afirma que, embora tenha sido possível realizar esta integração, esta não é uma

tarefa fácil. Santos Junior (2007, p. 38), no entanto, aponta a reutilização e a performance como os principais motivos para utilizar esta solução.

A API JNI conta com uma ampla documentação. No próprio site da Oracle é possível encontrar vários exemplos e tutorias de utilização. Além disso, ela conta com o Javadoc, um formato de documentação próprio da plataforma Java que oferece informações detalhadas das classes da API e que se integra automaticamente aos ambientes de desenvolvimento (IDEs).

2.3.6 Java RMI

Java RMI (Remote Method Invocation) é uma das abordagens da tecnologia Java para prover as funcionalidades de uma plataforma de objetos distribuídos. Segundo Ricarte (2002 apud BORSOI, 2004, p. 4), "através da utilização da arquitetura RMI, é possível que um objeto ativo em uma máquina virtual Java possa interagir com objetos de outras máquinas virtuais Java, independentemente da localização dessas máquinas virtuais". Assim como o JNI, o Java RMI também é livre e possui ampla documentação.

O Java RMI oferece um excelente suporte a sistemas de objetos distribuídos em Java, porém, por se tratar de uma extensão da linguagem, não permite a integração com outras linguagens de programação. Assim como as demais tecnologias Java, o Java RMI possui uma documentação rica disponível no site da Oracle.

2.3.7 COM/DCOM/.NET

O COM (Component Object Model) é um framework da Microsoft para permitir a comunicação entre processos. A arquitetura COM permite que objetos sejam escritos de forma independente de linguagem de programação, pois obriga que estes objetos definam interfaces para a comunicação.

A partir do COM surgiram outras tecnologias como o DCOM (Distributed Component Object Model), que permite a implementação de objetos COM de forma distribuída e o .NET framework que é baseado em DCOM e segue princípios parecidos.

A principal barreira destas tecnologias é que, por serem proprietárias da Microsoft, possuem certa dependência dos recursos e serviços do Windows, limitando assim, seu uso em outras plataformas. Segundo Siqueira (2005, p. 33), "a Microsoft tem procurado argumentar

que esta plataforma é independente de qualquer sistema operacional, contudo, implementações robustas de DCOM existem somente para Windows".

As tecnologias COM, DCOM e .NET possuem ampla documentação na página de desenvolvimento oficial da Microsoft a MSDN (Microsoft Developer Network). A MSDN oferece uma documentação completa organizada em tópicos e que permite realizar buscas.

2.3.8 CORBA

CORBA (Common Object Request Broker) é uma arquitetura para desenvolvimento de sistemas de objetos distribuídos criada em 1991 pela OMG (Object Management Group), uma organização internacional que reúne diversas empresas.

A arquitetura CORBA permite a comunicação entre objetos em linguagens de programação diferentes de forma transparente. Para permitir esta comunicação, CORBA utiliza uma linguagem chamada IDL (Interface Definition Language) que define as interfaces dos objetos distribuídos.

Para trabalhar com CORBA é necessário obter uma implementação da arquitetura. Essas implementações são chamadas de ORB (Object Request Brokers) e são elas, por exemplo, que permitem o mapeamento das interfaces em linguagem IDL para as linguagens de programação comuns.

A arquitetura CORBA possui uma documentação completa da especificação no site da OMG. Além disso, cada implementação de CORBA tem sua própria licença e fornece sua própria documentação.

2.3.9 WebServices

O WebService é uma arquitetura que permite a uma aplicação expor suas funcionalidades na forma de interfaces de serviço acessíveis pela Internet através do protocolo HTTP (Hipertext Transfer Protocol). Da mesma forma que nas aplicações Web comuns, as operações de um WebService são acessadas através de uma URL (Uniform Resource Locator).

Os WebServices são descritos em uma linguagem chamada WSDL (WebService Description Language), na qual são definidos, por exemplo, os métodos disponíveis no

serviço e o protocolo utilizado para a troca de mensagens. Existem dois protocolos de troca de mensagens que são comumente utilizados pelos WebServices, o SOAP (Simple Object Access Protocol) e o REST (Representational State Transfer).

Tanto a descrição do WebService quanto os protocolos de troca de mensagens, são baseados na linguagem XML. Segundo Santos Junior (2007, p. 90), "os WebServices são componentes que permitem às aplicações enviar e receber dados em formato XML. Cada aplicação pode ter a sua própria linguagem, que é traduzida para uma linguagem universal, o formato XML". Desta forma, os WebServices permitem a integração de diferentes aplicações, independente da linguagem de programação em que são escritas.

Várias linguagens de programação, como Java e .NET, permitem transformar suas aplicações em WebServices com o mínimo de esforço e aproveitando as implementações existentes. Desta forma, os WebServices configuram uma tecnologia de alta reusabilidade.

Uma boa documentação da arquitetura de WebServices, bem como da linguagem WSDL e do protocolo SOAP pode ser encontrada no site da W3C (World Wide Web Consortium) que regulamenta estas tecnologias. Vários exemplos de implementação também estão disponíveis na página da W3Schools. Todas estas tecnologias são de formato aberto e, portanto não possuem licença ou restrições.

2.4 ANÁLISE COMPARATIVA

Após a pesquisa bibliográfica das tecnologias de integração existentes, foi realizada a análise comparativa destas tecnologias para determinar a relevância de cada uma delas para o trabalho. Durante a análise foram verificados, para cada tecnologia, quais dos requisitos definidos eram atendidos pela mesma e de que forma.

Após a verificação dos requisitos, a relevância foi determinada através do cálculo da média ponderada dos requisitos de integração, conforme demonstrado na Equação 1 e na Equação 2. A Tabela 2 mostra os dados levantados nesta análise, relacionando as tecnologias pesquisadas com os requisitos de integração e a relevância calculada.

Tabela 2. Análise comparativa das tecnologias

Tecnologia	Interoperabilidade	Reusabilidade	Licença	Documentação	Relevância
Arquivos textuais	Alta	Não contempla	Não possui	Não possui	55,00
CORBA	Alta	Contempla	Livre *	Rica	100,00
COM/ DCOM/ .NET	Média	Contempla	Livre	Rica	87,50
JNI	Média	Contempla	Livre	Rica	87,50
Java RMI	Baixa	Contempla	Livre	Rica	81,25
RPC	Alta	Não contempla	Não possui	Rica	70,00
Sockets	Alta	Não contempla	Não possui	Rica	70,00
XML	Alta	Não contempla	Não possui	Rica	70,00
WebService	Alta	Contempla	Não possui	Rica	100,00

* Depende da implementação CORBA

Conforme pode ser visto na tabela, as tecnologias que apresentaram menor índice de relevância (55,00 e 70,00) foram: arquivos textuais, XML, Sockets e RPC. O fator determinante para a eliminação destas tecnologias foi o fato de elas não contemplarem a reusabilidade das implementações existentes, um dos requisitos mais importantes para o mecanismo de integração.

O Java RMI apresentou um índice de relevância maior (81,25), pois permite a reusabilidade das implementações existentes. No entanto, foi prejudicado pela baixa interoperabilidade. Por se tratar de uma tecnologia Java, o Java RMI só permite a integração com outras aplicações escritas em Java.

Já as tecnologias COM/DCOM/.NET da Microsoft e o JNI da Oracle, apresentaram o segundo maior índice de relevância (87,50), pois ambas contemplam a reusabilidade das implementações existentes. No entanto, ainda não são a solução ideal por possuírem interoperabilidade média. O JNI, por exemplo, permite a integração direta com C e C++, mas para integrar com outras linguagens é necessário criar uma "ponte" utilizando uma destas duas linguagens. Já as tecnologias Microsoft estão limitadas á plataforma Windows, e consequentemente ás linguagens utilizadas pela mesma: C, C++, VisualBasic e C#.

Por fim, as tecnologias que apresentaram maior índice de relevância (100,00) foram CORBA e WebServices, pois ambas as atendem de forma satisfatória a todos os requisitos estabelecidos. Embora estas duas tecnologias tenham empatado, a arquitetura CORBA é a escolhida para a implementação do projeto pelos seguintes motivos:

- O foco dos WebServices está na integração de aplicativos rodando em diferentes máquinas através da Internet, enquanto que, o foco do PortugolCore está na integração de duas aplicações (o PortugolCore e a aplicação que o utiliza) em ambiente Desktop localizadas na mesma máquina;
- Os WebServices dependem de um aplicação Web Server para funcionar; e
- Mesmo que o mecanismo de integração seja implementado em CORBA, ainda será possível transformá-lo em um WebService (caso isto se torne necessário) utilizando qualquer linguagem de programação que dê suporte à sua construção.

3. DESENVOLVIMENTO

Após o estudo e a análise comparativa das várias tecnologias de integração, decidiu-se utilizar a arquitetura CORBA para a implementação do mecanismo de integração, pois esta atende a todos os requisitos de integração estabelecidos (Interoperabilidade, Reusabilidade, Licença e Documentação).

Na primeira parte deste capítulo (COMPREENDENDO O CORBA), é realizado um estudo um pouco mais aprofundado da arquitetura CORBA, visando reunir o conhecimento necessário para a elaboração do projeto.

Já na segunda parte, é realizado o projeto e a do sistema em si, apresentando a forma como o sistema deverá ser implementado. O projeto é descrito de forma textual e conta com o auxílio de figuras para facilitar a compreensão do mesmo.

Por último, é feito o planejamento do TCC II, onde é apresentado o cronograma das atividades a serem realizadas e a metodologia a ser utilizada no desenvolvimento destas atividades.

3.1 COMPREENDENDO O CORBA

"O padrão CORBA foi definido em 1991 pela OMG, uma organização internacional que reúne empresas vendedoras de sistemas de informação, de desenvolvimento de software, universidades e utilizadores" (BORSOI, SCHULTZ, 2004, p. 2).

Coulouris, Dollimore e Kindberg (2007, p.710) definem CORBA como sendo "um projeto de middleware que permite as aplicações se comunicarem umas com as outras independentemente de suas linguagens de programação, de suas plataformas de hardware e software, das redes pelas quais se comunicam e de seus desenvolvedores".

Para Siqueira (2005, p. 28), "a arquitetura CORBA é um conjunto complexo de partes que possuem suas particularidades e especificações, que tem como objetivo comum, promover um ambiente que atenda às especificações definidas pela OMG para o CORBA".

Para entender como a integração com CORBA funciona é necessário entender as partes que compõem sua arquitetura. Estas partes serão descritas de forma resumida a seguir e estão ilustradas na Figura 8.

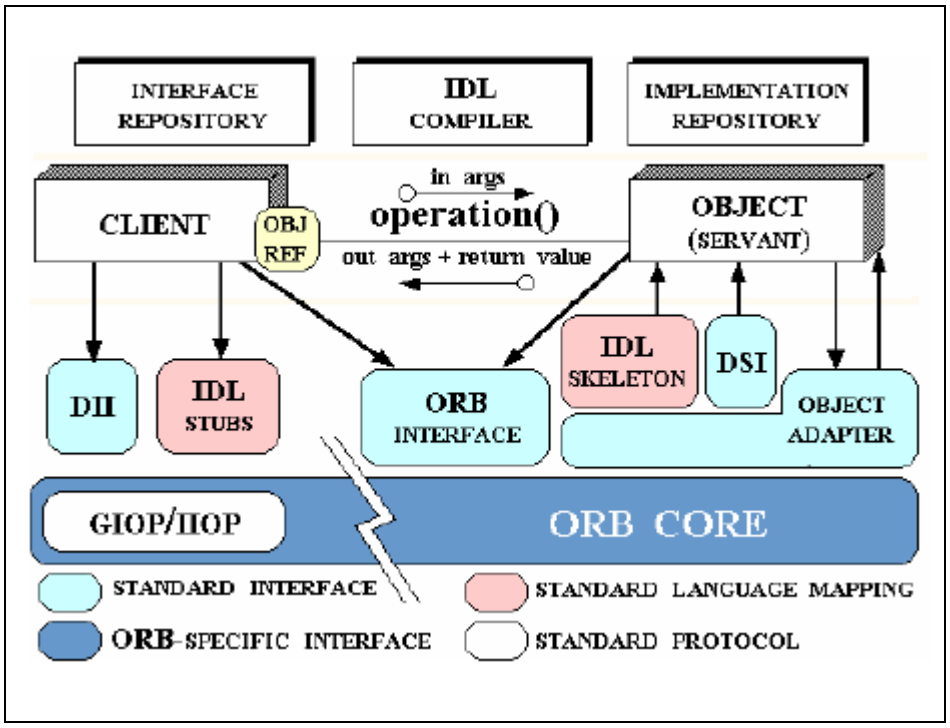

Figura 8. Arquitetura CORBA

Fonte: Borsoi (2004)

Objeto CORBA

Objetos CORBA são objetos escritos na linguagem nativa da aplicação e possuem uma referência para um objeto remoto. Segundo Siqueira (2005, p.23), "cada objeto tem uma interface definida, a qual deve conhecer e saber requisitar um serviço a outro objeto". Estas interfaces são definidas na linguagem IDL e permitem que seus métodos sejam invocados no objeto remoto.

Cliente (Client)

Clientes são as aplicações que enviam requisições aos objetos CORBA. Este nome remete à arquitetura de cliente/servidor geralmente presente em sistemas distribuídos. Uma aplicação em CORBA pode assumir diferentes papéis, podendo ser cliente no escopo de uma requisição e servidora no escopo de outra requisição.

Servidor (Server)

Servidores são as aplicações que detêm os objetos CORBA e recebem as requisições feitas pelos clientes. Da mesma forma que os clientes, os servidores podem assumir ambos os papéis em momentos diferentes.

Servente (Servant)

Os serventes são objetos escritos na linguagem de programação do servidor e implementam as funcionalidades dos objetos CORBA. Para cada objeto CORBA deve existir um objeto servente.

Agente de requisição de objetos (ORB)

Agente de requisição de objetos ou ORB (Object Request Broker) é o elemento da arquitetura CORBA responsável pela comunicação entre os objetos remotos:

> Os ORBs promovem interoperabilidade de sistemas de objetos distribuídos, pois eles permitem que usuários desenvolvam sistemas através da junção de objetos de diferentes fornecedores, podendo se comunicar uns com os outros através do ORB. Os ORBs manipulam a transformação de estrutura de dados internos dos processos de e para a sequência de bytes, que é transmitida pela rede. Além disso, expõem funções como transações distribuídas, serviços de diretório ou escalonamento de tempo de execução (WIKIPEDIA, 2011).

Linguagem IDL

A linguagem IDL (Interface Definition Language) “especifica um nome e um conjunto de métodos que os clientes podem solicitar” (COULOURIS, DOLLIMORE e KINDBERG, 2007, p. 712), em outras palavras, ela descreve as interfaces dos objetos CORBA de forma independente de plataforma ou linguagem de programação. Segundo Nardi (2003), essa característica é ponto chave para a utilização de CORBA em sistemas heterogêneos e integração de aplicações desenvolvidas separadamente.

Para a definição de suas interfaces, a IDL oferece suporte a dois conjuntos de tipos de dados: os tipos primitivos e os tipos construídos. Segundo Coulouris, Dollimore e Kindberg (2007, p. 722) os tipos primitivos suportados pela IDL são: “short(16 bits), long (32 bits), unsigned short, unsigned long, float (32 bits), double (64 bits), char, boolean (TRUE,

FALSE), octet (8 bits) e any". Os tipos construídos por sua vez, representam tipos de dados mais complexos e são descritos na Tabela 3.

Tabela 3. Tipos construídos da linguagem IDL do CORBA

Tipo	**Exemplos**	**Uso**
Sequence	typedef sequence <Shape, 100> All; typedef sequence <Shape> All sequências limitadas e não limitadas de Shapes	Define um tipo para uma sequência de elementos de comprimento variável de um tipo IDL especificado. Pode ser especificado um limite superior para o comprimento.
String	string name; typedef string <8> SmallString; sequências limitadas e não limitadas de caracteres	Define uma sequência de caracteres, terminada pelo caractere nulo. Pode ser especificado um limite superior para o comprimento.
Array	typedef octet uniqueId[12]; typedef GraphicalObject GO[10][8]	Define um tipo para uma sequência multidimensional de elementos de comprimento fixo de um tipo IDL especificado.

Record	struct GraphicalObject {
	string type;
	Rectangle enclosing;
	Boolean isFilled;
	};
		Define um tipo para um registro contendo um grupo de entidades relacionadas. Structs são passadas por valor em argumentos e resultados.

Enumerated	enum Rand (Exp, Number, Name);	O tipo enumerado na IDL faz o mapeamento de um nome de tipo para um pequeno conjunto de valores inteiros.

Union	union Exp switch (Rand) {
	case Exp: string vote;
	case Number: long n;
	case Name: string s;
	};
		A união da IDL permite que um tipo de determinado conjunto de tipos seja passado como argumento. O cabeçalho é parametrizado por um enum, que especifica qual membro está em uso.

Fonte: Coulouris, Dollimore e Kindberg (2007)

A IDL suporta ainda um tipo de dados especial chamado "Object", o qual é uma referencia a um objeto remoto. Além disso, dá suporte para o disparo de exceções nos métodos da interface e permite a definição de atributos, que são campos encapsulados automaticamente por métodos "get" e "set".

Na linguagem IDL os métodos definidos nas interfaces podem receber parâmetros e retornar valores. Estes valores podem ser de qualquer um dos tipos de dados primitivos ou construídos. "Cada parâmetro é marcado como sendo de entrada, saída ou ambos, usando-se

as palavras-chaves in, out ou inout" (COULOURIS, DOLLIMORE e KINDBERG, 2007, p.712). Os parâmetros de entrada (in) são enviados ao objeto remoto no servidor. Os parâmetros de saída (out) são recebidos pelo cliente na resposta da invocação de um método no objeto remoto. Os parâmetros de entrada-saída (inout) "trafegam" em ambos os sentidos.

Por último, a linguagem IDL permite que suas interfaces e definições de tipos sejam agrupadas em unidades lógicas denominadas de módulos (modules). Os módulos funcionam de forma semelhante ao namespace do C++ e ao package do Java, definindo um escopo e impedindo a colisão entre nomes iguais.

Mapeamento de linguagens

"Uma vez que as interfaces estejam definidas, estas devem ser compiladas de modo que os tipos de dados e as operações sejam mapeados na linguagem de programação utilizada para o desenvolvimento de servidores e clientes" (NARDI, 2005, p.16).

Este mapeamento é feito utilizando-se o compilador IDL da implementação CORBA escolhida. Existem padrões de mapeamentos definidos para várias linguagens: Java, C, C++, SmallTalk, COBOL, Ada, Lisp, Python, entre outras. No entanto, uma implementação de CORBA pode oferecer suporte para apenas uma ou algumas dessas linguagens.

Stub

Os Stubs são gerados na linguagem de programação do cliente durante o processo de compilação da IDL. Em linguagens orientadas a objetos eles correspondem a uma classe proxy cujo papel é empacotar os argumentos da requisição ao objeto remoto e desempacotar as exceções e resultados da resposta.

Esqueleto (Skeleton)

Os esqueletos são gerados na linguagem de programação do servidor durante o processo de compilação da IDL. Os esqueletos realizam o processo inverso ao dos Stubs, desempacotando os argumentos recebidos na requisição e empacotando as exceções e valores de retorno na resposta.

Adaptador de Objeto (Object Adapter)

Adaptadores de objeto são a interface através da qual as implementações de objetos CORBA têm acesso aos serviços oferecidos pelos ORBs. Entre estes serviços estão: geração e interpretação das referências de objeto, invocação de método, ativação e desativação das implementações de objetos e mapeamento entre as referências de objeto e suas implementações.

3.2 ELABORAÇÃO DO PROJETO

3.2.1 Requisitos não funcionais

O mecanismo de integração será desenvolvido durante o TCC II e deverá atender os seguintes requisitos não funcionais:

- O sistema deverá ser desenvolvido utilizando a arquitetura CORBA;
- O sistema deverá suportar a integração com pelo menos uma linguagem de programação diferente da linguagem Java; e
- O sistema deverá ser implementado de forma que o PortugolCore não se torne dependente dele para funcionar.

Para atender ao último requisito, será necessário separar o código-fonte do mecanismo de integração em "módulos de integração". Haverá um módulo de integração para cada linguagem de programação e cada um desses módulos deverá ser codificado como um projeto independente, de forma que possa ser compilado em uma biblioteca a ser incluída somente nas aplicações que utilizarem o mecanismo de integração. A Figura 9 ilustra este esquema.

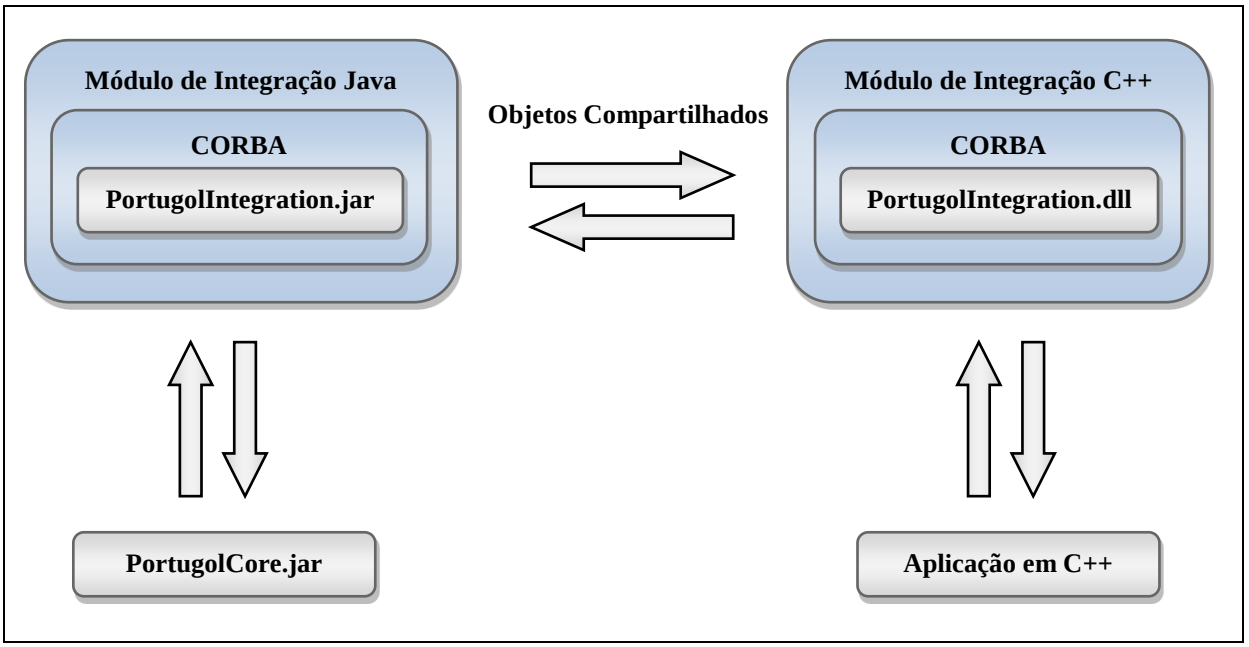

Figura 9. Estrutura do mecanismo de integração

Conforme ilustrado na figura, o módulo de integração Java deverá ser responsável por acessar os objetos do PortugolCore e disponibilizá-los aos outros módulos através do CORBA. Os demais módulos deverão receber os objetos do PortugolCore através do CORBA e disponibilizá-los às aplicações.

3.2.2 Integração em CORBA

Para a criação dos módulos de integração, deverão ser realizados os passos definidos por Borsoi (2004, p. 8) para a construção de programas em CORBA. Estes passos são transcritos de forma generalizada a seguir:

1. Primeiro passo: definir na linguagem IDL as interfaces dos objetos que estarão acessíveis nas aplicações clientes e servidoras.
2. Segundo passo: compilar as interfaces IDL nas linguagens de programação do servidor e do cliente. A partir desta compilação serão gerados arquivos de código-fonte para cada interface IDL definida.
3. Terceiro passo: incluir os arquivos de código-fonte nas aplicações do cliente e do servidor e, implementar no servidor as funcionalidades dos objetos, utilizando as interfaces e classes definidas nesses arquivos.

No primeiro passo, todos os objetos que serão compartilhados entre o PotugolCore e as aplicações (consultar a Tabela 7 no APÊNDICE B) deverão ser transcritos em interfaces IDL. Para compreender melhor, o Quadro 9 e o Quadro 10 apresentam o código-fonte da classe “Programa” do PortugolCore e sua interface IDL equivalente.

```
package br.univali.portugol.nucleo;
[...]

public final class Programa
{
    public void executar(final String[] parametros) {  [...] }

    public void interromper()  { [...] }

    public void setFuncaoInicial(String funcaoInicial) { [...] }

    public void setEntrada(Entrada entrada) { [...] }

    public void setSaida(Saida saida)  { [...] }

    [...]
}
```

Quadro 9. Classe "Programa" do PortugolCore

```
module br
{
    module univali
    {
        module portugol
        {
            module nucleo
            {
                  interface Programa
                  {
                        void executar(in string[] parametros);

                        void interromper();

                        void setFuncaoInicial(in string funcaoInicial);

                        void setEntrada(in Entrada entrada);

                        void setSaida(in Saida saida);
                  };
              };
         };
     };
};
```

Quadro 10. Interface IDL da classe "Programa"

No segundo passo, as interfaces criadas deverão ser compiladas na linguagem Java e em cada uma das linguagens de programação que serão suportadas pelo mecanismo de integração. Após a compilação das interfaces IDL, serão gerados vários arquivos de código-

fonte na linguagem-alvo selecionada. A Tabela 4 apresenta um exemplo genérico dos arquivos de código-fonte que são gerados ao compilar uma interface IDL em Java.

Tabela 4. Arquivos gerados pelo compilador IDL

Código-fonte	**Descrição**	**Local**
ObjetoOperations.java	Interface do objeto CORBA: Expõe as operações definidas na interface IDL.	Cliente / Servidor
Objeto.java	Objeto CORBA: Implementa as operações da interface IDL.	Cliente / Servidor
_ObjetoStub.java	Stub: empacota os argumentos da requisição.	Cliente
ObjetoPOA.java	Esqueleto: desempacota os argumentos da requisição.	Servidor
ObjetoHelper.java	Classe auxiliar: converte referências remotas de objetos CORBA para a classe equivalente	Cliente / Servidor
ObjetoHolder.java	Classe portadora: trata os argumentos out e inout da IDL, pois estes não podem ser mapeados diretamente para Java	Cliente / Servidor

Cada arquivo gerado implementa a funcionalidade de um dos componentes da arquitetura CORBA. O diagrama de classes no APÊNDICE A, demonstra o relacionamento entre as classes e interfaces geradas neste exemplo. Este mesmo modelo de relacionamento se aplicará ás classes e interfaces geradas a partir do objetos compartilhados do PortugolCore.

No terceiro passo, os arquivos gerados deverão ser incluídos nos projetos dos módulos de integração e os objetos compartilhados do PortugolCore, deverão ser modificados para implementarem as interfaces definidas nestes arquivos. Por exemplo, a classe “Programa” demonstrada anteriormente, seria modificada para implementar a interface definida no arquivo “ProgramaOperations;java”. A Figura 10 ilustra o processo de construção do módulo de integração.

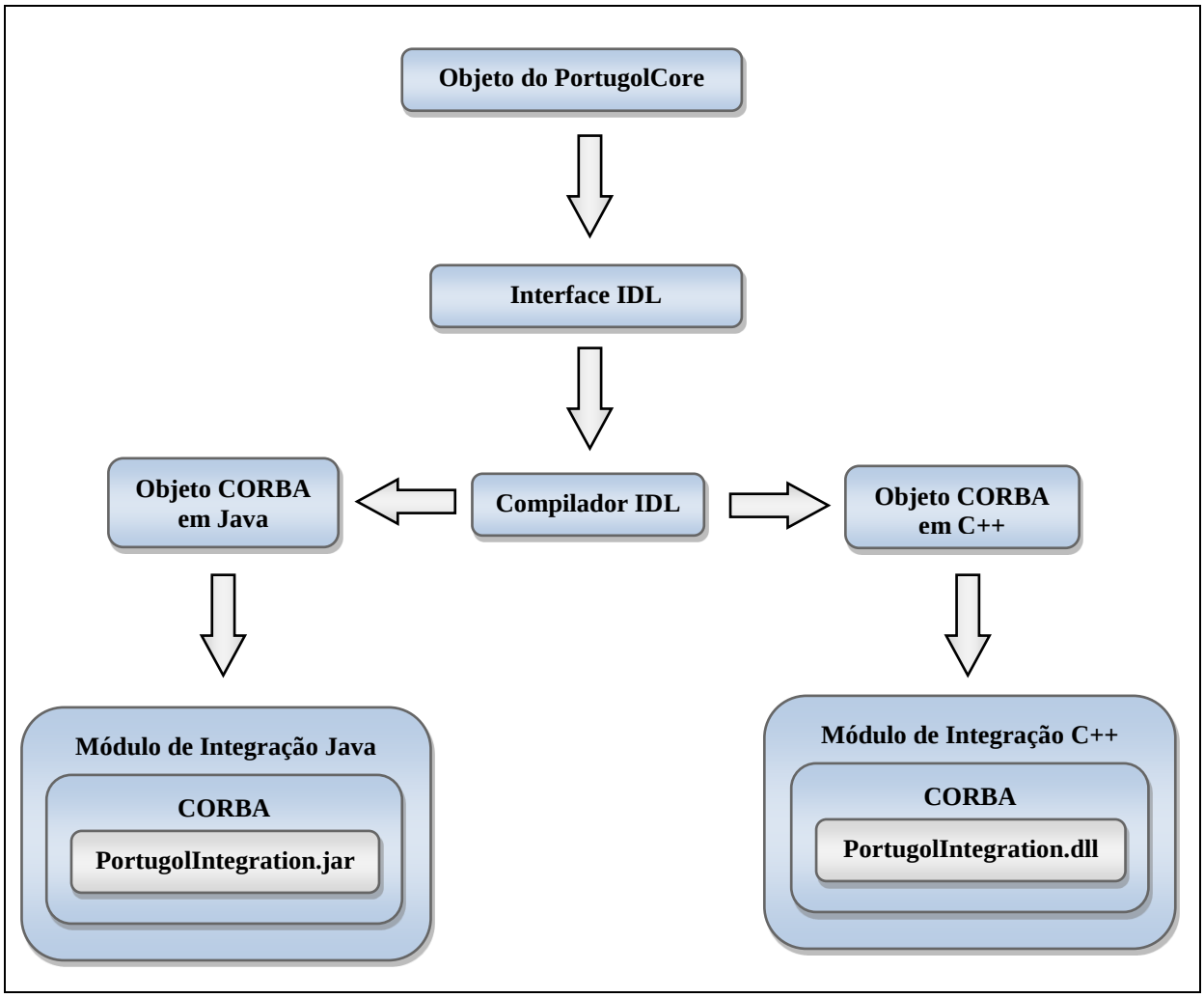

Figura 10. Passos para a implementação do sistema

Para compilar as interfaces IDL será necessário adquirir implementações do CORBA para cada uma das linguagens a serem suportadas. Cummins (2002, p.V) estima que existam no mercado mais de 70 implementações da arquitetura CORBA (ORBs), a maioria delas preparadas para uso geral. A Tabela 5 exibe uma relação de algumas implementações CORBA com que poderão ser utilizadas no desenvolvimento do projeto.

Tabela 5. Implementações CORBA

Nome	**Linguagens Suportadas**	**Licença**
Engine Room CORBA	C, C++, Java, Perl	Livre
Fnorb	Python	OpenSource
ISP C++ ORB	C++	---
JacORB	Java	LGPL
LuaORB	Lua	Livre
MICO	C++	OpenSource (GNU)
MTDORB	ObjectPascal (Delphi/Kylix)	BSD
omniORB	C++, Python	GPL/LGPL

Nome	Linguagens Suportadas	Licença
opalORB	Perl	OpenSource
OpenORB	Java	BSD
ORBit	C, Perl	GPL/LGPL
R2CORBA	Ruby	OpenSource
Remoting.Corba	C#	OpenSource
TAO (The ACE ORB)	C++	OpenSource
VBOrb	VisualBasic	GPL

A implementação do sistema proposto foi realizada seguindo o projeto elaborado anteriormente. Para que o sistema pudesse ser construído e se comportasse da forma ilustrada na Figura 9, foi necessário dividir o mecanismo de integração em duas partes: (i) módulo Java e; (ii) módulo C#.

3.2.3 Módulo Java

Dentro do mecanismo de integração, o módulo Java possui três funções principais: (i) gerenciar o serviço de nomes necessário para o funcionamento da arquitetura CORBA; (ii) gerenciar o ciclo de vida do ORB Java; e (iii) empacotar os objetos Java vindos do PortugolCore dentro dos objetos CORBA.

Para que a arquitetura CORBA possa funcionar, ela depende que um serviço de nomes (Naming Service) esteja ativo na máquina servidora. O serviço de nomes é responsável por criar um identificador único (IOR - Interoperable Object Reference) para cada objeto CORBA instanciado, o qual será utilizado para localizar os objetos durante as transações. Várias implementações do CORBA proveem este serviço, e no caso do Java não é diferente. Todas as versões do Java possuem este serviço implementado nativamente na forma de um arquivo executável (orbd.exe no Windows, orbd no Linux) distribuído juntamente com o JDK ou JRE.

Durante o desenvolvimento, optou-se por utilizar o serviço de nomes nativo do Java, para que o mecanismo de integração não ficasse amarrado a uma implementação específica do CORBA. Desta forma, o módulo Java fica encarregado de inicializar e finalizar o processo orbd no início e no término da execução, utilizando a API Java para criação de processos.

Além do serviço de nomes, a arquitetura CORBA necessita que haja uma implementação do ORB em cada linguagem de programação que for se comunicar com os objetos em CORBA. O Java também possui uma implementação nativa do ORB, disponível através de um conjunto de classes no pacote *"org.omg"*. Novamente o módulo Java fica

encarregado de instanciar estes objetos e, inicializar e finalizar o ORB no início de no término da execução.

Por último, conforme definido nos requisitos não funcionais do projeto, o mecanismo de integração deveria ser implementado sem gerar uma dependência com o código fonte já existente do PortugolCore. Para garantir este requisito, o módulo Java utiliza o padrão de projeto Proxy para empacotar os objetos vindos do PortugolCore dentro de objetos CORBA.

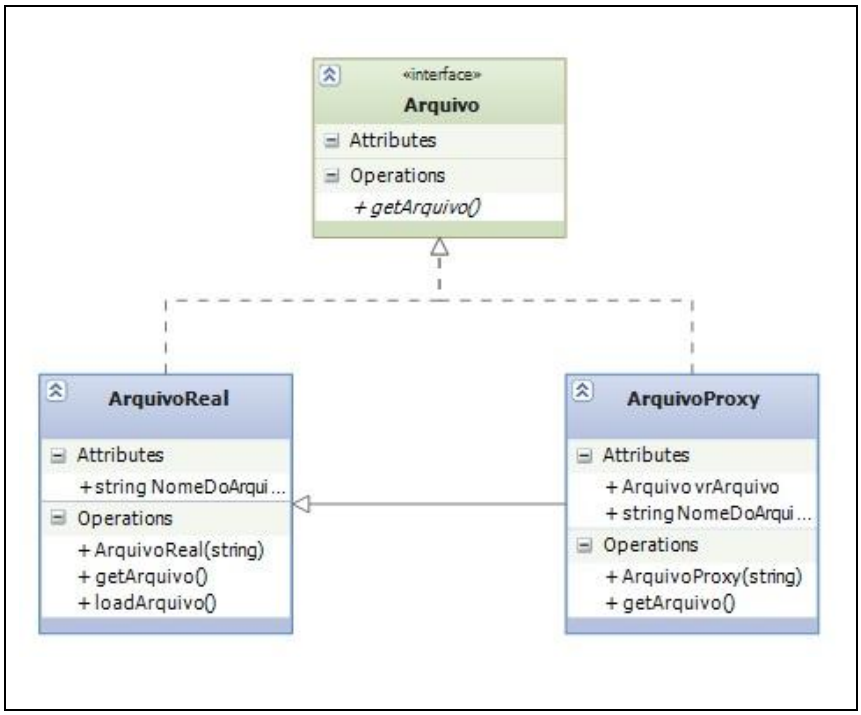

Figura 11. Padrão de projeto Proxy

No padrão Proxy, ilustrado pela Figura 11, os objetos Proxy possuem a mesma interface (expõem os mesmos métodos) e uma instância do objeto que se deseja encapsular. Quando um método é chamado no objeto Proxy, o objeto Proxy invoca o mesmo método na instancia do objeto encapsulado, esta estratégia é denominada delegação.

No módulo Java, os objetos Proxy estendem os objetos POA (Portable Object Adapter) gerados pelo compilador IDL, os quais por sua vez, implementam as interfaces definidas na linguagem IDL. Para cada objeto do PortugolCore a ser compartilhado (consulte a Tabela 7 para ver a relação completa destes objetos), existe uma interface IDL e portanto, um objeto Proxy também.

```
package br.univali.portugol.integracao.asa;
[...]

public final class NoCadeiaProxy extends NoCadeiaPOA
{
    private POA rootPOA;
    private br.univali.portugol.nucleo.asa.NoCadeia noCadeiaPortugol;
    private NoCadeia referencia;
    private TrechoCodigoFonte trechoCodigoFonte;

    private NoCadeiaProxy(br.univali.portugol.nucleo.asa.NoCadeia
noCadeiaPortugol, POA rootPOA)
    {
        this.rootPOA = rootPOA;
        this.noCadeiaPortugol = noCadeiaPortugol;
    }

    public static NoCadeia
empacotar(br.univali.portugol.nucleo.asa.NoCadeia noCadeiaPortugol, POA
rootPOA)
    {
        if (noCadeiaPortugol != null)
        {
            try
            {
                NoCadeiaProxy noCadeiaProxy = new
NoCadeiaProxy(noCadeiaPortugol, rootPOA);
                org.omg.CORBA.Object referencia =
rootPOA.servant_to_reference(noCadeiaProxy);
                NoCadeia noCadeiaCORBA = NoCadeiaHelper.narrow(referencia);
                noCadeiaProxy.referencia = noCadeiaCORBA;

                return noCadeiaCORBA;
            }
            catch (WrongPolicy excecao)
            {
                throw new UNKNOWN(excecao.getMessage());
            }
            catch (ServantNotActive excecao)
            {
                throw new UNKNOWN(excecao.getMessage());
            }
        }

        return null;
    }

    @Override
    public String getValor()
    {
        return noCadeiaPortugol.getValor();
    }

    @Override
    public TrechoCodigoFonte getTrechoCodigoFonte()
    {
        if (this.trechoCodigoFonte == null)
        {
            this.trechoCodigoFonte =
TrechoCodigoFonteProxy.empacotar(this.noCadeiaPortugol.getTrechoCodigoFonte
(), rootPOA);
```

```
        }

        return this.trechoCodigoFonte;
    }

    @Override
    public Any aceitar(VisitanteASA visitante) throws ExcecaoVisitaASA
    {
        return visitante.visitarNoCadeia(this.referencia);
    }

    @Override
    public boolean estaEntreParentesis()
    {
        return noCadeiaPortugol.estaEntreParentesis();
    }
}
```

Quadro 11. Exemplo de objeto Proxy do módulo Java

O módulo Java, ao ser compilado, gera um arquivo JAR, que deverá ser incluído nas aplicações que utilizarem os serviços do PortugolCore, conforme ilustrado na Figura 9.

3.2.4 Módulo C#

Dentro do mecanismo de integração, o módulo C# possui duas funções principais: (i) inicializar e finalizar o módulo Java; e (ii) gerenciar o ciclo de vida do ORB C#.

Como descrito anteriormente, o módulo Java ao ser compilado, gera um arquivo JAR, que deverá ser incluído juntamente com o modulo C# na aplicação. Uma das funções do módulo C# é inicializar o módulo Java de forma transparente à aplicação. Isto é feito através da criação de um processo Java no Sistema Operacional (java.exe no Windows, java no Linux) passando como argumento o arquivo JAR do módulo Java. A criação do processo é feita em tempo de execução utilizando a API C# e equivale à seguinte chamada de sistema: “java –jar caminho/do/jar/portuol-integracao.jar”.

O módulo C# é responsável ainda, por inicializar e finalizar o ORB C# no início e no término da execução do sistema. Para este projeto, a biblioteca IIOP.NET foi escolhida como implementação do ORB em C#. Para que o módulo C# possa utilizá-la, dois arquivos devem ser incluídos na aplicação: “portugol-integracao.dll” e “IIOPChannel.dll”. O primeiro arquivo é gerado pelo compilador IDL da biblioteca e contém as interfaces definidas no arquivo IDL. O segundo arquivo é gerado durante a compilação da biblioteca e contém a implementação do ORB.

Ao ser compilado, o módulo C# também gera um arquivo (“portugol-integracao-csharp.dll”) que deverá ser incluído na aplicação. No total, são 7 arquivos que devem ser incluídos na aplicação para que o mecanismo de integração possa funcionar. A Tabela 6 mostra a relação destes arquivos.

Tabela 6: Arquivos necessários para o funcionamento do mecanismo de integração

Nome do Arquivo	**Descrição**
antlr-runtime-3.4.jar	Biblioteca ANTLR, utilizada pelo PortugolCore para realizar o parsing do código fonte
IIOPChannel.dll	Biblioteca IIOP.NET. Contém a implementação C# do ORB.
portugol-integracao.dll	Gerado pelo compilador IDL da biblioteca IIOP.NET. Contém as interfaces dos objetos.
portugol-integracao.jar	O módulo de integração Java.
portugol-integracao-csharp.dll	O módulo de integração C#.
portugol-nucleo.jar	Arquivo JAR do PortugolCore. Contém os serviços disponibilizados pelo Portugol.
portugol-relator-erros.jar	Biblioteca utilizada pelo PortugolCore para gerar relatórios de erros para detecção de bugs.

Por último, o diagrama da Figura 12 mostra o funcionamento do mecanismo de integração como um todo, bem como a comunicação entre os módulos Java e C#.

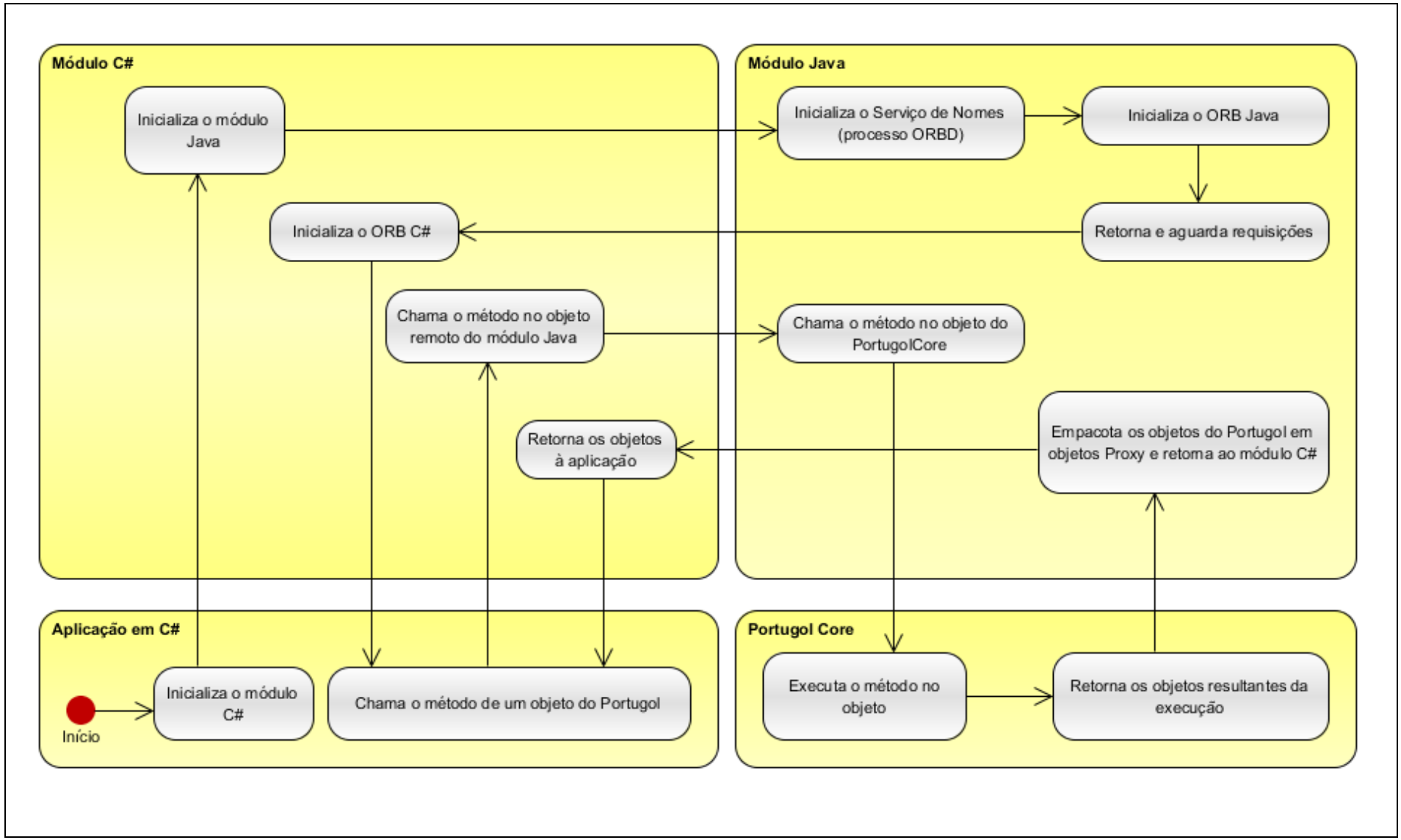

Figura 12: Funcionamento do mecanismo de integração

3.3 ESTUDO DE CASO – FERRAMENTA BIPIDE

Para validar a implementação do mecanismo de integração desenvolvido, foi realizado um estudo de caso com a ferramenta BIPIDE. O estudo de caso foi realizado com o intuito de identificar e corrigir bugs no mecanismo de integração, de forma a torná-lo passível de utilização.

3.3.1 Um pouco sobre o BIPIDE

O BIPIDE é uma ferramenta desenvolvida na UNIVALI pelo mestrando Paulo Vinícius Vieira utilizando a linguagem C#. O objetivo principal da ferramenta é simular a execução de algoritmos no processador BIP (Basic Instruction Processor). Maiores detalhes sobre o BIPIDE podem ser obtidos em (Viera, 2009 e Vieira, Raabe e Zeferino, 2011).

Para isto, o BIPIDE permite a programação de algoritmos escritos em uma variação da linguagem Portugol que suporta um conjunto reduzido de comandos. Os programas escritos em Portugol são traduzidos na linguagem assembly do BIP e então simulados. A Figura 13, mostra o BIPIDE em execução.

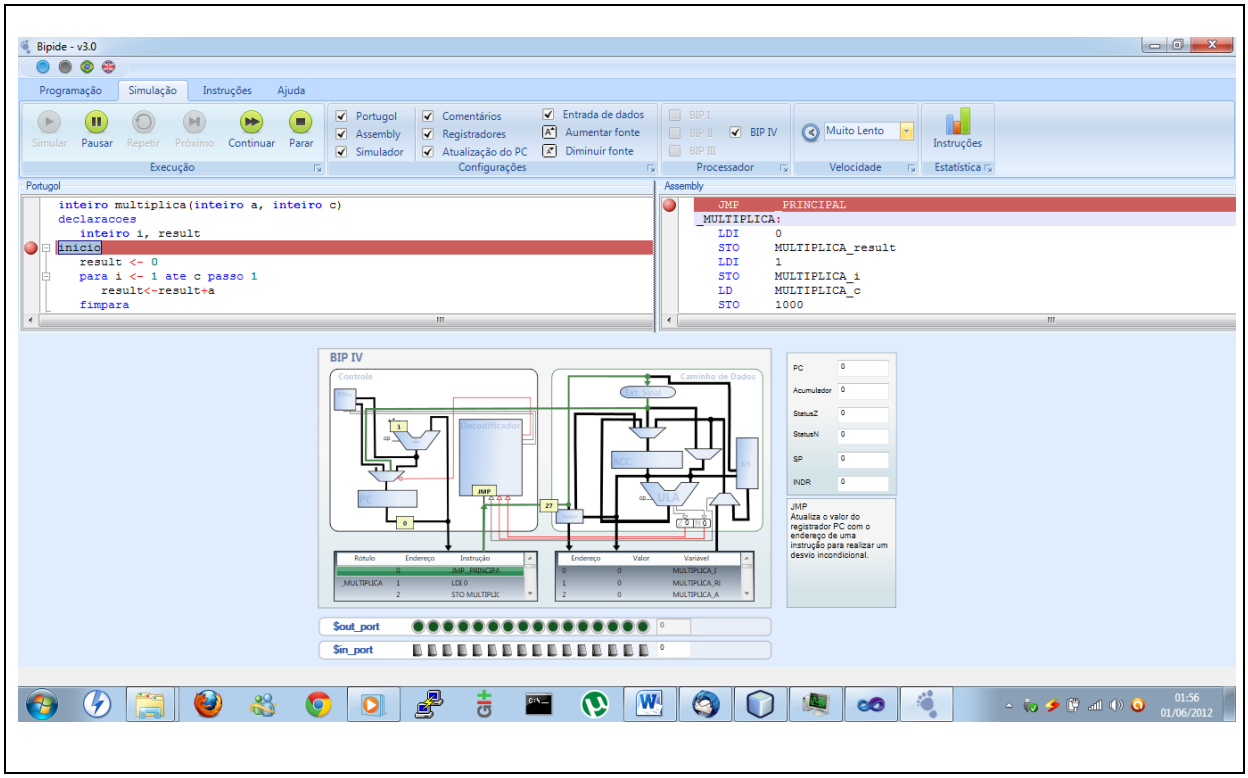

Figura 13: Ferramenta BIPIDE em execução

O BIPIDE foi escolhido para o estudo de caso por dois motivos específicos. Em primeiro lugar, por ser uma aplicação que já está em uso pelos alunos. Isto permite validar o potencial de integração do sistema desenvolvido, com ferramentas já existentes, e assim, agregar valor ao trabalho.

Em segundo lugar, por existir o interesse do grupo de pesquisa responsável pelo projeto, em adotar a nova sintaxe do Portugol 2.0 na ferramenta. Neste caso, a possibilidade de integração diminuiria o esforço necessário para esta adaptação, já que boa parte das funcionalidades desejadas seriam disponibilizadas pelo PortugolCore e não necessitariam ser implementadas.

3.3.2 Testes realizados e resultados obtidos

Foi estabelecido como critério de validação do mecanismo de integração, que todas as funcionalidades do PortugolCore pudessem ser utilizadas sem erros dentro da ferramenta. Se qualquer uma das funcionalidades apresentasse erros, a implementação seria invalidada.

Com base neste critério, cada uma das funcionalidades do PortugolCore foi submetida a testes dentro do BIPIDE, a saber: análise e tratamento de erros, execução de programas e geração de código intermediário (ASA).

Para testar a análise e tratamento de erros do Portugol, foram escritos algoritmos em Portugol contendo vários tipos de erros sintáticos e semânticos. Esta funcionalidade seria considerada validada caso fosse possível submeter estes algoritmos à análise do Portugol, capturar a lista de erros sintáticos e semânticos gerados durante a análise e exibi-los na interface da ferramenta. Conforme mostra a Figura 14, a funcionalidade passou em todos os testes realizados e, portanto, foi validada.

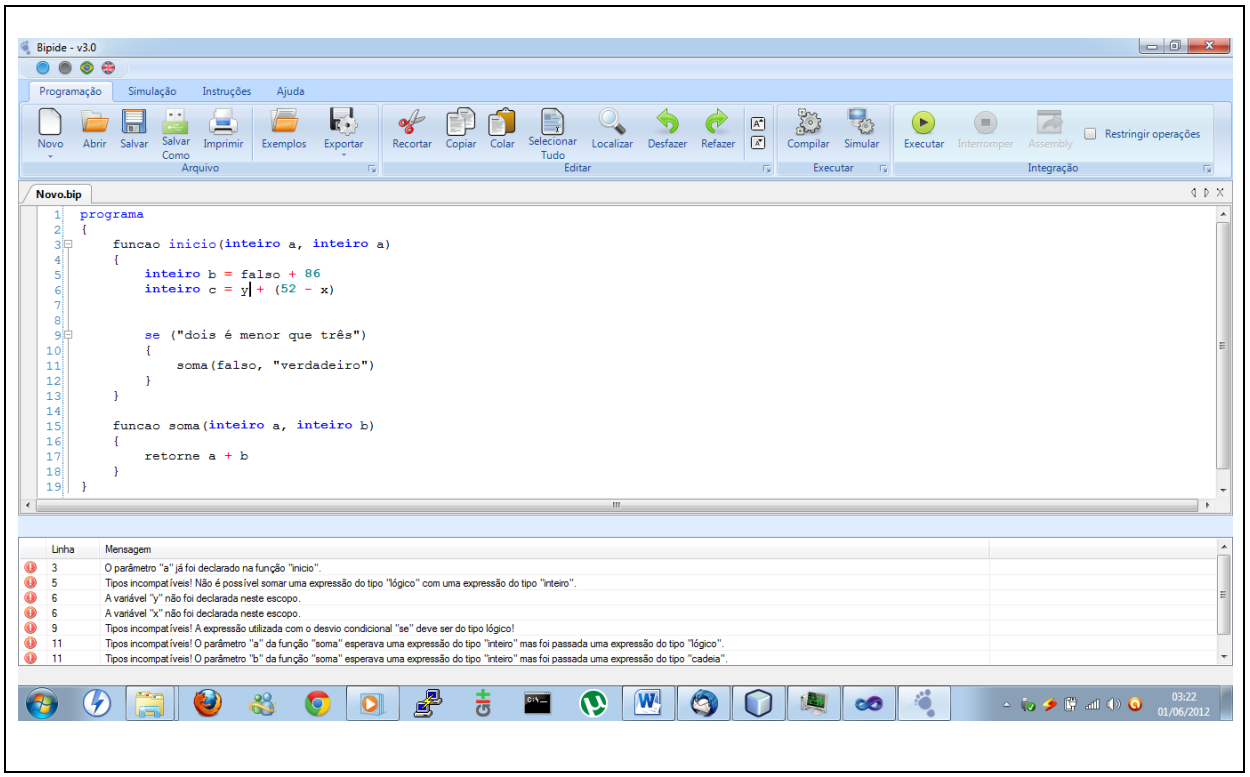

Figura 14: Teste da análise de erros do PortugolCore no BIPIDE

Para testar a execução de programas do PortugolCore, foram escritos algoritmos em Portugol livres de erros e contendo instruções de entrada e saída de dados. Esta funcionalidade seria considerada validada caso atendesse a dois requisitos: (i) se fosse possível inicializar e finalizar a execução de um programa através de botões na interface da ferramenta; e (ii) se fosse possível ler dados de entrada do usuário e escrever os dados de saída na interface da ferramenta.

Para possibilitar este teste, a ferramenta foi adaptada para implementar as interfaces *Entrada*, *Saida* e *ObservadorExecucao*, necessárias à esta funcionalidade. A interface de entrada de dados foi implementada utilizando um formulário simples, com uma caixa de texto e um botão de confirmação. A interface de saída de dados foi implementada utilizando um formulário com uma caixa de texto de fundo preto e fonte branca, simulando um console.

Nos primeiros testes realizados, a execução do programa iniciava corretamente, mas na hora de realizar a entrada e saída dos dados ocorriam exceções no CORBA. Após exaustivos testes e várias investigações, descobriu-se que, o problema estava em implementar as interfaces *Entrada*, *Saida* e *ObservadorExecucao* em uma única classe. A solução adotada foi implementar cada uma destas interfaces em uma classe diferente. O motivo deste

comportamento ainda não foi descoberto, mas após fazer esta alteração a funcionalidade passou em todos os testes, sendo, portanto validada, conforme mostra a Figura 15.

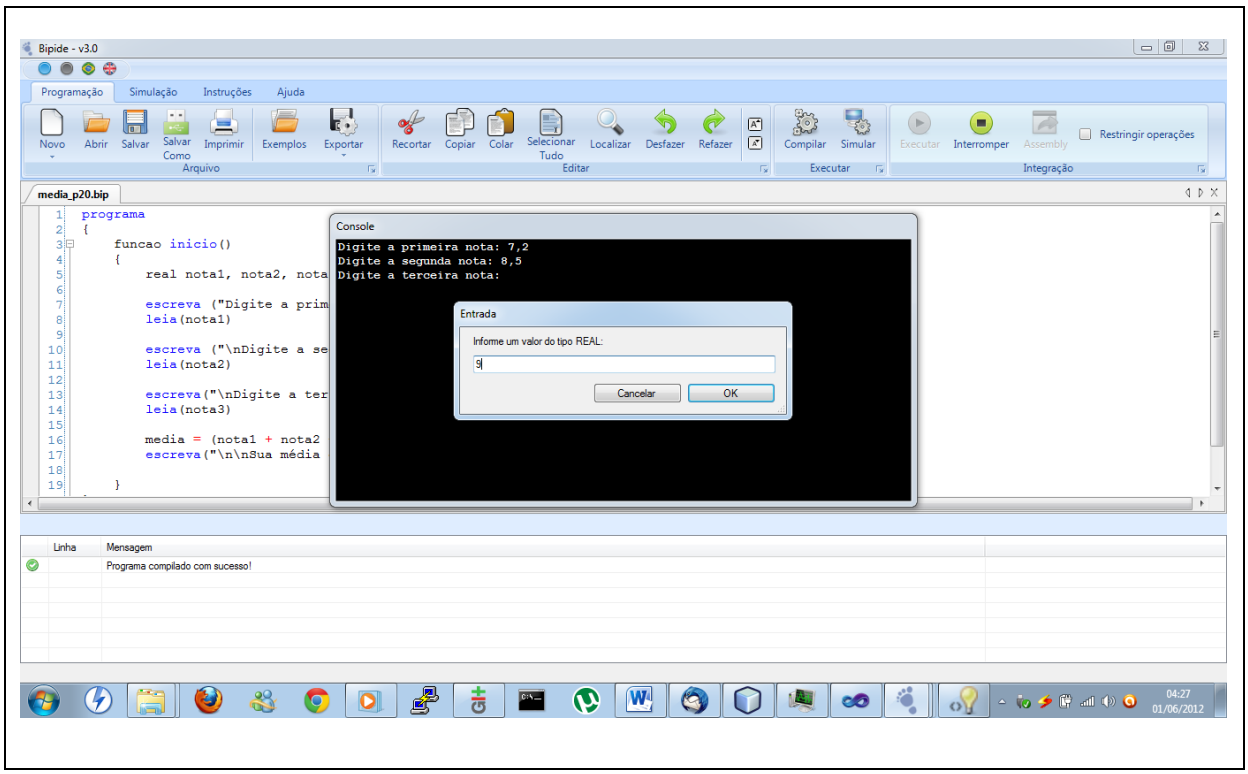

Figura 15: Teste da execução de programas no BIPIDE

Outro problema encontrado na execução dos programas foi o desempenho na saída de dados. O BIPIDE levou cerca de 20 segundos para executar um laço de repetição com 1000 iterações escrevendo na saída de dados. O mesmo algoritmo levou cerca de 150 milissegundos para executar no PortugolStudio. Será necessário realizar um estudo mais aprofundado para determinar as causas deste problema, no entanto, isto não invalida a implementação, já que o foco deste trabalho está na interoperabilidade e não no desempenho.

Para testar a geração de código intermediário (ASA) do PortugolCore, foram escritos em C# dois algoritmos de caminhamento na ASA. Os algoritmos implementam a interface *VisitanteASA* definida no PortugolCore, e portanto, utilizam o padrão de projeto Visitor para percorrer a ASA. Esta funcionalidade seria considerada validada se os algoritmos conseguissem percorrer todos os nós da árvore necessários para realizar seu processamento, sem que fossem geradas exceções do CORBA.

O primeiro algoritmo funciona como uma extensão do analisador semântico do Portugol. Ele percorre a ASA procurando nós que representam instruções não suportadas pelo

processador BIP e para cada nó encontrado, gera um erro que será exibido na janela principal do BIPIDE. O segundo algoritmo percorre a ASA gerando o código assembly para o processador BIP do programa escrito em Portugol.

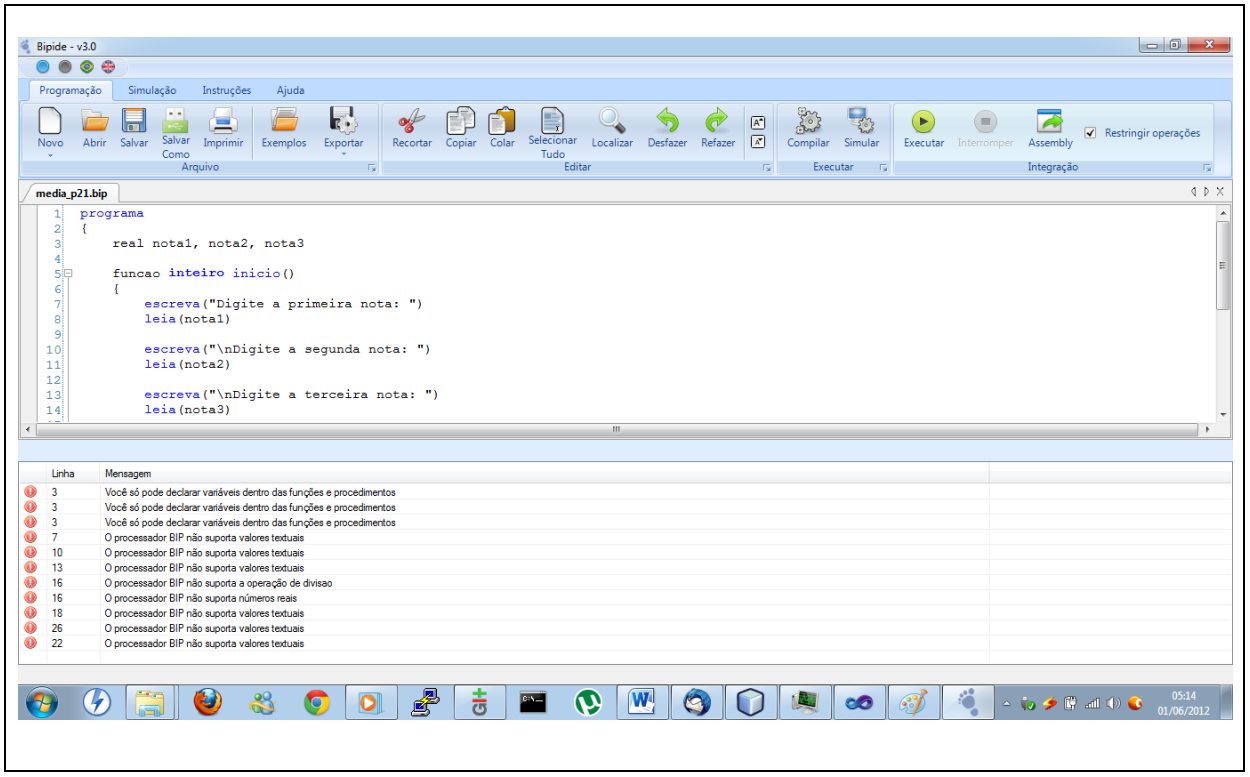

Figura 16: Teste da geração de código intermediário no BIPIDE (algoritmo 1)

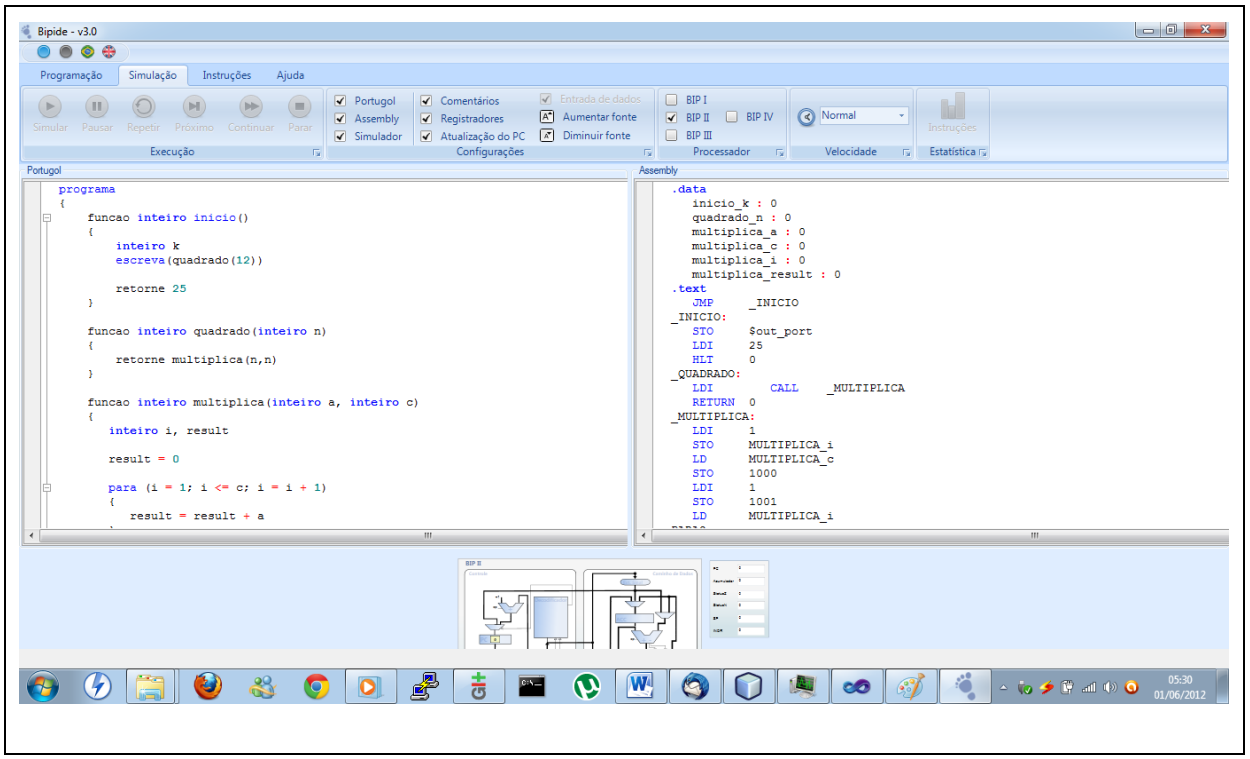

Figura 17: Teste da geração de código intermediário no BIPIDE (algoritmo 2)

Os dois algoritmos foram executados com vários programas de teste escritos em Portugol e conseguiram percorrer corretamente a ASA gerando o resultado esperado, validando portanto esta funcionalidade, conforme mostram a Figura 16 e a Figura 17.

Não foram encontrados problemas durante os testes da funcionalidade de geração de código intermediário. Faz-se aqui a observação de que os algoritmos criados para percorrer a ASA foram desenvolvidos para testes e, portanto são apenas parcialmente funcionais. O algoritmo para geração de código assembly, por exemplo, só consegue gerar o assembly das instruções mais simples do Portugol.

Durante o desenvolvimento, foram identificadas algumas limitações do módulo C# ocasionadas pela utilização da biblioteca IIOP.NET. A primeira limitação está relacionada ao fato de a biblioteca ter sido construída em cima da versão 2.0 da plataforma .NET. Isto dificulta ou impossibilita a utilização do módulo de integração com aplicações desenvolvidas em versões anteriores da plataforma, pois obriga que estas sejam migradas para a versão 2.0.

Outra limitação está relacionada ao compilador IDL da biblioteca, que ao ser executado gera a DLL (Dynamic Link Library) contendo as interfaces para os objetos do PortugolCore. O problema é que a DLL é gerada na versão mais atual da plataforma .NET que estiver em uso na máquina. Durante o desenvolvimento do módulo C#, por exemplo, a versão da plataforma instalada na máquina era a versão 4.0. Isto fez com que o código fonte do BIPIDE (que usava a versão 3.5) tivesse que ser migrado para a versão 4.0 para possibilitar a utilização da biblioteca. Aparentemente não há como alterar este comportamento a não ser modificando o código fonte da biblioteca. Uma possível solução, embora trabalhosa, seria executar o compilador IDL em várias máquinas, cada uma com uma versão da plataforma instalada.

No mais, todos os testes realizados com as funcionalidades disponibilizadas pelo PortugolCore foram bem sucedidos. Sendo assim, o mecanismo de integração foi validado e já está pronto para ser utilizado.

4. CONCLUSÕES

Neste trabalho foi proposta a adaptação do PortugolCore para permitir sua integração com ferramentas escritas em outras linguagens de programação. Esta adaptação foi proposta para aproveitar o potencial do PortugolCore e suas funcionalidades e assim, contribuir com outros projetos e grupos de pesquisa.

Para alcançar tal objetivo, primeiramente foram estabelecidos todos os requisitos de integração que se esperava obter destas tecnologias, a saber: interoperabilidade, reusabilidade, licença e documentação. Tais requisitos foram definidos a partir das necessidades do projeto.

Com os requisitos já estabelecidos, foi realizada uma pesquisa bibliográfica acerca das tecnologias de integração existentes, buscando identificar a melhor solução a ser adotada para a implementação do sistema. Considerou-se que seria escolhida a tecnologia que atendesse a maior parte destes requisitos. A partir dos dados levantados durante a pesquisa bibliográfica, foi realizada uma análise comparativa das tecnologias e a arquitetura CORBA foi escolhida.

Em seguida foi feito um estudo mais aprofundado da arquitetura CORBA visando compreender melhor os detalhes da arquitetura, e assim, reunir o conhecimento necessário para a elaboração do projeto. Após o estudo, o projeto do mecanismo de integração foi elaborado, e nele foi definido que o mecanismo seria dividido em módulos de integração, um para cada linguagem de programação a ser suportada. Também foi estabelecido que o mecanismo de integração deveria ser implementado sem tornar o PortugolCore dependente dele.

A implementação do mecanismo foi realizada seguindo o projeto elaborado e observando todos os requisitos estabelecidos. Desta forma, o sistema foi dividido em dois módulos de integração: um módulo Java e um módulo C#. Para a implementação do módulo Java, optou-se por utilizar as implementações CORBA existentes na plataforma, a fim de não gerar dependências externas. Já na implementação do módulo C#, foi escolhida a biblioteca IIOP.NET por ser desenvolvida com os recursos nativos da plataforma .NET e pela facilidade na sua utilização. Também foram observadas boas práticas de programação no desenvolvimento, como o padrão de projeto Proxy, que foi empregado no módulo Java para garantir ao PortugolCore a independência do mecanismo de integração.

Para validar a implementação realizada, foi feito um estudo de caso com a ferramenta BIPIDE, desenvolvida na UNIVALI na linguagem C#. No estudo de caso, a ferramenta foi modificada para utilizar cada uma das funcionalidades disponibilizadas pelo PortugolCore. Os testes revelaram alguns bugs na implementação do mecanismo de integração e algumas limitações no módulo C# impostas pelo uso da biblioteca IIOP.NET. Após as devidas correções, todos os testes foram refeitos e passaram com sucesso, validando a implementação e viabilizando a utilização do mecanismo em aplicações C#.

No TCC I, também havia sido proposto um estudo de caso com uma aplicação a ser desenvolvida para validar a integração com a linguagem C++. Para realizar este estudo de caso era necessário baixar uma implementação CORBA em C++. Nenhuma das implementações pesquisadas possuía disponível uma versão já compilada, sendo necessário baixar o código fonte e compilar utilizando ferramentas como Cygwin, CMake e NMake. Ao tentar compilar as bibliotecas, ocorriam diversos erros de dependência com outras bibliotecas que também deveriam ser baixadas e compiladas. Em virtude do tempo que seria gasto para resolver estas dependências, e que poderiam prejudicar o andamento do trabalho, optou-se juntamente com orientador por não realizar este estudo de caso, deixando-o para um trabalho futuro. No entanto, a não realização deste estudo, não inviabilizou a validação do mecanismo desenvolvido, já que foi possível integrar com a linguagem C#.

Por último, foram criados três artefatos de documentação: (i) uma documentação no formato HTML abordando a sintaxe do Portugol 2.0, (ii) uma documentação em formato Javadoc de todas as classes e pacotes do código fonte do PortugolCore; e (iii) uma documentação com o passo a passo para a integração do PortugolCore com aplicações em C# utilizando o mecanismo de integração desenvolvido. Estas documentações foram publicadas juntamente com o código fonte em um repositório de projetos OpenSource no endereço: https://github.com/UNIVALI-L2S/Portugol.

Todos os objetivos propostos no trabalho foram cumpridos e acredita-se que ele irá contribuir com a comunidade científica e com outros grupos de pesquisa facilitando o desenvolvimento de novas ferramentas de auxílio ao aprendizado de algoritmos. Sugere-se ainda como principais trabalhos futuros: (i) o desenvolvimento de novos módulos de integração para dar suporte a outras linguagens de programação; e (ii) a adaptação do BIPIDE para utilizar efetivamente a linguagem Portugol 2.0.

REFERÊNCIAS BIBLIOGRÁFICAS

AHO, A. V.; LAM, M. S.; SETHI, R.; ULLMAN, J. D. **Compiladores:** princípios, técnicas e ferramentas. 3.ed. Editora Pearson, 2007.

BISHOP, J.; HORSPOOL, R. N.; WORRAL, B. **Experience in integrating Java with C# and .NET.** Concurrency Computat.: Pract. Exper. 2003; 00:1-18.

BORSOI, T. B.; SCHULTZ, R. E. O. **Estudo comparativo entre CORBA e Java RMI**. In: Congresso Anual de Tecnologia de Informação. 2004, São Paulo. SP, 2004.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Sistemas distribuídos:** conceitos e projeto. 4.ed. Porto Alegre: Bookman, 2007.

CUMMINS, F. A. **Integração de sistemas:** EAI – Enterprise Application Integration: Arquiteturas para integração de sistemas e aplicações corporativas. 1.ed. Rio de Janeiro: Campus, 2002.

HOSTINS, H.; RAABE, A. L. A. **Auxiliando a aprendizagem de algoritmos com a ferramenta Webportugol**. In: WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO - CONGRESSO DA SBC, 27., 2007, Rio de Janeiro. Anais... Rio de Janeiro: SBC, 2007. p. 96-105.

MIRANDA, E. M. **Uma ferramenta de apoio ao processo de aprendizagem de algoritmos**. 2004. Dissertação de Mestrado (Programa de Pós-Graduação em Ciência da Computação) - Universidade Federal de Santa Catarina, Florianópolis, 2004.

NARDI, A. R. **Componentes CORBA**. 2003. Dissertação de Mestrado (Ciência da Computação) – Universidade de São Paulo, São Paulo, 2003.

RAABE, A. L. A.; SILVA, J. M. C. **Um ambiente para atendimento às dificuldades de aprendizagem de algoritmos**. In: XIII WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO, 2005, São Leopoldo. RS, 2005.

SANTOS JUNIOR, A. L. **Integração de sistemas com Java**. 1.ed. Rio de Janeiro: Brasport, 2007.

SEBESTA, R. W. **Conceitos de linguagens de programação**. 5.ed. Porto Alegre: Bookman, 2003.

SIQUEIRA, L. L. **Estudo comparativo entre plataformas de suporte a ambientes virtuais distribuídos**. 2005. Dissertação de Mestrado (Ciência da Computação) – Universidade Federal de Uberlândia, Uberlândia, 2005.

TOGNI, J. D.; **Uma Proposta para Auxiliar a Interoperabilidade entre Ambientes e Ferramentas de CAD para Microeletrônica**. 2005. Dissertação de Mestrado (Ciência da Computação) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2005.

VIEIRA, P.; ZEFERINO, C. A.; RAABE, A. L. A. **Bipide: Ambiente de Desenvolvimento Integrado para Utilização dos Processadores BIP no Ensino de Programação**. XX

VIEIRA, Paulo Viniccius. Bipide: Ambiente de Desenvolvimento Integrado para Utilização dos Processadores BIP no Ensino de Programação. 2009. Trabalho de Conclusão de Curso. (Graduação em Ciência da Computação) - Universidade do Vale do Itajaí.

VIEIRA, P. V. ; RAABE, André Luís Alice ; ZEFERINO, Cesar Albenes . Bipide Ambiente de Desenvolvimento Integrado para a Arquitetura dos Processadores BIP. Revista Brasileira de Informática na Educação, v. 18, p. 32-43, 2010.

Simpósio Brasileiro de Informática na Educação - SBIE2009, 2009, Florianópolis - SC. Anais do XX Simpósio Brasileiro de Informática na Educação, 2009. v. 1.

WIKIPEDIA. **Object Request Broker**. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2011. Disponível em: <http://pt.wikipedia.org/w/index.php?title=Object_request_broker&oldid=23572575>. Acesso em: 15 dez. 2011.

APÊNDICES

APÊNDICE A. EXEMPLO DE INTEGRAÇÃO EM CORBA

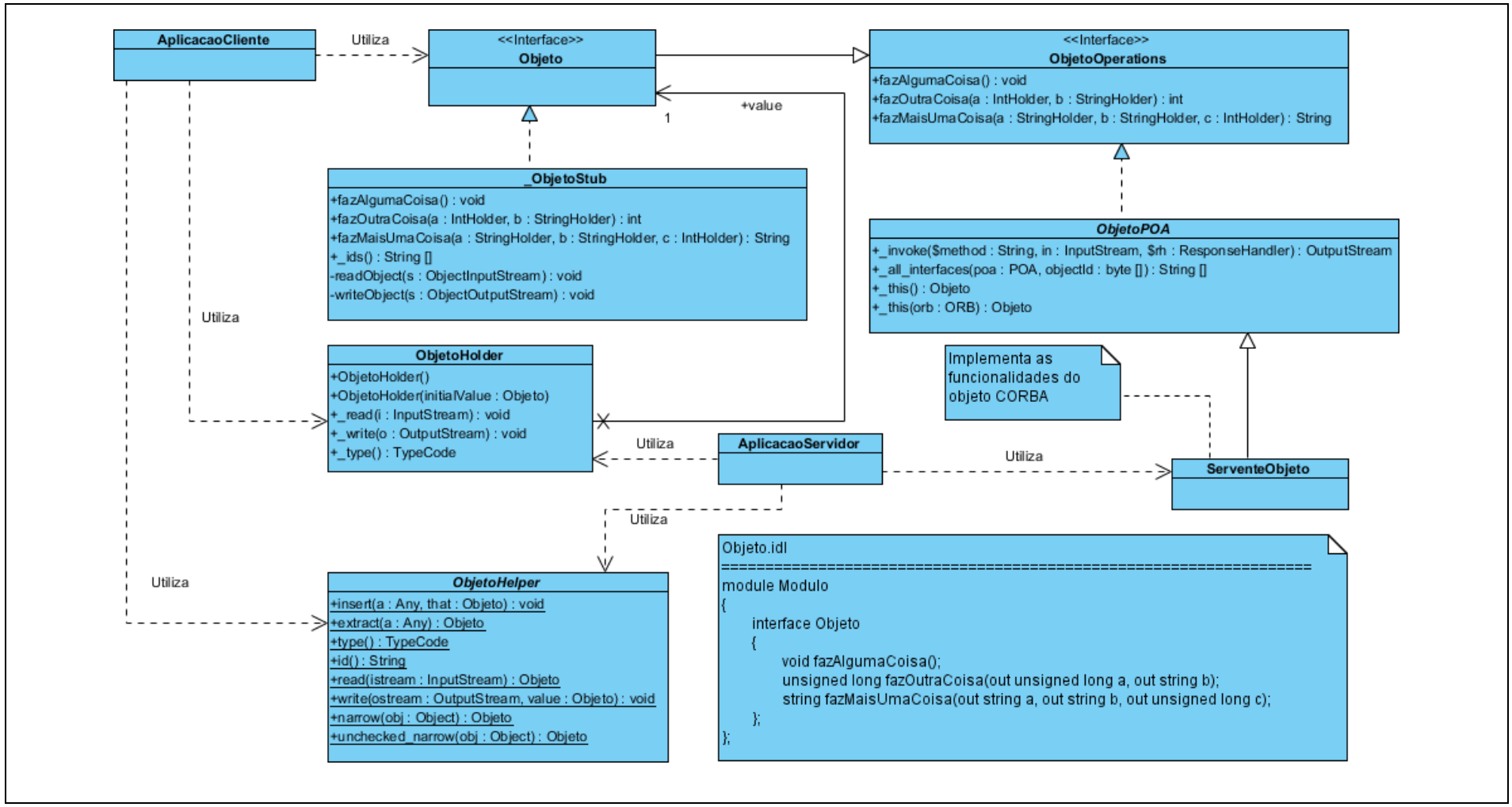

Figura 18. Exemplo de integração em CORBA

APÊNDICE B. OBJETOS DO PORTUGOLCORE

APRESENTAÇÃO

Neste apêndice é apresentada a relação das classes do PortugolCore, que são visíveis a outras ferramentas. No total são oitenta e cinco classes organizadas em ordem alfabética. As colunas da direita mostram a quais funcionalidades disponibilizadas pelo PortugolCore a classe está relacionada.

Legenda
A.T.E - Análise e Tratamento de Erros **G.C.I** - Geração de Código Intermediário **E.P** - Execução de Programas

Tabela 7. Objetos compartilhadas do PortugolCore

Classe	A.T.E	G.C.I	E.P
ArvoreSintaticaAbstrata		✔	✔
ArvoreSintaticaAbstrataPrograma		✔	✔
Aviso	✔		
AvisoAnalise	✔		✔
AvisoValorExpressaoSeraArredondado	✔		
Entrada			✔
Erro	✔		✔
ErroAbreFechaParenteses	✔		
ErroAlgoritmoContemErros	✔		
ErroAnalise	✔		
ErroCadeiaIncompleta	✔		
ErroCompilacao		✔	
ErroEscopoNaoFoiAbertoCorretamente	✔		
ErroEscopoNaoFoiFechadoCorretamente	✔		
ErroEscopoSimples	✔		
ErroExecucao			✔
ErroExecucaoNaoTratado			✔
ErroExpressaoEsperada	✔		
ErroExpressaoIncompleta	✔		
ErroExpressaoTipoLogicoEsperada	✔		
ErroFuncaoInicialNaoDeclarada			✔
ErroIndiceVetorInvalido			✔
ErroNomeIncompativel	✔		
ErroNomeSimboloEstaFaltando	✔		
ErroNumeroParametrosPassadosFuncao	✔		
ErroOperacaoComExpressaoConstante	✔		
ErroOperandoEsquerdoAtribuicaoConstante	✔		

Classe	A.T.E	G.C.I	E.P
ErroPalavraReservadaEstaFaltando	✔		
ErroParametroRedeclarado	✔		
ErroParsingNaoTratado	✔		
ErroSemantico	✔		
ErroSimboloNaoDeclarado	✔		
ErroSimboloRedeclarado	✔		
ErroSintatico	✔		
ErroTipoDeDadoEstaFaltando	✔		
ErroTipoParametroIncompativel	✔		
ErroTiposIncompativeis	✔		
ErroTiposIncompativeis2	✔		
Mensagem	✔		✔
ModoEncerramento			✔
NoBloco		✔	✔
NoCadeia		✔	✔
NoCaracter		✔	✔
NoCaso		✔	✔
NoChamadaFuncao		✔	✔
NoDeclaracao		✔	✔
NoDeclaracaoFuncao		✔	✔
NoDeclaracaoMatriz		✔	✔
NoDeclaracaoVariavel		✔	✔
NoDeclaracaoVetor		✔	✔
NoDecremento		✔	✔
NoEnquanto		✔	✔
NoEscolha		✔	✔
NoExpressao		✔	✔
NoFacaEnquanto		✔	✔
NoIncremento		✔	✔
NoInteiro		✔	✔
NoLogico		✔	✔
NoMatriz		✔	✔
NoMenosUnario		✔	✔
NoNao		✔	✔
NoOperacao		✔	✔
NoPara		✔	✔
NoParametro		✔	✔
NoPare		✔	✔
NoPercorra		✔	✔
NoReal		✔	✔
NoReferencia		✔	✔
NoReferenciaMatriz		✔	✔
NoReferenciaVariavel		✔	✔
NoReferenciaVetor		✔	✔
NoRetorne		✔	✔
NoSe		✔	✔
NoVetor		✔	✔

Classe	**A.T.E**	**G.C.I**	**E.P**
ObservadorExecucao		✔	✔
Operacao		✔	✔
Portugol	✔	✔	✔
Programa		✔	✔
Quantificador		✔	✔
ResultadoAnalise	✔		
ResultadoExecucao			✔
Saida			✔
TipoDado		✔	✔
TrechoCodigoFonte		✔	
VisitanteASA		✔	