5CONTECSI June 04 - 06, 2008 São Paulo, Brazil

5th International Conference on Information Systems and Technology Management
5º Congresso Internacional de Gestão da Tecnologia e Sistema de Informação
De 04 a 06 de Junho de 2008 - São Paulo - Brasil

PS-973

APPLYING THE RESPONSIBILITY-DRIVEN PROCESS TO CREATE A SUBFRAMEWORK FOR FORMULA SYNTACTIC VALIDATION

Rafael Hornung (Universidade Tecnológica Federal do Paraná – Campus Ponta Grossa, Paraná, Brasil) - rafaelhornung@gmail.com
Simone N. Matos (Universidade Tecnológica Federal do Paraná – Campus Ponta Grossa, Paraná, Brasil) - snasser@utfpr.edu.br
Clovis T. Fernandes (Instituto Tecnológico de Aeronáutica, São Paulo, Brasil) - clovistf@uol.com.br

Syntactic validation of mathematical formulas is a theme approached by several applications. However, the user of these applications can have correction difficulty when its formula holds inconsistencies. Instances of syntactic inconsistencies are: unbalanced parentheses or two operating without an operator among them. This work presents the process for the creation of a domain subframework for the syntactic validation of mathematical formulas. This subframework was built using the responsibility-driven process and implemented with free tools. The proposed subframework analyzes syntactically the formula using Lexical and Syntactic Analysis algorithms. These algorithms were developed with JFlex and CUP tools. In the case of an inconsistent formula, the subframework presents more detailed error message, facilitating its correction.

Keywords: Lexical and Syntactic Analyzers, Mathematical Formulas, Domain Framework, Subframework.

APLICANDO O PROCESSO DIRIGIDO POR RESPONSABILIDADES PARA A CRIAÇÃO DE UM SUBFRAMEWORK PARA VALIDAÇÃO SINTÁTICA DE FÓRMULAS

INTRODUÇÃO

Uma fórmula, para matemática, é a representação dos cálculos que devem ser efetuados para se obter um determinado resultado. Geralmente uma fórmula é composta pela iteração entre números, operadores e variáveis, sendo essa iteração também conhecida por expressão.

A validação sintática de fórmulas, dentre elas, pode-se citar as seguintes, entre outras: adição, subtração, média, é um tópico abordado por alguns aplicativos, tais como: Axiom (2007), Yacas (2007), Maxima (2007), Eigenmath (2007), entre outros.

Durante a análise de alguns destes aplicativos verificou-se que todos possuíam mecanismos próprios para a determinação de inconsistências sintáticas. Porém, uma dificuldade encontrada foi com relação a interpretação das mensagens retornadas, além de não serem implementados em linguagem nativa. Na maior parte dos casos, as mensagens são incompletas, dificultando a sua interpretação e posterior correção por parte do usuário.

Além disso, os aplicativos não podem ser reusados em uma nova aplicação que necessita de análise sintática, pois são ferramentas específicas.

Neste artigo apresenta-se o desenvolvimento de um aplicativo reutilizável, que realiza uma análise detalhada em uma fórmula, e retorna, para o caso de encontrar uma inconsistência, uma mensagem mais detalhada do erro.

O aplicativo ou subframework proposto neste trabalho foi implementado com ferramentas gratuitas e desenvolvido pelo processo de Matos e Fernandes (2006) voltada ao desenvolvimento de framework de domínio. Ressalta também, que o mecanismo de validação de fórmulas usado reúsa os algoritmos de análise léxica e sintática gerados pelo JFlex (2007) e CUP (2007).

Este artigo tem a seguinte organização. Seção 1 apresenta a motivação de se desenvolver um analisador sintático de fórmulas. Seção 2 relata algumas definições sobre framework, bem como seus benefícios, reúso e classificação. O domínio de fórmulas, algumas abordagens e ferramentas que podem ser utilizadas para sua análise léxica e sintática são apresentados na Seção 3. A Seção 4 descreve alguns aplicativos gratuitos que são usados para validação sintática de fórmulas. Apresenta também uma breve análise comparativa entre eles. A Seção 5 apresenta como o subframework proposto neste trabalho foi desenvolvido, ilustrando seu processo de criação, seu funcionamento interno, sua usabilidade e as ferramentas utilizadas. A Seção 6 apresenta as conclusões deste trabalho, bem como, recomendações de trabalhos futuros.

1. FRAMEWORK NA LITERATURA ESPECIALIZADA

São várias as definições encontradas na literatura a respeito de frameworks, entre elas as mais conceituadas são as propostas por Johnson and Foote (1988), Johnson (1997) e Gamma et al (1994). Johnson and Foote (1988) propõe um framework como sendo um:

> "Conjunto de classes que definem um projeto abstrato de soluções para uma família de problemas relacionados" (Johnson and Foote, 1988).

Gamma et al (1994) complementa a definição proposta por Johnson and Foote (1988), incorporando a finalidade da utilização de um framework, conceituando-o como um:

> "Conjunto de classes cooperando, tanto abstrata quanto concretas, que produzem um projeto reusável para uma categoria específica de software" (Gamma et al, 1994).

Johnson (1997) aprimorou a definição de framework abordando a sua finalidade em termos de reúso de uma forma mais clara do que a proposta por Gamma et al (1994), redefinindo framework como um:

> "Projeto reutilizável de uma parte ou o todo de um sistema, que é representado por um conjunto de classes abstratas e pelo modo que elas interagem" (Johnson, 1997).

Com base nas definições propostas por Johnson e Gamma et al, pode-se concluir de forma resumida, que um framework é um conjunto de classes que interagem formando uma aplicação semi-completa e reutilizável, com a finalidade de resolver problemas comuns existentes em várias aplicações, podendo ainda ser aplicado a diversos domínios.

1.1. Classificações de um Framework

Um framework pode ser classificado de diversas formas, entre elas a mais utilizada é quanto ao seu propósito ou escopo (Taligent, 1994):

- Aplicação ou de Infra-Estrutura: Cobre funcionalidades aplicáveis a vários domínios, como por exemplo, frameworks para sistemas operacionais, comunicação, redes, construção de interfaces, entre outros.
- Domínio: Cobre funcionalidades aplicáveis há um domínio específico, como por exemplo, frameworks para telecomunicações, manufatura, controle de produção, multimídia, engenharia financeira, otimização de preço de venda, entre outros.
- Suporte ou de Integração de Middleware: Oferecem serviços de baixo nível, normalmente utilizados na integração de aplicações distribuídas e seus componentes, como por exemplo, frameworks CORBA ORB, DCOM, entre outros.

Portanto, os frameworks de aplicação e de suporte preocupam-se basicamente com problemas internos de desenvolvimento de software, já os frameworks de domínio têm como objetivo apoiar o desenvolvimento de aplicações dirigidas aos usuários e produtos em domínios específicos.

1.2. Custos e Benefícios

Com a utilização de frameworks os desenvolvedores podem focar seus esforços na área específica do sistema que está construindo, pois uma grande parte das classes comuns ao domínio já estão desenvolvidas, testadas, depuradas e validadas. Isso gera uma diminuição considerável na quantidade de linhas de código a serem implementadas (Maldonado, 2007). Devido a esse fato, um desenvolvedor quando opta em utilizar um framework melhora o nível de modularidade, reusabilidade, extensibilidade e manutenibilidade do sistema que está desenvolvendo.

Porém, isso gera alguns custos para os seus usuários como, por exemplo, um maior esforço para a aprendizagem do framework, já que não foi ele que o projetou. Outra dificuldade é quanto à documentação, que muitas vezes é escassa ou incompleta.

Mesmo com a existência de alguns custos a utilização de frameworks ainda é de grande ajuda, pois os benefícios da sua utilização superam as desvantagens em longo prazo.

1.3. Reúso

O reúso é uma das principais metas buscadas pelos desenvolvedores de software (Maldonado, 2007). Porém, a sua implementação em grande escala era difícil, sendo apenas possível à reutilização de pequenos componentes. Devido ao avanço do paradigma da orientação a objetos, a tecnologia para o reúso de grandes componentes vem tornando-se possível.

Esses fatos acabaram gerando uma grande motivação para a utilização de padrões de projeto (Gamma et al, 1994) e framework, que provêem um aumento considerável no reúso e consequentemente uma diminuição no tempo para a produção de uma aplicação.

Um framework pode ser classificado quanto ao processo de reúso das suas classes durante a sua extensão como sendo: caixa branca, caixa preta ou caixa cinza (Yassin and Fayad, 2000), sendo que:

- Caixa Branca (*White Box*): Está fortemente ligada às características da linguagem orientada a objetos, sendo o reúso e a extensão das funcionalidades desenvolvidas por meio da herança de classes e da implementação de métodos.
- Caixa Preta (*Black Box*): Possibilita a extensão através de interfaces para componentes que podem ser “plugados” ao framework, tendo o reúso das funcionalidades aplicadas por meio da composição ou da definição de interfaces para os componentes.
- Caixa Cinza (*Gray Box*): Evita as desvantagens dos frameworks, Caixa Branca e Caixa Preta. Esse tipo de framework possui flexibilidade e extensibilidade, além de esconder informações que não são necessárias para os desenvolvedores de aplicações.

A Figura 1 representa um reúso caixa cinza, que é um híbrido do tipo de reúso caixa branca e caixa preta.

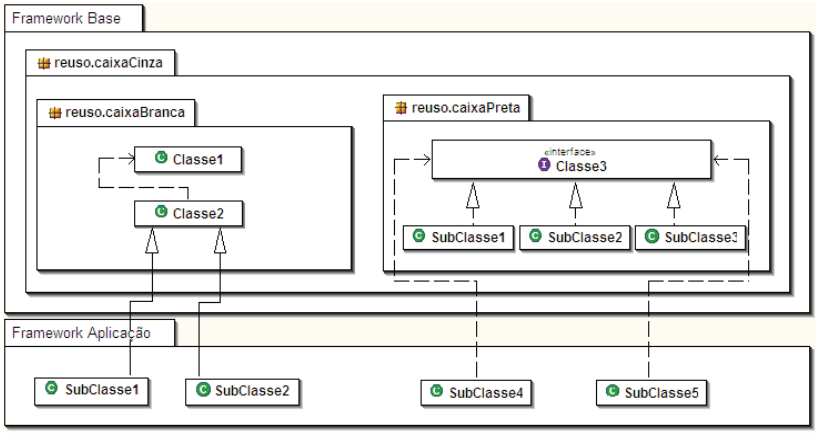

Figura 1 – Representação de um framework com reúso caixa cinza.

Considerando a Figura 1, o reúso caixa cinza é obtido por meio de herança, ligação dinâmica e interface. A classe *Classe2* possui os métodos que serão sobrepostos e em alguns casos implementados nas classes: *SubClasse1* e *SubClasse2*, representando o reúso caixa branca. Já a interface *Classe3*, que é formada por uma composição de objetos, e é apenas utilizada pelas classes: *SubClasse4* e *SubClasse5*, não tendo acesso aos códigos, representando um reúso caixa preta.

2. ANÁLISE DE FÓRMULAS NA LITERATURA

São poucas as definições encontradas na literatura que definem explicitamente o que é uma fórmula ou uma expressão. As definições encontradas para fórmula são as citadas por Tersariol (1997) e pelo sitio Wikipedia (2007a), os quais definem uma fórmula como, respectivamente:

> "Expressão de um preceito ou principio; receita; expressão matemática para resolução dos problemas relativos a certa questão" (Tersariol, 1997).
>
> "A Fórmula é a síntese de um raciocínio, e não um subterfúgio para não raciocinar" (Wikipedia, 2007a).

Já quanto à expressão as definições encontradas foram citadas por Matos (2001), Grasseschi, Andretta and Santos (1995) e pelo sitio Wikipedia (2007b; 2007c), definindo uma expressão como:

> "Uma expressão ... é qualquer seqüência finita de símbolos de seu alfabeto" (Matos, 2001)
>
> "... codificação, em linguagem matemática, de uma série de eventos quantitativos" (Grasseschi, Andretta and Santos, 1995).
>
> "Expressão numérica é na matemática um estilo de conta que inclui todas as operações fundamentais: a divisão, multiplicação, subtração e adição. Os sinais gráficos usados são colchetes, chave e parênteses" (Wikipedia, 2007b).

> "Uma expressão matemática é um combinação de números, operadores e variáveis. Exemplo: {[(9/3)+5]*12}" (Wikipedia, 2007c).

Com base nas definições citadas acima, este trabalho define uma fórmula como sendo uma representação simbólica de uma informação, composta por expressões matemáticas, que por sua vez é formada pela iteração entre números, operadores, variáveis e os sinais gráficos como: chave e parênteses. Um exemplo de fórmula utilizada pela área de contabilidade é ilustrado na Figura 2 (Receita Federal, 2007):

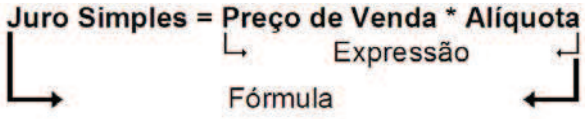

Figura 2 – Exemplo de fórmula.

Como pode ser visto na Figura 2, a fórmula de Juro Simples é formada por uma expressão, composta pela multiplicação de duas variáveis: **Preço de Venda** e **Alíquota**.

2.1. Método Utilizado para a Análise de Fórmulas

Para que um computador consiga executar um programa, ele precisa compreender o seu código. Por esse motivo, utiliza-se um compilador para realizar a conversão do código do programa e o do código de alto nível em um código compreensivo pela máquina, ou seja, um código binário (Ricarte, 2007).

A fim de executar essa tarefa, um compilador utiliza-se dois passos. O primeiro é uma análise do código fonte. O segundo é a síntese do programa em uma linguagem de máquina, sendo essa última não comentada neste trabalho por não interessar ao desenvolvimento do mesmo.

A análise do código fonte é formada por três fases, demonstradas a seguir (Ricarte, 2003):

- Análise Léxica: aplicação do conceito de reconhecimento de sentenças para agrupar as seqüências de caracteres em "palavras" ou tokens.
- Análise Sintática: estrutura do programa é analisada a partir do agrupamento de tokens.
- Análise Semântica: realiza as análises que não podem ser feitas durante a análise sintática, como por exemplo: verificação dos tipos, verificação da unicidade da declaração de variáveis, entre outras.

Este artigo irá tratar apenas das duas primeiras etapas, Análise Léxica e a Análise Sintática. Para que essas etapas sejam realizadas é necessário que o compilador tenha o conhecimento das expressões regulares válidas e da gramática da linguagem. Para maiores informações sobre estes assuntos consultar o trabalho de Regex (2007).

2.2. Ferramenta para a Geração de Analisadores Léxicos e Sintáticos

A geração de rotinas que realizam a análise léxica e sintática pode ser realizada de forma manual, porém, para linguagens mais complexas essa estratégia de desenvolvimento torna-se trabalhosa (Ricarte, 2003).

Com a finalidade de resolver esse problema, foram desenvolvidas algumas ferramentas que a partir da definição das expressões regulares e a gramática da linguagem, geram rotinas que realizam a analise léxica e sintática. As ferramentas mais conhecidas são:

- Lex: Gerador de Analisadores Léxicos (Lex, 2007).
- Yacc: Gerador de Analisadores Sintáticos (Yacc, 2007).

Tanto o Lex quanto o Yacc foram desenvolvidas com a finalidade de gerarem rotinas para a linguagem de desenvolvimento C. Atualmente há diversas implementações similares que trabalham com outras linguagens de programação que não a linguagem C.

As ferramentas escolhidas para o desenvolvimento deste trabalho foram: JFlex (versão 1.4.1) e CUP (versão 0.10j), como geradores de analisadores léxicos e sintáticos respectivamente.

O JFlex é um gerador de Analisadores Léxicos para Java e implementado em Java (JFlex, 2007). O seu objetivo é gerar uma rotina em Java, ou seja, uma classe que contenha as especificações léxicas válidas para a linguagem. Essa classe é gerada a partir de um arquivo, criado pelo usuário, que contém as especificações léxicas válidas para a linguagem que se deseja criar.

O CUP (*Constructor of Useful Parsers*) é um gerador de Analisadores Sintáticos LALR (*LookAhead Left-Right*) para Java. O seu objetivo, assim como o JFlex, é gerar uma rotina em Java a partir de um arquivo que contenha a gramática para o qual o analisador sintático será utilizado (CUP, 2007).

O JFlex foi uma ferramenta desenvolvida para ser facilmente utilizada em conjunto com o CUP. Para a sua utilização basta utilizar a diretriz %cup na especificação léxica do seu JFlex.

3. ANÁLISE COMPARATIVA DE ALGUNS APLICATIVOS GRATUITOS QUE FORNECEM SUPORTE A FÓRMULAS MATEMÁTICAS

Nesta seção serão analisadas algumas características dos aplicativos gratuitos: Axiom (AXIOM, 2007), Eigenmath (EIGENMATH, 2007), Maxima (MAXIMA, 2007) e Yacas (YACAS, 2007). Porém, alguns detalhes como passos para o desenvolvimento das expressões, funções aceitas ou os tipos de cálculos realizados não serão abordados por não constituírem o foco deste trabalho. As características analisadas foram as seguintes:

- **Análise Léxica**: Reconhecimento dos caracteres, ou tokens, válidos para os aplicativos, assim como o tratamento das possíveis inconsistências léxicas em expressões informadas.
- **Análise Sintática**: Reconhecimento da gramática utilizada pelos aplicativos, bem como o tratamento das possíveis inconsistências sintáticas em expressões informadas.
- **Análise Semântica**: Por se tratar de um tópico não abordado em profundidade neste trabalho, será apenas verificada se existe algum tratamento para possíveis inconsistências semânticas em expressões informadas. Por exemplo, divisão por zero.

As seções a seguir foram desenvolvidas com a ajuda de algumas informações retiradas dos manuais dos aplicativos e de sua execução prática.

3.1. Axiom

O Axiom é um aplicativo que se utiliza do computador para resolver alguns problemas matemáticos, sendo especialmente útil para os cálculos simbólicos e pesquisas matemáticas no desenvolvimento de novos algoritmos matemáticos (Axiom, 2007).

Análise Léxica

Caracteres Aceitos pelo Aplicativo

Na Tabela 1 tem-se os caracteres válidos ao aplicativo Axiom.

Tabela 1 – Caracteres aceitos pelo aplicativo Axiom.

^	a - z	=	<	´	@	$	+	[ ]	~	_	( )	?	"
**	A - Z	:=	>	,	#	%	-	{ }	!	..	:	\	\/
*	0 - 9	==	<=	.	|	&	;	/\	->	/	>=	::	%%

Tratamento de Erros

Na ocorrência da informação de um caractere não conhecido pelo aplicativo, é exibida uma mensagem de erro como a ilustrada na Figura 3.

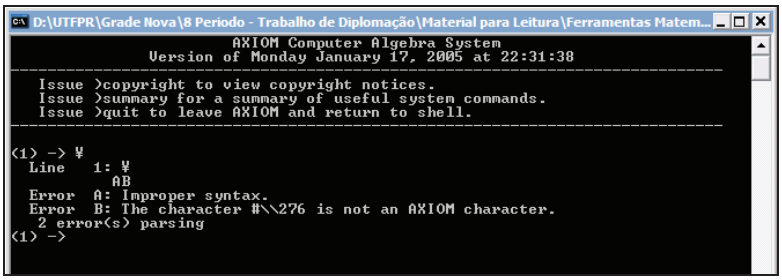

Figura 3 – Mensagem de erro do aplicativo Axiom para um caractere desconhecido.

Análise Sintática

Gramática do Aplicativo

Os principais operadores matemáticos aceitos pelo aplicativo são: Exponeciação (^ ou **), Multiplicação (*), Divisão (/), Adição (+), Subtração (-).

A ordem de precedência é: ^ ou **, * ou /, + ou -. Essa ordem pode ser alterada com a adição de parênteses, chaves ou colchete. No aplicativo Axiom, uma expressão é formada por: “ID + OP + ID”, onde:

- ID (Identificador) é composto por números, positivo ou negativo, ou variável.
- OP (Operador) é composto pelos operadores matemáticos já demonstrados.

Estas expressões podem ainda ser separadas por parêntese, chaves ou colchete, como já citado.

Tratamento de Erros

Na ocorrência de uma inconsistência sintática, como por exemplo, um parêntese desbalanceado o aplicativo exibe uma mensagem, como a ilustrada na Figura 4.

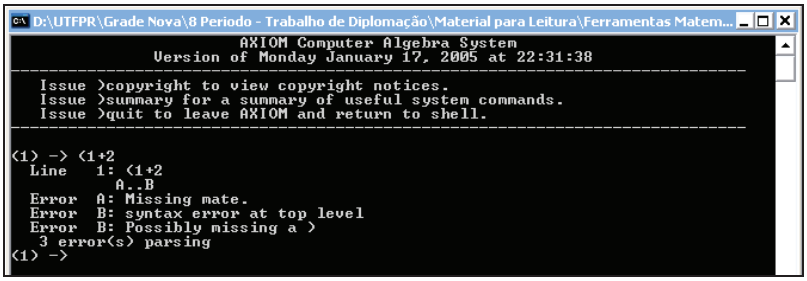

Figura 4 – Mensagem de inconsistência sintática no aplicativo Axiom.

Análise Semântica

Um exemplo de inconsistência semântica é a divisão por zero, pois o seu resultado é impossível de ser determinado. Quando se tenta fazer uma divisão por zero no aplicativo Axiom, o resultado exibido pelo aplicativo é “division by zero”.

3.2. Eigenmath

Esse aplicativo é utilizado para cálculos avançados, como integrais, inversão de matrizes, fatoração, entre outros (Eigenmath, 2007).

Análise Léxica

Caracteres Aceitos pelo Aplicativo

Na Tabela 2 tem-se os caracteres válidos ao aplicativo Eigenmath.

Tabela 2 – Caracteres aceitos pelo aplicativo Eigenmath.

^	a - z	=	<
*	A - Z	,	>
/	0 - 9	.	<=
+	( )	!	>=
-	[ ]	"	

Tratamento de Erros

Na ocorrência da informação de um caractere não válido, o aplicativo exibe uma mensagem de erro que pode ser visualizada na Figura 5.

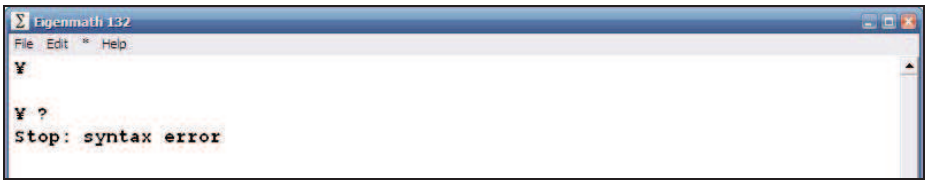

Figura 5 – Mensagem de erro do aplicativo Eigenmath para um caractere desconhecido.

Análise Sintática

Gramática do Aplicativo

Os principais operadores matemáticos aceitos pelo aplicativo são: Exponeciação (^), Multiplicação (*), Divisão (/), Adição (+), Subtração (-). A ordem de precedência é: ^, * ou /, + ou -. Essa ordem pode ser alterada com a adição de parênteses.

Assim como já demonstrado e explicado no aplicativo Axiom, no aplicativo Eigenmath, uma expressão é formada por: “ID + OP + ID”.

Tratamento de Erros

Na ocorrência de uma inconsistência sintática, como por exemplo, um parêntese desbalanceado o aplicativo exibe uma mensagem, como a demonstrada na Figura 6.

Figura 6 – Mensagem de inconsistência sintática no aplicativo Eigenmath.

Análise Semântica

Como já citado anteriormente, um exemplo de inconsistência semântica é a divisão por zero. No aplicativo Eigenmath, quando se tenta realizar esta operação o resultado obtido é “divide by zero”.

3.3. Maxima

O Maxima é um sistema para manipulação de expressões simbólicas e numéricas. O seu desenvolvimento foi baseado na ferramenta Macsyma, uma ferramenta conhecida desenvolvida em 1960 na Universidade de Massachusetts (Maxima, 2007).

Análise Léxica

Caracteres Aceitos pelo Aplicativo

Na Tabela 3 tem-se os principais caracteres válidos ao aplicativo Maxima.

Tabela 3 – Caracteres aceitos pelo aplicativo Maxima.

^	a - z	=	<	$	;	+	[ ]	?	( )	!
**	A - Z	:=	>	%	::	-	{ }	~	"	.
*	0 - 9	:	<=	#	@	,	|	/	>=	\

Tratamento de Erros

A Figura 7 ilustra o resultado obtido no aplicativo Maxima para um caractere que os outros aplicativos reconheceram como sendo inválido.

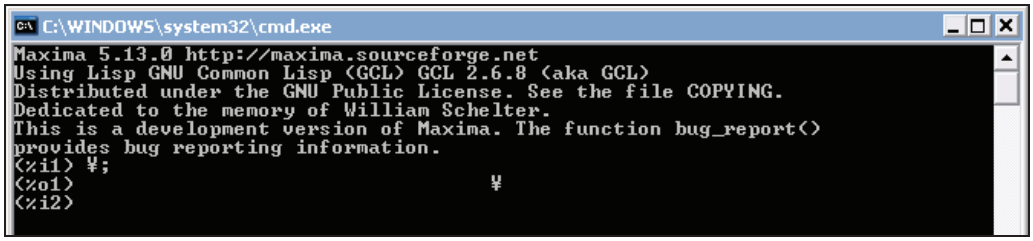

Figura 7 – Resultado obtido pelo aplicativo Maxima para um caractere pouco utilizado.

Diferente dos demais aplicativos analisadas, o Maxima aceita a maior parte dos caracteres, reconhecendo os mesmos como variáveis.

Análise Sintática

Gramática do Aplicativo

Os principais operadores matemáticos aceitos pelo aplicativo são: Exponenciação (^ ou **), Multiplicação (*), Divisão (/), Adição (+), Subtração (-). A ordem de precedência é ^ ou **, * ou /, + ou -. Essa ordem pode ser alterada com a adição de parênteses, chaves ou colchete. No aplicativo Maxima, assim como nos anteriores, uma expressão é formada por: “ID + OP + ID + ;”, porém, possui uma diferença, a necessidade do “ ; “ para indicar o fim da expressão.

Tratamento de Erros

Na ocorrência de uma inconsistência sintática, como por exemplo, um parêntese desbalanceado o aplicativo exibe uma mensagem que pode ser visualizada na Figura 8.

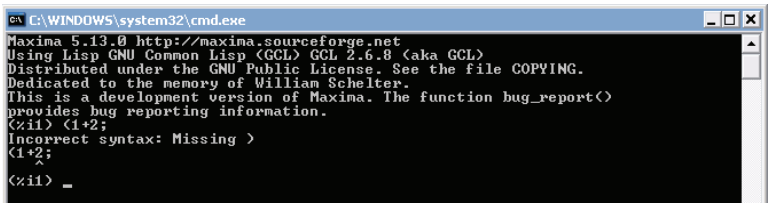

Figura 8 – Mensagem de inconsistência sintática no aplicativo Maxima.

Análise Semântica

Assim como nos aplicativos anteriores, a divisão por zero foi um mecanismo escolhido para a verificação da existência de mecanismos de detecção de inconsistências semânticas. No aplicativo Maxima, o resultado exibido por esta operação é “Division by 0 – na error...”.

3.4. Yacas

Assim como as anteriores o Yacas é um aplicativo que realiza a manipulação de expressões matemáticas com o uso do computador (Yacas, 2007).

Análise Léxica

Caracteres Aceitos pelo Aplicativo

Os caracteres válidos ao aplicativo Yacas são ilustrados na Tabela 4.

Tabela 4 – Caracteres aceitos pelo aplicativo Yacas.

^	a - z	{ }	=	<<	%
*	A - Z	<	:=	>>	,
/	0 - 9	>	:	!=	/@
+	( )	<=	;	++	!
-	[ ]	>=	"	--	@

Tratamento de Erros

Na ocorrência de um caractere não válido, o aplicativo exibe uma mensagem de erro como ilustrado na Figura 9.

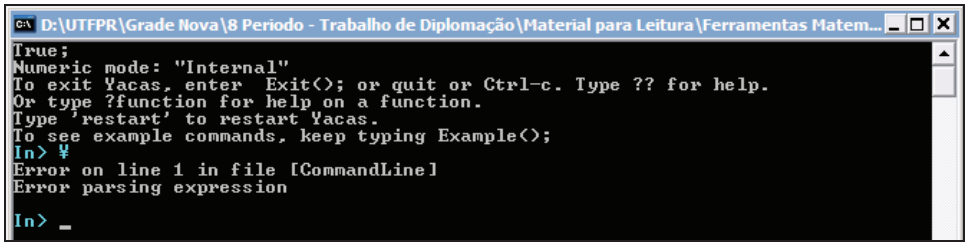

Figura 9 – Mensagem de erro do aplicativo Yacas para um caractere desconhecido.

Análise Sintática

Gramática do Aplicativo

Os principais operadores matemáticos aceitos pelo aplicativo são: Exponenciação (^), Multiplicação (*), Divisão (/), Adição (+), Subtração (-). A ordem de precedência é: ^ ou **, * ou /, + ou -. Essa ordem pode ser alterada com a adição de parênteses ou chaves

Assim como o aplicativo Maxima, no Yacas uma expressão é formada por: “ID + OP + ID + ;”, porém, ao contrário do Maxima a adição do “;” para indicar o fim da expressão é opcional.

Tratamento de Erros

Na ocorrência de uma inconsistência sintática, como por exemplo, um parêntese desbalanceado o aplicativo exibe uma mensagem, como a demonstrada na Figura 10.

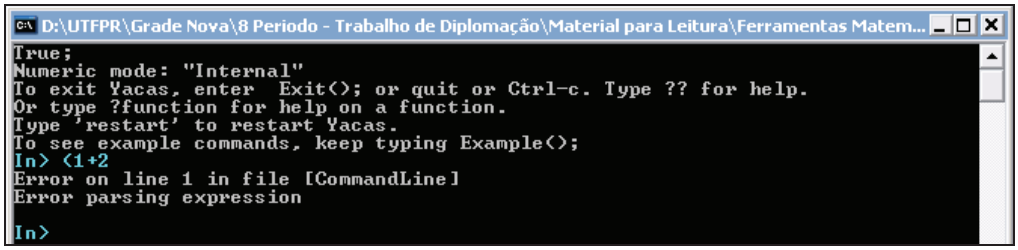

Figura 10 – Mensagem de inconsistência sintática no aplicativo Yacas.

Análise Semântica

Quando se tenta fazer uma divisão por zero no aplicativo Yacas, o resultado, ao contrário dos outros aplicativos, não é uma mensagem de erro, mais sim “Infinity;” .

3.5. Análise Comparativa entre os Aplicativos

Uma característica comum a todos os aplicativos analisados por este trabalho, é o fato de serem de distribuição gratuita, tendo em alguns casos o seu código-fonte também disponível para consulta.

Uma das características analisadas nos aplicativos, quanto a Análise Léxica, foi quanto aos caracteres válidos, entre eles podem-se citar os demonstrados na Tabela 5.

Tabela 5 – Caracteres em comum entre os aplicativos analisadas.

^	Expoente	**( )**	Parêntese
*****	Multiplicação	**[ ]**	Chaves
/	Divisão	**<**	Menor
+	Adição	**>**	Maior
-	Subtração	**<=**	Menor Igual
a - z	Letras de a até z (Concatenados ou não)	**>=**	Maior Igual
A - Z	Letras de A até Z (Concatenados ou não)	**"**	Aspas Duplas
0 - 9	Números de 0 até 9 (Concatenados ou não)	**,**	Vírgula
=	Igualdade	**!**	Interrogação (Função)

As mensagens de erro encontradas nesta etapa demonstram que todos os aplicativos possuem mecanismos para definição de caracteres válidos. Isso ocorre pelo fato que todos os aplicativos demonstram erros semelhantes para o caso de encontrarem caracteres inválidos as mesmas.

Quanto à etapa de Análise Sintática, pode-se destacar que as expressões matemáticas aceitas pelos aplicativos analisados são semelhantes, possuindo pequenas diferenças, como no caso dos aplicativos Maxima e Yacas, onde o “ ; ”

é utilizado para indicar o fim da expressão, além do fato de algumas delas não aceitarem a chaves ou o colchete em suas expressões.

Com isso pode-se concluir que uma forma comum quanto à gramática dos aplicativos seria: “ID + OP + ID”, onde:

- ID (Identificador) é composto por números, positivo ou negativo, ou variável.
- OP (Operador) é composto pelos operadores matemáticos “^, *, /, +, -”.

Pode-se ainda utilizar o parêntese para facilitar a visualização ou modificação da ordem de execução dos operadores.

As mensagens de inconsistência sintática, apesar de não serem compreensivas ao usuário, foram encontradas em todos os aplicativos para o caso de um parêntese desbalanceado, demonstrando que os aplicativos possuem mecanismos para a detecção dos mesmos.

Como a Análise Semântica não será abordada pela ferramenta proposta por este trabalho, aqui será apenas comentado sobre à existência de mecanismos de validação quanto à semântica das expressões informadas.

Para isso foram realizados alguns testes de possíveis inconsistências semânticas em todos os aplicativos. Por exemplo, a divisão por zero, onde todos os aplicativos, com exceção do Yacas apresentaram mensagens de erro. O aplicativo Yacas apresentou como resultado o valor infinito.

Com isso pode-se concluir que os aplicativos analisados possuem mecanismos para a detecção de inconsistências semânticas. Porém, eles não podem ser implementados em domínios diferentes dos utilizados pelos programas aqui analisados.

4. PROCESSO DE CRIAÇÃO DO SUBFRAMEWORK DE DOMÍNIO PARA ANÁLISE DE FÓRMULAS

O objetivo geral do subframework proposto por este trabalho é ser uma ferramenta que realize a validação sintática de fórmulas matemáticas. Algumas das características que o difere das aplicações analisadas na Seção 4 são as seguintes:

- Ser reutilizável em uma nova aplicação. Isso será demonstrado em um trabalho futuro.
- Apresentar mensagens de inconsistência de fórmulas matemáticas através de informações mais detalhadas.
- Facilitar a alteração de mensagens de erro pelo desenvolvedor.
- Não realizar análise semântica das fórmulas.

Cabe salientar que este subframework proposto fará parte do framework de otimização de preços de venda do Projeto de Pesquisa do Grupo de Engenharia de Software. Por esse motivo, foi considerado um subframework. Mas, esse analisado individualmente, é considerado um framework de domínio, pois possui aspectos comuns e específicos implementados separadamente.

4.1. Ferramentas Utilizadas

O subframework proposto foi implementado na linguagem de programação Java (SUN, 2007), sendo utilizado as ferramentas descritas a seguir:

- Eclipse 3.1.2 (Eclipse, 2007): É uma plataforma (IDE) focada no desenvolvimento de ferramentas e aplicações de software. Uma de suas características é a forte orientação ao desenvolvimento baseado em plugins, procurando atender as diferentes necessidades de diferentes programadores (Javafree, 2007).
- Omondo 2.1.0 (Omondo, 2007): É um plugin para o Eclipse que auxilia na construção de diagramas UML. Com este plugin é possível criar diagramas de classe, seqüência, estados, use cases, atividades, etc. Alterações no diagrama automaticamente se refletem no código-fonte e vice-versa. Outro recurso interessante é a capacidade de fazer Engenharia Reversa (Lecheta, 2007).
- GEF-ALL 3.1.1. Permite criar um rico editor gráfico a partir de seu modelo, é um plugin necessário para a utilização do Omondo (Lecheta, 2007).
- JEM-SDK 1.1.0.1. Biblioteca para modelagem, utilizado internamente para fazer outros projetos, plugin necessário para a utilização do Omondo (Easyeclipse, 2007).

Para mais informações sobre as estas ferramentas, assim como passos para instalação, pode-se consultar o relatório desenvolvido pelo Grupo de Pesquisa de Engenharia de Softwares (GPES, 2007). Além destas ferramentas, foram utilizadas as já citadas na Seção 3, a saber, JFlex e CUP.

4.2. Metodologia

Como relatado anteriormente, o subframework proposto, de forma individual, pode ser considerado um framework. Por esse motivo, utilizou-se o processo dirigido por responsabilidades de Matos and Fernandes (2006) para o seu desenvolvimento. A Figura 11 apresenta as fases propostas por essa abordagem.

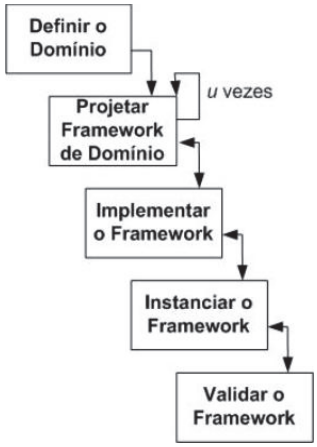

Figura 11 – Fases da abordagem de Matos and Fernandes (2006)

A fase *Definir o Domínio* é utilizada com o objetivo de permitir que o desenvolvedor escolha o domínio que o framework irá pertencer. Neste trabalho, utilizou-se o domínio que o Grupo de Pesquisa em Engenharia de Software

necessitava para a criação do módulo de validação sintática de fórmulas matemáticas. Por isso, a Seção 3 apresentou o estudo no domínio de validação sintática de fórmulas.

A fase de *Projetar Framework de Domínio* é executada de forma a permitir a criação do framework em partes, onde *u* é a quantidade de partes já desenvolvidas, sendo que o framework da aplicação evolui de maneira a sempre adaptar-se a novos subframeworks ou corrigir implementações inadequadas ou não tão satisfatórias.

A Fase Projetar Framework de Domínio, é formada por duas subfases: Compreender Aplicação Exemplo e Definir Framework Base e de Aplicação conforme ilustrado na Figura 12.

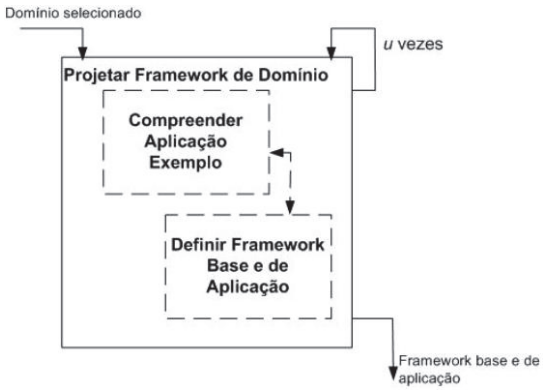

Figura 12 – Subfases da fase *Projetar Framework de Domínio* de Matos and Fernandes (2006)

Neste trabalho inicio-se pela subfase *Compreender Aplicação Exemplo*, que permitiu um conhecimento mais detalhado sobre as quatro aplicações descritas na Seção 4 e a partir desta análise, definiu-se as expressões regulares e a gramática que foram implementadas. Mais informações serão apresentadas nas Seções 5.2.1 e 5.2.2.

A segunda subfase, *Definir Framework Base e de Aplicação*, ilustrado na Figura 12, permitiu a elaboração e modelagem do subframework. Para esta subfase analisou-se as aplicações descritas na Seção 4 e verificou-se entre elas os aspectos comuns, *Framework Base,* e os específicos, *Framework de Aplicação*. Este levantamento foi detalhado na Seção 4.5 em que se comparou os aplicativos. O diagrama de classe gerado por essa subfase será ilustrado na Figura 13 da Seção 5.3.

Na Fase *Implementar o Framework*, implementou-se na linguagem de programação Java as classes que serão apresentadas na Seção 5.3.

Na Fase *Instanciar o Framework* utilizou-se de uma aplicação de teste para simular a usabilidade do subframework em uma situação real. Isso será abordado na Seção 5.4.

Por fim, a última fase *Validar o Framework*, será implementada quando o framework proposto pelo Grupo de Pesquisa (GPES, 2007) para otimização de preço de venda estiver concluído.

5.2.1. Expressões Regulares

O padrão de texto válido ao subframework desenvolvido por este trabalho é ilustrado no Código 1. Esse padrão foi proposto através da análise dos aplicativos descritos na Seção 4, levando-se em consideração os símbolos que são destinados as fórmulas matemáticas e não booleanas. Por esse motivo, nem todos os símbolos foram considerados.

```
LineTerminator = \r|\n|\r\n
WhiteSpace = {LineTerminator} | [ \t\f]
ID_NUM = 0 | [1-9]* "," [0-9]* | [1-9][0-9]*
ID_STR = [A-Za-z_][A-Za-z_0-9]*
        OP = ["^"] | ["*"] | ["/"] | ["+"] | ["="]
```

Código 1 – Representação das Expressões Regulares utilizadas pelo Subframework.

Além destes, foram considerados também os caracteres que indicam o parêntese e a subtração. Abaixo segue uma descrição para o Código 1:

- As duas primeiras linhas indicam os caracteres de controle, como: espaço, mudança de linha, retorno e tabulação. Esses caracteres serão considerados inválidos para o subframework proposto.
- A representação “ID_NUM” representa os *Identificadores do tipo Numérico*, sendo assim considerado os caracteres de “0” até “9”. Pode-se ainda utilizar o caractere ‘ , ’ para números não inteiros, ou seja, os decimais.
- A representação “ID_STR” representa os *Identificadores do Tipo Texto*, ou seja, os caracteres de “a” até “z, tanto minúsculo quanto maiúsculo. Pode-se ainda ser composto pelo caractere ‘ _ ’ e números de “0” até “9”, com a condição que estejam intercalados entre os caracteres “a” até “z”.
- A representação “OP” traz os *Identificadores do Tipo Operador*, sendo composto pelo sinal de exponenciação, multiplicação, divisão e a soma. O operador subtração esta indicado em outra parte do arquivo, facilitando assim a sua implementação para os números negativos.

5.2.2. Gramática

A gramática válida ao subframework proposto é ilustrada no Código 2. Essa foi desenvolvida através da análise dos aplicativos descritos na Seção 4.

```
expr        ::=  expr OP term
                 | expr SUB term
                 | SUB term
                 | term
                 ;
 term       ::=  LPAREN expr RPAREN
                 | ID_NUM
                 | ID_STR
                 ;
```

Código 2 – Representação da Gramática utilizada pelo Subframework.

Abaixo segue uma descrição para o Código 2:

- A primeira regra determina que uma expressão “expr”, é formada:
  - por ela mesma, um operador “OP” e um termo “term”, ou
  - por ela mesma, o operador de subtração “SUB” e um termo, ou
  - o operador de subtração e um termo, ou
  - um termo.
- A segunda regra determina que um termo “term” é formado:
  - pela primeira regra entre parênteses, “LPAREN” para ‘ (’ e “RPAREN” para ’) ’, ou
  - por um identificador do tipo numérico “ID_NUM”, ou
  - por um identificador do tipo texto “ID_STR”.

Com a utilização desta gramática foi possível verificar se uma fórmula matemática informada pelo usuário está correta ou não.

4.3. Arquitetura do Subframework de Domínio para Análise de Fórmulas

A Figura 13 ilustra o diagrama de classes do subframework proposto, facilitando assim a compreensão da sua arquitetura básica. Uma versão preliminar da arquitetura de alto nível deste subframework pode ser encontrada em Hornung et al. (2007).

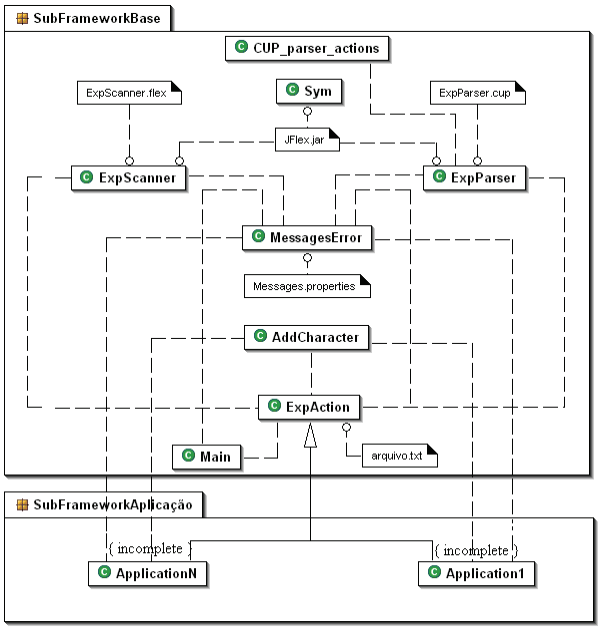

Figura 13 – Diagrama de Classes do Subframework proposto.

A arquitetura da Figura 13 está dividido em dois módulos, o *SubFrameworkBase* e o *SubFrameworkAplicação*. O *SubFrameworkBase*, representa os aspectos comuns das aplicações, é composto pelas classes base do subframewok. Por sua vez, o *SubFrameworkAplicação* é composto pelas classes especificas que são criadas pelo desenvolvedor que deseja reusar o subframework. Os tópicos seguintes detalham cada componente ilustrado na Figura 13.

ExpScanner.flex

O arquivo *ExpScanner.flex* contém as especificações léxicas do aplicativo desenvolvido por esse trabalho.

ExpParser.cup

O arquivo *ExpParser.cup*, contém as especificações sintáticas do aplicativo aqui proposto.

Messages.properties

O arquivo *Messages.properties*, é utilizado como um centralizador de mensagens. Seu intuito é facilitar a sua manutenção, assim como possíveis alterações nas mensagens a serem retornadas pelo aplicativo.

JFlex.jar

O arquivo *JFlex.jar*, é um arquivo Java compactado composto pelas principais classes das ferramentas JFlex e CUP. Sua utilização é necessária para a execução da análise léxica e sintática do aplicativo desenvolvido por esse trabalho.

Classe Main.java

A classe *Main*, é utilizada para testar o aplicativo desenvolvido por este trabalho.

Classe ExpAction.java

A classe *ExpAction*, é responsável por criar um arquivo de texto contendo a fórmula que será passada a ela.

Classe MessagesError.java

A classe *MessagesError*, é composta pelos métodos necessários para a manipulação do arquivo *Messages.properties*.

Classe ExpScanner.java

A classe *ExpScanner*, foi criada pelo aplicativo JFlex, a partir das especificações léxicas disponibilizadas pelo arquivo *ExpScanner.flex*. Nesta classe foram feitas algumas modificações, tais como:

- Objeto ZZ_CMAP: alterado para aceitar todos os caracteres do conjunto *Basic* do *Unicode*, sendo 255 no total.
- Método ZZ_ERROR_MSG: as mensagens originalmente em inglês foram traduzidas para o português.

Classe ExpParser.java

A classe *ExpParser*, é criada pelo aplicativo CUP, a partir das especificações sintáticas disponibilizadas pelo arquivo *ExpParser.cup*. Nesta classe a única alteração feita foi a remoção de uma parte do código-fonte, sendo criada a classe *CUP_parser_action* com esse código.

Classe CUP_parser_actions.java

A classe *CUP_parser_actions*, foi criada contendo uma parte do código-fonte da classe *ExpParser*. Essa separação foi realizada com o intuito de facilitar a compreensão do funcionamento do mesmo.

Classe Sym.java

A classe *Sym*, é criada pelo aplicativo CUP, sendo composta pelos símbolos válidos, assim como seus respectivos valores.

Arquivo.txt

Este arquivo é gerado em tempo de execução pelo aplicativo, sendo composto pela fórmula matemática informada pelo usuário e que devera ser validada sintaticamente.

Classe AddCharacter.java

A classe *AddCharacter*, é utilizada com a finalidade de adicionar novos caracteres a aplicação, assim como modificar o tipo dos caracteres que já fazem parte da aplicação. Com a sua utilização o programador pode modificar os caracteres válidos ao aplicativo desenvolvido por este trabalho, podendo assim utilizá-lo de acordo com o seu domínio.

Classe Application1.java

Esta classe é um exemplo de aplicação específica, em que o programador pode modificar ou adicionar novos caracteres.

Classe ApplicationN.java

Esta classe foi desenvolvida apenas para demonstrar que existe a possibilidade da criação de diversos aplicativos específicos, em que os caracteres válidos são diferentes do analisado por este trabalho. Neste caso, isso foi representado na Figura 13 pela palavra reservada da notação UML-F (FONTOURA et. al, 2000) “{incomplete}”.

4.4. Funcionamento Interno e Aplicação do Subframework

A seguir serão demonstrados os passos que são realizados para que a expressão matemática possa ser considerada como válidas ou não, assim como os possíveis erros na fórmula que possam ocorrer. Para melhorar a compreensão dos passos uma seqüência numérica foi utilizada e está ilustrada na Figura 14.

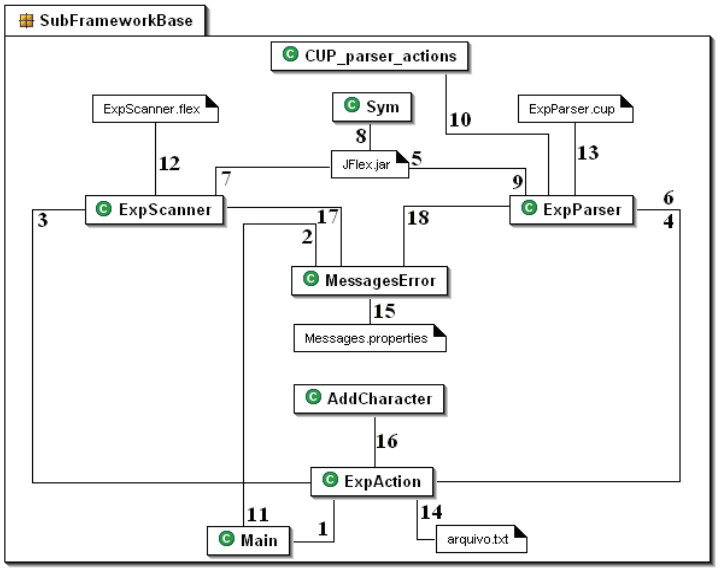

Figura 14 – Diagrama de Classes demonstrando os passos do Subframework.

Os passos são explicados abaixo:

- **1** – A classe *Main* instância a classe *ExpAction* passando para ela a expressão a ser avaliada e o nome do arquivo onde essa expressão será armazenada.
- **2** – A classe *MessagesErro* é instanciada e passada para a classe *ExpAction* através do método *init_validate*.
- **3** – O método *init_validate* da classe *ExpAction* cria o arquivo de texto indicado pelo passo 15. Logo após, a classe *ExpScanner* é instanciada e recebe o arquivo de texto contendo a expressão e a classe *MessagesError.*
- **4** – A classe *ExpParser* é instanciada na classe *ExpAction*, passando-se para ela a classe *ExpScanner* e *MessagesError*.
- **5** – A classe *ExpParser* que herda os métodos da classe *lr_parser*, que esta contida no arquivo compactado JFlex.jar., instância na classe *lr_parser* a classe *ExpScanner*, e localmente a classe *MessagesError*.
- **6** – A classe *ExpAction* chama o método *parser* da classe *lr_parser* através da classe *ExpParser.*
- **7** – O método *parser* executa algumas funções, entre elas destaca-se a chamada ao método *scan*. O método *scan* retorna os tokens, um por vez de forma recursiva, através do método *next_token* da classe *ExpScanner*.
- **8** – A principal ação realizada pelo método *next_token* é retornar o valor do símbolo representado pelo token atual. Para isso, ele utiliza a classe *Sym* que contém os símbolos válidos e seus respectivos valores.
- **9** – Os valores que representam os símbolos são passados para o método *do_action* da classe *ExpParse*r.
- **10** – O método *do_action* da classe *ExpParser* instância a classe *CUP_parser_action*, que utiliza os valores passados pela classe *ExpScanner,* verifica-se os símbolos estão sintaticamente corretos. A verificação sintática é feita através das regras de gramática que haviam sido criadas inicialmente no arquivo ExpParser.cup. Após a verificação de todos os tokens, a validação sintática na fórmula esta concluída.

- **11** – Por fim, a classe *Main* verifica se existe alguma mensagem de inconsistência na classe *MessagesError*. Caso sim as exibe. Caso contrário, retorna a mensagem que a expressão esta correta.
- **12** – O arquivo *ExpScanner.flex* contém as expressões regulares válidas ao subframework, com o auxílio da ferramenta JFlex a classe *ExpScanner* é criada.
- **13** – O arquivo *ExpParser.cup* contém a gramática válida ao subframework, com o auxilio da ferramenta CUP a classe *ExpParser* é criada.
- **14** – O arquivo de texto é criada pela classe *ExpAction* no momento de sua instanciação. Esse arquivo contém a expressão matemática que o subframework irá validar sintaticamente. A nomeação deste arquivo de texto deve seguir o padrão do Windows para nomes de arquivos.
- **15** – O arquivo *Messages.properties* contém as mensagens de erro que podem ser encontradas durante a utilização da subframework aqui proposta.
- **16** – A classe *AddCharacter* é apenas utilizada quando o domínio analisado por esse subframework não é válido ao aplicativo que deseja-se desenvolver, aqui podem ser adicionado novos caracteres assim como o seu tipo.
- **17** – Quando o método *next_token* é executado os tokens contidos na expressão são avaliados. Caso não seja valido, é passado para a classe *MessagesError* qual o caractere que torna o token inválido.
- **18** – A classe *ExpParser* recebe da classe *CUP_parser_action* o resultado da gramática se ela é valida ou não. Caso algum erro seja encontrado é passado para a classe *MessagesError* a posição do token que torna a expressão inválida sintaticamente.

A Figura 15 ilustra o resultado obtido através da execução dos passos descritos acima. Neste caso, a expressão matemática informada é “a!a”, que contenha um caractere invalido, no caso o ‘ ! ’. Observe que a mensagem de erro retornada pelo subframework proposto é mais explicativa do que as mensagens retornadas pelos aplicativos analisados na Seção 4.

Figura 15 – Execução de uma fórmula que contém um caractere inválido.

A Figura 16 ilustra uma expressão matemática, “a++a”, que contém uma inconsistência sintática, o que torna a sua gramática inválida.

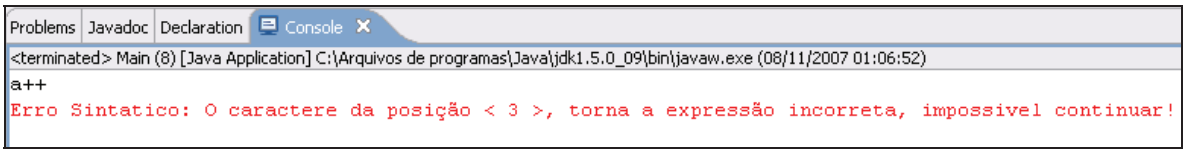

Figura 16 – Execução de uma fórmula que contém uma inconsistência na gramática.

A Figura 17 ilustra uma expressão matemática, “a+a”, que foi informado corretamente.

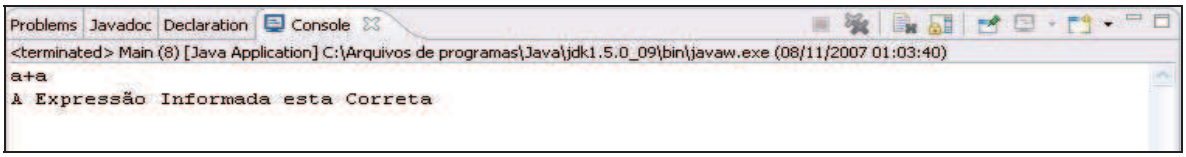

Figura 17 – Execução de uma fórmula correta.

5. CONCLUSÕES E RECOMENDAÇÕES

Este trabalho apresentou o desenvolvimento de um subframework de domínio para a validação sintática de fórmulas matemáticas. Durante o processo de validação o subframework é capaz de retornar mensagens de erro que auxiliam o usuário para a criação de suas fórmulas.

O subframework foi gerado utilizando os conceitos de framework e por isso pode ser reusado por outras aplicações que necessitem da análise sintática de fórmulas. Além disso, foi implementado com ferramentas gratuitas e seu código-fonte ficará disponibilizado na internet.

A gramática implementada pelo subframework levou em consideração a análise de domínio de quatro aplicativos. Para a realização da validação de fórmulas matemáticas, este subframework reusou e adaptou os algoritmos de análise léxica e sintática implementados pelas ferramentas JFlex e CUP.

Portanto, o subframework proposto neste trabalho traz vantagens tanto para usuários quanto para desenvolvedores. Do ponto de vista do usuário as vantagens do uso deste aplicativo são as seguintes: informação sobre erros na fórmula, dica de localização do erro, entre outras. Por sua vez, do ponto de vista do desenvolvedor o reúso e a facilidade de criar novas aplicações a partir do subframework são as principais contribuições. A criação de novas aplicações a partir deste subframework será demonstrando em um próximo artigo.

Poderá ser objeto de futuras pesquisas a utilização deste subframework em outros domínios, assim como o desenvolvimento de uma interface que facilite essa operação.

Além disso, é interessante o desenvolvimento de uma comparação entre a abordagem escolhida por esse trabalho para a análise em fórmulas, com outras possíveis ferramentas ou técnicas que também realizam esta função.

6. REFERÊNCIAS

AXIOM. **MatchAction and Axiom**. Disponível em <http://wiki.axiom-developer.org/FrontPage> acesso em 24-ago-2007.

CUP. **Parser Generator for Java**. Disponível em <http://www.cs.princeton.edu/~appel/modern/java/CUP/> acesso em 24-ago-2007.

EASYECLIPSE. **Eclipse Tools**. Disponível em <http://www.easyeclipse.org/site/plugins/eclipse-tools.html> acesso em 05-mar-2007.

ECLIPSE. **Eclipse - an open development platform**. Disponível em <http://www.eclipse.org/> acesso em 24-ago-2007.

EIGENMATH. **Eigenmath**. Disponível em <http://eigenmath.sourceforge.net/> acesso em 24-ago-2007.

FONTOURA, M.; PREE, W.; RUMPE, B. UML-F: A modeling language for object-oriented frameworks. In: European Conference on Object-Oriented Programming, 2000, Sophia Antipolis and Cannes, France. **Proceedings ECOOP – Lecture Notes in Computer Science 1850 Springer-Verlag**, 2000.

GAMMA, E. et al. **Design Patterns: Elements of Reusable Object-Oriented Software**. New York: Addison-Wesley, 1994. 424p.

GPES. **Grupo de Pesquisa de Engenharia de Software**. Disponível em <http://www.pg.utfpr.edu.br/coinf/gpes/> acesso em 24-ago-2007.

GRASSESCHI, Maria Cecília C.; ANDRETTA, Maria Capucho; SANTOS, Aparecida Borges dos. **PROMAT 4: Projeto Oficina de Matemática**. São Paulo, FTP, 1995.

HORNUNG, R., MATOS, S. N., BELMONTE, D. L. **Arquitetura de um Modelo Adaptativo Baseado em Framework para Validação Sintática de Fórmulas**. V EPCC, Maringá-PR, 2007.

JAVAFREE. **Eclipse**. Disponível em <http://www.javafree.org/wiki/Eclipse> acesso em 05-mar-2007.

JFLEX. **The Fast Scanner Generator for Java**. Disponível em <http://www.jflex.de/> acesso em 24-ago-2007.

JOHNSON, R. E.; FOOTE, B. Designing Reusable Classes. **Journal of the Object-Oriented Programming**, v.1, n.2, p.22-35, 1988.

JOHSON, R. E. Frameworks = (Components + Patterns). **Communications of the ACM**, v.40, n.10, p.39-43, 1997.

LECHETA, Ricardo Rodrigues. **Omondo EclipseUML**. Disponível em

<http://www.guj.com.br/content/articles/eclipseUML/eclipseUML.pdf> acesso em 05-mar-2007.

LEX. **Generator of Lexical Analysis Programs**. Disponível em <http://plan9.bell-labs.com/magic/man2html/1/lex> acesso em 24-ago-2007.

MALDONADO, J. C. et al. **Padrões e Frameworks de Software**. Disponível em <http://www.icmc.sc.usp.br/%7Ertvb/apostila.pdf> acesso em 19-jun-2007.

MATOS, Simone Nasser. **Especificação Formal e Implementação de um Protótipo para a Linguagem Paralog**. Curitiba, 2001, 132 f. Dissertação. Curso de Pós-Graduação em Informática, Universidade Federal do Paraná.

MATOS, S. N.; FERNANDES, C.T. Defining the Architectural Design of Framework through a Group of Subframework Created from Frozen and Hot Spots. In: International Conference on Software Engineering Advances, 2006, Tahiti. **Proceedings IEEE Computer Society Press**, 2006.

MAXIMA. Maxima – **A GPL CAS based on DOE- MACSYMA**. Disponível em <http://maxima.sourceforge.net/> acesso em 24-ago-2007.

OMONDO. **The Live UML Company**. Disponível em <http://www.eclipsedownload.com/> acesso em 24-ago-2007.

RECEITA FEDERAL. **Simples - Microempresa (ME) e Empresa de Pequeno Porte (EPP)**. Disponível em <http://www.receita.fazenda.gov.br/PessoaJuridica/DIPJ/2005/PergResp2005/pr108a200.htm> acesso em 24-ago-2007.

REGEX. **Expressões Regulares**. Disponível em <http://www.regex.pro.br/wiki/Home> acesso em 24-ago-2007.

RICARTE, Ivan Luiz Marques. **Programação de Sistemas: Uma Introdução.** São Paulo, 2003. 188 páginas. Apostila de Programação de Sistemas – Universidade Estadual de Campinas.

SUN. **Java Technology**. Disponível em <http://java.sun.com/> acesso em 24-ago-2007.

TALIGENT. **Building object-oriented frameworks**. A Taligent White Paper. 1994.

TERSARIOL, Alpheu. **Minidicionário da língua portuguesa 2ª Edição**. Rio Grande do Sul: Edelbra, 1997.

WIKIPEDIA. **Fórmula**. Disponível em <http://pt.wikipedia.org/wiki/F%C3%B3rmula> acesso em 24-ago-2007a.

—— . **Expressão numérica**. Disponível em <http://pt.wikipedia.org/wiki/Express%C3%A3o_num%C3%A9rica> acesso em 24-ago-2007b.>

—— . **Expressão matemática**. Disponível em < http://pt.wikipedia.org/wiki/Express%C3%A3o_matem%C3%A1tica> acesso em 24-ago-2007c.>

—— . **Símbolos Matemáticos**. Disponível em < http://pt.wikipedia.org/wiki/S%C3%ADmbolos_matem%C3%A1ticos> acesso em 24-ago-2007d.>

YACAS. **The Yacas computer algebra system**. Disponível em <http://yacas.sourceforge.net/homepage.html> acesso em 24-ago-2007.

YACC. **Yet Another Compiler-Compiler**. Disponível em <http://plan9.bell-labs.com/magic/man2html/1/yacc> acesso em 24-ago-2007.

YASSIN, A., FAYAD, M. E. Application frameworks: A survey. In: FAYAD, M. E., JOHNSON, R. E. **Domain-Specific Application Frameworks: Frameworks Experience by Industry**. New York: John Wiley & Sons, 2000. Cap. 29 p.615-632.