Faculdade de Tecnologia Senac RS
Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

ROTEIRO DO RELATÓRIO DE PROJETO PARCIAL

IDENTIFICAÇÃO

Aluno: Felipe Nascimento de Moura
Endereço Residencial: R. Duque de Caxias,278, apto. 102
Bairro: Centro CEP: 90010-180
Cidade: Porto Alegre UF: RS
Telefone(s): 51 – 3024 5696
E-mail(s): 51 – 8425 5530

Título:

Desenvolvimento de um interpretador de linguagens de alto nível, modularizado e escalável, capaz de ser integrado com diferentes linguagens e bancos de dados.

Orientador:

Guilherme Bertoni Machado

Apresentação Geral:

Este documento contextualiza o desenvolvimento de um interpretador de linguagens de auto nível, modularizado e escalável, capaz de ser integrado com diferentes linguagens e bancos de dados e usado por meio de diferentes interfaces. Os diagramas e arquivos de documentação estão todos em inglês. O Inglês foi definido como idioma padrão para a documentação devido ao fato de o projeto ser *Open Source* e pode, futuramente, contar com a colaboração de pessoal externo, e o inglês é um idioma amplamente abordado na área de desenvolvimento de softwares em âmbito global.

O sistema é todo desenvolvido baseando-se em uma estrutura de API(*Application Program Interface*), a qual pode ser consumida por diferentes interfaces, incluindo requisições HTTP (*HyperText Transfer Protocol*), o que permitirá a terceiros o desenvolvimento de sua própria interface, esteja ela rodando em um *browser* ou não, consumindo a API em forma de serviço baseado no modelo *SaaS (Software As A Service),* com referências ao padrão REST (*REpresentational State Transfer*).

Um dos focos deste projeto *Open Source* é lembrar que a mente humana tem a habilidade de criar, inovar ou tornar algo artístico, enquanto as máquinas são infinitamente superiores em tratamento lógico e matemático. Isso nos remete à conclusão de que devemos deixar para as máquinas todas as tarefas que se possa automatizar, liberando tempo para a mente humana trabalhar em tarefas que exijam suas habilidades. Trata-se de fazer um melhor uso do que se tem ao alcance em cada uma das pontas (Gams, Paprzycki, Wu; 1997).

O modelo de desenvolvimento atual já é relativamente antigo e tende a evoluir muito brevemente para uma forma ainda mais próxima da comunicação interpessoal humana, o que inspira o desenvolvimento deste projeto.

Para a realização deste trabalho será utilizado o conceito de linguagem de programação discreta, criado pelo autor deste documento, que o cita desta forma:

> *"Uma linguagem de programação discreta consiste em uma linguagem baseada em grupos variáveis de padrões e instruções com alguns marcadores padrões com a habilidade de se adaptar conforme seu uso fazendo uma transformação da forma original de alto nível para uma forma padrão em mais baixo nível"* (Moura, 2008).

Este conceito prevê que a linguagem de programação deva seguir grupos independentes de regras, sendo passível de agregação para novos grupos, ou adaptação para grupos depreciados. Isto deverá tornar a própria linguagem de programação capaz de aprender e se adaptar conforme o uso sofrendo variações inclusive conforme tradições ou costumes da região onde está sendo empregada.

Tal interpretador deverá manter alguns princípios do projeto já existente denominado theWebMind, também desenvolvido pelo autor deste trabalho de conclusão. Trata-se de uma nova versão, mais robusta e com mais embasamento científico, de forma mais escalável e receptiva à futuras alterações.

Definição do Problema:

O desenvolvimento de softwares vem há muito se adaptando e enriquecendo, adotando novas tecnologias e equipamentos cada vez mais acessíveis. O crescimento da demanda por novas aplicações é inegável e embora haja o desenvolvimento de inúmeras aplicações inovadoras nas mais diversas áreas, ainda há, e sempre haverá uma grande chamada por softwares comerciais, empresariais ou focados em determinados nichos do mercado.

O que se observa é justamente a adoção de um padrão contínuo, estudado em faculdades e cursos sobre como implementar regras e padrões no desenvolvimento de software. Porém, se ao desenvolver um software, implementamos muitos padrões e regras, devemos pensar seriamente no fato de termos máquinas hoje em dia com muito maior capacidade para implementar tais regras.

Eis o ponto onde a preocupação por uma melhor utilização do conhecimento e habilidades artísticas e relacionadas a criatividade, por parte do cérebro humano. O tempo investido na padronização e implementação de regras durante o desenvolvimento de um determinado software poderia ser aplicado em outras áreas, ou mesmo, aprimorando as já existentes.

Para a evolução dos softwares, é imprescindível que mudemos a forma como pensamos e desenvolvemos aplicações. As novas metodologias para desenvolvimento, novas técnicas e tecnologias, além de artefatos para análise e equipamentos hoje disponíveis e a acessibilidade das informações oferecem um grande leque de opções quanto a forma como desenvolveremos o próximo aplicativo, e cabe a nós, analistas e desenvolvedores decidir qual a melhor abordagem. Uma evolução nestas áreas é inevitável.

O desenvolvimento das primeiras versões deste projeto já provou que com o processamento de uma máquina comum, é possível computar muito rapidamente tais padrões, de forma quase instantânea e confiável.

Este projeto busca solucionar o seguinte problema: Seria possível, um código extremamente simples baseado em uma linguagem seminatural, desenvolver uma aplicação capaz de gerar um sistema completo, incluindo o seu banco de dados, classes, formulários, documentação, etc?

Baseando-se na questão do parágrafo anterior, os seguintes desafios devem ser resolvidos:

- Cadastrar novos projetos a serem desenvolvidos, bem como novos membros que atuarão no desenvolvimento dos mesmos. Poder-se-á especificar o SGBD (Sistema de Gestão de Bases de Dados) que se deseja usar para o desenvolvimento, o idioma a ser utilizado para programação e também a linguagem ou *framework* a serem usados para geração de um resultado final e funcional.
- Desenvolver uma estrutura de interpretação de Português e Inglês, idiomas nos quais o usuário deverá ter liberdade para escrever, bastando seguir pequenos padrões para ter uma aplicação funcional gerada, ou mesmo documentação e diagramação. Esta plataforma deverá estar preparada para conectar em um SGBD e gerar a base de dados conforme o necessário.
- Criar uma interface a fim de viabilizar o próprio usuário a criar seus próprios *plugins* ou módulos geradores de código e documentação, possibilitando o crescimento do projeto através da comunidade *open source*.

A plataforma precisará ter baixo acoplamento para que possa manter uma boa escalabilidade de serviços e módulos.

Objetivos:

Conforme os desafios citados na definição do problema, para que seja possível o desenvolvimento de um interpretador de linguagens de alto nível, modularizado e escalável, capaz de ser integrado com diferentes linguagens e bancos de dados, os seguintes objetivos devem ser alcançados:

- Definir uma estrutura padronizada e organizada para tratamento posterior a partir do código criado em alto nível;
- Desenvolver uma API para integração com a estrutura gerada, proporcionando meio para mineração e interação com os dados armazenados;
- Disponibilizar uma ferramenta para manipulação de projetos e usuários;

Análise de Tecnologias e Ferramentas:

Linguagens de programação usadas:

- PHP: (*PHP Hypertext Processor*) Linguagem de programação fracamente tipada que roda no servidor, lidando com requisições HTTP;
- PL/SQL: (*Procedural Language/Structured Query Language*) Linguagem padronizada para criação e pesquisa em banco de dados;
- XML: (*Extensable Markup Language*) Grupo de regras e padrões para estruturação de informações em documentos;
- shellscript: Script capaz de interagir com o interpretador de linhas de comando do Sistema Operacional;
- Javascript: Linguagem de programação que roda no cliente, quando em uma arquitetura cliente/servidor, normalmente executada em navegadores web;
- CSS: *(Cascade Style Sheet) Lista encadeada de estilos. É amplamente usada para padronizar a forma como as informações são exibidas na tela, normalmente, também em um navegador;*
- HTML5.

Ferramentas para o desenvolvimento:

- Netbeans 6.9: IDE avançada para desenvolvimento em diversas linguagens de programação, incluindo PHP e Javascript;
- Postgres: SGBD(*Sistema Gerenciador de Banco de Dados*) open source;
- MySQL: SGBD amplamente usado para bases de dados pequenas e ágeis;
- SQLite: SGBD que trabalha com arquivos no próprio formado do sistema

operacional, muito rápido, porém, pouco robusto.

Extensões:

- PDO: (*PHP Data Objects*);
- Finfo: (*File Info*) extensão que permite acesso mais detalhado aos arquivos no Sistema Operacional;
- Mcrypt: Biblioteca com diversos tipos diferentes de algoritmos para encriptação de dados;
- Xdebug: Ferramenta que enriquece a forma como o PHP exibe mensagens de erro, além de permitir um tratamento passo a passo de cada tarefa, erro ou exceção;
- PHPUnit: Ferramenta para criação e execução de testes unitários em PHP;
- ReadLine: Extensão que oferece diversas funcionalidades para execução do PHP em linha de comando, com um console mais interativo;
- SQLite: Extensão para integração com bases de dados SQLite;

Sistemas operacionais: Ubuntu 10, Windows XP/Seven/Server, MacOSX;

Browsers a serem suportados: Firefox 3.6+, Safari 5+, Chrome 5.0+, Opera 10.0+;

Versionador: GIT (*Global Information Tracker*) Um sistema gerenciador de versões em arquivos distribuídos.

Licença MIT: Massachusetts Institute of Technology *Open Source license.* Esta licença garante o uso, alteração e distribuição do software livremente, com tanto que se mantenha o crédito dos criadores iniciais, explícito.

Descrição da Solução:

O sistema será todo composto por módulos capazes de interagir uns com os outros, mas também capazes de oferecer funcionalidades independentemente do resto da estrutura. Estes módulos deve interagir conforme o necessário, mas não devem afetar o sistema como um todo.

Haverá um módulo específico para a interpretação do código escrito, tal módulo será dividido em outros submódulos, como validador léxico, flexionador de substantivos, identificador de *tokens,* interpretador de padrões e regras e analisador de estatísticas e probabilidades. Desta forma, podemos integrá-lo a um outro grupo, capaz de usar as informações e estruturas tratadas para fazer levantamentos e deduções, além de utilizar os mesmos para a sua aprendizagem.

Outro grupo de classes deverá ser responsável pela integração com banco de dados e padronização, bem como normalização do mesmo.

Em outro conjunto de classes, deverá haver o tratamento para sua API, de forma a possibilitar acesso de aplicações externas a fim de fazer uma mineração nos dados levantados pelo novo motor do theWebMind.

Na solução proposta, módulos podem ser considerados pacotes de classes e as requisições podem vir tanto via linha de comando, quanto por HTTP simulando o uso de REST.

Rest foi escolhido como modelo para conexões via HTTP por ser de muito rápida aprendizagem e manter um padrão que a própria estrutura do protocolo HTTP oferece nativamente. Além disto, acaba melhor aproveitando os dados enviados e recebidos adotando o padrão Json. Todavia, não foi implementada toda a interface de RestFull, que prevê diferentes cabeçalhos para cada tipo de requisição, adotando tão somente o Post como método de recebimento de solicitações.

Existe a possibilidade de instalar a aplicação no servidor para um uso mais fácil por meio de linhas de comando.

Ao efetuar a instalação, uma base de dados em SQLite será iniciada.

Todos os projetos e usuários serão armazenados nesta base de dados, ilustrada pela Figura 1, que também ficará como responsável por manter o

histórico das alterações. Nesta base de dados, cada trecho do código gerado é armazenado separadamente, como classes, métodos, propriedades, interfaces, entre outros. Estes itens vinculam-se em forma hierárquica a fim de identificar a estrutura dinamicamente, ainda mantendo um status que indica qual a versão atual.

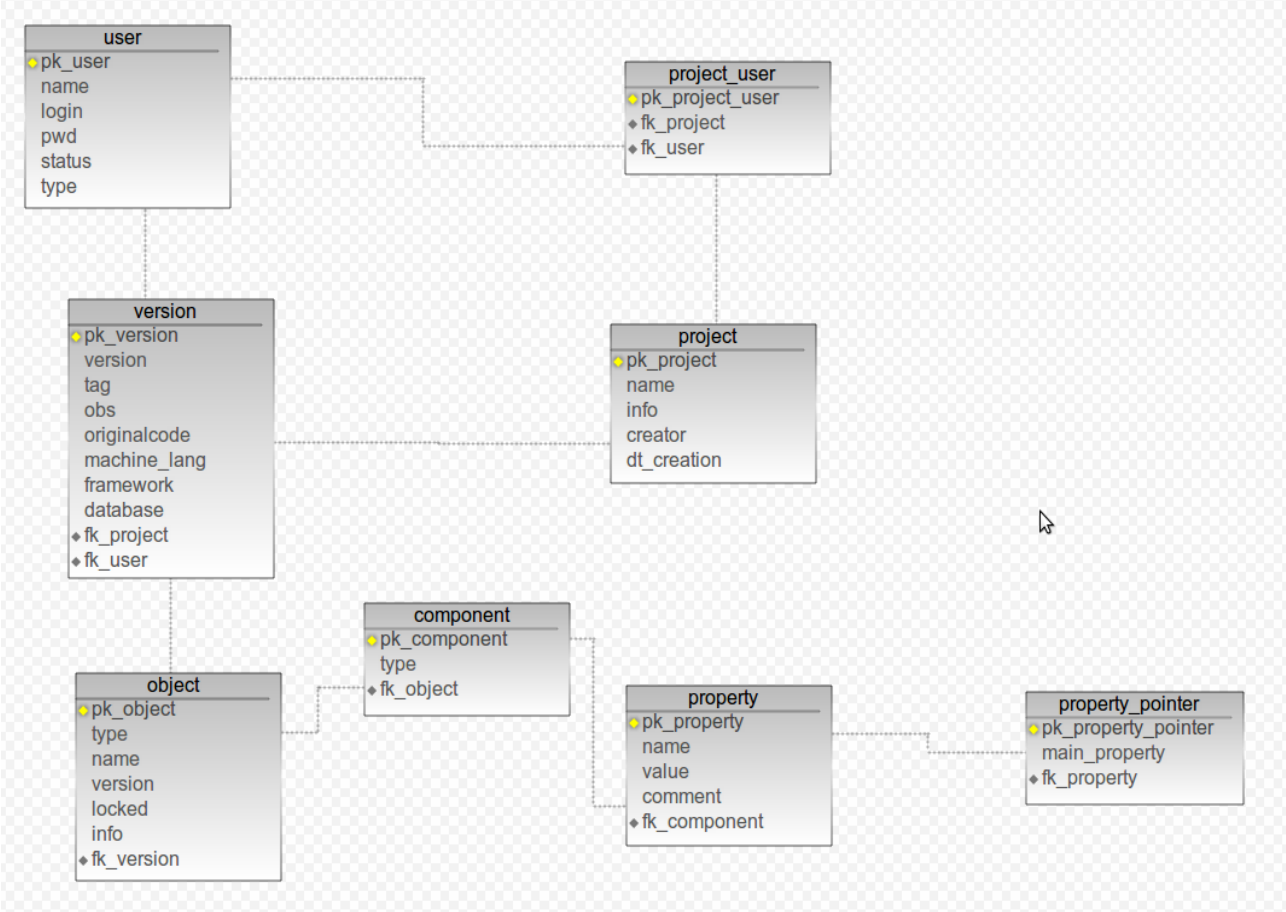

Figura 1: Diagrama ER com a estrutura das tabelas na base de dados em SQLite

O usuário precisará autenticar-se com um login e senha antes de executar determinados comandos.

As tarefas disponíveis no sistema são dividias em dois principais níveis, a *camada de programas* onde encontram-se os comandos executados diretamente pelo usuário, e a *camada do cortex*, camada onde encontram-se as classes e conjuntos de classes responsáveis pela interpretação, “tokenização”, padronização e normalização das informações, e também para geração dos códigos em uma etapa final. Além destas duas camadas de funcionalidades, haverão também as classes para o controle da comunicação com o usuário implementando L10N para *localization,* com diferentes idiomas.

Há também uma camada de *integração de plugins*. O sistema oferecerá suporte a *plugins* por meio de chamadas padronizadas para cada *plugin*, aceitando em sua descrição o que será o gatilho, o *trigger*, que disparará a chamada de cada *plugin*. Estes *plugins* são basicamente uma classe seguindo um padrão documentado, estendendo a classe MindPlugin e implementando a interface *Plugin*.

Todas as demais classes estão classificadas como *operacionais*. A arquitetura de camadas do sistema é ilustrada pela Figura 2.

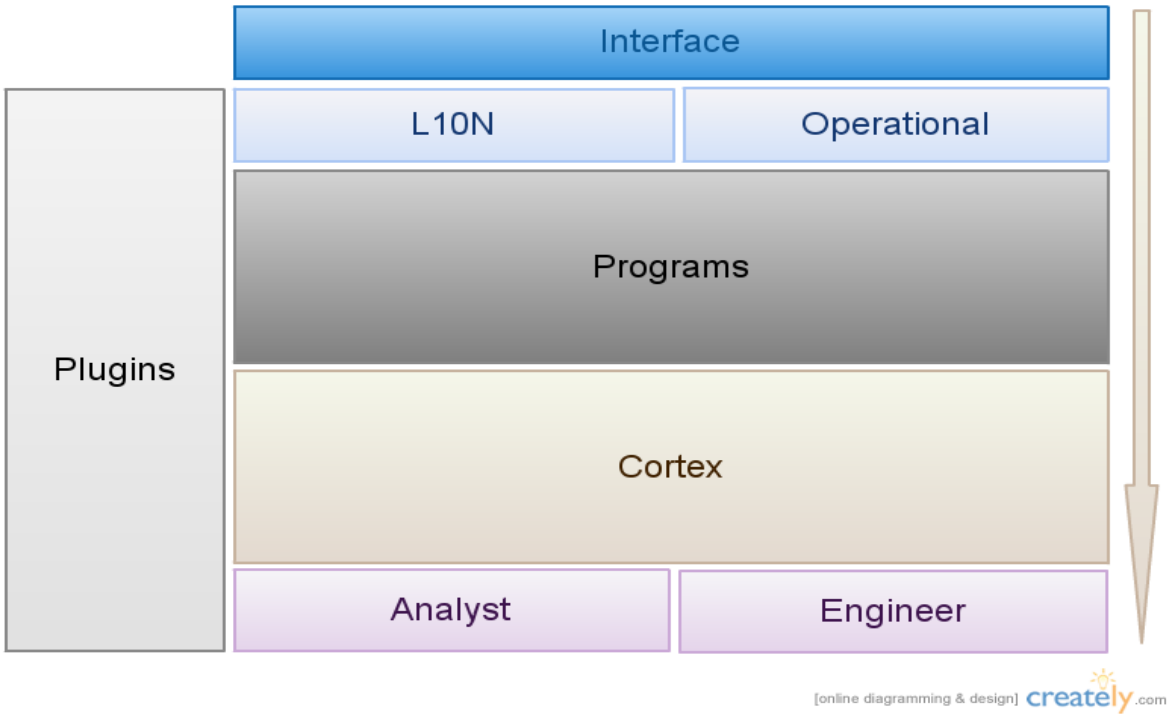

Figura 2: Camadas da aplicação

Para padronização, serão transportadas informações em formado JSon, quando a requisição vier por meio do protocolo HTTP.

O recebimento de informações via HTTP se fará por meio do método POST, seguindo a nomenclatura dos programas acessíveis pelo usuário autenticado, e seus parâmetros, que podem ser vistos através do comando *help* de cada programa.

Caso a requisição vier por uma chamada via linha de comando, o retorno se dará diretamente por mensagens exibidas no console.

Abordagem de Desenvolvimento:

A metodologia adotada será uma variação da Scrum, pois não terá as reuniões diárias, porém contará com a organização de *sprints* e documentação de metas com muitas pequenas entregas. A tabela com os dados dos *sprints* está disponível junto ao cronograma, neste documento.

Também serão usadas técnicas de *Test Driven Development* (TDD), para testes e implementação (Beck, 2003).

A maior parte do sistema manter-se-á com uma abordagem Orientada a Objetos, porém, implementando também funcionalidades REST e tratando as aplicações do sistema como serviço, lembrando SaaS.

As alterações feitas no projeto estarão sendo versionadas com o controlador de versões GIT, sendo assim, todo o histórico das alterações estará seguro e atualizado em http://github.com/felipenmoura juntamente com todos os *logs* e documentação. Por se tratar de um projeto *open source*, existe a possibilidade de outros colaboradores alterarem algum código ou documento, todavia, por hora apenas o autor deste artigo está cadastrado como administrador do projeto com permissões de escrita, e também, para alterações futuras por parte de novos membros, sempre constará um *log* seguro e preciso de cada arquivo alterado, inserido ou removido do servidor.

Serão adotados os comentários de *Annotation*, ou seja, todas as classes, *packages/namespaces*, interfaces e demais blocos de código estarão devidamente comentados seguindo um padrão que possibilita a geração automática de documentação, ao término do desenvolvimento, o que agilizará a documentação final da API e também das próprias classes.

Importante notar também que, justamente por se tratar de um projeto *Open Source*, grandes alterações podem ocorrer desde refatorar um módulo na análise, até reescrever classes de código, devido a possíveis intervenções no decorrer do desenvolvimento. Estas intervenções podem ocorrer em função de novas pesquisas, novos projetos a serem incorporados, novos contatos com pessoas de diversas áreas.

Há contato já iniciado com pesquisadores nas mais diversas áreas de estudo de idiomas em Inglês, Português e Espanhol, podendo-se expandir conforme novos contatos e oportunidades surjam.

Arquitetura do Sistema:

Este sistema contará com uma estrutura, conforme já citado anteriormente, dividida em algumas camadas.

A camada *L10N* fica como responsável pela exibição de mensagens para o usuário, enquanto a camada de *classes operacionais* fica responsável por receber e lidar com as requisições trazidas pela camada da *interface*.

A camada *programs* controlará os programas que podem ser carregados, enquanto a camada *cortex* é a responsável pela análise, compreensão e transformação do conteúdo digitado pelo usuário.

A camada do *cortex* é subdividida em dois novos níveis, o *engineer* e o *analyst*, um responsável pela geração de arquivos e base de dados, e outro pela análise léxica e sintática e levantamento de padrões e *tokens*, respectivamente. Na parte de análise, haverá uma identificação de verbos, substantivos e quantificadores, que deverão ser flexionados para sua forma canônica, ou seja, para o singular, no gênero masculino caso haja flexão de gênero (Gonzalez et al, 2003).

A Figura 3 apresenta o diagrama parcial de classes do projeto. Este diagrama mostra a estrutura principal das classe e suas relações, mas alguns métodos e propriedades ainda serão adicionados conforme eu desenvolvimento do sistema, e novas pesquisas e contatos.

A Figura 4 define o mapa mental do projeto. Este mapa mental mostra as funcionalidades e módulos que compões o projeto em altíssimo nível. A Figura 5 apresenta uma forma mais detalhada do módulo Córtex e suas tarefas.

A Figura 6 mostra o diagrama de sequência do *lifetime* de uma requisição para análise de um código de alto nível. A Figura 7 ilustra o fluxo de uma requisição desde a chamada por parte do usuário, seja ela por HTTP ou por linha de comandos, até as ações da aplicação.

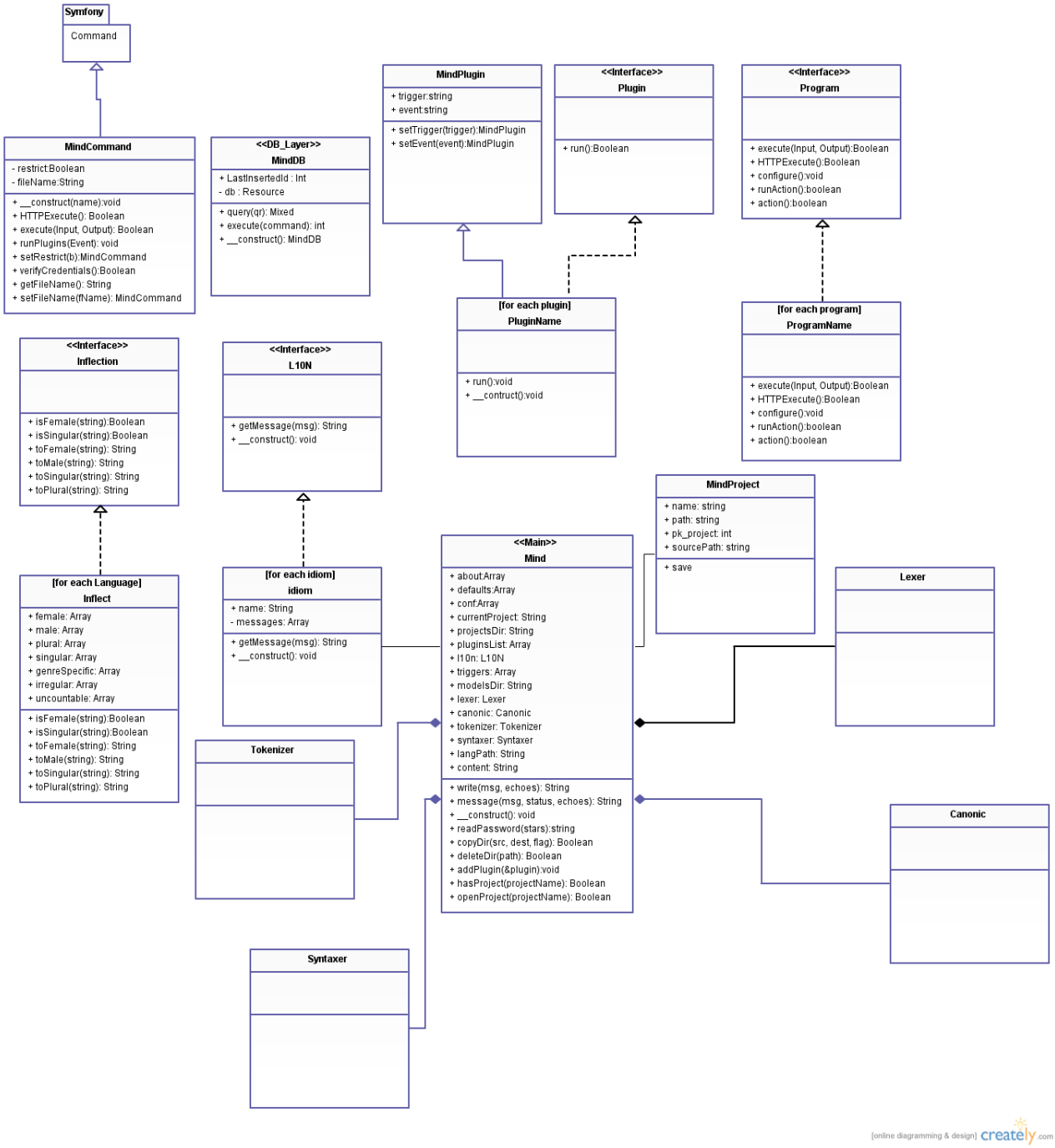

Figura 3. Diagrama de classes parcial

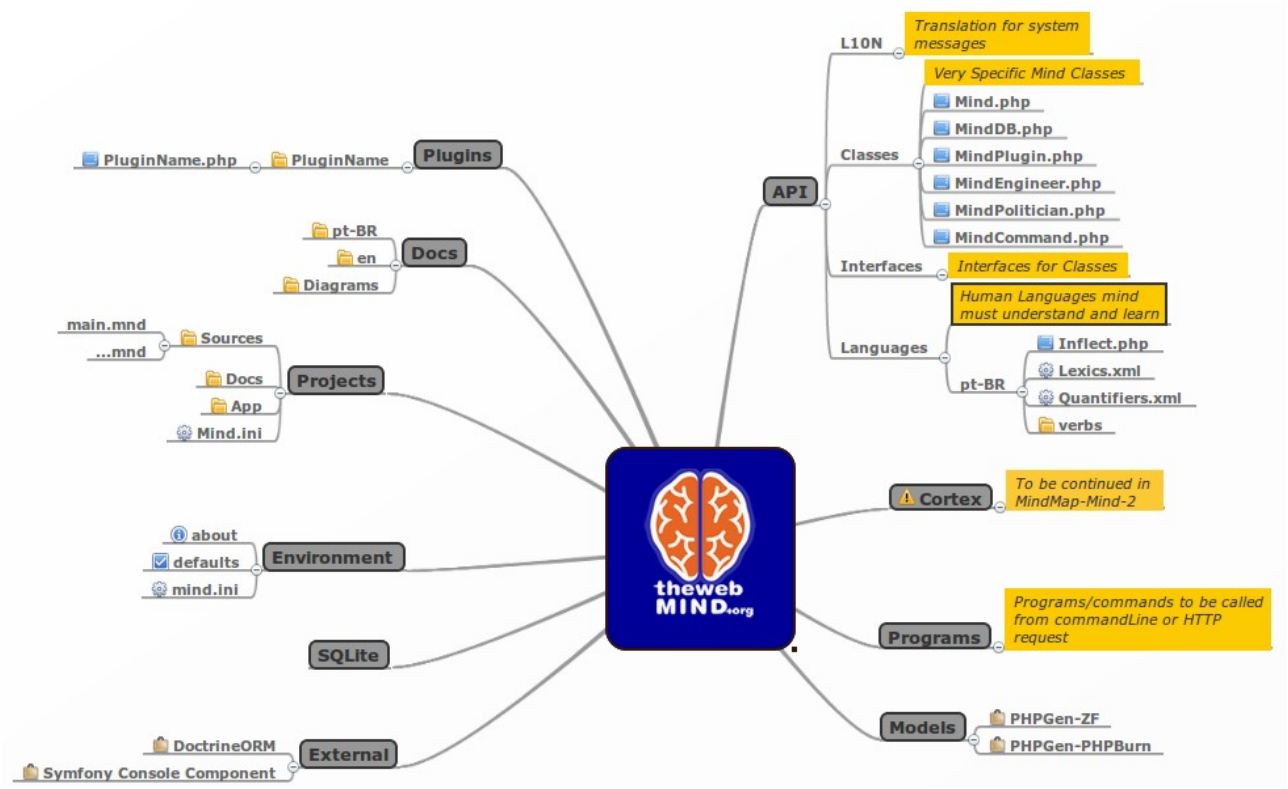

Figura 4. Funcionalidades e módulos que compões o projeto em altíssimo nível

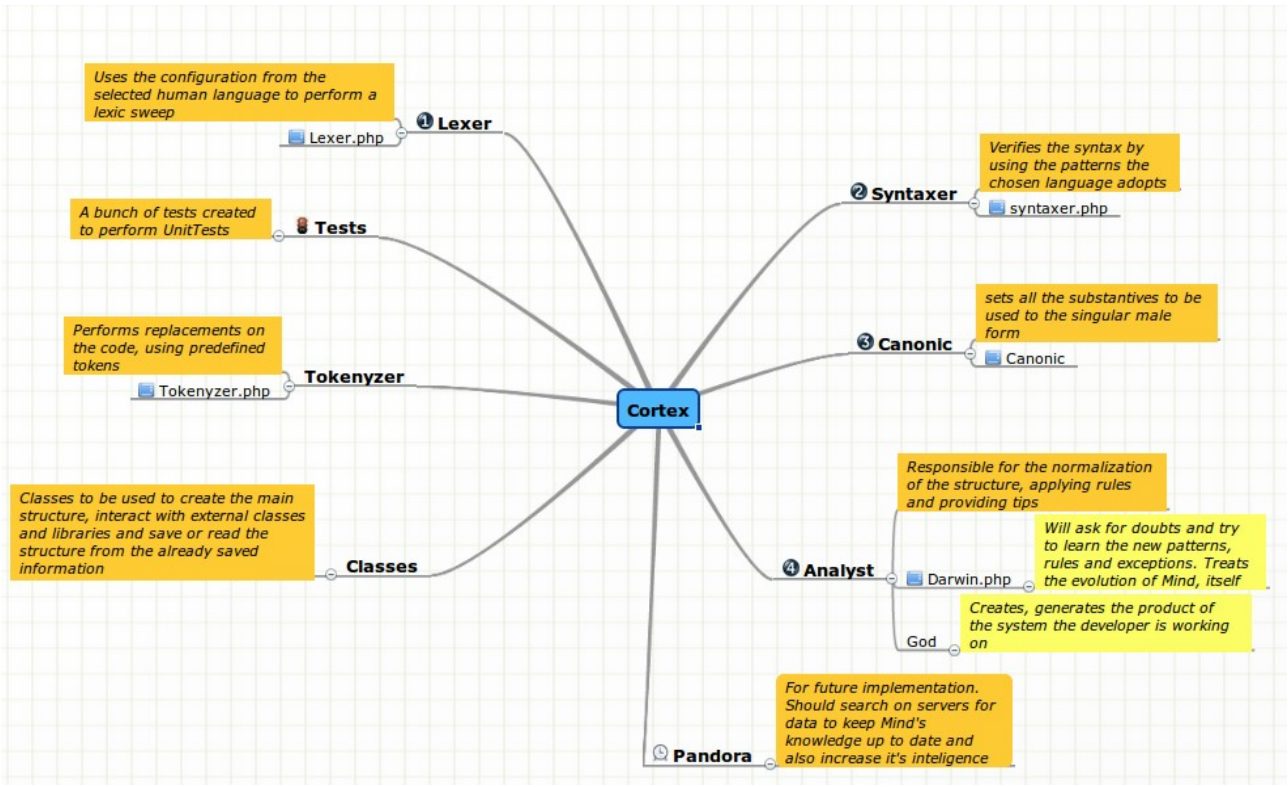

Figura 5. Uma forma mais detalhada do módulo Córtex e suas tarefas

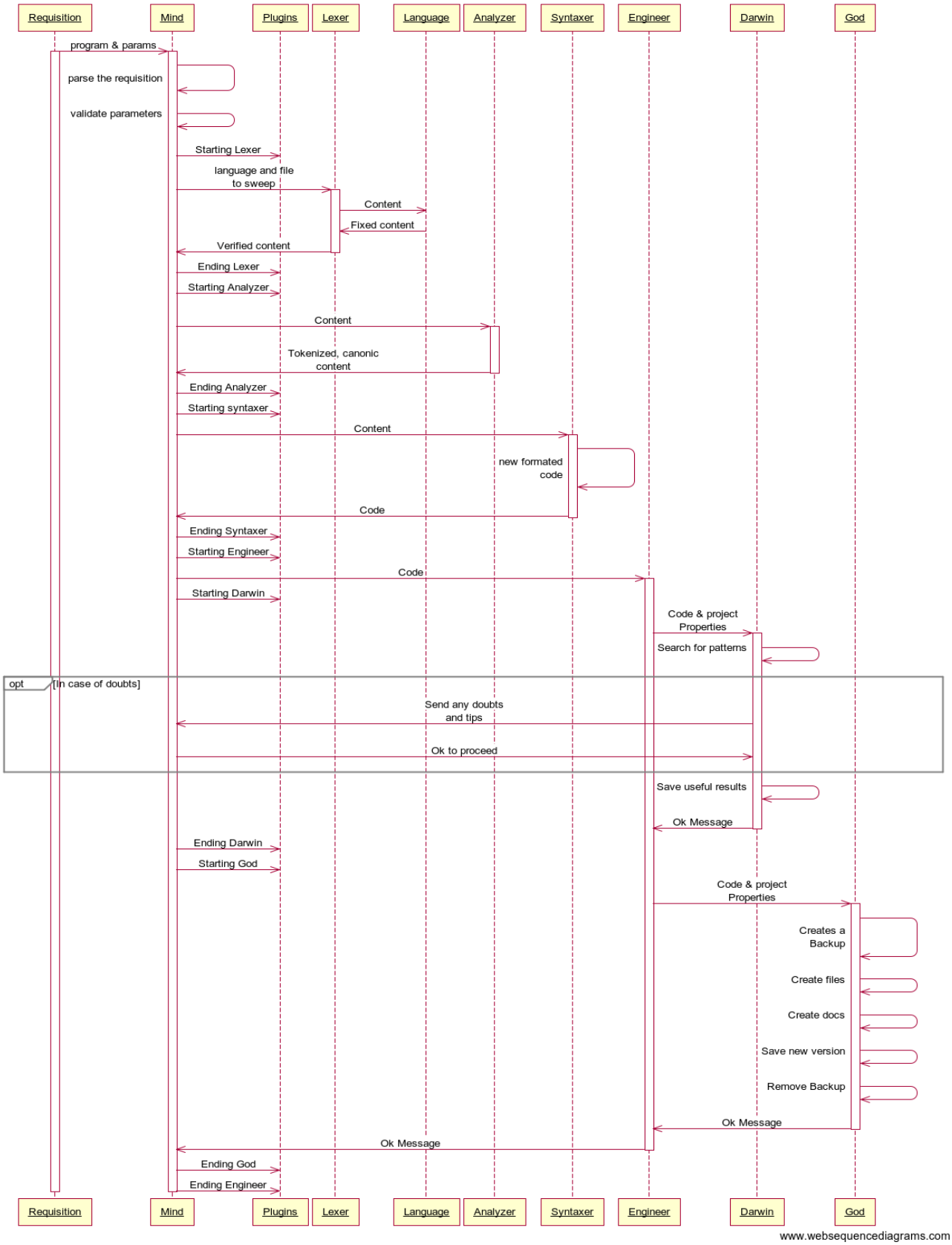

Figura 6. Diagrama de sequência de uma requisição para análise de um código de alto nível

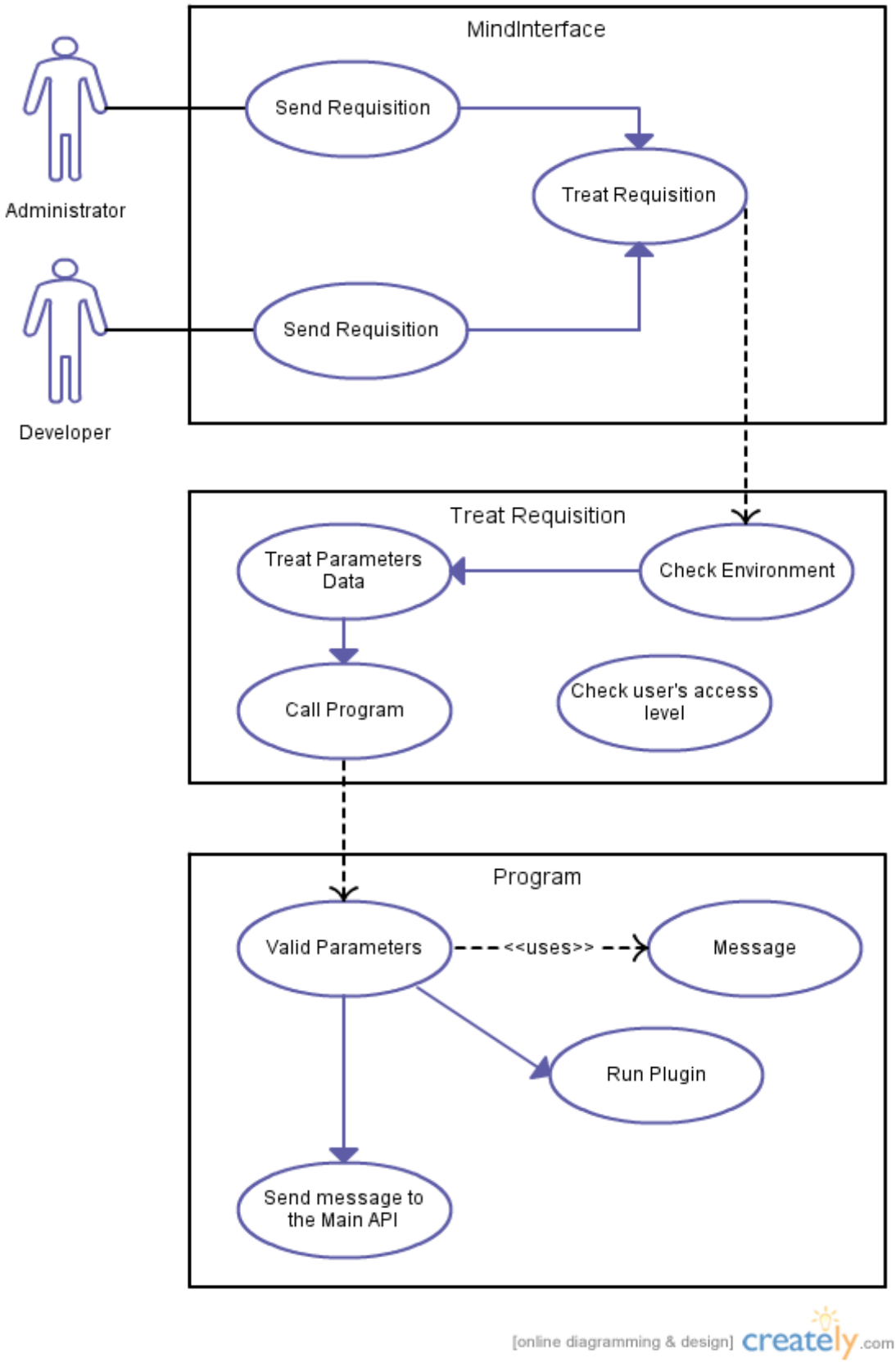

Figura 7. Fluxo de uma requisição

Funcionamento do Sistema

Neste protótipo, temos disponível a integração dos ambientes por meio da API, já aceitando requisições tanto por linha de comando quanto pelo protocolo HTTP.

É possível cadastrar novos usuários com informações como login e senha. E também pode-se cadastrar novos projetos. A ferramenta para autenticação de usuário também está logando.

O analisador do texto digitado pelo usuário já teve seus trabalhos iniciados e já conta com a classe *Inflect* funcionando para o idioma Português, transformando termos para o singular ou plural, e para o feminino ou masculino.

A camada de controle léxico também está funcionando, eliminando caracteres que não devam entrar no idioma selecionado para o projeto, além de determinar as divisões de palavras e linhas.

Há testes já implementados para as funcionalidades da classe *Inflect*.

Validação:

A forma de validação a ser feita de duas maneiras.

Na primeira, haverá uma sequência de testes unitários automatizados a serem executado a fim de validar funcionalidades de classes e módulos. Num segundo momento, haverá um questionário para validação das opiniões de ao menos 10 usuário que tenham testado o novo *core*. Tais usuários serão analistas e desenvolvedores ativos na comunidade *Open Source*.

O questionário terá como metodologia, uma avaliação mais qualitativa que quantitativa, sendo composto por poucas questoes, focadas na opinião e impressão tida durante a experiência com o software, oferecendo liberdade para expressar opiniões sobre melhorias, problemas ou ideias para atualizações. Uma melhor definição para esta metodologia é referida pelo ibope (IBOPE, 2010).

Serão cinco perguntas, cujas respostas serão usadas para gerar uma nuvem de termos, visando encontrar os termos mais relevantes. As conclusões a partir deste questionário virão baseadas nesta nuvem de termos, e nas ideias propostas.

Os testes serão escritos com o uso de Testes Unitários com a utilização do PHPUnit.

A primeira bateria de testes, ilustrada pela Figura 8, atinge 100% de acertos, executada sobre a classe *Inflect* para flexão de substantivos em português, alterando ou detectando seu gênero e número.

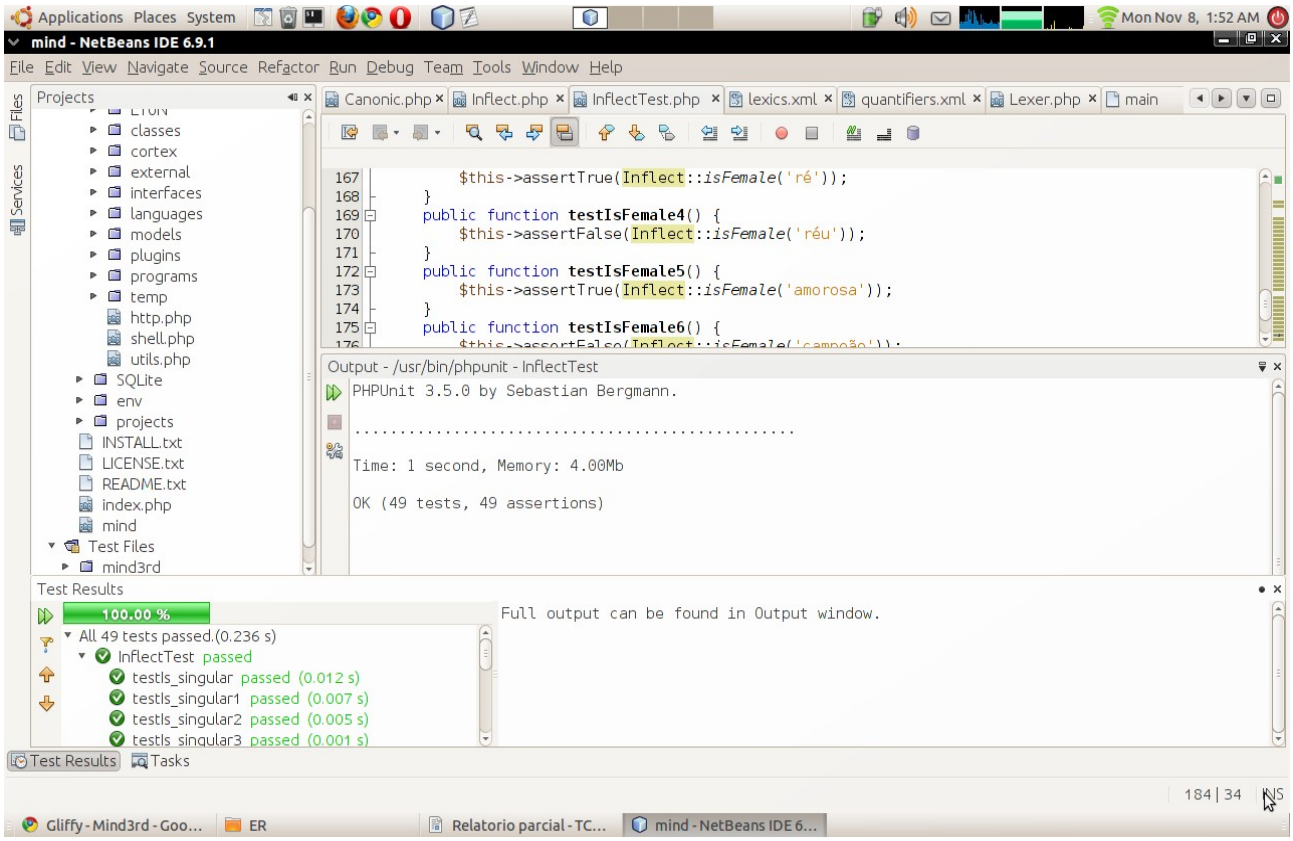

Figura 8. Relatório dos testes executados a partir de uma IDE.

Cronograma:

O cronograma deste TCC foi todo baseado em *Sprints* de 15 dias, com datas programadas e concluídas, status e pontuação indicando o grau de dificuldade.

Pode-se também visualizar neste cronograma, separada por *sprints*, o produto de cada tarefa.

Abaixo, segue a legenda para a tabela de product backlog:

Types:

Documento

Feature, um trecho do sistema funcionando;

Correção de algum problema encontrado;

Pesquisa, envolve também contatos e reuniões.

Status:

Ok: Tarefa concluída e adicionada à documentação;

Omlg: Tarefa concluída, mas ainda demanda de testes para se adicionada à documentação;

And: Tarefa sendo desenvolvida no momento;

Pen: Tarefa ainda não iniciada

Pts:

Indicam o grau de dificuldade em uma escala de 1 a 8.

ID	Produto	Tipo	Status	Pts.	Entrega
Sprint 1					05/09/10 a 20/09/10
1	Plano de trabalho	Documento	Ok	6	10/09/10
2	MindMap 1	Documento	Ok	4	15/09/10
3	Diagrama de sequência versão 1	Documento	Ok	4	20/09/10
Sprint 2					21/09/10 a 05/10/10
4	Diagrama de classes versão 1	Documento	Ok	4	39/09/10
5	Criação das interfaces(OO)	Feature	Ok	2	02/10/10
6	Diagrama de sequência versão 2	Documento	Ok	4	05/10/10
7	MindMap 2	Documento	Ok	2	05/10/10
Sprint 3					05/10/10 a 20/10/10
8	Diagrama de classes versão 2	Documento	Ok	4	07/10/10
9	Bootstrap para requisições	Feature	Ok	8	15/10/10
10	Analizador léxico	Feature	Ok	8	20/10/10
Sprint 4					21/10/10 a 04/11/10
11	Classe Inflect em portguês	Feature	Ok	8	29/11/10
12	Testes unitários para Inflect	Feature	Ok	1	01/11/10
13	Diagrama ER	Documento	Ok	2	02/11/10
14	Classes para controle da base SQLite	Feature	Ok	2	04/11/10
Sprint 5					05/11/10 a 16/11/10
15	Programas básicos(clear,exit,use...)	Feature	Ok	6	10/11/10
16	Programa analyse	Feature	Omlg	8	16/11/10
17	Programas create e show	Feature	Omlg	6	16/11/10
18	Relatório de projeto parcial	Documento	Ok	6	16/11/10

Planejamento futuro:

ID	Produto	Tipo	Status	Pts.	Entrega
Sprint 6					17/11/10 a 01/12/10
19	Identificador de verbos	Feature	Pen	6	26/11/10
20	Evolução da classe Inflect em Pt.	Feature	Pen	5	30/11/10
Sprint 7					18/12/10 a 01/01/11
21	Tokenizer	Feature	Pen	6	24/11/10
22	Padronização da estrutura do conhecimento	Pesquisa	Pen	8	01/01/11
Sprint 8					02/01/11 a 16/01/11
23	Controle e estrutura para plugins	Feature	Pen	4	08/01/11
24	Padronização da estrutura dos módulos do Cortex	Pesquisa	Pen	8	16/01/11
Sprint 9					17/01/11 a 31/01/11
25	Programas para gerência de projetos	Feature	Pen	4	20/01/11
26	Programas para gerência de usuários	Feature	Pen	4	24/01/11
27	Programas para gerência de plugins	Feature	Pen	4	31/01/11
Sprint 10					01/02/11 a 15/02/11
28	Classes responsáveis pela aprendizagem	Feature	Pen	8	12/02/11
29	Classes para armazenamento em arquivos	Feature	Pen	4	15/02/11
Sprint 11					16/02/11 a 02/03/11
30	Classe para classificação de dados	Feature	Pen	4	19/02/11
31	Classes para tratamento de erros	Feature	Pen	4	26/02/11
32	Classes para integração com o usuário	Documento	Pen	4	02/03/11
Sprint 12					03/03/11 a 18/03/11
33	Classes para definição de dúvidas, dicas e tomadas de decisão parte 1	Feature	Pen	8	11/03/11
34	Classes para definição de dúvidas, dicas e tomadas de decisão parte 2	Feature	Pen	8	18/03/11
Sprint 13					19/03/11 a 02/04/11
35	Módulo inicial para geração de códigos	Feature	Pen	8	22/04/11
36	Ferramentas para geração da base de dados	Feature	Pen	8	02/04/11
Sprint 14					03/04/11 a 18/04/11
37	Módulo para geração de códigos parte 2	Feature	Pen	8	14/04/11
16	Ferramenta para visualização de cógido gerado	Feature	Pen	6	18/04/11
Sprint 15					19/04/11 a 03/05/2011
38	Ferramenta para edição de código gerado	Feature	Pen	4	25/04/11
39	Documentação para o desenvolvedor	Documento	Pen	8	03/05/11

Sprint 16					04/05/2011 a 19/05/2011
40	Site para divulgação e documentação	Documento	Pen	8	16/05/11
41	Questionário para avaliação e testes	Feature	Pen	8	19/05/11
Sprint 17					20/05/2011 a 31/05/2011
42	Análise dos resultados do questionário	Documento	Pen	4	26/05/11
43	Entrega do relatório e feedback	Documento	Pen	8	31/05/11

Referências:

Beck, K. Test-Driven Development by Example, Addison Wesley, 2003

Gonzalez, M.; Toscani, Daniela; Rosa, Letícia; Dorneles, Rita; Lima, Vera L. S. Normalização de itens lexicais baseada em sufixos. XVI Brazilian Symposium on Computer Graphics and Image Processing - (SIBGRAPI). I Workshop em Tecnologia da Informação e Linguagem Humana, São Carlos, 2003.

IBOPE. Conheça os tipos de pesquisa realizados pelo Grupo IBOPE. 2010. Disponível em: http://tinyurl.com/22vg28w

M. Gams, M. Paprzycki, and X. Wu (Eds.). 1997. Mind Versus Computer: Were Dreyfus and Winograd Right?. IOS Press, Amsterdam, The Netherlands, The Netherlands.

Moura, Felipe Nascimento de. theWebMind Project Documentation. 2008. Disponível em: http://docs.thewebmind.org/

Aho, Alfred; Sethi, Ravi; Ullman , Jeffrey D.; Compilers Principles, Techniques and Tools.

Briscoe, Ted; Carroll, John; Generalized Probabilistic LR Parsing of Natural Language (Corpora) with Unification-Based Grammars.

Richardson, Leonard; Ruby, Sam; “RESTful Web Services”

LX Center, “Language Resources and Technology for Portuguese” disponível em: http://lxcenter.di.fc.ul.pt/services/pt/

Componentes reutilizados:

-Algumas bibliotecas da versão anterior do próprio sistema;
-PDO, *drive* para abstração da camada de banco de dados no PHP;
-SQLite e PHP-SQLite
-Symfony Console
-Biblioteca Inflection para flexão de substantivos em inglês encontrada em http://php.net