

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:

TheWebMind project - mind3rd release

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, tratando-se de uma padronização para documentação e considerado até mesmo uma boa prática no desenvolvimento.

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.

Este documento está disponível no link <http://goo.gl/bfyBs>.

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, Felipe, 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. Esta nova versão deverá fazer um tratamento mais detalhado e inteligente de situações, muitas das quais, não previstas na versão anterior.

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 há 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 desenvolver uma aplicação capaz de interpretar um código extremamente simples baseado em uma linguagem seminatural de extremo alto nível, em diferentes idiomas, e com tais informações fazer uma análise e gerar a base de dados aplicando regras como

normalização ou otimização, documentação e por que não, o próprio código da aplicação?

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.

Tal escopo, a ser abordado neste projeto deve contemplar o desenvolvimento núcleo, responsável pela interpretação da linguagem de alto nível, gerando uma base de conhecimento reutilizável, capaz de ser plugada a outras saídas. Tais saídas poderão ser desenvolvidas posteriormente, de forma separada por demais interessados em colaborar com o projeto. Para testes e demonstração, será desenvolvida uma forma de visualização dos dados estruturados nesta base de conhecimento.

Uma API será desenvolvida juntamente neste escopo a fim de possibilitar uma maior integração, e de permitir um desenvolvimento menos encapsulado.

Uma vez que o básico para o funcionamento deste sistema exigirá um cadastro de usuários e projetos, mesmo indiferentemente dos módulos tercerizados, futuramente agregados, tais ferramentas, para manuseio de usuários e projetos também será entregue.

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:

- Desenvolver o core, capaz de interpretar um texto em altíssimo nível usando grupos pontuais de regras, de acordo com o idioma selecionado;
- Estruturar de forma a permitir adição de novos idiomas para interpretação;
- Possibilitar integração com plugins, que podem ser adicionados à aplicação;
- Desenvolver a estrutura necessária para que se possa adicionar novos drivers capazes de lidar com diferentes SGBDs;
- 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:

As ferramentas para o desenvolvimento deste sistema foram selecionadas baseadas no conhecimento já adquirido com experiência de uso ou conselhos de pessoas influentes. Estas ferramentas já foram, em geral, amplamente usadas e testadas, o que oferece maior credibilidade.

Todas as ferramentas utilizadas são Open Source, e gratuitas. Isto influencia no crescimento do projeto por não afetar orçamento, e também por utilizar ferramentas de muita aceitação no mercado, as quais facilmente pode-se encontrar profissionais qualificados, além de reportar qualquer tipo de problema com muito maior acesso a informações e fontes.

Linguagens de programação usadas:

- PHP: (*PHP Hypertext Processor*) Linguagem de programação fracamente tipada que roda no servidor, lidando com requisições HTTP. Esta linguagem de programação trabalha apenas no lado servidor, porém é de muito fácil instalação, servirá muito bem aos propósitos do sistema, pois será necessário centralizar a base de conhecimento dos projetos em um servidor principal, e também oferece funcionalidades para modo linha de comando do console, o *PHPCli*, citado abaixo;
- PL/SQL: (*Procedural Language/Structured Query Language*) Linguagem padronizada para criação e pesquisa em banco de dados. É utilizada para a manutenção do conhecimento absorvido sobre o projeto a ser desenvolvido, apoiando-se em *SQLite*, também a ser detalhado em seguida;
- XML: (*Extensible Markup Language*) Grupo de regras e padrões para estruturação de informações em documentos. Com esta linguagem de marcação torna-se muito prático e rápido a mineração em informações não tão numerosas, além de ter alguns interpretadores(parsers) nativos no PHP;
- Shellscript: Script capaz de interagir com o interpretador de linhas de comando do Sistema Operacional. Será utilizado amplamente para testar a API desenvolvida;
- Javascript: Linguagem de programação que roda no cliente, quando em uma arquitetura cliente/servidor, normalmente executada em navegadores web. Será muito útil para demonstrarmos o funcionamento da aplicação por meio de requisições *Ajax(Asynchronous JavaScript and XML)*, a fim de consumir a mesma API utilizada pelo console;
- 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. Será utilizada para enriquecer a visualização dos resultados nos testes da aplicação;
- HTML: (*Hyper Text Markup Language*) Linguagem base a ser utilizada para qualquer saída do servidor, direcionada a um navegador quando a requisição for pelo protocolo HTTP. Também será utilizada para construir as telas de teste da aplicação.

Ferramentas para o desenvolvimento:

- Netbeans 6.9: IDE avançada para desenvolvimento em diversas linguagens de programação, incluindo PHP e Javascript. Oferece recursos de *auto-complete* e documentação altamente desenvolvidos, além de contar nativamente com funcionalidades oferecidas por outras IDEs somente em forma de plugins, como gerenciador *SVN(Subversion)*, GIT ou integração com XDebug e PHPUnit;
- PostgreSQL: SGBD(*Sistema Gerenciador de Banco de Dados*) open source. Esta base de dados será analisada a fim de detectarmos padrões muito

similares aos adotados pelo SGBD Oracle ou SQLServer, a fim de identificar qualquer notoriedade ou dependência que deva ser abordada, visando a geração de código para esta base, e manutenção dos dados da mesma, futuramente;

- MySQL: SGBD amplamente usado para bases de dados pequenas e ágeis. Aplica-se o mesmo interesse comentado acima, sobre PostgreSQL;
- SQLite: SGBD que trabalha com arquivos no próprio formato do sistema operacional, muito rápido, porém, pouco robusto. Esta base de dados é ágil e conta com integração nativa em diversas linguagens e plataformas. Será utilizada para armazenar e oferecer suporte a buscas rápidas por dados já abstraídos de análises feitas sobre o sistema a ser desenvolvido ou mesmo, sobre sistemas desenvolvidos anteriormente.

Extensões:

- PDO: (*PHP Data Objects*) Responsável por abstrair a camada da base de dados. Esta extensão, além de facilitar o manuseio da base de dados da própria aplicação, também ajudará a lidar com a base de dados do sistema a ser desenvolvido;
- Finfo: (*File Info*) extensão que permite acesso mais detalhado aos arquivos no Sistema Operacional. Esta extensão oferece maior controle, rigidez e consequentemente segurança, ao lidar com arquivos no servidor;
- 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, em modo debug;
- PHPUnit: Ferramenta para criação e execução de testes unitários em PHP. A utilizaremos a fim de manter cada módulo constantemente testado individualmente;
- ReadLine: Extensão que oferece diversas funcionalidades para execução do PHP em linha de comando, com um console mais interativo. Normalmente esta extensão já está instalada por padrão e será utilizada para os momentos em que a requisição chega à API por um protocolo não HTTP, ou seja, por linha de comando em console;
- SQLite: Extensão para integração com bases de dados SQLite;

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

Os sistemas operacionais proprietários serão utilizados para testes da aplicação, que deverá rodar naturalmente em qualquer dos ambientes citados. O desenvolvimento da aplicação se dará em Linux, um Sistema Operacional Open Source e gratuitos.

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

Tratam-se de navegadores de internet robustos, que oferecem algumas das funcionalidades de última geração, além de se aproximarem muito dos requisitos dos padrões formalizados pela W3C(World Wide Web Consortium), além de serem navegadores ágeis, seguros e, com poucas exceções, rodam gratuitamente em diferentes sistemas operacionais;

Versionador: GIT (*Global Information Tracker*) Um sistema gerenciador de versões em arquivos distribuídos. A versão anterior do projeto está versionada com SVN, porém, GIT é capaz de integrar-se a uma base SVN sem problemas, é mais performático e oferece alguns recursos não contemplados pelo SVN

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 explicitamente o crédito dos criadores iniciais.

Descrição da Solução:

O sistema deverá ser capaz de interpretar um texto em altíssimo nível e fazer a análise sobre o problema, gerando uma estrutura que ofereça possibilidades reais para adição de novas ferramentas e funcionalidades. Deve ter a habilidade de interpretar o texto em diferentes idiomas e a partir disto aplicar todas as normas e regras possíveis, além de tentar seguir o que é aconselhado pelas boas práticas.

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 serão futuramente armazenados 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.

A base atual já suporta um controle sobre o histórico das análises feitas sobre o código escrito pelo usuário e sua evolução, bem como logs.

A aplicação é capaz de, ao salvar o andamento, salvar apenas as alterações em uma nova versão para cada entidade ou propriedade alterada, mantendo as demais sem alteração alguma.

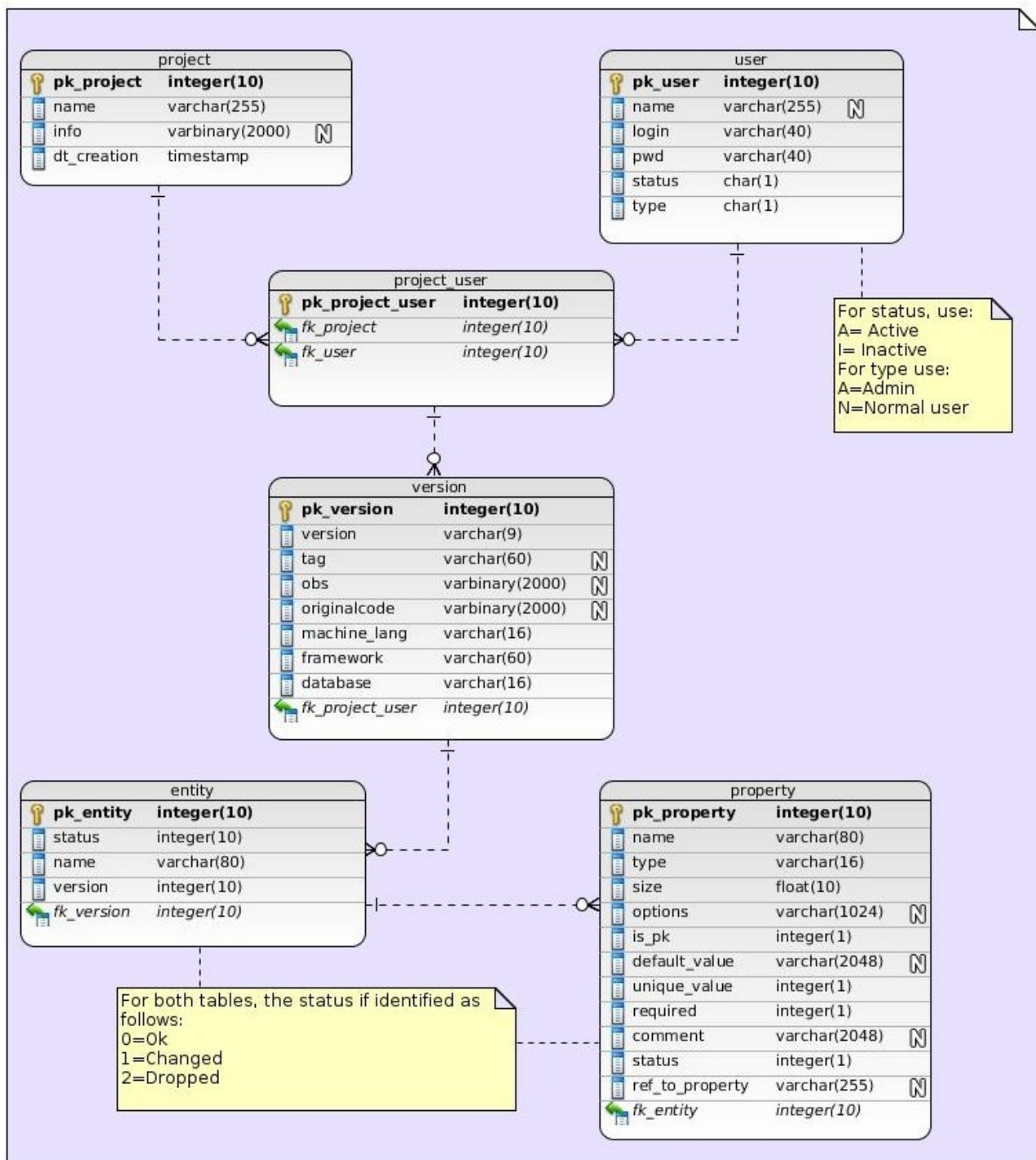


Figura 1: Diagrama ER com a estrutura das tabelas na base de dados interna, em SQLite

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

As tarefas disponíveis no sistema são divididas 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 pela geração dos códigos em uma etapa final. Além destas duas camadas de funcionalidades, haverá 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 em uma visão macro, é ilustrada pela Figura a seguir(Figura 2).

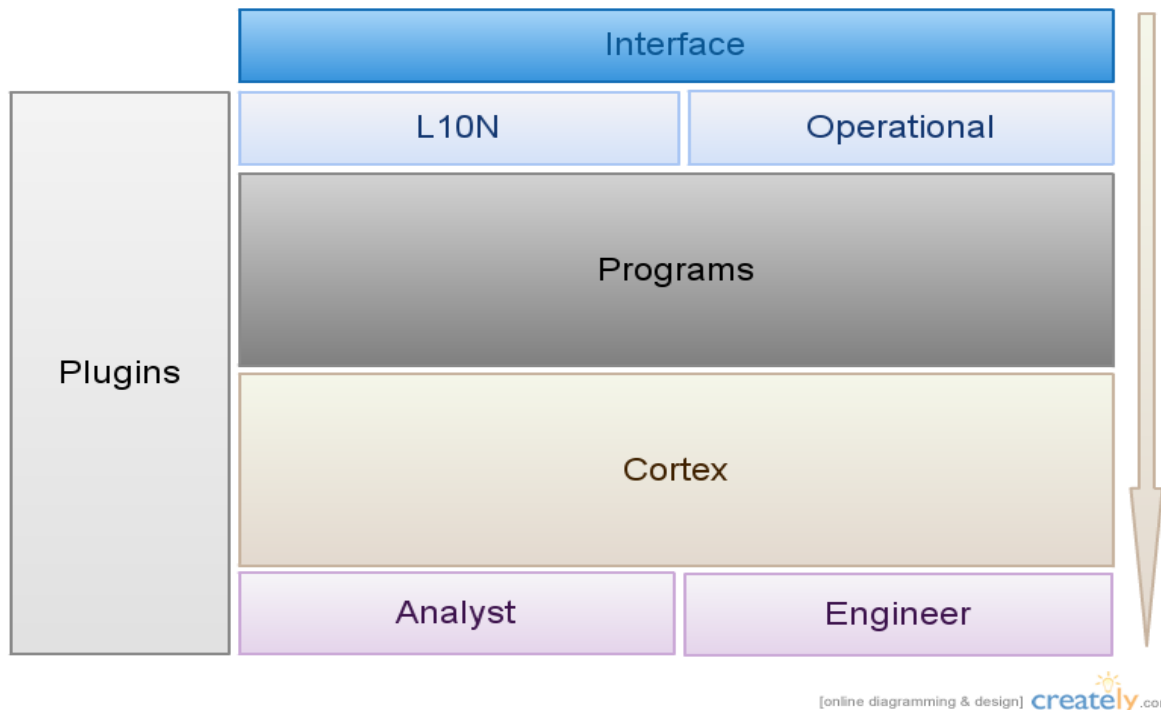


Figura 2: Camadas da aplicação em visão macro

Tais camadas devem ser vistas de forma semântica, uma vez que não necessariamente haverá divisões como pacotes ou classes específicas para cada uma delas, porém, tendo elas suas tarefas com objetivos e aspectos específicos. Vejamos algumas definições mais detalhadas de cada camada:

A camada de *Interface* será a responsável por receber as requisições e retornar as saídas oferecidas pela aplicação. Corresponderia às portas da API.

As camadas *L10N* e *Operational* ligam-se à camada de interface, sendo as classes e documentos estruturados da camada *L10N*, responsáveis pela identificação do idioma e definição de que termos deverão ser exibidos para mensagens de saída da aplicação em si. As classes relacionadas à camada *Operational* são as classes responsáveis pelo meio de ligação e validação, bem como preparação do ambiente e da requisição recebida pela interface, antes de passar, ou não, tal requisição já estruturada para a camada de *Programas*. Em ambas as camadas, *plugins* já são acionados para eventos cadastrados, interagindo com as classes da camada de *Plugins*.

Após tratada e definida como válida pela camada *Operational*, a requisição segue seu fluxo até a camada *Programas*. Esta camada identifica o programa e os parâmetros a serem usados, e é responsável pela execução propriamente dita, do programa, lembrando que neste ponto, a requisição tratada já indifere da origem, caso por meio de console ou por HTTP. Os programas podem ser variados, sendo desde o programa *clear* para limpar a tela do console ou *exit* para destruir a sessão do usuário, até programas muito mais complexos. Esta camada também faz integração com a camada de *Plugins*, disparando chamadas conforme eventos registrados pelos mesmos em forma de *listeners*.

que “escutam” a espera do disparo de um evento.

A camada denominada como *Cortex* é acessada por alguns dos programas, conforme a necessidade. Esta camada é responsável por tarefas mais complexas e delicadas, como identificação de padrões e probabilidades, aplicação de regras e levantamento de dúvidas, como dados relatados como desconhecidos ou pouco dominados pela aplicação. Esta camada faz conexão com a camada de *Plugins*, e também de *Programs*, podendo retornar imediatamente qualquer problema identificado. Esta camada fará uma análise lexica sobre os dados solicitados e validará as requisições, permissões e uma compreensão básica, além de ter classes responsáveis por regras específicas, como a *Inflex*, porém, para tarefas mais complexas, irá se conectar à camada *Analyst* ou *Engineer*.

A camada *Analyst* receberá as instruções a partir da camada *Cortex* e aplicará as regras e formulas necessárias para tentar abstrair o maior número de informações ricas possível. Estas informações serão retornadas à camada *Cortex* para que sejam transformadas em padrões que possam ser usado posteriormente. Estas classes serão as responsáveis por identificar verbos, substantivos, quantificadores e demais padrões nos contextos.

Já a camada *Engineer*, como o nome já diz, será a responsável pela geração de códigos e base de dados, bem como documentação automática. As classes e elementos nesta camada se alimentarão da camada *Cortex* a fim de buscar as informações já padronizadas e prontas para uso. Esta camada deverá interagir com uma última e tercerizada camada, a parte da aplicação, denominada “módulo” ou “model”, que poderá ser desenvolvida por terceiros e acoplada à aplicação a fim de usar as informações e estruturas de dados levantadas pelo Mind para escrever códigos, arquivos e documentação em linguagens de programação diferentes, para SGBDs diferentes ou mesmo com frameworks diferentes.

Para padronização, serão transportadas informações em formato JSON, quando a requisição vier por meio do protocolo HTTP, o que garante maior performance e também e oferece maior liberdade ao desenvolvedor do front-end para manusear tais informações a fim de exibi-las na tela de forma mais personalizada.

O recebimento de informações via HTTP se fará por meio do método POST, o que o difere do padrão mais rígido de RESTful, o qual exigiria o uso de métodos específicos para necessidades específicas, porém, não foge da semântica do sistema, visto que o conceito de chamada de programas de uma API pode ser visto como uma “postagem”. Estas informações enviadas por Post estarão 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 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. Este diagrama está recordado em 4 partes, dispostos neste documento de forma lateral para melhor visualização, estando disponível online no link:

<http://goo.gl/uZYtZ>.

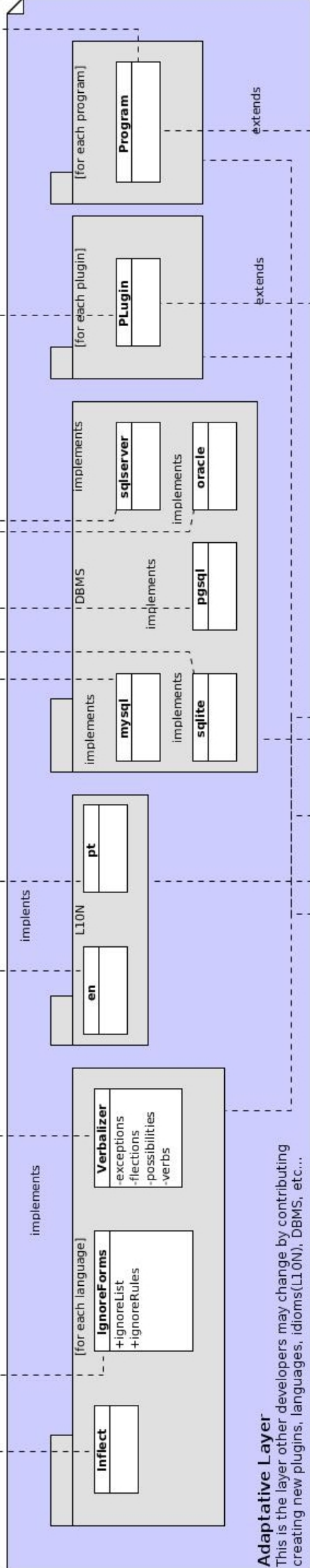
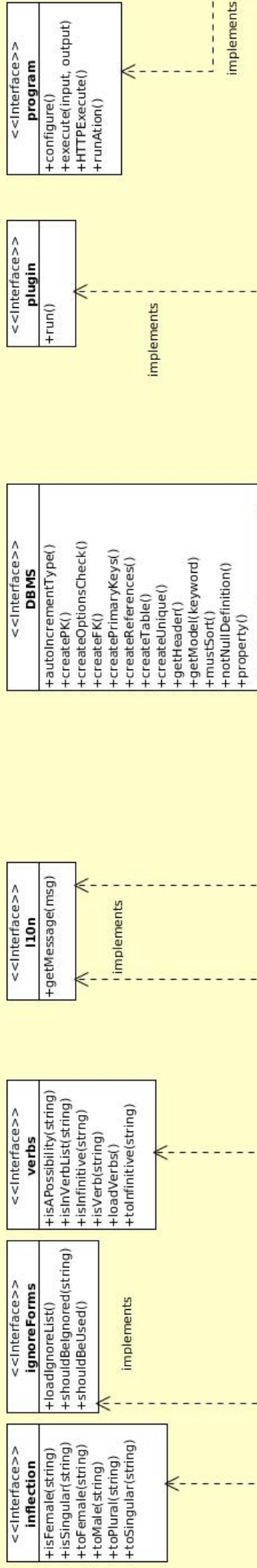
A Figura 4 mostra o diagrama de sequência do *lifetime* de uma requisição para análise de um código de alto nível. Os itens indicados no diagrama de sequência não são necessariamente classes, mas abstrações sobre as operações de classes ou packages de classes relacionados.

A Figura 5 define o mapa mental do projeto. Este mapa mental mostra as funcionalidades e módulos que compões o projeto em altíssimo nível. A Figura 6 apresenta uma forma mais detalhada do módulo Córtex e suas tarefas.

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.

Interfaces Layer

This is a layer, or group of objects in which all the interfaces will be found. These interfaces will be used by the *Adaptation Layer*.



Adaptive Layer

This is the layer other developers may change by contributing creating new plugins, languages, Idioms(L10N), DBMS, etc...

MindGeneric Classes

These classes are for generic use for all other classes' level

MindProperty
+comment +definition +key +refBy +refTo +default +name +options +required +size +type +unique +isKey(String expression) +isKnown(String type) +isProperty(String expression) +isRequired(String expression) +isUnique(String expression) +constructor(String expression) +parse() +setAsKey() +setAsWeakKey() +setDefault(String) +setName(String) +setOptions(Array) +setRefTo(MindEntity en, MindProperty pro) +isRequired(boolean) +setSize(float) +setType(String) +setUnique(boolean)

MindProject
+currentSource +sourceContent +analyze(boolean commit, boolean echo) +hasProject(String projectName) +import(String src) +loadIdiom(String idiom) +loadSource(String source) +loadSources() +openProject(AssocArray projectData) +setUp()

MindRelation
+focus +linkType +linkTypes +max +min +quantifiers +rel +verb +name +opposite +treated +uniqueRef +constructor(String name) +rename(String) +setEntities(MindEntity e, MindEntity e2) +setFocus(MindEntity) +setLinkType(String) +setMax(mixed) +setMin(mixed) +setRel(MindEntity) +setUsedVerb(String)

Mind
+about +conf +defaults +autoloadPaths +canonic +content +curlang +currentProject +l10n +langPath +lexer +modelsDir +originalContent +pluginList +projectsDir +ref +syntaxer +tokenizer +triggers +addPlugin(MindPlugin) +autoloadRegisterPath(String) +copyDir(String src, String dest, Boolean) +deleteDir(String dir) +hasProject(String projectName) +isInstalled() +message(String msg, String status, boolean) +openProject(AssocArray) +readPassword(String) +write(String msgKey, Boolean)

MindDir
+copyDir(String src, String dest, Boolean) +deleteDir(String path)

IndEntity
+refBy +refTo +name +linkTable +pkcs +properties +relations +relevance +selfRef +isEntity(String definition) +constructor(String name) +addAutoPK(boolean unique) +addProperty(MindProperty) +addRefBy(MindRelation) +addRefTo(MindEntity) +getRefBy() +getRefTo() +hasHardKey() +hasProperty(String propName) +removeProperty(String propName) +removeRefBy(String refName) +removeRefTo(String refName) +setSelfRefered(String)

MindPlugin
+description +event +links +name +trigger +version +attribute +addPlugin(MindPlugin) +setEvent(String) +setTrigger(String)

MindDB
+db +lastInsertedId +execute() +query()

MindSpeaker
+message(String msg, String status, bool) +write(String key, bool echo, args)

VersionManager
+cleanUp() +commit() +setUp() +getCurentVersion() +getElapsedTime() +init()

MindCommand
+restrict +fileName +construct(String name) +HTTPExecute() +operation() +execute(Input, Output) +getFileName() +readPassword() +runAction() +runPlugins(String event) +setFileName(String) +setRestrict(boolean) +verifyCredentials()

Project
+data +description +framework +originalCode +tag +versionId +addNewVersion() +getCurrentEntities(Int vs, String name, Int id) +getProperties(Array entity) +insertEntity(MindEntity Int vs, Int status) +insertProperty(MindProperty p, Int key) +markAsChanged(Array en) +markAsDropped(Array en)

ProjectFactory
+constructor(Array projData, Int version) +areDifferent(Array en1, MindEntity en2) +commit() +getCurrentVersion(Int version) +getEntity(mixed idOrName) +saveEntities(Array curEntityCollection)

Query
+fks +pks +closingQuery +query +createForeignKeys() +createPrimaryKeys(String qr, Array table) +createProperties(String qr, Array table) +parseDetails(Array prop, Array table) +setUp()

TableSort
+aux +ordered +original +unordered +addToOrderedList(Int qrKey) +isOrdered(Int qrKey) +sort(Array queries) +tableRefTo(Array table)

QueryFactory
+dbms +mustSort +queries +showHeader +addQuery(String command, Array tale, String template) +buildQuery(Array table, String qrCommand) +getQueryString(String command) +setUp(String dbDriver) +showQueries(boolean decorated, boolean raw) +sortTable()

DAO

DQB

uses

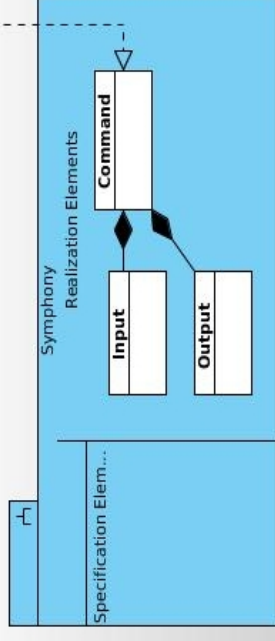
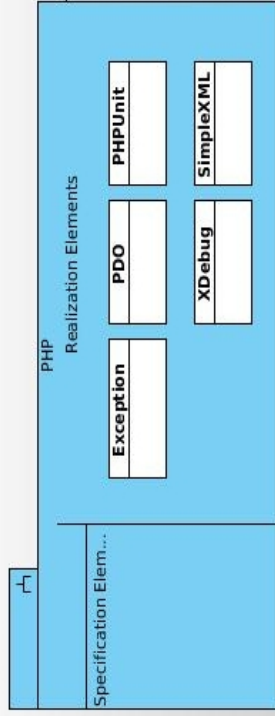
extends

extends

realizes

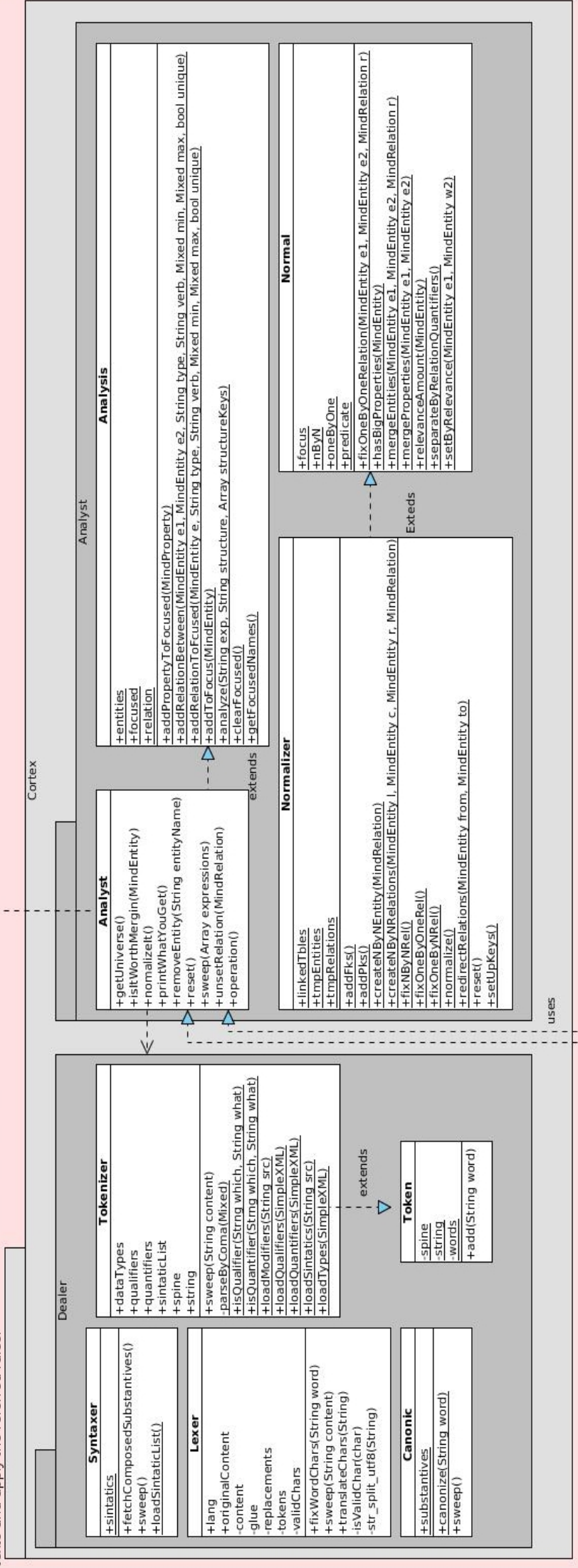
External Components

These classes come from external projects/packages not changed in this project

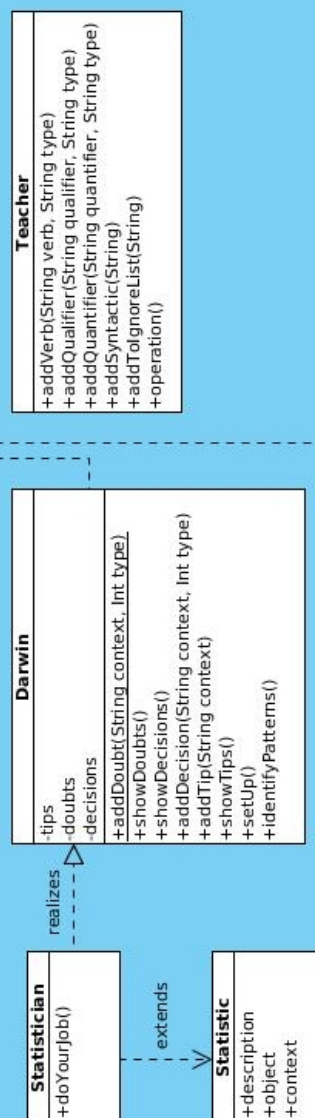


Cortex(the core of the system)

Here, are the main classes used to analyze and understand high level texts and apply the referred rules.

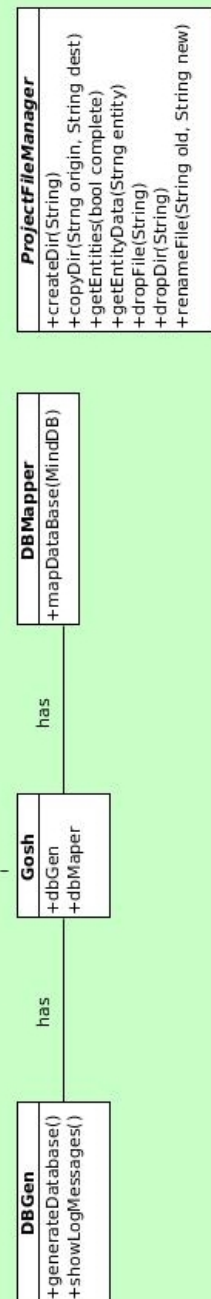


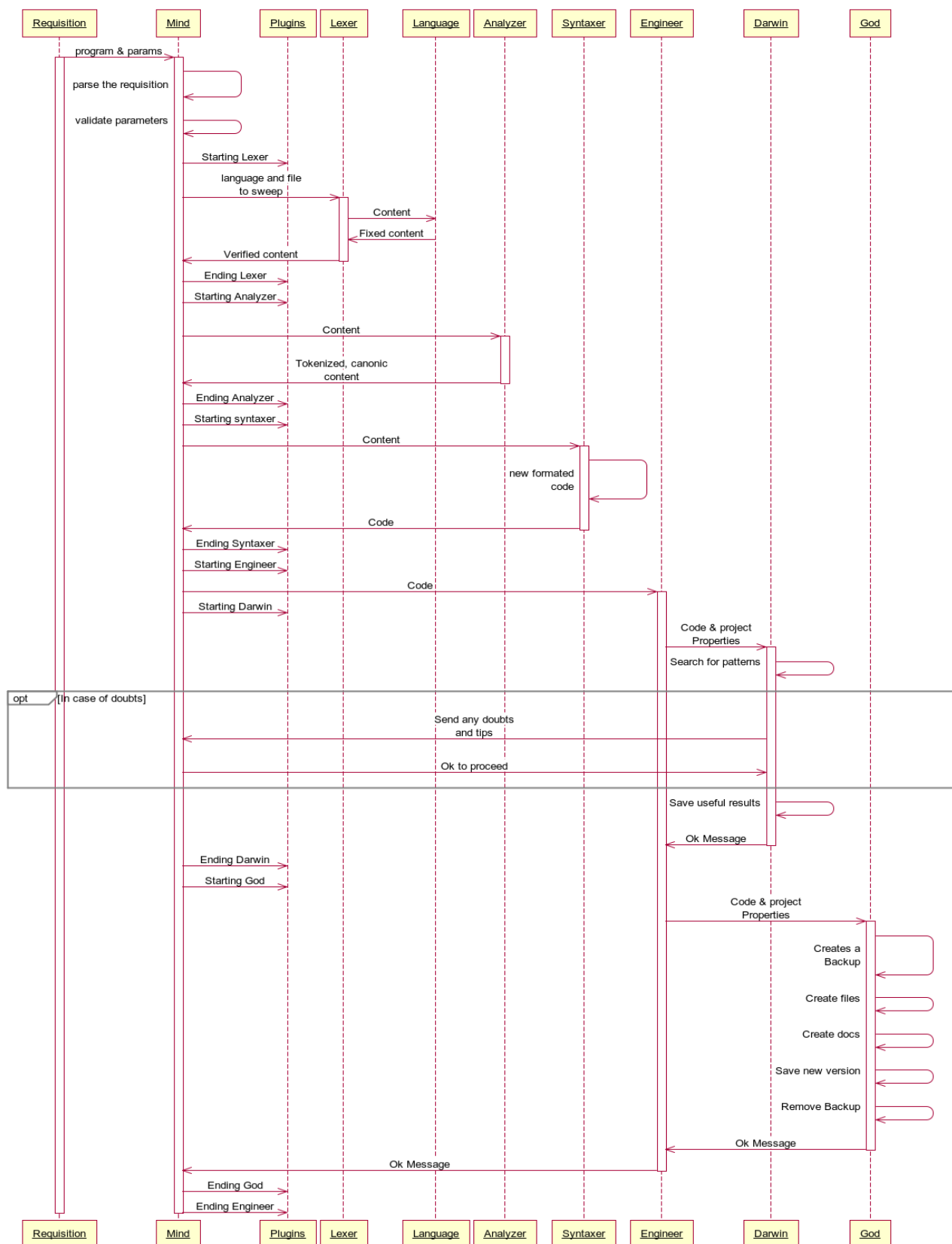
Scientia



Theos

In this layer are found the classes responsible for the "miracle" of the database or files generation from the analyzed source.





www.websequencediagrams.com

Figura 4. Diagrama de sequência de uma requisição para análise de um código de alto nível

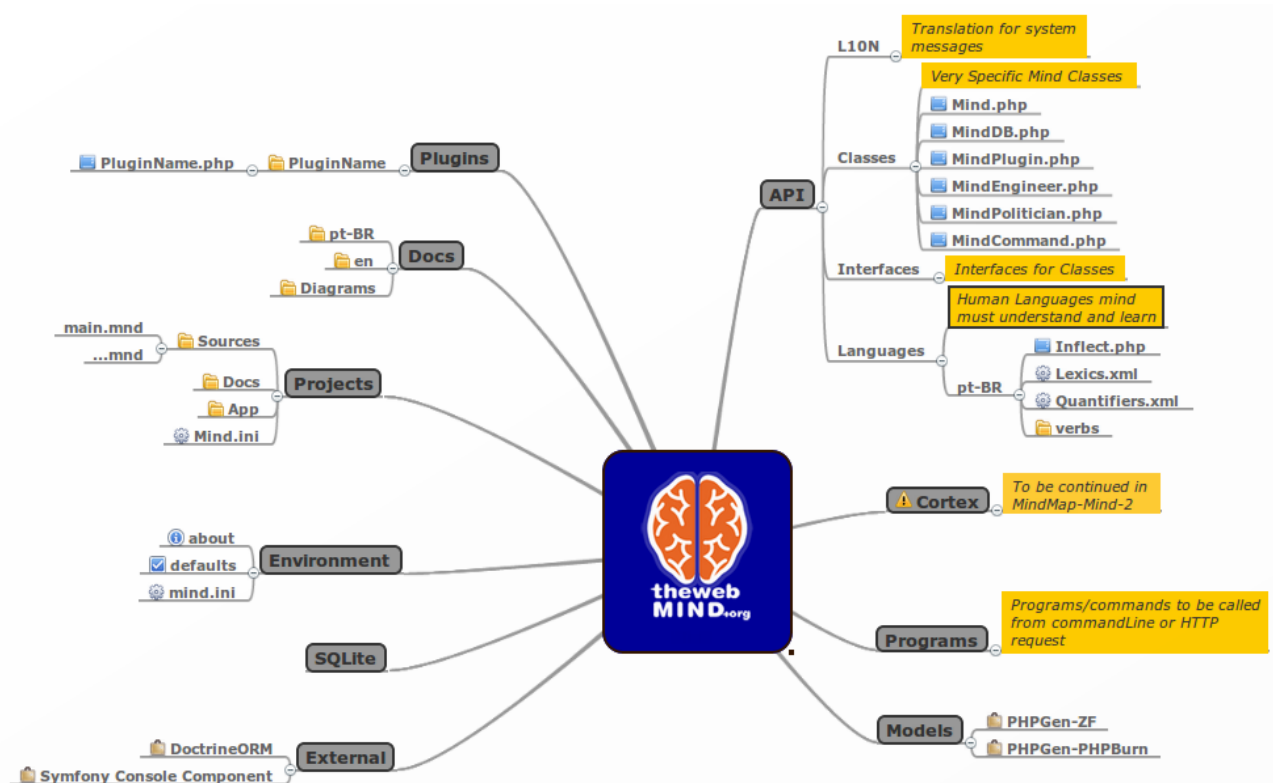


Figura 5. Funcionalidades e módulos que compõem o projeto em altíssimo nível

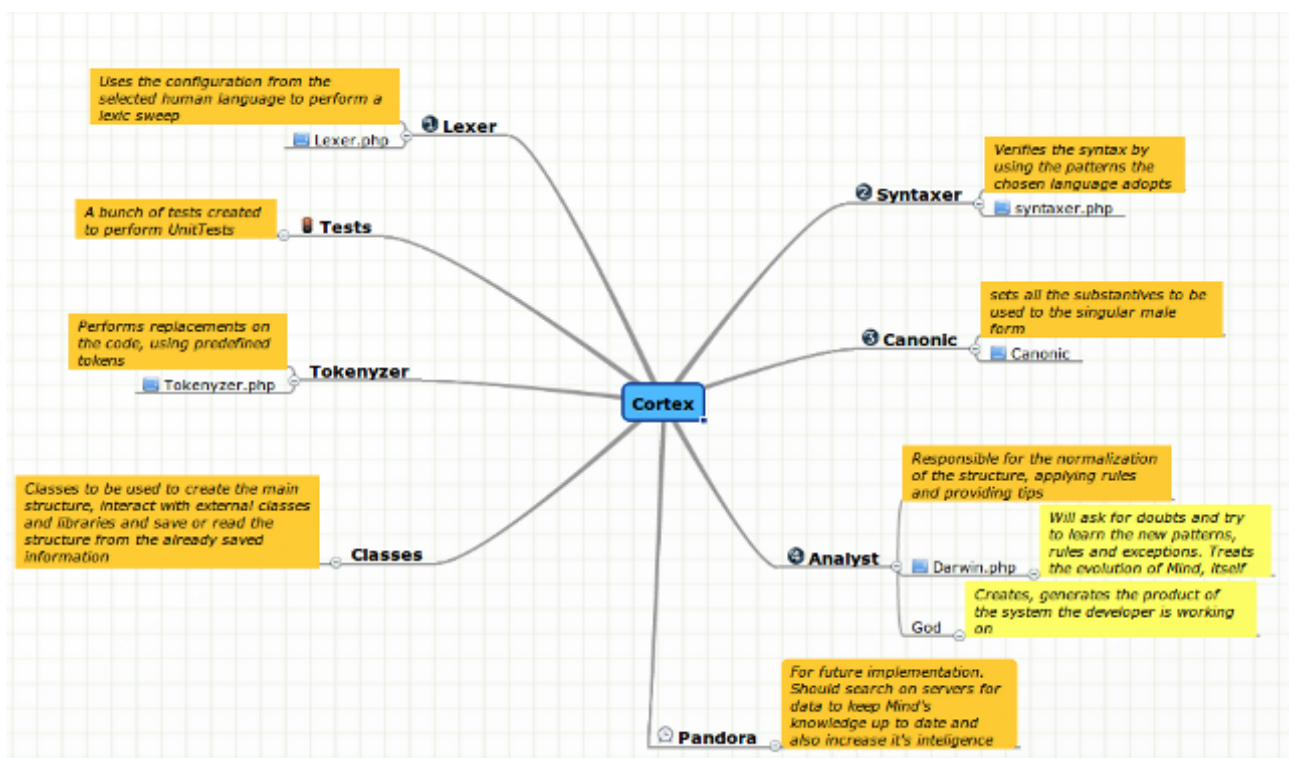


Figura 6. Uma forma mais detalhada do módulo CórteX e suas tarefas

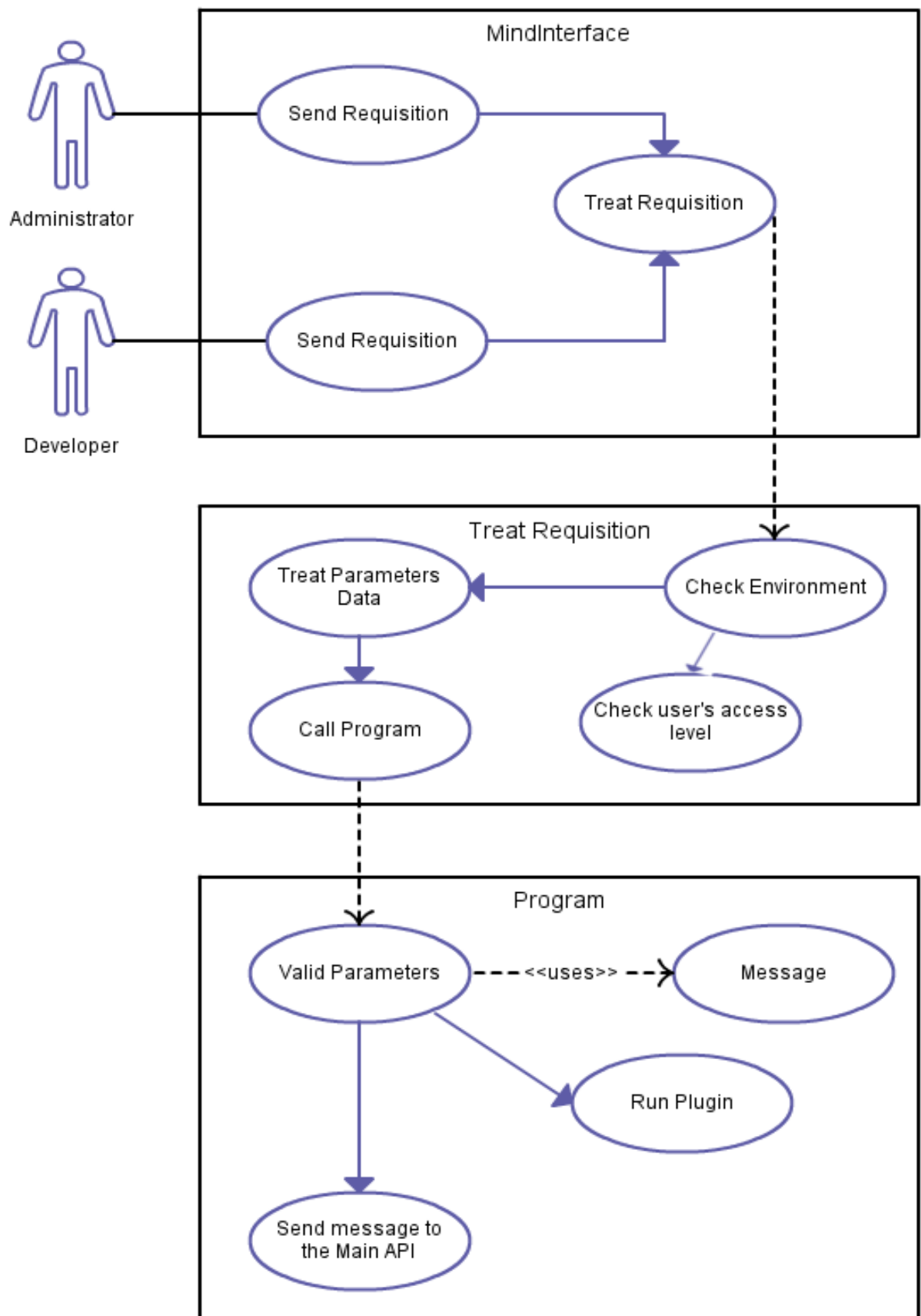


Figura 7. Fluxo de uma requisição

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. As ferramentas relacionadas a login e validação de cesso também estão operantes.

O analisador do texto (a ser digitado pelo usuário) já teve seus trabalhos iniciados e já conta com a classe *Infect* funcionando para os idiomas Português e Inglês, 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.

Existe também as classes responsáveis pela análise sintática, que verifica as expressões dentro de uma oração que podem ser tratadas como uma afirmação a ser analisada pelo sistema. Para isto, diversas formas de Tokens são utilizadas, juntamente com técnicas de *Button-up parsing*.

Há testes unitários implementados para as funcionalidades da classe *Infect*, responsável pela.

Uma interface web simples foi desenvolvida com métodos básicos para demonstração das funcionalidades e também para evidenciar como pode ser fácil a integração de uma ferramenta externa por meio do uso da API desenvolvida.

O usuário deve se autenticar no sistema, acessando tanto via linha de comando(console) quanto por alguma interface web(via HTTP). Assim, o usuário terá acesso aos projetos em que está envolvido, podendo alterar ou criar projetos novos. Para cada projeto, o usuário poderá especificar o idioma em que pretende programar, a linguagem de programação que pretende usar e o SGBD que será adotado para a aplicação final.

Após o usuário selecionar um projeto no qual trabalhará, ele poderá executar instruções como *analyze* para que o seu código seja analisado e compreendido pelo sistema, e *commit* para que o mesmo seja persistido na base de dados.

Programas como *show* e *dqb* exibirão informações a respeito da estrutura compreendida pelo sistema, como as entidades identificadas e os comandos DDL para interação com a base de dados selecionada.

Cada programa(ou comando disparado tanto pelo console quanto por alguma requisição HTTP) pode disparar a execução de plugins. Isto permite o desenvolvimento de plugins que são definidos por QUAL programa os disparará(o *trigger*, ou gatilho) e em que evento(no caso, *before* para antes, ou *after* para após a execução da instrução).

A seguir, este documento tratará de um detalhamento maior a respeito das regras aplicadas pelo sistema.

- **Do interpretador**

O interpretador do sistema é responsável por identificar qual o idioma adotado para o projeto atual, carregar seu grupo de regras específico e aplicar tais regras ao texto digitado. Os códigos digitados pelo usuário são encontrados no diretório *source* de cada projeto. O usuário pode ter mais de um arquivo neste diretório, incluindo novos diretórios, porém, o arquivo *main.mnd* é requerido.

Demais arquivos(também .mnd) poderão ser vinculados ao arquivo fonte principal por meio do seguinte comando:

@import <nome do arquivo>

É importante notar que, mesmo caso o comando import esteja dentro de um comentário, ele incluirá o arquivo sugerido.

O interpretador se encarregará de aplicar as regras do idioma selecionado, como levar um substantivo para sua forma canônica(singular, usualmente masculino), além de fazer o mesmo com verbos(por exemplo, o verbo “ensinam” deverá ser tratado como ensinar”).

Tal interpretador terá a habilidade de identificar e “tokenizar” o texto, a fim de torná-lo analisável pelas demais ferramentas do sistema.

Todavia, existe um problema quanto a tais padrões. Ainda não se faz viável o uso da semântica em tal interpretador, e por esta razão, tal ferramenta pode se confundir, como segue o exemplo: Descrevendo a seguinte oração:

rádio tem pilha.

O interpretador entenderá que pilha trata-se de um verbo(pilhar), resultando em uma sequência não válida de tokens e esta oração será ignorada. Para corrigir tal situação, ou caso o usuário precise ou deseje indicar uma entidade cujo nome seja um verbo, pode-se “escapar” tal nome por meio do caractere “\”, da seguinte maneira.

rádio tem \pilha.

É importante ressaltar que o interpretador NÃO tratará(aplicará a forma canônica) para verbos em primeira pessoa ou no passado, visto que o mesmo tem interesse em tratar da descrição de uma aplicação a ser desenvolvida.

O interpretador identificará a ausência do quantificador menor(a cardinalidade mínima de uma relação) como 0, caso esta seja omitida. No caso da omissão do quantificador maior(a cardinalidade máxima), o valor 'n' será adotado, indicando a ligação como múltipla para aquela direção.

O interpretador utiliza de artefatos específicos para as regras na linguagem discreta. Tais artefatos padrões tratam-se dos *qualificadores*, responsáveis por definições específicas sobre determinados itens, como por exemplo, definir uma propriedade como obrigatória ou uma ligação entre entidades como uma possibilidade. Além dos qualificadores, existem os *quantificadores*, que definem os valores para as cardinalidades.

Listagens também são utilizadas para que se possa definir um número finito de verbos, palavras chaves a serem ignoradas e continuadores(por exemplo, o “,” e o “e” podem ser considerados continuadores de uma oração.

- **Da análise**

A análise será feita sobre o texto já interpretado, com o auxílio dos *tokens*, identificados também pelo interpretador.

A análise do sistema se dá seguindo e aplicando determinadas regras definidas no estudo da análise de sistemas.

Perante tais regras, as atitudes, ou decisões tomadas pela aplicação seguirão o padrão apresentado a seguir:

Para ligações múltiplas entre entidade(*:n/*:n)

- Exemplo de uso:

professor tem muitos alunos e cada aluno tem vários professores.

- Possibilidades:

Caso 1: 0:n/0:n

Caso 2: 0:n/1:n ou 1:n/0:n

Caso 3: 1:n/1:n

- Premissas:

N/A

- Decisões:

Caso 1, 2 ou 3: Verifica se existe alguma entidade cujo nome seja a união entre as duas entidades em questão, se sim, usa ela como entidade de ligação, caso contrário cria uma nova entidade com tal nome. O formato de tal nome é definida por:

Nome da entidade cujo nome for mais curto, o caractere “_”(que pode ser alterado no arquivo de configurações) e o nome da outra entidade.

A tabela definida como entidade de ligação passará a referenciar ambas as entidades anteriores. Após isto, a relação entre as entidades anteriores é removida, mantendo a ligação apenas entre as entidades por meio da entidade de ligação.

Para relacionamentos simples(*:1/*:n)

- Exemplo de uso:

aluno tem um professor.

- Possibilidades:

Caso 1: 0:1/0:n ou 0:n/0:1

Caso 2: 1:1/1:n ou 1:n/1:1

- Premissas:

N/A

- Decisões:

Caso 1 ou 2: É adicionada uma *foreign key*(chave estrangeira) na tabela de menor cardinalidade, referenciando-se à tabela de maior cardinalidade.

Para ligações unárias(*:1/*:1)

- Exemplo de uso:

Toda igreja tem um padre, e cada padre tem uma única igreja.

- Possibilidades:

Caso 1: 1:1/1:1

Caso 2: 0:1/0:1

Caso 3: 0:1/1:1 ou 1:1/0:1

- Premissas:

-A quantidade de campos que definem uma tabela como extensa,

bem como a quantidade de campos de tamanho grande ou quantidade de campos *não nulos* estão definidos no arquivo de configuração e podem ser editados;

-A pontuações de uma tabela equivalem à verificação quanto a:

:Quantidade de campos na tabela;

:Quantidade de campos de tamanho grande, na tabela;

:Quantidade de campos *não nulos* na tabela;

-A entidade mais relevante é identificada por:

:ser referenciada por mais entidades que a outra entidade;

:ter pontuação mínima de 3 pontos;

▪ **Decisões:**

Caso 1: *Mesclar entidades(merge)*

Move todas as propriedades da entidade mais fraca(menos relevante) para a mais forte;

altera todas as relações entre a tabela mais fraca e demais tabelas, para que apontem para a mais forte;

Caso 2: *mesclar ou corrigir as entidades*

Caso a entidade menos relevante tenha menos de 3 pontos, mescla as entidades.

No caso de ambas as entidades tenham pontuação maior de 3, exclui a relação entre a menos relevante e a mais relevante, e então adiciona a chave da mais relevante na entidade menos relevante marcando-a como uma foreign key referenciando a mais relevante, e também como primary key para que juntas, sejam únicas adotando as chaves de ambas como uma primary key composta;

Caso 3: *Corrigir as entidades*

Exclui a relação entre a menos relevante e a mais relevante, e então adiciona a chave da mais relevante na entidade menos relevante marcando-a como uma foreign key referenciando a mais relevante, e também como primary key para que juntas, sejam únicas adotando as chaves de ambas como uma primary key composta;

• **Da comunicação/interação**

As classes relacionadas à L10N são responsáveis pela nacionalização das mensagens. É uma camada que abstrai o idioma usado, permitindo que as demais classes manifestem as mensagens por meio de um identificador. Através deste identificador, as classes de nacionalização são capazes de encontrar no idioma utilizado pelo usuário(não necessariamente o mesmo utilizado no projeto atual) qual mensagem exibir para a interface console, ou a enviar como resposta a uma requisição HTTP.

Tais mensagens estão classificadas pelos status de Falha por meio do identificador *[Fail]* ou por sucesso, com o identificador *[Ok]*. Outras mensagens podem ser enviadas para a saída sem o uso de um selo de status. É possível o uso de *type specifier*(especificações de tipo) na mensagem, permitindo assim que se envie uma variável à classe L10N, a fim de ter uma mensagem personalizada, com o termo passado concatenado na mensagem(exemplos de *type specifiers*: %s, %d, %f...).

- **Das dúvidas, sugestões e decisões**

O sistema está preparado para interagir com a inteligência gerada a partir da análise do texto descrito pelo usuário de forma a identificar determinados padrões e definir se tomará uma decisão sobre cada expressão. Todavia, há situações em que o sistema pode deparar-se com uma dúvida.

As dúvidas são definidas por situações desconhecidas, como um tipo de propriedade não identificado pelo interpretador ou expressões sem nenhuma forma de sintaxe válida. Dúvidas também são adicionadas no caso de *conflito de afirmações*, onde o usuário descreve uma situação de uma forma, e em seguida descreve a mesma situação de outra forma.

As dúvidas são enviadas às classes Darwin, a fim de serem tratadas ou, conforme definidas, exibidas ao usuário oferecendo uma forma de tratamento.

As dúvidas permitirão uma interação com o usuário a fim de evoluir a base de conhecimento do sistema, em relação a um determinado idioma. Com elas, o sistema pode evoluir aprendendo novos verbos, qualificadores, quantificadores ou tipos de propriedades.

As sugestões são provenientes de padrões identificados pelo sistema a fim de otimizar a forma como o texto descrito pelo usuário é interpretado. Tais sugestões podem vir de acordo com as seguintes situações:

- Confirmação de idioma: Caso nenhuma sintaxe tenha sido identificada em toda a descrição do usuário, pode ser pelo fato do idioma selecionado estar errado.
- Redundância na declaração: Caso sejam identificadas declarações de repetidas para entidades e relações, ou atributos de entidades.
- Uso inadequado de opções de valores: Caso o usuário defina muitas opções válidas para determinado atributo, a dica será transformar tal atributo em uma nova entidade, na forma de uma *lookup table*, uma tabela mais focada em pesquisa. O mesmo para casos onde campos em tabelas diferentes tenham as mesmas opções.
- Otimização por generalização/especialização: Uma dica será exibida indicando uma especialização para tabelas cujas (todas) propriedades sejam iguais e repetidas.
- Identificação de termos possivelmente enganados: O usuário pode ter escrito um termo que por engano pode ter ficado com uma letra errada, ou faltando. O sistema deverá ter a habilidade de perceber que a entidade outrora chamada por "*usuário*" agora pode ter sido confundida, ao ser descrita como "*usário*", mas não necessariamente ser tal entidade. Por este motivo, esta situação enquadra-se como uma dica vinda do sistema.

- **Da sintaxe** (sintaxe discreta)

A sintaxe discreta trata-se de uma determinada sintaxe base, adotada por uma linguagem que siga o formato/conceito de *linguagem de programação discreta*. As linguagens de programação discreta procuram por um pequeno grupo de regras básicas e a partir delas identificam padrões por meio de outros grupos variáveis de regras. Para a *MND*, a linguagem de programação discreta desenvolvida para este projeto(a ser mais detalhadamente descrita em outro artigo), o grupo de regras base

são os que seguem:

- Definição de elementos no universo(entidades): Para a definição de entidades ou elementos no universo descrito pelo usuário, a regra que se sucede busca por uma sintaxe aprovada(descrita por um modulo de idioma externo) aplicando regras para a canonização dos elementos(também regidas por um grupo de regras externo definido pelo idioma selecionado).

É feita uma busca, então, na lista atual de elementos e caso o identificador não tenha sido encontrado, ele é adicionado a lista de entidades. Um exemplo de descrição seria este:

professor tem aluno.

Onde professor e aluno seriam identificados como entidades no universo definido pelo usuário.

- Definição de ligações entre entidades: As ligações entre as entidades contidas no universo de uma aplicação podem ser definidas por palavras chaves, normalmente na forma de um verbo (previamente trazido para sua forma canônica) podendo contar com demais identificadores a fim de indicar sua cardinalidade, também definidos em um grupo externo, carregado, de regras de acordo com o idioma. Os exemplos a seguir demonstram três formas diferentes para uma mesma definição de ligação entre entidades:

professor tem aluno.

Professor tem muitos alunos.

Professor pode ter nenhum ou vários alunos.

Lembrando que para os casos onde a cardinalidade mínima é omitida, zero é assumido, enquanto que para casos onde a cardinalidade máxima é omitida, 'n' é assumido, como citado anteriormente.

- Definição de modificadores de ligação: Correspondem a forma como se descreve o elo entre duas entidades a fim de acusar uma cardinalidade mínima. Ligações indicadas como *must*(obrigação) acarretarão em uma cardinalidade mínima 1, enquanto ligações fracas, ou definidas como *possibilities*(possibilidades) resultarão em uma cardinalidade mínima 0, como são demonstrados nos exemplos a seguir:

Professor deve ter aluno.

Professor pode ter aluno.

Para o primeiro exemplo, a cardinalidade mínima será de 1 pois cada professor precisa obrigatoriamente ter algum aluno, enquanto que no segundo exemplo, isto não é uma obrigação(sendo este o padrão, na omissão de um modificador).

Novamente, as palavras chave *deve* e *pode* são provenientes de um grupo de regras e definições carregadas a partir de um idioma.

- Definição de propriedades e tipos: Propriedades fazem parte das entidades e podem ser definidas por meio do caractere ":" logo após o nome da propriedade, seguido obrigatoriamente pelo tipo que tal propriedade corresponde sendo o tipo, uma variável oriunda do grupo de regras do idioma.

Professor tem nome:string.

Professor tem idade:inteiro.

O detalhamento de tal propriedade pode ser expresso entre parênteses logo após a definição do tipo.

Os possíveis valores para detalhamento podem ser:

- default: todo valor entre aspas contido entre parênteses. Para definição de valores default em forma literal(valores numéricos e chamadas para funções) inicia-se o valor com o caractere "=";
- tamanho: qualquer valor numérico entre os parênteses;
- requerido: define se aceitará valor nulo ou não;
- unico: se aquela propriedade deve ser única na tabela;
- opções: definido entre chaves, definido pelo valor seguido pelo caractere "=", seguido pela descrição. Utilizando o caractere "|" como separador de opções. Ex.: {F=Feminino|M=Masculino}

Exemplos de definição de propriedade detalhada:

aluno tem sexo:char(1, obrigatório, {F=Feminino|M=Masculino}).

aluno tem cpf:int(único, obrigatório).

aluno tem status:varchar(16, "Ok").

aluno tem idade:inteiro("=0").

aluno tem data de cadastro:data("=now()").

Note que a definição dos detalhes de uma propriedade não exige uma ordem específica.

- Definição composta: A definição, tanto para propriedades pode ser feita de forma composta por meio de identificadores indicados pelo idioma selecionado. No caso do português, um identificador amplamente adotado seria o "de". Para o caso de uma composição que não contará com uma preposição auxiliar, indica-se o uso de um hífen ou do caractere "_".

Ex.: Pessoa tem carta de crédito.

Pessoa tem documento-pendente.

Para os exemplos, os espaços e o hífen serão substituídos pelo caractere "_" respectivamente, sendo a preposição removida.

Note que caracteres especiais também são trazidos para uma forma canônica e que para o segundo exemplo, o caractere "é" se tornaria "e", resultando em "carta_credito".

- Definição múltipla: Tanto para definição de propriedades quanto para ligações entre entidades, é possível a identificação múltipla. Para isto, o idioma selecionado provisionará algumas expressões de continuidade(como o vírgula, "quanto" ou "e").

Tanto professor quanto aluno tem nome:string e idade:int.

Atribuindo então as propriedades nome e idade para ambas as entidades definidas na expressão.

Todavia, note que uma expressão deve admitir apenas a definição de propriedades OU de ligações, tornando a seguinte oração inapropriada:

Professores e alunos tem nome:string e chefe.

- **Das tecnologias**

Para o desenvolvimento deste sistema foram adotadas técnicas e tecnologias expressivas para sistemas interpretadores e compiladores, bem como geradores.

Expressões regulares são a base para identificação de termos e sintaxes. Porém foi necessária a implementação de *pseudo-código* e *programação dinâmica* para criação em tempo de execução das expressões regulares de acordo com o que é definido pelo idioma selecionado. Isto torna

dinâmicas, as regras de interpretação adotadas pelo sistema permitindo uma evolução das regras por meio de uma aprendizagem por parte do sistema, bem como uma maior customização do usuário.

Por meio de *tokens*, o sistema se torna capaz de identificar as sintaxes válidas identificadas pelo interpretador e desta forma aplicar as regras definidas no idioma selecionado, lembrando que a forma como os tokens são aplicados e identificados também derivam de regras definidas no idioma.

Por usar técnicas de *bottom-up parsing*, tanto as expressões quanto as palavras identificadas pelos tokens são canonizadas e permitem um maior controle por parte do sistema.

Um sistema de controle por *templates* e *models* é adotado para situações onde se faz necessária a criação, cópia ou geração de dados.

As classes originadas do projeto conhecido como *Symphony* são utilizadas para o tratamento de comandos de entrada, e formatação de saídas para o console, além do tratamento de parâmetros passados por meio do mesmo. Em uma camada acima desta, a classe *MindCommand* se encarrega de abstrair este controle e também de oferecer suporte às requisições HTTP alocando as variáveis de parâmetros em uma variável de ambiente, adotada por ambas as interfaces ligadas à API.

O uso da ferramenta PHPUnit possibilitou a execução em lote de *testes unitários*, automatizando os testes. Os testes unitários aumentam a confiança sobre o software na hora de verificar sua qualidade, visto que apuram seus resultados esperados e os comparam com os obtidos em uma quantidade grande de testes para situações variadas.

O PDO é a camada de abstração do banco de dados adotada para o desenvolvimento deste sistema, possibilitando uma interação com as bases de dados variadas. Todavia, foi preciso a criação de classes para cada SGBD específico visando construir os comandos *DDL* para cada sistema de banco de dados.

A biblioteca JQuery foi utilizada para a criação da ferramenta web usada para demonstração do uso da API por meio de requisições HTTP, a fim de auxiliar a criação de objetos *Ajax*.

A linguagem de programação PHP foi utilizada justamente por oferecer todas as ferramentas necessárias para o desenvolvimento das ferramentas do sistema, tornando possível o desenvolvimento para interação tanto via console quanto via HTTP por meio do uso de um servidor web e do *PHPcli*, extensão que oferece suporte a execução da linguagem via linha de comando. A versão mínima da linguagem é a 5.3. É importante ressaltar isto devido às novas ferramentas oferecidas por esta versão da linguagem como uso de *namespaces* e *passagem de objetos por referência*.

Como base de dados para a aplicação, foi escolhido o SQLite devido a sua leveza e portabilidade, estando disponível nos mais variados sistemas operacionais e ter um bom suporte via PHP.

- **Da API**

A API da aplicação oferece suporte à comunicação por meio do uso da execução do programa por linha de comando em console, e também por requisições HTTP. Para fazer tal tratamento, é preciso passar qual o sub-programa(ou programa interno) a ser executado e seus respectivos

parâmetros. Utilizando o console, após acessar o sistema por meio da execução do comando “mind”, o console se abre para execução dos programas internos. A passagem de parâmetros para cada subprograma é feita seguindo o padrão de passagem de parâmetros do console do sistema operacional.

Existem alguns parâmetros que são adicionados para cada programa só sistema, por padrão, como o parâmetro *-help*.

Exemplo da chamada de um programa interno do sistema:

```
mind> use project demo
```

Sendo *use* o programa interno executado e *project* e *demo* os parâmetros passados.

Via requisições HTTP, a solicitação do programa a ser executado será feita por meio da passagem do parâmetro “program” pelo método POST. Os parâmetros devem então ser passados de acordo com seus respectivos nomes. Abaixo, segue o exemplo da mesma requisição demonstrada acima, utilizando uma requisição HTTP utilizando Ajax por com o uso da biblioteca JQuery.

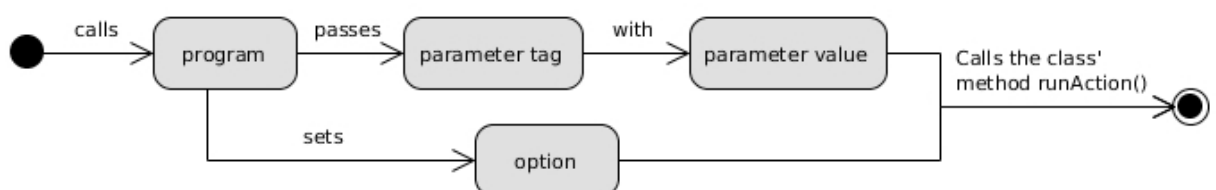
```
$.ajax({  
    type:'POST',  
    url:'server/mind',  
    data:{ program:'use',  
           what:'project',  
           name:'demo'},  
    success: function(ret){  
        doSomething(ret);  
    }  
});
```

Neste exemplo, a requisição ajax buscará a url definida para o seu servidor onde o sistema *mind* está configurado enviando pelo método POST os parâmetros *program* com o nome do programa a ser executado e os parâmetros *what* e *name*. Na função definida para o evento *success* receberá por parâmetro o retorno emitido pelo servidor.

Para o caso de requisições HTTP, caso o retorno seja uma lista de itens, este, será transformado no padrão JSON a fim de oferecer um maior controle para a aplicação em *client-side* e também para manter um padrão estável.

O nome para cada parâmetro pode ser encontrado por meio do uso do parâmetro *--help* para cada programa, visto que novos programas serão adicionados com a evolução do sistema ou personalização por parte do usuário.

Para ambos os casos, parâmetros opcionais podem ser simplesmente omitidos. Parâmetros podem ser classificados como parâmetros compostos ou opções. Os parâmetros compostos possuem uma tag identificadora (como *project*) e um valor para o mesmo, enquanto *parameter options* são definidos por apenas uma instrução (como *-help* ou *-d*).



Validação:

A forma de validação deverá ser feita de duas maneiras.

Na primeira, haverá uma sequência de testes unitários automatizados a serem executados 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*.

Estes usuários receberão acesso ao código fonte do projeto, além da aplicação em si, e terão 10 dias para testá-lo e entregar uma avaliação a ser tomada por meio do formulário supracitado.

O questionário terá como metodologia uma avaliação mais qualitativa que quantitativa, sendo composto por poucas questões 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 8.1, atinge 100% de acertos (em 49 testes executados), executada sobre a classe *Infect* para flexão de substantivos em português, alterando ou detectando seu gênero e número.

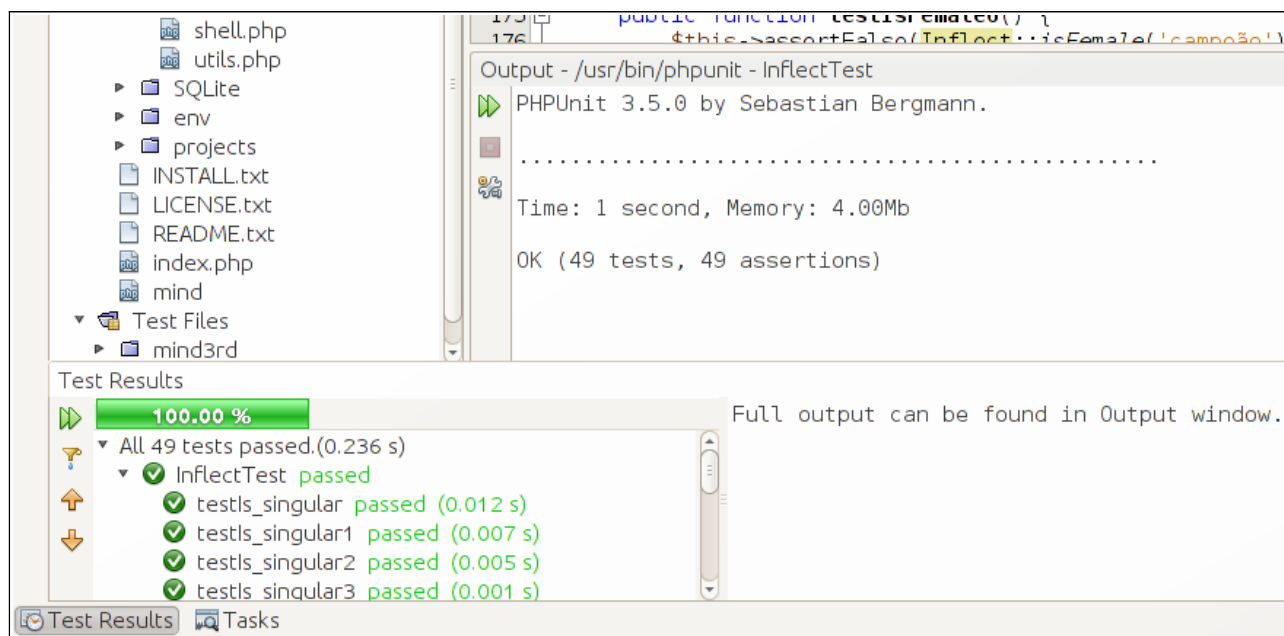
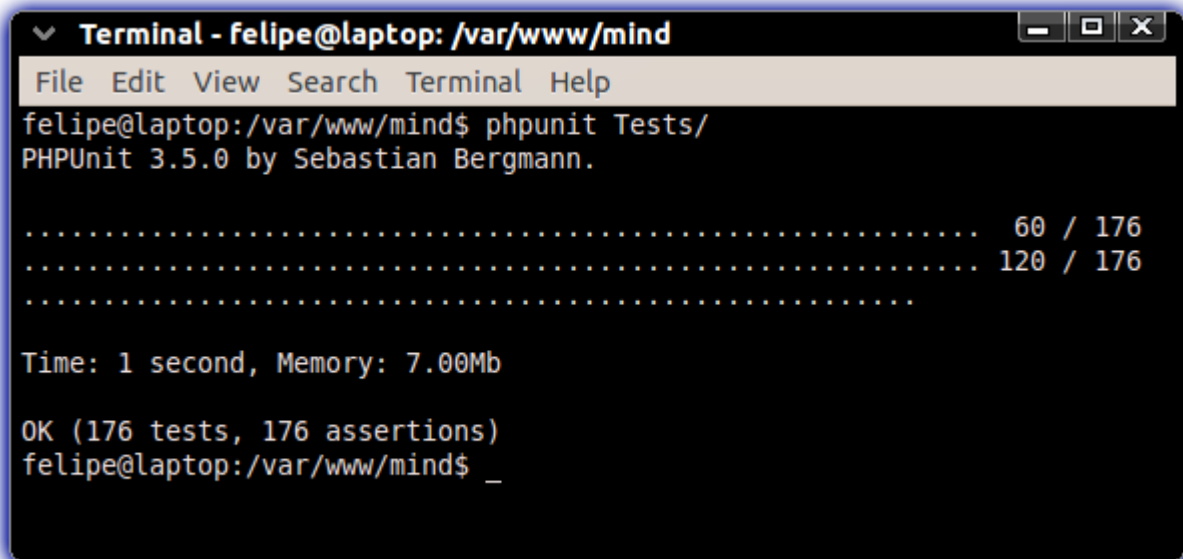


Figura 8.1. Relatório dos testes executados a partir de uma IDE.

Em uma segunda bateria de testes, foram executados 176 testes unitários para asserções em chamadas de métodos diversos a fim de manter certificar que nada havia sido quebrado com a adição das novas funcionalidades, e também que tais funcionalidades estavam retornando exatamente o que se esperava, como é demonstrado na Figura 8.2.

A terminal window titled "Terminal - felipe@laptop: /var/www/mind" with a menu bar (File, Edit, View, Search, Terminal, Help). The command "phpunit Tests/" has been executed. The output shows "PHPUnit 3.5.0 by Sebastian Bergmann." followed by a progress bar with dots and the numbers "60 / 176" and "120 / 176". Below this, it says "Time: 1 second, Memory: 7.00Mb" and "OK (176 tests, 176 assertions)". The prompt "felipe@laptop: /var/www/mind \$" is followed by an underscore character.

```
Terminal - felipe@laptop: /var/www/mind
File Edit View Search Terminal Help
felipe@laptop:/var/www/mind$ phpunit Tests/
PHPUnit 3.5.0 by Sebastian Bergmann.

..... 60 / 176
..... 120 / 176
.....

Time: 1 second, Memory: 7.00Mb

OK (176 tests, 176 assertions)
felipe@laptop:/var/www/mind$ _
```

Figura 8.2. 176 testes unitários executados com sucesso pela ferramenta PHPUnit via console

Para a validação qualitativa, as seguintes questões vem sendo preparadas:

- 1) Por quanto tempo, em média, testou o sistema?
() menos de 3 horas () entre 3 e 6 horas () mais de 6 horas
- 2) Defina o grau de dificuldade ao tentar instalar a aplicação:
- 3) Defina sua impressão quanto à nova abordagem, por meio do uso de uma API consumida tanto por meio de comandos via console quanto via requisições HTTP:
- 4) Você identificou algum erro na interpretação do sistema sobre o seu código digitado? Alguma decisão tomada erroneamente?
- 5) As mensagens de ajuda ou informação foram úteis e bem definidas, estando de acordo com cada situação?
- 6) Descreva sua visão a respeito da aplicação quanto à usabilidade, performance, confiabilidade e demais itens que gostarias de citar.
- 7) Para o aumento da confiabilidade deste questionário, deixe uma breve descrição de seu trabalho, experiência e interesse, podendo identificar-se, caso desejado.

Tal questionário será distribuído entre as pessoas selecionadas para a fase de testes, que deverão ser acadêmicos, professores, desenvolvedores e analistas Sênior e membros reconhecidos e renomados na comunidade open source ou no meio onde atuam.

Cronograma:

O cronograma deste TCC foi todo baseado em *Sprints* de 15 dias, com datas programadas, 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 ser 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	30/09/10
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 português	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	Ok	8	16/11/10
17	Programas create e show	Feature	Ok	6	16/11/10
18	Relatório de projeto parcial	Documento	Ok	6	16/11/10

ID	Produto	Tipo	Status	Pts.	Entrega
Sprint 6		17/11/10 a 01/12/10			
19	Identificador de verbos	Feature	Ok	6	26/11/10
20	Evolução da classe Inflect no idioma Português(Pt).	Feature	Ok	5	30/11/10
Sprint 7		18/12/10 a 01/01/11			
21	Tokenizer	Feature	Ok	6	24/11/10
22	Padronização da estrutura do conhecimento	Pesquisa	Ok	8	01/01/11
Sprint 8		02/01/11 a 16/01/11			
23	Controle e estrutura para plugins	Feature	Ok	4	08/01/11
24	Padronização da estrutura dos módulos do Cortex	Pesquisa	Ok	8	16/01/11
Sprint 9		17/01/11 a 31/01/11			
25	Programas para gerência de projetos	Feature	Ok	4	20/01/11
26	Programas para gerência de usuários	Feature	Ok	4	24/01/11
27	Evolução da classe Inflect para o idioma inglês(En).	Feature	Ok	4	31/01/11
Sprint 10		01/02/11 a 15/02/11			
28	Classes responsáveis pela comunicação com usuário em diferentes idiomas	Feature	Ok	8	12/02/11
29	Classes para armazenamento em arquivos	Feature	Ok	4	15/02/11
Sprint 11		16/02/11 a 02/03/11			
30	Classe para classificação de dados	Feature	Ok	4	19/02/11
31	Classes para tratamento de erros	Feature	Ok	4	26/02/11
32	Classes para interação com o usuário	Documento	Ok	4	02/03/11
Sprint 12		03/03/11 a 18/03/11			
33	Classes para geração de comandos DDL para diferentesSGBDs(DQB)	Feature	Ok	8	11/03/11
34	Classe de geração de DDL usando a camada DQB para pgSQL e Mysql	Feature	Ok	8	18/03/11
Sprint 13		19/03/11 a 02/04/11			
35	Evolução da ferramenta de geração de DDL	Feature	Ok	8	22/04/11
36	Preparo da documentação para entrega do relatório	Feature	Pen	8	02/04/11
Sprint 14		03/04/11 a 18/04/11			
37	Módulo para geração de arquivos e diretórios no projeto pela API	Feature	Pen	8	14/04/11
Sprint 15		19/04/11 a 03/05/2011			
38	Ferramentas para geração da base de dados	Feature	Pen	4	25/04/11
39	Documentação para o desenvolvedor	Documento	Pen	8	03/05/11

ID	Produto	Tipo	Status	Pts.	Entrega
Sprint 16		04/05/2011 a 19/05/2011			
40	Site para divulgação	Documento	Pen	8	16/05/11
41	Questionário para validação	Feature	Pen	8	16/05/11
42	Refinamento da documentação a ser entregue como relatório para o TCC, e slides para apresentação à banca	documento	Pen	6	19/05/11
Sprint 17		20/05/2011 a 31/05/2011			
43	Análise dos resultados do questionário	Documento	Pen	4	26/05/11
44	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:

- Conceitos da versão anterior do próprio sistema;
- PDO, *drive* para abstração da camada de banco de dados no PHP;
- SQLite e PHP-SQLite;
- Symfony Console, grupo de classes dedicadas ao manuseio do console;
- PHPUnit, ferramenta para automação de testes uniários;
- Classe Inflection para flexão básica de substantivos(apenas para o idioma inglês) encontrada em <http://php.net>

Ferramentas externas para análise e desenvolvimento

Sequence diagram

<http://www.websequencediagrams.com/>

ER Diagram

<http://thewebmind.org/> (second version)

MindMap

<http://www.mindmeister.com/>

Class Diagram

<http://creatly.com>

<http://www.visual-paradigm.com/product/vpuml/> (Community version)

General charts

<http://www.lucidchart.com/>