Uma Implementação de Arquitetura para Sistemas Web

Aluno: Rodrigo Buás Lopes

Orientador: Julio Cesar Sampaio do Prado Leite

Introdução

Há aproximadamente três anos estou envolvido com o desenvolvimento de softwares de pequeno e médio porte, mais especificamente aplicações web. Desde então, estive empenhado em estudos sobre arquiteturas [1] [2] [3], frameworks, padrões de design [4], e padrões para web [5] [6] [7] com o objetivo de descrever uma arquitetura de software voltada para web.

Nestas experiências, pude perceber que muitos desenvolvedores, mais especificamente os desenvolvedores web, em geral, não utilizavam um framework, e muitas vezes não conseguiam seguir nenhum padrão estabelecido, ou algo do tipo. E, no decorrer do desenvolvimento, acabavam deixando escapar conceitos importantes. Um exemplo recorrente é a excessiva utilização de técnicas condenadas e obsoletas, como para resolver o problema de incompatibilidade entre browsers, utilizar bifurcação de código. Outro problema muito encontrado é o desenvolvimento onde o modelo e a interface gráfica estão fortemente acoplados, o que certamente irá dificultar na manutenção e atualização posteriores.

O trabalho aqui proposto pretende criar representações arquiteturais que sejam úteis sob a ótica de arquitetura de software e ao mesmo tempo não conflitem com as características básicas de software livre, onde o documento central de desenho é o próprio código.

A principal motivação é descrever uma arquitetura, onde a parte responsável pelo processamento central esteja o máximo possível desacoplada da parte de que se refere à interface gráfica com o usuário. Para que seja possível numa mesma aplicação, utilizar diferentes tecnologias para o front-end. Consequentemente terá que ser criado entre elas uma porta de comunicação para diversas linguagens.

Na primeira parte são apresentados conceitos fundamentais e o paradigma Web. Em seqüência será apresentado o estudo realizado sobre a teoria envolvida, as tecnologias, técnicas e linguagens testadas e as ferramentas utilizadas.

Na terceira parte, baseados nos conceitos estudados sobre padrões, macro e micro-arquitetura, e arquitetura para web, é mostrado, usando conceitos de engenharia reversa [8] e re-engenharia [9], como foi o processo de levantamento e especificação de requisitos funcionais e não-funcionais, e como foi criada a nova proposta de desenho de arquitetura.

Em seqüência, será mostrado um exemplo de utilização deste framework proposto, através de uma evolução sobre o C&L, software para descrição de cenários e léxicos.

Exemplo Lúdico

Meu professor orientador costuma dizer que uma boa imagem vale mais que mil palavras. Neste documento, serão apresentados muitos conceitos, que na maioria dos casos, são difíceis de serem explicados somente através da teoria. Como nos exemplos aqui não será possível associar uma imagem, creio que a melhor alternativa é estimular ao leitor a criação desta imagem em sua própria mente.

Para solucionar este problema e ser mais didático, usarei no decorrer deste documento um recurso lúdico para ilustrar os exemplos que seguirão. Neste, serão usados personagens reais em situações não reais.

Trata-se de um site fictício que disponibiliza receitas culinárias na internet, o “MestreCuca”. Onde podem ser inseridas novas receitas, removidas as ultrapassadas, inserir comentários e atribuir notas para as receitas.

A idéia do site começou com um grupo de amigos que gostava de comer, cozinhar, cozinhar e comer, beber e cozinhar e comer. Seja com amigos, com a família, mulher, namorada, amante... Enfim, cada um gostava dos prazeres da gula pelos mais variados motivos.

Em lugares distintos, os amigos raramente se encontravam, mas sentiam necessidade de trocar informações sobre pratos, receitas, novidades e dicas de cozinha. Com isso, um dos amigos, o Bill, deu a idéia do site.

Bill Gates sentia muita fome durante sua jornada de trabalho, que duravam horas e mais horas. Costumava pedir para sua mulher cozinhar para ele, mas como cozinha não era a especialidade dela resolveu pegar algumas receitas com seu amigo Vinicius. Vinicius de Moraes, amante declarado de todas as mulheres, além da poesia e da música, utilizava a culinária como arte em suas conquistas amorosas.

Jô, o Soares, esse gosta de comer. Entediado por passar madrugadas em claro sem uma boa comida feita na hora, logo se juntou ao grupo com novas idéias. Consigo, levou para o grupo seu amigo Ray Charles, que adorava cantar cozinhando, porém, sempre cozinhava às cegas.

O ministro Gilberto Gil, nosso último personagem, foi convidado a entrar no eclético grupo de amigos depois de um farto jantar de comida baiana com Bill e Vinicius na casa de Albert Einstein, que se tornou uma espécie de guru do grupo após apresentar estudos da teoria da relatividade culinária.

1. Paradigma Web

1.1 – Preliminares

Neste capítulo, abordarei conceitos aos quais farei referência durante todo esse projeto. Muitos desses conceitos não são inéditos, e podem ser encontrados, de uma forma ou de outra, espalhados pela web [5] [6] [7] [23].

Esse capítulo tem como proposta, inicialmente contextualizar o leitor ao ponto que o mesmo entenda a visão do autor sobre o meio no qual estaremos inseridos. Neste serão abordados os pontos necessários para o entendimento sobre o tema, tais como arquitetura, protocolos, segurança e funcionamento.

1.2 – Visão Geral

A World Wide Web (WWW) foi concebida inicialmente para ser um repositório de dados navegável como um hypertexto. Onde pessoas de lugares diferentes poderiam cambiar informações, estudos ou qualquer publicação textual escritas uma linguagem de marcação. Devido a sua grande popularidade, logo evoluiu e adaptou-se, aos desejos dos usuários, para suportar também diferentes tipos de extensões, que diziam respeito não só as melhorias visuais, mas também a navegação e a recuperação dos dados.

Essa brusca evolução deixou a web um pouco desestruturadas, uma vez que cada fabricante de navegadores disponíveis tentava impor o seu padrão com funcionalidades diferentes, o resultado foi uma total interoperabilidade entre eles. Chegando até a ter sites que eram perfeitos num determinado navegador e em outros simplesmente não aparecia. O que força os empresários, donos de negócios interessados em publicar sua companhia na web, a ter seu site desenvolvido diversas vezes, uma para cada navegador, resultando num gasto muito excessivo e nem sempre obtendo resultados desejáveis.

Criado em 1994, com o intuito de promover avanços integrados para web, o W3C (World Wide Web Consortium) elaborou especificações e diretrizes, as quais denomina de recomendações, para garantir a compatibilidade entre plataformas e entre tecnologias que fazem uso da web. Assim como outras entidades que chegaram para definir os padrões que seriam adotados, como, por exemplo, a ECMA (European Computer Manufacturers Association), que é responsável pelo ECMAScript, padrão da linguagem conhecida com JavaScript.

Todos esses esforços desses grupos estão transformando a web, propiciando um ambiente mais rentável, confiável e com vantagens para o desenvolvimento de aplicações de pequeno e médio porte. Emergidos dentro de um escopo de flexibilidade, pela facilidade de uso, combinada com a descentralização de processamento, aliada ao conceito de multi-plataforma, fizeram da web uma alternativa viável para uma classe muito grande de softwares e aplicações.

1.3 – ARQUITETURA BÁSICA

O modelo de arquitetura adotado na web hoje é o modelo cliente-servidor. Os servidores, locais aonde estão armazenados as informações encontradas na web como páginas, websites e documentos, utilizam protocolos de comunicação e fornecem serviços que são requisitados pelos clientes, os browsers.

A arquitetura cliente-servidor classifica seus componentes básicos em interface-gráfica, lógica do programa e dados. Podendo eles estar tanto no servidor, como também no cliente. Por exemplo, podemos armazenar dados no servidor, usando um servidor de banco de dados, como também podemos armazenar dados no cliente, escrevendo em arquivos no disco onde o cliente está rodando, e.g. usando cookies; podemos também ter a lógica do programa

no servidor, com interpretadores de scripts server-side, e.g. PHP, ASP e CGI, ou então ter essa lógica no cliente, e.g. utilizando JavaScript.

1.4 – PROTOCOLO HTTP

HTTP(Hiper Text Transport Protocol) é um protocolo usado na comunicação na web. Tecnicamente, é um protocolo de nível de aplicação, destinado a sistemas de informações hipermídia distribuídos.

Consiste em trocas de pedidos, respostas e pacotes de informações. Funciona da seguinte forma: um cliente estabelece uma conexão TCP a uma porta no servidor e envia um pedido de um recurso, então, o servidor recebe o pedido, envia a resposta e fecha a conexão.

Mais informações sobre este tópico podem ser encontradas em[13][14].

1.5 – SEGURANÇA

Existe uma preocupação muito forte com relação à segurança dos dados na web. Com razão, uma vez que a web é um ambiente suscetível a falhas, pois, em geral, utiliza protocolos de comunicação não confiáveis. Esse fato deixa os dados de nossas aplicações bastante vulneráveis a diversos tipos de ataque. Para minimizar esse problema existem protocolos seguros que provêm um nível de confiabilidade bem maior.

O protocolo de segurança SSL – Secure Socket Layer possibilita que comunicações pela rede tenham garantia de privacidade. Garantida pelo uso de técnicas de criptografia, com a composição dos métodos criptográfos conhecidos como “PublicKey” e “SecretKey”.

Simplificadamente, criptografia é uma maneira de codificar uma informação de forma que somente com uma senha seja possível fazer a decodificação. Essa senha é chamada de chave criptográfica, e deve ser mantida em segredo para garantir a privacidade.

1.6 – O NOVO PARADIGMA

Devido aos avanços e as novas tecnologias, que hoje tendem a se padronizar, o desenvolvimento de aplicações, ou softwares de pequeno e médio porte para web, começou a ganhar adeptos entre os desenvolvedores e interessados. Consequentemente, assim como o desenvolvimento para aplicações web amadureceram, os desenvolvedores tiveram que se adaptar aos novos paradigmas impostos[6].

Esses novos conceitos modificam bastante os conceitos mais tradicionais de programação e arquitetura de software, uma vez que aplicações web executam em um ambiente distribuído, onde cada parte que compõe o software pode estar rodando em diversa maquinas em diferentes redes. Deixando para processamento na máquina cliente somente a parte relacionada à interface visual.

Muitos programadores poderiam se assustar com tudo isso, pois para simples aplicações, poderiam pensar que seriam obrigados a criar códigos imensos para sistemas distribuídos. Porém, os servidores web utilizam infra-estrutura de redes já existentes, deixando transparente para os usuários o seu uso, e através de protocolos, permitem que as aplicações troquem informações e requisitem serviços. Tornando irrelevante para o desenvolvedor a topologia física ou lógica da rede.

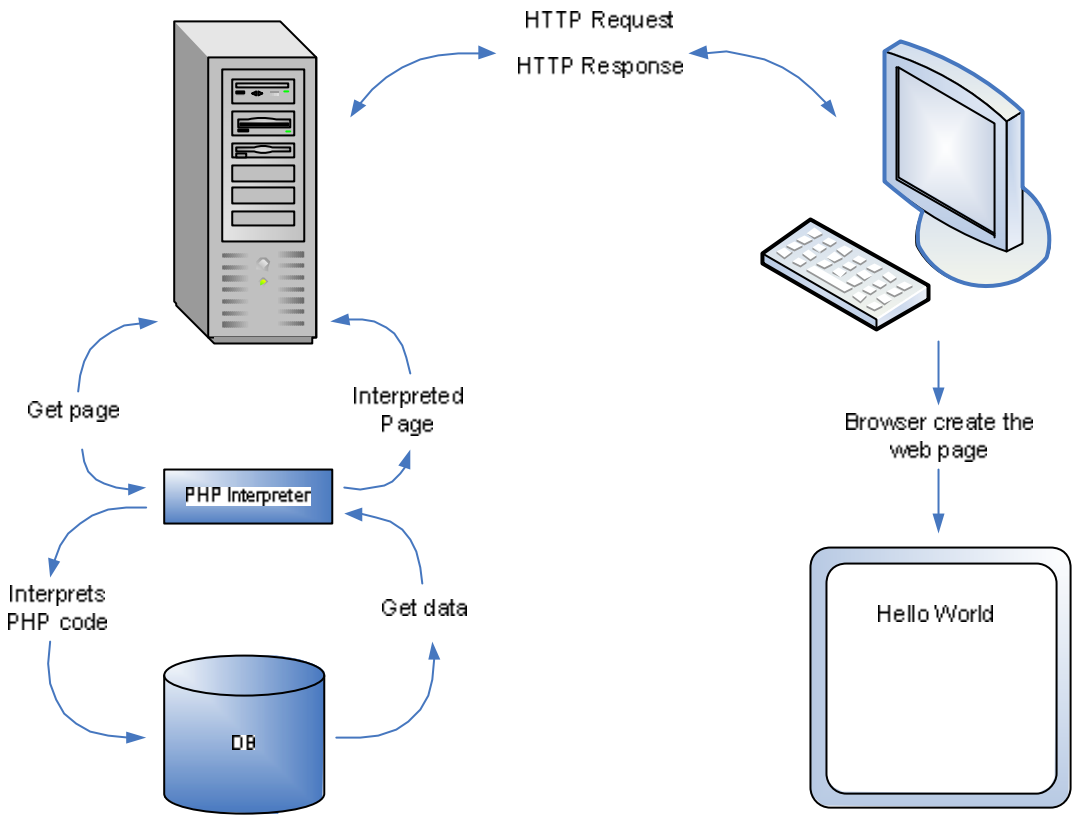
Talvez pela dificuldade e demora na padronização[5], ou até por conseqüência das necessidades dos interessados em aplicações para web, o desenvolvimento de software para web não acompanhou os avanços e estudos que foram realizados nas áreas de desenvolvimento de softwares tradicionais e sistemas distribuídos. Deixando escapar, entre outros, conceitos de construção e conceitos arquiteturais.

1.7 – A NOVA WEB

"Conhece-te a ti mesmo – isto é, torna-te consciente de tua ignorância - como sendo o ápice da sabedoria, que é o desejo da ciência mediante a virtude".

Seguindo o lema do filosofo Sócrates, antes de adicionar novos conceitos ao processo de desenvolvimento, e descrever uma arquitetura bem estruturada que atenda as necessidade dos desenvolvedores, permitindo que sejam desenvolvidas aplicações que preencham os requisitos dos interessados, precisamos antes entender o funcionamento de um processo no ambiente onde estaremos inseridos.

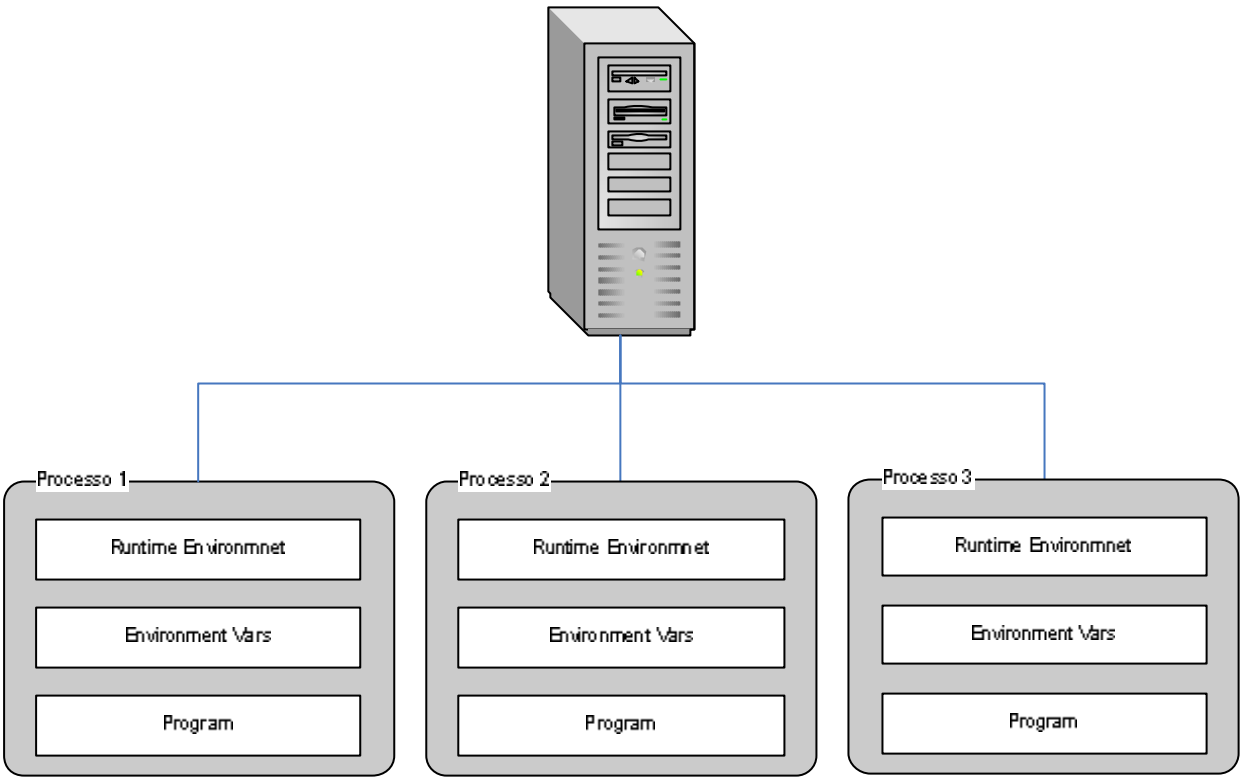
O diagrama a seguir ilustra a principal estrutura das partes envolvidas em um processo na web. Este engloba em um sistema cliente-servidor, o servidor web, um interpretador de linguagem de script, um servidor de banco de dados, que pode estar ou não em máquinas diferentes do servidor web, e do lado do cliente um "browser".



❒ Figura 1.1 - Processo de comunicação entre cliente e servidor na web.

O processo[13][14] ilustrado na figura 1.1 inicia quando um cliente, browser utilizado pelo usuário, faz uma requisição http para o servidor web. O servidor então armazena o endereço e a porta pela qual esta sendo feita a comunicação com o cliente, então carrega a página selecionada. Parte dos dados carregados na página pode conter códigos que precisam ser interpretados, então, se necessário, a envia para o interpretador. Este por sua vez tratará de interpretar e processar o que estava codificado, inclusive fazendo requisições de outros serviços, como por exemplo, de banco de dados, conexões FTP(File Transport Protocol) ou serviços do sistema de arquivos.

Quando os dados estão prontos, o interpretador entrega ao servidor web as informações de resposta, que por sua vez repassará as informações, pela respectiva porta, para o endereço armazenado. Repassando então o resto do processamento para o cliente, browser, que receberá essas informações e saberá interpretá-las. A figura a seguir ilustra como é a arquitetura em relação a seus clientes. Separando a comunicação por portas(SAP's do nível mais alto da arquitetura Internet).


Processo 1
Runtime Environmnet
Environment Vars
Program
Processo 2
Runtime Environmnet
Environment Vars
Program
Processo 3
Runtime Environmnet
Environment Vars
Program

desenhada com esses princípios, e não almejava atender a esses propósitos. Essas limitações a tornam uma ferramenta de desenvolvimento fraca se comparada a linguagens com Java, VB, Delphi e C++. E mesmo com os avanços e as novas linguagens de apoio, como XML, DOM e JavaScript, e técnicas AJAX, entre outras, ainda não existe, até aonde essas pesquisas me levaram, uma descrição de arquitetura simples e consistente que tenha como objetivo atender as necessidades do paradigma web descrito neste capítulo.

2. Web Compatível e Acessível

2.1 – INTRODUÇÃO

Neste capítulo serão levantadas questões e conceitos relacionados a compatibilidade e acessibilidade de conteúdo web. Serão abordadas as vantagens e a importância de ser acessível. Em seguida, a partir destes conceitos e dos demais apresentados neste documento, apresentarei minha visão de um padrão estrutural adequado às necessidades de hoje.

2.2 – EFEITOS DA COMPATIBILIDADE

É comum encontrar espalhados pela rede sites mal formulados, com tabelas aninhadas em vários níveis e com excessivo uso de tags desnecessárias e fora do padrão. Essas e outras redundâncias triplicam a largura de banda necessária para que os usuários possam carregar nossas páginas.

Como uma bola de neve, as conseqüências da má elaboração do conteúdo disponibilizado podem se tornar enormes. As conseqüências imediatas são contratar mais horas de programadores e pagar por mais espaço em servidores. Posteriormente, serão percebidas conseqüências maléficas: o número de acessos de usuário de redes de baixa velocidade, que certamente diminuirá, pois muitos desistirão de esperar longos minutos para carregar uma simples página; ser obrigado a re-programar o conteúdo do site em função do lançamento de novas versões ou de atualizações nos softwares que utilizam esse conteúdo.

Antes de surtir efeitos dos esforços de padronização, pressionados por seus contratantes que desejavam, com toda razão, maior acessibilidade na internet, não querendo restringir seu público à usuários de determinados browser’s, muitos desenvolvedores eram forçados a desenvolver um mesmo projeto, ou partes dele, particularmente para atender diversos softwares usuário.

```
if(document.layers)                                    //SE FOR NETSCAPE 4.7X
OU <
{
        document.write("<ilayer id='test'>");
        //…
}
else if(document.getElementById && !document.all)  //SE FOR NETSCAPE 6
{
        document.write("<div id='test'>");
        //…
}else           //SE FOR EXPLORER
{
        document.write("<div id='test'>");
        //…
}
```

Essa técnica, bifurcação de código, ficou muito famosa por ser dispendiosa e deselegante, porém era muito utilizada para garantir a compatibilidade de sites em browser's de diferentes fabricantes. E hoje, em função de seus malefícios é obsoleta e condenada.

A compatibilidade de um site está relacionado aos padrões. Manter o conteúdo conforme os padrões estipulados é equivalente a dizer que nós fizemos a nossa parte para deixar nosso site compatível com versões futuras de leitores web. Pois de resto, órgãos como W3C e desenvolvedores de browser's terão que fazer. E como tudo indica que estão fazendo a parte deles, cabe a nós fazer a nossa.

2.3 – ACESSIBILIDADE

É crescente o número de usuários de PDA's e celulares com acesso a internet. Esses dispositivos não dispõem do mesmo poder de processamento que um computador tradicional. Até eletrodomésticos estão entrando na lista de dispositivos que podem ler o conteúdo disponibilizado na web. São dispositivos mais restritos, porém, não podemos excluir esses milhões de usuários de público alvo de nossos sites.

Ser acessível não significa que todos devem ver as páginas do mesmo modo. Acessibilidade diz respeito a conteúdo e informações. Diz respeito a tornar todo o conteúdo (textos, imagens e multimídia) disponíveis ao usuário. O ponto crucial aqui é que todos os usuários de dispositivos de diferentes poderes de processamento tenham acesso às mesmas informações, independente de plataforma, hardware ou software utilizado.

Com pouco esforço qualquer programador, mesmo os iniciantes, são capazes de construir sites acessíveis. Basta que tenham em mente a importância e a relevância de ser acessível.

2.4 – TECNOLOGIA SERVER-SIDES

Tecnologias server-side viabilizam aplicações mais avançadas e mais hábeis via web, pois agregam à simples páginas estáticas dinamicidade e serviços poderosos, como por exemplo, acesso a banco de dados e conexões remotas entre outras.

Tecnicamente, o que esse tipo de tecnologia faz é processar informações e construir dinamicamente uma página com conteúdo descrito em linguagem de marcação, i.e. HTML, XHTML, ou XML. O que esses processadores distribuem é um conteúdo que funciona melhor quando está semântica e claramente estruturado. E é nesse ponto que ocorre a maioria das falhas.

Acessibilidade e padrões tem muita coisa em comum. Ambos garantem que nosso trabalho será disponível, de forma útil, ao maior número de pessoas. Porém estar fora dos padrões estabelecidos pode ser garantia de mau funcionamento e problemas futuros, que podem levar a estagnação do software e até a obsolência.

2.5 – W3C – WORLD WIDE WEB CONSORTIUM

Fundada por Tim Berners em 1994 a World Wide Web Consortium é um consórcio de empresas de tecnologia para desenvolver padrões - protocolos comuns e fóruns abertos que promovem a evolução e a interoperabilidade na web.

Os sites que seguem os padrões estabelecidos pelo W3C são acessíveis a qualquer pessoa, independente de plataforma, hardware e software utilizados.

É dividida em comitês que se dedicam, cada um, a uma tecnologia voltada para apresentação de conteúdo na internet. Esses comitês desenvolvem recomendações que passam por um processo de maturidade. O primeiro é “Working Draf”(WD), depois “Candidate Recommendation”(CR), “Proposed Recommendation”(PR) e finalmente “W3C Recommendation”(REC). As recomendações estão sob a licença “royalty-free patent”, possibilitando que qualquer um os implemente eles.

Diferentemente de outras entidades de padrões internacionais, o W3C não tem um programa de certificação, pois considera que este criaria barreiras, restringindo a comunidade adepta, e atrapalharia a disseminação dos padrões.

2.6 – Acessibilidade Especial – Introdução ao Novo Mundo

A história mostra que a sociedade pode ser cruel com minorias, entretanto, como uma balança, tenta sempre reparar suas falhas, mesmo que tarde. Deficientes físicos sempre estiveram à margem da sociedade durante décadas.

Leis governamentais exigem que pessoas com deficiência física participem de atividades sociais, tenham acesso a informações e serviços com direitos iguais. A indústria civil foi uma das primeiras que tiveram que se adaptar para atender as necessidades de deficientes e seguir essas leis.

Na computação, o termo acessibilidade não se restringe à softwares. Existem milhões de ferramentas que permitem que portadores de deficiência tenham acesso às informações em um computador. Dentre essas estão leitores de telas para deficientes visuais, sintetizadores e reconhecedores de voz, teclados virtuais...

Hoje em dia existe um esforço muito grande para que todo o conteúdo disponibilizado na web seja acessível também a essa parte da população. Mas para isso, não basta apenas seguir os padrões web. Precisamos estar atentos às necessidades desses usuários.

Imagine um deficiente visual que deseja acessar o site do “MestreCuca”. Para isso, certamente ele irá usar um leitor de tela. Contudo, se o site estiver mal formatado, ou não programado para esse tipo de usuário, o leitor de tela pode ler nomes de botões, figuras e outras marcações sem significado antes de chegar a ler as informações relevantes no site.

Esses e outros problemas que teremos que estar atentos quando formos criar um conteúdo para web são discutidos em [5].

2.7 – Validação

Existem na internet sites que fornecem validadores de sitaxe de diversas linguagens. Esses validadores são muito úteis, pois podem avaliar se a página que estamos gerando está dentro dos padrões estabelecidos. Abaixo segue uma lista de validadores para diversas linguagens:

- W3C's Markup Validator - http://validator.w3.org/
- Validators - http://uitest.com/en/analysis/
- Site Check - http://uitest.com/en/check/
- CSS – http://www.htmlhelp.com/tools/validator/
- XML – http://validator.aborla.net/

2.8 – PADRÕES ESTRUTURAIS

No cerne deste trabalho está a pesquisa sobre padrões estruturais na web e como é dividia conceitualmente. Neste capítulo apresentarei aspectos que fundamentaram este projeto final de graduação. Mostrarei também a evolução destes conceitos e como eles se aplicam na prática. No decorrer deste documento o leitor atento poderá identificar diversas alusões aos conceitos apresentados neste capítulo.

No desenvolvimento de software existe uma preocupação constante com a separação entre conteúdo da aplicação e o modo como é apresentado. Mais adiante veremos padrões que tentam resolver essa separação, como por exemplo MVC. Mas antes trataremos dos conceitos relacionados a esta separação. Tratando mais especificamente do contexto web, um software pode ser dividido nas seguintes partes:

- Dados - É o conjunto de informações relevantes. São somente os dados da aplicação, e não o modo como serão apresentados, como se comportarão ou como serão tratados. Podem estar armazenados em banco de dados, em arquivos binários, ou em arquivos textuais.
- Processamento/Processing - Se refere ao modo como são processadas as informações. É o modelo da aplicação, onde se encontram as rotinas, classes e funções que manipulam os dados da aplicação. Junto com os dados, compõem a parte central do software.
- Comportamento/Behavior - Diz respeito ao código responsável por como nossa aplicação agirá em resposta as ações dos usuários. Código responsável pelo tratamento de eventos – ação e reação. Em suma, todo código responsável por como a aplicação se comportará.
- Estrutura de Apresentação/Presentation Structure - Está relacionada ao modo como estes dados estarão estruturados, independente de onde estão armazenados e como serão apresentados.

- Estilo de Apresentação/Presentation Style - É a parte da aplicação onde definimos como serão apresentados os dados ao usuário. Basicamente, onde definimos o layout da interface gráfica, desde o posicionamento de elementos, cores, bordas e até a tipagem utilizada.

A separação das partes descritas anteriormente em websoftware's acompanhou a evolução das tecnologias e dos padrões web, veja na figura abaixo. As tecnologias indicadas na figura 2.1 serão comentadas mais tarde nos próximos capítulos.

A primeira vista poderíamos achar que a separação e as diversas tecnologias poderiam dificultar o processo de desenvolvimento. Causando demora e confusão nos desenvolvedores. Pelo contrário, considero que esta separação agrega inúmeras vantagens para a aplicação. Dessa forma, podemos colocar processos em paralelo, e.g., deixar uma equipe de desenvolvedores responsáveis pelo conteúdo da aplicação e em como estaria a estrutura desses dado, enquanto que outra equipe trabalha no design gráfico da aplicação.

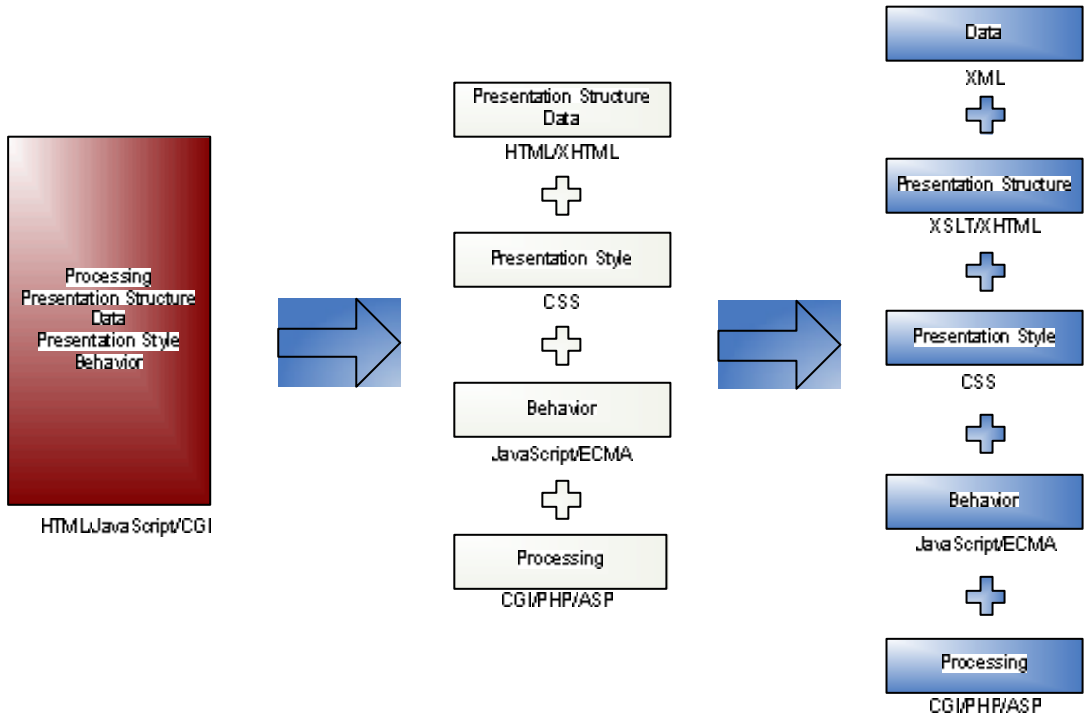
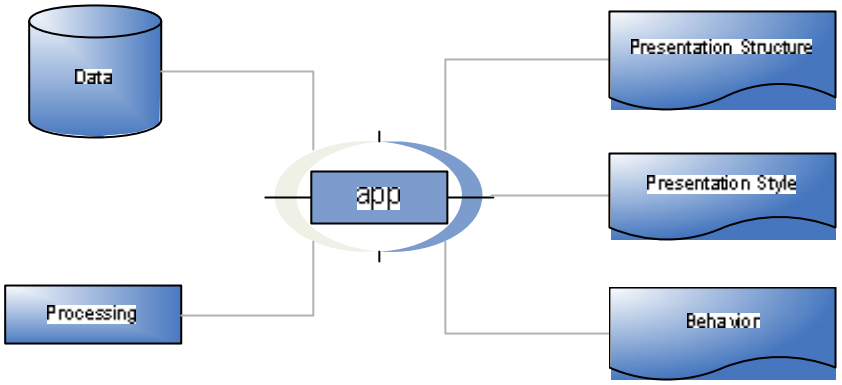


Figura 2.1 – Evolução da separação do conteúdo de web software's

Observe que dessa forma deixamos pessoas especializadas em cada área trabalhando independentemente do outro. A figura abaixo ilustra essa divisão da estrutura interna básica de uma aplicação web tradicional adequada às necessidades de hoje.



❒ Figura 2.2 – Divisão da estrutura de uma aplicação web.

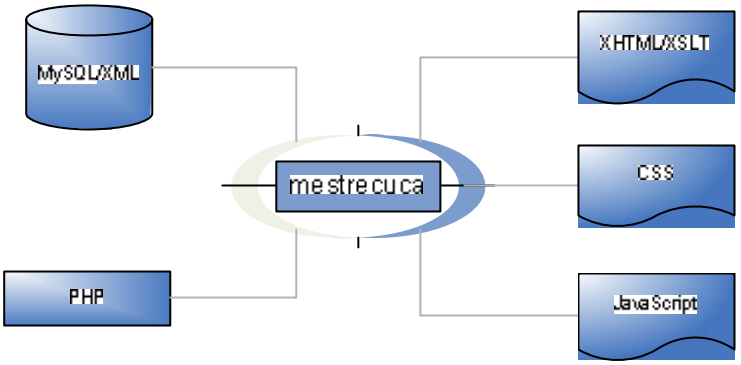
Definidos esses componentes, poderíamos classificar como independentes ou persistentes em uma aplicação, os dados e processamento do modelo da aplicação. Conseqüentemente, para cada dispositivo ou software com possibilidades de recursos diferentes, teríamos que nos preocupar somente com as partes de apresentação, comportamento e talvez também a estrutura.

Para elucidar ainda mais, vamos analisar esses conceitos inseridos em nosso exemplo lúdico. O “MestreCuca” está ligado a um banco de dados que armazena receitas. Neste encontram-se tabelas de receitas e autores de receitas, tabelas de relacionamento e tudo mais que se refere aos dados da aplicação.

Para o processamento das informações do site, Bill, mesmo tendo a sua disposição o ASP(Active Serve Page) da Microsoft, outra linguagem de script server-side, optou por utilizar PHP para poder realizar testes com a concorrência. Criou classes e scripts que recuperam as receitas e dados relacionados a ela do banco de dados. Esses dados são repassados em forma de XML.

Acompanhando esse XML, estão as folhas de estruturas de apresentação, escritas com uma combinação de XSLT/XHTML, e as folhas de CSS, que determinam o design gráfico da aplicação. A parte de comportamento do site foi desenvolvida em JavaScript, já que no cliente a aplicação será acessada via browser.

Para o leitor que ficou um pouco perdido com essa sopa de letrinhas, não se preocupe. Mais adiante, nos capítulos posteriores abordarei novamente sobre essas tecnologias.



❒ Figura 2.3 – Divisão estrutural do MestreCuca

3. Engenharia de Requisitos

3.1 – Introdução

Neste capítulo irei apresentar uma síntese de estudos realizados na área de engenharia de requisitos. Esta área é uma área bastante extensa e motivo de diversos estudos. Neste relatório, farei uma breve introdução sobre requisitos[25][26], e ressaltarei sua relevância e sua aplicação na engenharia de software.

Na seqüência, apresentarei um resumo de estudos relacionados a descrição de cenários e léxicos[26][3] – técnica para descrição de requisitos que baseia parte deste trabalho final de graduação.

3.2 – Visão Geral

Um requisito é um uma necessidade singular documentada de o que um produto particular deve ser ou realizar. Uma sentença que expressa as necessidades dos clientes. É comumente utilizado, de modo formal, em engenharia de sistemas ou engenharia de software.

Na abordagem clássica de engenharia, conjuntos de requisitos são usados como entradas dos estágios de design do processo de desenvolvimento do produto.

O objetivo da engenharia de requisitos é a definição do conjunto de requisitos de um software. A coleção de requisitos então define as características ou funcionalidades do sistema desejado, mas não diz como deve ser implementado tais requisitos no sistema. Menções nos

requisitos descrevendo como um sistema deve ser implementado são conhecidas como “vícios de implementação” (implementation bias).

A fase de desenvolvimento de requisitos pode ser precedida por uma fase de análise conceitual ou um estudo de aplicabilidade, que consiste em um estudo preliminar realizado antes do início de um projeto cujo objetivo é estimar sua chance de sucesso. Envolve uma análise de todas as possíveis soluções de um problema e a recomendação da melhor solução a ser adotada, e a avaliação de como a solução se encaixa na empresa. A fase de requisitos pode ser decomposta em elicitação de requisitos, análise, definição e especificação.

3.3 – FATORES NO DESENVOLVIMENTO DE REQUISITOS

Requisito é a função ou capacidade que um cliente espera que um produto atenda. São tipicamente classificados em três categorias:

- Requisitos funcionais – descrevem funcionalidades do sistema ou tarefas que o sistema deve realizar.
- Requisitos não funcionais – descrevem propriedades que o sistema deve ter. Estão relacionados com a qualidade do sistema no que diz respeito ao desejo do contratante (ex: performance, disponibilidade, acessibilidade, segurança).
- Restrições – limitam o desenvolvimento de alguma forma, como para qual sistema operacional será desenvolvido o software, ou qual linguagem de programação deve ser usada para implementar o software.

> *“O processo de definir requisitos é inerentemente incompleto, tendo em vista a grande complexibilidade do mundo, É óbvio, no entanto, que sempre estaremos procurando ter requisitos os mais completos possíveis.”.*

Requisitos são notoriamente difíceis de serem assimilados em um nível ideal. Usualmente, expert users são acionados para prover um elo entre usuários e desenvolvedores. Esses expert users são capazes de expressar requisitos funcionais de modo que sejam facilmente traduzíveis em uma característica de design do sistema, e, ao mesmo tempo, compreendidos pelo usuário final.

3.4 – A IMPORTÂNCIA DOS REQUISITOS PARA QUALIDADE DE SOFTWARE

Em engenharia de sistemas, requisito é uma descrição do propósito de um sistema, isto é, o que ele deve fazer realmente. Sistemas podem ter de dezenas a milhares de requisitos. O conjunto de requisitos condicionam a qualidade do software.

Assim como acontece em outros produtos, os consumidores de softwares, cada vez mais, exigem qualidade e preço. Mas a qualidade de um software não está somente ligada ao produto final. Também diz respeito aos processos utilizados para gerar o produto.

Podemos iniciar o processo de auditoria de qualidade na descrição dos requisitos não funcionais que o sistema deverá atender. Assim, abrangemos ainda na definição dos requisitos aspectos funcionais – funcionalidade do sistema, e aspectos não funcionais – critério de qualidade.

3.5 – CARACTERÍSTICAS DE REQUISITOS

Devemos descrever os requisitos de modo que todos sejam testáveis ao final da implementação, caso existam requisitos que não se enquadrem nesses parâmetros, é necessário que estes sejam alterados ou descartados. Também devemos estar atentos ao descrever requisitos para que tenham as seguintes características:

- Necessários
- Singulares (sem ambigüidades)
- Concisos
- Consistentes
- Completos
- Alcançáveis
- Verificáveis

3.6 – ANÁLISE DE REQUISITOS

A análise de requisitos, em engenharia de software, é um termo usado para descrever todas as tarefas que envolvem a instigação, delimitação de escopo e definição sistema novo ou alterado. A analise de requisitos é uma parte importante do processo de engenharia de software, onde analistas de negócio (business analysts) ou desenvolvedores de software identificam as necessidades ou requisitos de um cliente.

Cada dia mais a definição e análise de requisitos vem ganhando importância. Pois contratados e contratantes de serviços de TI vêm tomando consciência de que uma falha nesta fase pode implicar em insatisfação ou funcionamento não esperado do software.

A maior dificuldade da análise de requisitos é superar o gap de comunicação entre usuários e uma empresa IT. Por isso é considerado um campo de especialidade melhor exercido por experts (business e system analysts), que podem superar esse gap.

Existem também diversas técnicas e metodologias para definir requisitos. Mais adiante veremos uma metodologia para descrição de requisitos – a descrição de cenários.

3.7 – RASTREABILIDADE DE REQUISITOS

Rastrear é definido como marcar, seguir e conferir. Muitos autores [11] [12] atribuem a rastreabilidade de requisitos sendo o ponto base da gerência de requisitos no processo associado a qualidade do desenvolvimento de softwares. E é vista na literatura como fator de qualidade.

Hoje em dia, é comum em um processo de desenvolvimento que os requisitos sejam modificados, seja por parte do dos interessados, steakholders. Quando mudam sua perspectiva sob um determinado ponto, ou por parte da equipe de desenvolvimento, onde os motivos seriam vários, entre eles, quando identificam erros, alterações no contexto, ou quando são encontradas necessidades não identificadas.

A rastreabilidade de requisitos é utilizada para explicitar relacionamento entre requisitos, arquitetura e implementação. E pode ser instaurada por um conjunto de elos, links, entre os requisitos e suas fontes, como por exemplo outros requisitos ou imposições dos steakholders ou da tecnologia disponível, e entre requisitos e seus destinos, como por exemplo os componentes que o implementam.

Uma especificação de requisitos de software são rastreáveis se a origem de cada requisito é clara e facilita a referência para cada requisito no futuro desenvolvimento ou na documentação.

Rastreabilidade de requisitos está na habilidade de descrever e seguir a vida do requisito, para as duas direções, anterior e posterior.

- Forward Traceability: capacidade de rastrear um requisito até seu componente.
- Backward Traceability: capacidade de rastrear um requisito até sua fonte.

❒ Figura 3.1 – Direções da rastreabilidade de requisitos

Rastreamos “forward” quando os requisitos forem modificados e queremos projetar e avaliar os impactos da mudança. Refere-se aos aspectos da vida do requisito para sua inclusão na especificação. E rastreamos “backward” quando temos uma modificação e queremos entender os requisitos, investigando a informação usada para elicitar a modificação dos requisitos. Se refere aos aspectos da vida do requisito que resultaram da sua inclusão na especificação.

3.8 – CENÁRIOS E LÉXICOS

A descrição de cenários é uma técnica para descrição de requisitos em linguagem natural de situações enfatizando o comportamento. Tais situações podem ser descritas sob o foco de funcionalidade do sistema e funcionamento específico. Descrevemos os cenários utilizando símbolos do Léxico.

Tem o objetivo de retratar requisitos de forma que a comunicação entre projetistas, usuários e desenvolvedores se torne mais fácil.

Os cenários e os símbolos do léxico devem seguir dois princípios fundamentais:

- Princípio do vocabulário mínimo: deve-se utilizar um conjunto restrito de palavras para descrever noção e impacto. Preferencialmente contendo termos de linguagem natural não pertencentes ao LEL.
- Princípio da circularidade: estabelece que os símbolos do LAL devem definir-se utilizando o máximo outros símbolos do LEL. Desta forma é reduzida a ambigüidade e maximiza o uso dos símbolos do LEL. Além de permitir navegação entre cenários e léxicos.

O LAL(Léxico ampliado da linguagem), ou Léxico, é uma espécie de dicionário do vocabulário empregado na descrição do cenários. Descreve os termos(símbolos) utilizados nas

sentenças dos cenários. Esses termos retratam dois aspectos: noção(denotação) e impacto(conotação). Cada símbolo é descrito pelos seguintes itens:

- Nome: nome do símbolo. Nome ao qual serão feitas referências. Um símbolo pode ter mais de um nome – sinônimos.
- Impacto: como o símbolo afeta o macrosistema, que tipo de influência exerce. Cada símbolo do LEL deve conter um ou mais impactos.
- Noção: onde é definido o significado do símbolo. Deve conter uma ou mais sentenças que ajudam a entender o significado do símbolo no macrosistema.
- Tipo: pode ser um objeto,

Nível	Projeto
verbo	representam ações que são realizadas no macrosistema.
objeto	representam elementos passivos; recebem ações indicadas pelos símbolos do tipo verbo.
sujeito	representam pessoas ou organizações que executam as ações indicadas pelos símbolos do tipo verbo.
estado	descreve características como localização ou condições específicas.

❒ Tabela 3.1 – Tipos de símbolos do LAL

Um cenário pode conter referências a outros cenários – subcenários e contém vínculos com os símbolos do LAL. É constituído dos seguintes elementos: título, ator, contexto, episódios, exceções, objetivo, recurso,

- Título: deve ser suficientemente explícito para entender o propósito do cenário.
- Objetivo: Deve ser coerente com o título do cenário e indica o que deve ser realizado no cenário.
- Contexto: é a entidade do cenário que descreve seu estado inicial. Se divide em contexto geográfico, contexto temporal e precondição. É utilizado para saber se é possível realizar o cenário.

- Ator: pessoa ou organização que têm um papel no cenário. Realiza ações nos episódios e deve estar presente em pelo menos um episódio.
- Recursos: deve existir pelo menos um recurso na lista de recursos. Dispositivo ou outro elemento pasivo que deve estar disponível no cenário. Preferencialmente deve ser um símbolo do LAL. Deve também ser mencionado em pelo menos um episódio do cenário.
- Episódios: É um conjunto de sentenças que determinam o comportamento do cenário. Cada episódio é a descrição de uma atividade dentro o cenário realizada por um ator, onde se utilizam recursos. Pode ser uma referência para outro cenário. Devem existir dois ou mais episódios em um cenário.
- Exceções: É uma entidade opcional. Afeta a totalidade do cenário. É a causa que provoca que um cenário não possa ser realizado.

4. Engenharia Reversa

4.1 – INTRODUÇÃO

Durante o projeto final de graduação, executei e exercitei diversas técnicas de engenharia reversa sem mesmo antes conhecer a teoria sobre o assunto.

Este capítulo se faz presente neste documento, pois é a teoria que embasa uma fase necessária neste projeto. Além de ser uma área de grande interesse meu, também é uma forma de documentar o aprendizado que obtive nesta área.

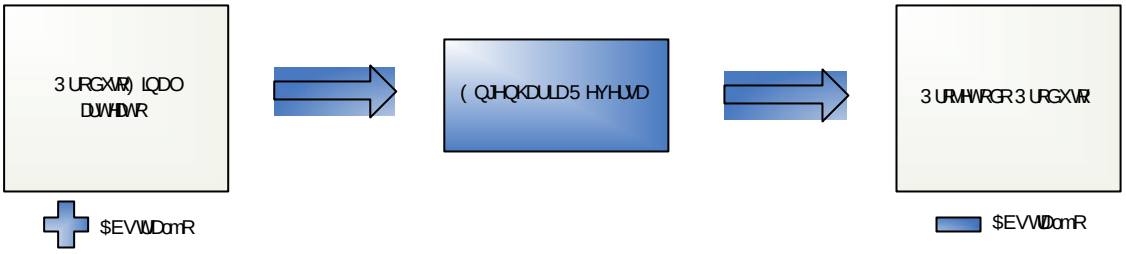
A seguir será apresentada uma síntese, sob a minha interpretação, do material sobre o assunto ao qual tive contato[8][9][22].

4.2 – VISÃO GERAL

O processo de engenharia reversa, hoje em dia, depende muito da criatividade da pessoa que o executa. É uma área para entendidos, estudiosos no assunto, técnicos familiarizados com a tecnologia do artefato estudado.

É o processo inverso da engenharia progressiva. É um processo de análise e estudo dos detalhes do funcionamento de um produto final, com o objetivo de extrair deste estudo seu

projeto, sem ter que copiar algo da implementação do original, nem muito menos violar leis de patente.



❒ Figura 4.1 – Engenharia reversa em grau de abstração.

Existem diversos motivos para realizar a engenharia reversa em um software ou hardware, abaixo estão alguns dos principais:

- Construir um novo produto com as mesmas funcionalidades
- Recuperação de código-fonte perdido
- Análise de código-fonte malicioso – detecção de vírus
- Correção de erros – quando o proprietário não esta mais disponível para corrigir
- Recuperação de código-fonte de terceiros
- Análise de produtos de concorrentes
- Segurança nacional - análise de produtos em situação militar

Aplicada em softwares, serve para compreender a estrutura e o sistema da aplicação. Pode ser aplicada de duas formas: análise do código ou entendimento da aplicação. Analisando somente o código, é muito difícil chegar aos propósitos de um sistema. Recuperar um projeto partindo do entendimento da aplicação é alcançar o projeto desde sua essência, engloba a análise de código, documentação existente e experiências pessoais.

Dependendo do objetivo da execução do processo, é necessário nível diferente de abstração no projeto do produto. Cada nível é chamado de nível de visão do projeto. Veja a tabela a seguir.

Nível	Descrição	Projeto
implementacional	Abstrai características da linguagem de programação	Esquema de implementação relacionando funções,

		declarações, inclusões...
estrutural	Abstrai detalhes da linguagem de programação para dar ênfase a sua estrutura	Esquema relacional entre os componentes do sistema
funcional	Abstrai a funcionalidade de um componente	Esquema que relaciona partes do programa às suas funcionalidades ou requisitos
de domínio	Abstrai o contexto em que o sistema está operando	Esquema relacional entre entidades da aplicação inserida em seu contexto como um todo

❒ Tabela 4.2 – Níveis de visão do projeto

Na história do desenvolvimento tecnológico existem vários casos famosos que contribuíram para a área. Como por exemplo o caso famoso dos desenvolvedores do kernel do Linux, no inicio do projeto, que foram praticamente obrigados a praticar a engenharia reversa para construir drivers de hardware cujo fabricante considerava que não valeria o investimento para desenvolvê-los.

Outro caso interessante aconteceu na Rússia, onde as leis de software são bem diferentes das nossas. Lá, quando compramos um software, ele se torna de nossa propriedade, nos permitindo usá-lo, aplicar engenharia reversa, abri-lo, modificar e, uma vez modificado qualquer detalhe, podemos redistribuí-los como desejarmos, inclusive vendê-los.

No caso citado, um russo abriu o código-fonte da Macromedia, o modificou, e redistribuiu gratuitamente pela internet. E a Macromedia não pode fazer nada, já que as leis da Rússia o protegiam. Mas isso é só na Rússia, no Brasil isso não é permitido.

Em seguida apresentarei informalmente algumas técnicas bastante utilizadas. O foco deste capítulo não é ensinar, ou detalhar cada técnica. Minha intenção aqui é mostrar resumidamente algumas possíveis técnicas a serem empregadas num processo de engenharia reversa.

A meu ver, a engenharia reversa, na verdade, está ligado diretamente a dois fatores: conhecimento sobre a tecnologia do produto e criatividade. Poder de abstração, manter uma visão do quadro geral e não perder o foco, certamente são características necessárias.

4.3 – PROCESSO CLÁSSICO

Nos anos 80, quando a Compaq surgiu, seu pessoal utilizou esse processo para copiar o chip da BIOS de um PC. Muito preocupados em não violar as leis de patente eles adotaram o processo de engenharia reversa ao qual hoje o chamamos de processo clássico. Mais tarde, muitos utilizaram a mesma técnica. O processo tem os seguintes passos:

- Contrata-se um grupo de engenheiros para registrar todas as funcionalidades do artefato.
- Este grupo cria um documento com todas as especificações dessas funcionalidades.
- Contrata-se um novo grupo que não tem contato com o anterior nem com o artefato.
- Esse novo grupo desenvolve um artefato baseado nas especificações do primeiro.

4.4 – DECOMPILAÇÃO

Um software, quando compilado e linkado, “transforma” os códigos-fonte em um executável. O processo de decompilação é exatamente o inverso, a partir do executável, alcançar os códigos-fonte.

No processo de compilação, a maioria os nomes usados em variáveis, funções e classes são convertidos em endereços. Muitos compiladores também são capazes de otimizar o código gerado. Consequentemente, muitas vezes não é possível chegar ao código exatamente como no original, tendo como resposta códigos com nomes de variáveis sem sentido semântico. Portanto a decompilação não é um processo de engenharia reversa completo.

Existem diversos decompiladores para várias linguagens, como C/C++, Java, Delphi, Flash, entre outros. Estes podem ser facilmente encontrados espalhados pela internet.

4.5 – NORMALIZAÇÃO

O objetivo da normalização é simplificar o modelo que descreve a estrutura de um projeto, bem como evitar falhas, e facilitar o entendimento das relações entre suas entidades.

Muito usado em projetos de banco de dados, é um processo utilizado para eliminar redundância de dados seguindo algumas regras simples. Pretende aumentar a disponibilidade dos dados, bem como melhorar a estrutura de armazenamento.

Essa técnica vem sendo utilizada também como parte do processo de engenharia reversa. Depois de coletadas as informações sobre o artefato estudado, é empregada normalização nos dados adquiridos com o intuito de facilitar o entendimento sobre o que se deseja replicar.

A normalização pretende deixar um modelo em uma forma normal. Existem diversas formas normais, entre elas: Primeira Forma Normal(1FN), Segunda Forma Normal(2FN), Terceira Forma Normal(3FN) e Forma Normal de Boyce-Codd.

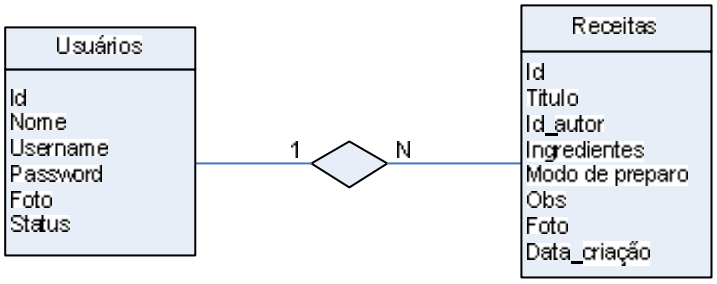
O processo de normalização tem como entrada um esquema relacional de fonte de dados estruturados, relatórios, arquivos, documentos, informações... Como saída, gera um esquema relacional para fonte de dados livre de redundâncias.

❒ Figura 4.3 – Entradas e saídas do processo de normalização.

4.6 – UML NO PROCESSO DE ENGENHARIA REVERSA

Tradicionalmente, um diagrama de classes UML ilustra como uma classe se relaciona com outras, quais são seus métodos e quais suas propriedades.

Utilizado na engenharia reversa como forma de documentação das informações coletadas sobre o artefato que deseja replicar, o diagrama UML ilustra as entidades (partes, componentes ou módulos de um artefato), como se encontram organizadas funcionalmente, como se relacionam, sua função efetiva e suas propriedades.



❒ Figura 4.4 – Exemplo de UML usada na engenharia reversa da aplicação do MestreCuca.

5. Re-Engenharia

5.1 – INTRODUÇÃO

Neste capítulo irei abordar alguns aspectos de reengenharia. Resumidamente tentarei mostrar um pouco da teoria envolvida[1][8][9][10][11][12], os processos e técnicas utilizadas aliadas ao meu ponto de vista sobre o assunto e a experiência obtida neste projeto final de graduação.

5.2 – VISÃO GERAL

Reengenharia: Método usado para reprojetar e reformar sistematicamente toda uma empresa, função e processo. Reconstrução de algo no mundo real buscando melhorias.

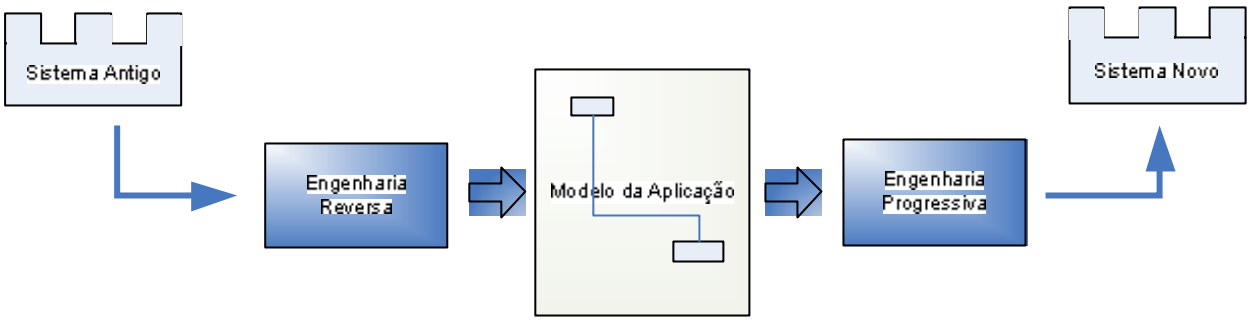
Na engenharia é uma reconstrução radical nos processos de uma organização, com o objetivo de agilizar processos, melhorar o rendimento, ou corrigir falhas.

Reengenharia de software é a modificações de um sistema de software, geralmente, feito para corrigir erros ou evoluir a aplicação – adicionar novas funcionalidades. Hoje em dia as técnicas de reengenharia são muito utilizadas também como forma de manutenção do sistema.

Diferentemente do desenvolvimento de software, que parte da descrição da especificação e vai até a implementação, a reengenharia parte de um artefato já construído. Reengenharia não é o mesmo que engenharia reversa, pois na reengenharia, busca-se melhorias e a reconstrução do sistema em questão. A engenharia reversa é uma etapa da reengenharia.

Foi definida [8] em duas partes, engenharia reversa e engenharia progressiva. A primeira extrai do sistema atual um modelo da aplicação contendo todas as informações reunidas. Esse

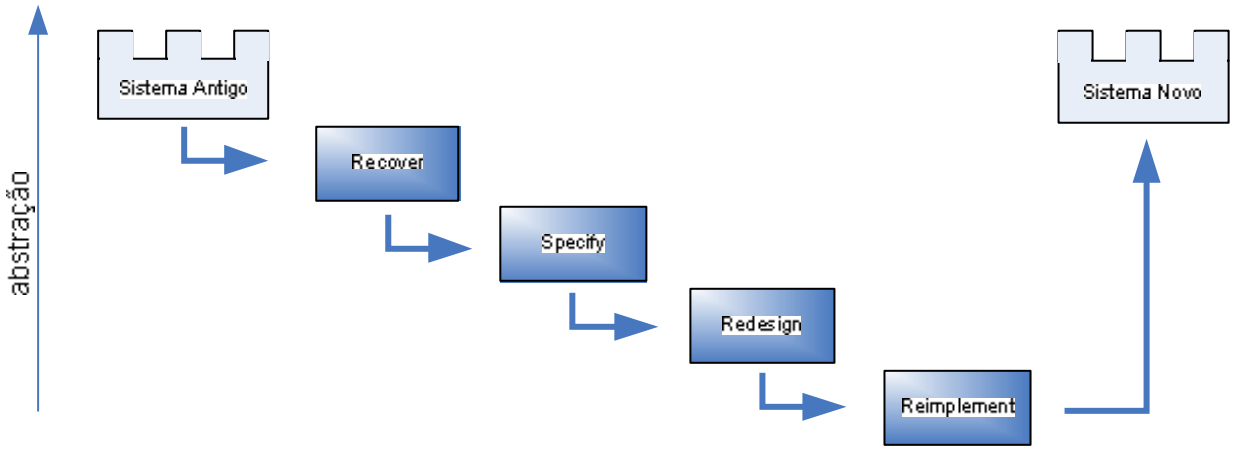
modelo irá fundamentar a segunda parte – engenharia progressiva, onde são feitas as modificações desejadas.



❒ Figura 5.1 – Processos da reengenharia

A engenharia reversa, já comentada neste trabalho, é utilizada como parte do processo de reengenharia com o objetivo de prover o entendimento do sistema. É a parte da reengenharia encarregada por tirar proveito dos esforços passados, para que o processo seguinte, de reconstrução do sistema, tenha seu tempo e custo de desenvolvimento reduzidos.

Podemos explicitar os processos ilustrados anteriormente dividindo-os nos seguintes subprocessos[9].



❒ Figura 5.2 – Sub-processos envolvidos na reengenharia

Recuperação/Recover é o processo baseado na engenharia reversa, tem o objetivo de desvendar os detalhes do artefato, identificar entidades, ações e funcionalidades. Essas informações servem de entrada para o próximo processo – especificação/specify, que ficará é

incumbida de prover para o processo seguinte uma relação das entidades e ações às suas funcionalidades. Gerando um modelo que favorece o entendimento da aplicação como um todo.

O processo de re-design/reestruturação, é iniciado identificando no modelo de entrada quais partes serão reusadas. Também nessa fase é possível a adição de novas funcionalidades e requisitos ao modelo da nova versão do software. O processo seguinte, re-implementação/re-implement, como o nome já bem diz, é onde são realizadas efetivamente as modificações na implementação, baseando-se na descrição do modelo recebido do processo anterior.

5.3 – CATEGORIAS

Estudos publicados [22] dividem a reengenharias em categorias de onde ocorrem em uma organização/empresa:

- Reengenharia de processos administrativos – reestruturação de processos ligados a administração da organização.
- Reengenharia de processos produtivos – modificação em ciclos de processos de produção de uma organização.
- Reengenharia de produtos – onde se encontra a reengenharia de software. É o exame, estudo e modificação de mecanismos internos ou funcionalidades de um sistema.

Diversos estudos qualificam e subdividem a reengenharia de software por diversos aspectos, atribuem diversas definições, que inclusive concordo. A leitura à qual melhor me adapto é a que subdivide a reengenharia pelo razão que levou ao processo de refatoração do sistema.

5.3.1 – MIGRAÇÃO DE PARADIGMA

Migração de um software legado, baseado em paradigma procedimental para o orientado a objetos. Em busca dos benefícios associados ao paradigma OO, tais como facilidade de manutenção e de evoluções futuras.

Um software legado é aquele que foi construído ao longo de anos, muitas vezes de forma não modularizada e utilizando técnicas hoje consideradas obsoletas. Mas que, por outro lado, executam tarefas úteis na organização, algumas até cruciais para o negócio.

Essa migração, assim como uma simples modificação num sistema desse tipo pode ser tarefa muito difícil, pois é necessário antes de qualquer mudança, fazer um estudo profundo das conseqüências e custos da modificação.

5.3.2 – APRIMORAMENTO DE SOFTWARE

Feito quando o objetivo da reengenharia é melhorar a qualidade do software, aprimorar algoritmos ou corrigir falhas, preservando as funcionalidades existentes e adicionando novas quando desejar. Reengenharia como forma de manutenção, quando o processo que é feito com o objetivo de realizar evolução e melhorias constantes.

5.3.3 – REFATORAÇÃO DE FUNCIONALIDADES

Quando em uma organização, mudamos alguns procedimentos, ou quando esta sofre um processo de reengenharia dos processos produtivos ou administrativos, somos forçados a executar reengenharia de software em seus sistemas.

5.4 – CONSIDERAÇÕES

No meu ponto de vista, ainda acrescento algumas considerações. A reengenharia como forma de manutenção constante, como um ciclo continuo de reconstrução e melhorias é um fator importantíssimo que influencia no tempo de vida útil de um software.

Creio que filosofar pode ajudar a chegarmos a conclusões mais brevemente. Considero que as técnicas de reengenharia devem ser pensadas e aplicadas desde o início de um projeto de desenvolvimento de software. Claro que isso pode soar um pouco contraditório e difícil de ser pensado. Isto é, quem irá pensar em uma melhoria de quando ainda estamos desenvolvendo algo? Mas é possível vislumbrar desde o início do desenvolvimento, ter uma documentação atualizada com a realidade atual do artefato. Dessa forma, poderíamos adiantar alguns processo descritos anteriormente neste documento, provendo uma base sólida para fundamentar a evolução para uma nova versão de nossos sistemas.

Contudo, em projetos de software gastamos muito tempo documentando códigos, documentando módulos, classes, criando modelos... E mesmo com muito esforço, a velocidade requerida em alguns ciclos de desenvolvimento do projeto acabam por deixar sua documentação obsoleta, atrasada em relação ao real estado do sistema. O que obriga ao gestor

do projeto a realocar pessoas de sua equipe, ou até contratar outros profissionais para suprir essa necessidade.

A solução para esses dilemas inspira várias propostas. Minha visão é a seguinte: utopicamente, uma documentação centrada no código, onde o código é a documentação principal do software. Aliada à geração automática de documentos, modelos e API's poderiam manter consistentes implementação e documentação.

É importante observar que de nada adianta se o código não foi criado seguindo as normas de boas práticas, e muito menos sem seguir as especificações das ferramentas utilizadas na geração dos documentos. Mais tarde, abordarei sobre técnicas, e ferramentas para gerar documentos desde baixos níveis à altos níveis de abstração.

6. Arquitetura de Software

6.1 – INTRODUÇÃO

Este capítulo é dedicado aos estudos sobre arquitetura de software que realizei durante minha graduação e principalmente durante o desenvolvimento do meu projeto final. Neste, farei uma "viagem" pelos modelos de arquitetura estudados, apontando suas características, vantagens e desvantagens.

Não apresentarei aqui nada de novo, nem nada de diferente da literatura existente sobre o assunto. Lembro que intenção aqui é compactar e documentar meus estudos [1][2][4][27] e esforços em aumentar meu conhecimento no assunto.

Antes de chegar aos modelos, apresentarei alguns breves tópicos que se fazem necessários, como historia motivos do surgimento desta linha de estudos e a importância da arquitetura de software.

6.2 – ORIGEM

A arquitetura de software foi fortemente influenciada pela arquitetura tradicional. Muitas das nossas preocupações e metodologias são resquícios do legado herdado desta área, inclusive os próprios nomes: arquitetura e padrão. O Dr. Cristopher Alexander foi o precursor nos estudos sobre o assunto. Iniciou uma nova forma de pensar em arquitetura.

“Ele acreditava que o estudo profundo das leis que governam uma determinada situação nos levaria a uma solução que seria aplicável aquela situação; bem como seria a solução de outras situações pelas mesmas leis [Alexander 77] [Alexander 75]”.

Com isso, desenvolveu formas de resolução de problemas que inspiram e baseiam diversos trabalhos voltados para softwares hoje em dia. Trazendo para o nosso vocabulário, poderíamos fazer a seguinte analogia:

O estudo dos requisitos de um software nos levaria a um estilo de arquitetura que seria aplicável no desenvolvimento daquele software; bem como seria aplicável ao desenvolvimento de softwares com os mesmos requisitos.

Assim, se tivéssemos uma base de conhecimento que relacionasse requisitos, estilos arquiteturais e fatos do desenvolvimento poderíamos saber, no desenvolvimento de um novo software, a partir do conjunto de requisitos, que estilo de arquitetura seria o mais apropriado princípio de reutilização.

6.3 – ARQUITETURA, FRAMEWORK E PADRÃO (REUSO/REUTILIZAÇÃO)

Mesmo com uma base extensa, a identificação ou escolha de um estilo arquitetural a partir dos requisitos solicitados para um software pode ser muito difícil. Em geral, nos dia de hoje, ainda despende muito da experiência do engenheiro de software.

Deixando de lado as dificuldades da escolha, acredito que uma vez selecionado, o estilo não pode nos deixar aleijados, nem mesmo engessados, dependentes do mesmo.

Entendo que reuso, que co-existe em outros graus, deve ser principalmente sobre experiência.

Um estilo de arquitetura apontado como apropriado ou um padrão indicado deve nos guiar, indicar o caminho, nos ajudar em tomadas de decisões, em função de permitir que a partir de uma experiência passada possamos retirar conhecimento necessário para poder avançar no desenvolvimento de novos projetos com bases mais sólidas.

Três formas de reuso têm destaque na engenharia de software, são elas:

- Arquitetura de software: mera estrutura comportamental dos elementos que formam um software, e suas restrições. [Shaw 96]
- Framework: códigos incompletos de algum domínio específico. [Buschmann 98]

- Padrões: é algo mais físico ; envolve codificação; pedaços de software que funcionam independentemente conforme concebidos em sua forma original; implementação orientada a objetos. [Gamma 95]

6.4 – CONCEITOS

6.4.1 - SUBSISTEMA

Sistema cuja operação não depende dos serviços fornecidos por outro subsistema. São compostos de módulos e têm interfaces definidas, as quais são utilizadas para a comunicação entre subsistemas.

6.4.2 - MÓDULO

Componente de sistema que fornece um ou mais serviços para outros módulos. Utiliza serviços de outros módulos e normalmente não é considerado um sistema independente.

6.4.3 – ACOPLAMENTO

O acoplamento diz respeito às informações de controle entre as entidades - inter - (módulos, camadas, objetos...). Quanto mais distantes umas das outras são as entidades, mais fraco é o acoplamento.

- Acoplamento Forte: informações compartilhadas
- Acoplamento Fraco: informações simples

De uma maneira geral, para produção de software, usamos acoplamento fraco. Desse modo torna mais fácil se desejarmos substituir algum componente, ficando atentos somente a sua interface e não a mais informações compartilhadas internamente como num acoplamento forte.

6.4.4 – COESÃO

A coesão diz respeito às informações dentro de uma entidade – intra. De maneira geral usamos a forte.

- Forte: funcional – a entidade trata de uma questão ou problema determinado.

- Fraco: aleatória – a entidade trata de partes de um problema, deixando outras partes para serem resolvidas por outras entidades.

6.5 – Representação Gráfica dos Modelos de Sistema

Existem diversos modos de representação gráfica de modelos de sistema. Abaixo estão listados os principais.

- Modelo estrutural estático: mostra os subsistemas ou componentes que devem ser desenvolvidos como unidades separadas.
- Modelo de processo dinâmico: mostra o sistema, como ele é organizado em processos run-time.
- Modelo de Interface: define os serviços oferecidos por cada subsistema em sua interface pública.
- Modelo de relacionamento: mostra relacionamentos como o fluxo de dados entre os subsistemas.

A estrutura e o estilo específico escolhido podem depender dos requisitos não funcionais do sistema. Os requisitos que exercem maior influência na escolha são:

- Desempenho: menor número de subsistemas com a menor comunicação possível entre esses subsistemas.
- Proteção: estrutura em camadas com proteção para as camadas mais internas e com alto nível de validação.
- Segurança: operações relacionadas com segurança num único subsistema; para evitar comunicação em excesso e redução de problemas com validação.
- Disponibilidade: arquitetura inclui componentes redundantes, para que seja possível substituir e atualizar componentes sem a interrupção do sistema.
- Facilidade de manutenção: componentes encapsulados de menor granulidade, para que possam ser rapidamente modificados. Produtores de dados separados dos consumidores e evitar estrutura de dados compartilhados.

6.6 – Modelos Arquiteturais

A primeira fase do projeto de arquitetura é um diagrama de blocos mostrando a decomposição do sistema em um conjunto de subsistemas que interagem entre si, possibilitando uma visão

abstrata do sistema. Neste tópico serão mostrados alguns dos principais modelos de estruturação de um sistema.

6.6.1 – MODELO DE REPOSITÓRIO

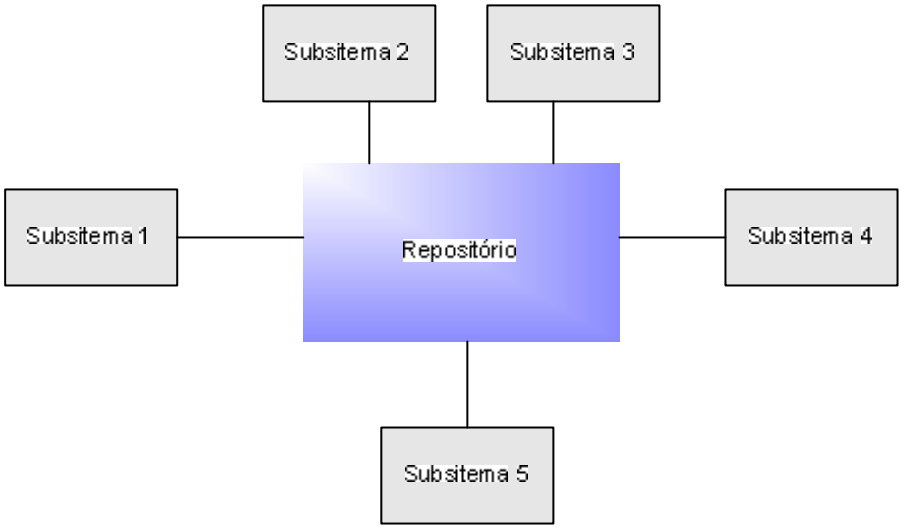
Para sistemas que são formados por subsistemas que precisam trocar informações, podemos optar por duas condições:

- cada subsistema mantém seu próprio banco de dados, e os dados são intercambiados com outros subsistemas através de troca de mensagens.
- todos os dados compartilhados são mantidos em um banco de dados central que pode ser acessado por todos os subsistemas.

No modelo de repositório é usado a segunda condição. Desta forma o sistema é visto contendo dois elementos distintos:

- blackboard: banco de dados central
- ks – knowledge source: componentes fontes de informação que executam operações no banco de dados

Uma particularidade do modelo repositório é que os componentes não podem se comunicar uns com os outros, exceto via repositório central – o blackboard. Que serve como um quadro de troca de mensagens entre objetos.



❒ Figura 6.1 – Modelo de Repositório

Esse modelo é adequado para aplicações em que os dados são gerados por um subsistema e utilizados por outro. Vantagens e desvantagens:

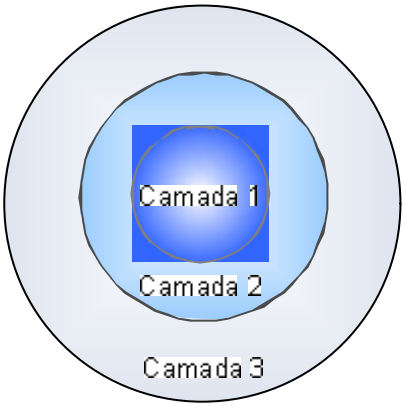
Vantagens	Desvantagens
Não há necessidade de transmitir os dados explicitamente de um sistema para outro	Evolução pode ser difícil, uma vez que um grande volume de informações é gerado de acordo com um modelo de dados estabelecido.
Os subsistemas que produzem dados não precisam saber como esses dados são utilizados pelo outro subsistema	Diferentes subsistemas podem ter diferentes requisitos para políticas de proteção, recuperação e backup. No modelo de repositório, a política imposta a todos os subsistemas é a mesma.
Facilidade para backup, segurança, controle de acesso e recuperação a partir de erros.	Pode ser difícil distribuir o repositório em uma serie de maquinas, podendo haver problemas de redundância e inconsistência de dados.
As ferramentas podem focalizar suas principais funções, já que o repositório teria seu subsistema controlador.	
Facilidade em integrar novas ferramentas, considerando que elas são compatíveis com o modelo de dados	

estabelecido.	

❒ Tabela 6.2 – Vantagens e desvantagens do modelo arquitetural - Repositório

6.6.2 – MODELO EM CAMADAS

Esse estilo é organizado de maneira hierárquica, onde cada camada tem o objetivo de prover serviços bem definidos para as camadas vizinhas. Organiza o sistema em camadas e modela a interface de subsistemas, possuindo para cada camada protocolos de comunicação com a mais próxima camada para cima e com a mais próxima para baixo. Dessa forma podemos facilmente alterar uma camada se conservarmos suas interfaces.



❒ Figura 6.3 – Modelo em Camadas

É difícil estruturar sistemas dessa maneira, pois dificulta e até impossibilita que alguns serviços implementados muito mais acima utilize alguns recursos básicos implementados numa camada muito mais inferior.

Vantagens	Desvantagens
Camadas podem ser reutilizadas, uma vez que possuem protocolos de comunicação para seus serviços.	Camadas ficam limitadas a serviços das próximas camadas.
Alterações são controladas, uma vez que alterações não podem atingir mais de outras duas camadas.	Aumento de processamento para seguir os protocolos de comunicação.

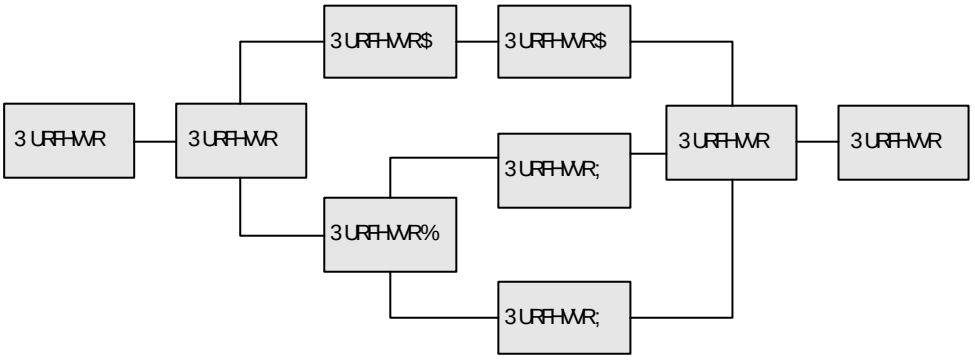
❒ Tabela 6.4 – Vantagens e desvantagens do modelo arquitetural – Camadas

6.6.3 – MODELO DE FLUXO DE DADOS

Neste modelo, decompomos os subsistemas em módulos. Cada módulo pode conter componentes que farão parte do sistema. Cada componente possui um conjunto de entradas e de saídas. As transformações funcionais em cada componente processam suas entradas e produzem saídas.

Quando as transformações são apresentadas como processo separado, chamamos esse modelo de tubos (pipe) ou filtro. Um pipe, ou pipeline é um conjunto de processamento de dados em serie, onde as saídas de um componente servem de entrada para outro componente. É possível introduzir buffers de entrada e saída entre os componentes consecutivos.

Em geral, esse modelo possui um grau de encapsulamento muito forte e cada pipe não tem conhecimento de interfaces com outros pipes.



❒ Figura 6.5 – Modelo de Fluxo de dados

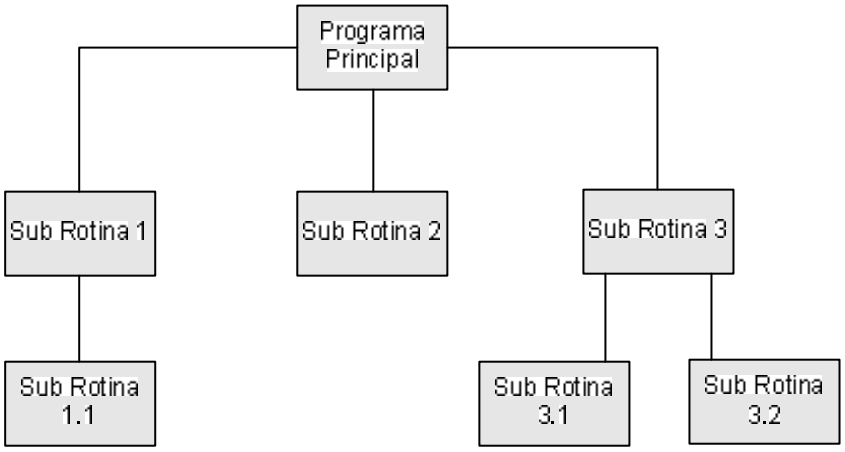
Vantagens	Desvantagens
Suporta reuso de transformações.	Necessidade de um formato comum para transferência de dados entre os processos.
A adição de novas transformações é simples.	Uma saída para transferência de dados é trabalhar com dados serializados,

É intuitiva.	mas isso aumenta o overhead (tempo) do sistema.
Simples de ser implementado.	

❒ Tabela 6.6 – Vantagens e desvantagens do modelo arquitetural – Fluxo de Dados

6.6.4 – MODELO DE SUBROTINAS

Muito utilizado nas décadas de 70 e 80. É caracterizada por um programa principal que chama diversos outros programas(subrotinas). Cada vez que um programa é chamado ele recebe o controle da execução e ao final devolve ao programa chamador.



❒ Figura 6.7 – Modelo de Subrotinas

Vantagens	Desvantagens
Facilidade de implementação de pequenos projetos.	Dificuldade de manutenção. Inviável para projetos grandes.
Facilidade no controle de execução.	Dificuldade de evolução. Cria softwares legados obsoletos.
Sem problemas de troca de informações em subsitemas.	Sem paralelismo de processamento.

- Tabela 6.8 – Vantagens e desvantagens do modelo arquitetural – Subrotinas

6.6.5 – MODELO ORIENTADO A OBJETOS

Neste modelo, pensamos em termos de "coisas" em vez de operações ou funções. O subsistema é decomposto em objetos que se comunicam. Cada objeto é um encapsulamento de informações e possui um estado e um conjunto definido de operações.

Os objetos pertencem a classes de objetos e chamam serviços fornecidos por outro objeto. A decomposição nesta forma diz respeito a classes de objetos, seus atributos e suas operações. Para um projeto seguindo o modelo orientado a objetos devem ser projetadas as classes de objetos e as relações entre as classes.

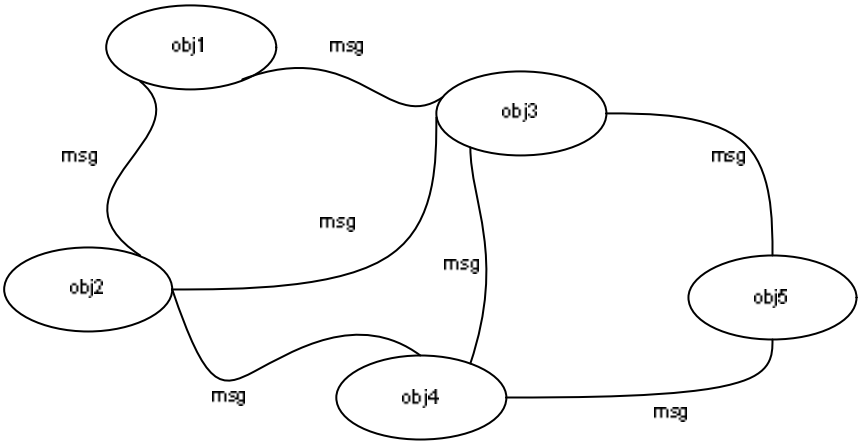
Classes de objetos podem ser organizadas em uma hierarquia de generalização ou de herança, que mostra o relacionamento entre as classes de objetos gerais ou mais específicos. Os relacionamentos entre objetos podem ser modelados descrevendo-se as associações entre classes de objetos.

Existem dois tipos de comunicação entre objetos:

- Síncrona: o objeto que faz uma requisição de serviço e espera que o objeto servidor conclua e o responda.
- Assíncrona: objetos podem executar simultaneamente com threads ou em objetos distribuídos.

Em alguns sistemas distribuídos, as comunicações com os objetos são implementadas diretamente como mensagens de texto. O objeto receptor analisa a mensagem, identifica o serviço e os dados associados e realiza o serviço requisitado.

É aconselhável que se use este modelo quando fazem parte do subsistema componentes que são inadequadamente acoplados.



Vantagens	Desvantagens
Fácil manutenção, objetos são modificados como entidades stand-alone.	Para qualquer modificação na interface de qualquer objeto, é necessário que se faça, antes, um estudo sobre os impactos que essa mudança pode causar.
Objetos podem ser compreendidos facilmente, uma vez que existe sempre um nítido mapeamento entre entidades do mundo real e seus objetos de controle do sistema.	Aumento no uso de memória para instanciar objetos.
Permitem relaxamento de visibilidade de um objeto em relação a outro.	
Objetos são reutilizáveis devido a seus níveis de encapsulamento	
Podemos facilmente trocar um objeto por um outro mantendo somente a sua interface.	

❒ Tabela 6.10 – Vantagens e desvantagens do modelo arquitetural – Orientado a Objetos

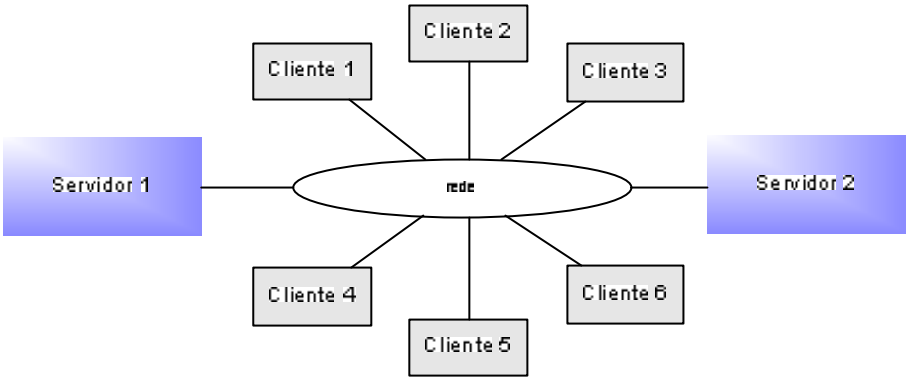
6.6.6 – MODELO CLIENTE-SERVIDOR

Este modelo possui os seguintes componentes:

- Um conjunto de servidores stand-alone para servir subsistemas

- Um conjunto de clientes que solicite serviços a outros subsistemas.
- Uma rede permite aos clientes acessar esses serviços.

É possível que haja várias instâncias de um programa cliente sendo executadas normalmente. Os clientes precisam saber os nomes de seus servidores, mas os servidores não precisam necessariamente saber o nome dos clientes.



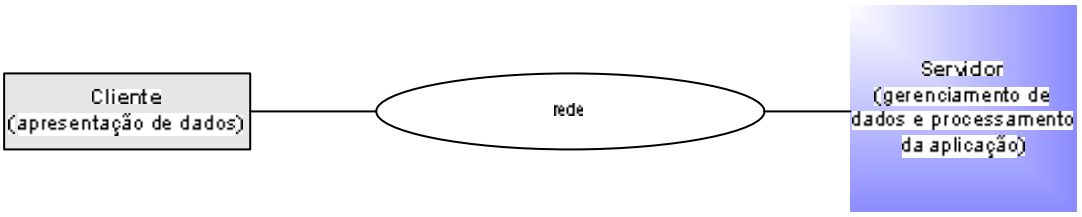
❒ Figura 6.11 – Modelo Cliente-Servidor de duas camadas – Cliente-Magro

Podemos entender o modelo de repositório como modelo cliente-servidor, onde o repositório seria o servidor e os subsistemas que se comunicam com o repositório seriam clientes.

Esse modelo pode ser aplicado de duas formas: o mais simples de duas camadas, e o de três camadas.

6.6.6.1 – Duas camadas - Modelo Cliente-Magro

Todo o processamento da aplicação e gerenciamento de dados é realizado no servidor. O cliente é responsável simplesmente por executar o software de apresentação.



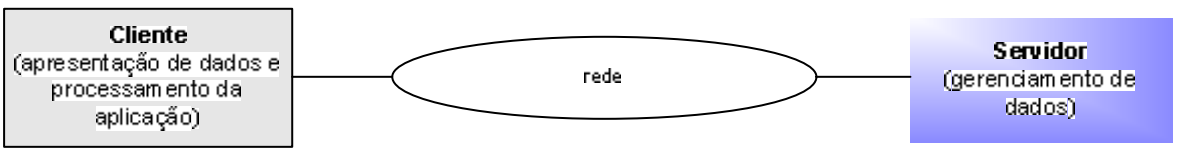
❒ Figura 6.12 – Modelo Cliente-Servidor de duas camadas – Cliente-Magro

Vantagens	Desvantagens
Aplicação no cliente fica bem mais leve. Só se encarrega de apresentar os dados recuperados do servidor.	Sobrecarrega o servidor. Exige um servidor de grande poder de processamento.
Centralização de processamento no servidor.	Desperdício de processamento do pc de usuários – desbalaciamento de carga.
Facilidade de manutenção, uma vez que a aplicação esta concentrada.	
Facilidade para atualizações.	

❒ Tabela 6.13 – Vantagens e desvantagens do modelo arquitetural – Cliente(magro)-Servidor

6.6.6.2 – Duas camadas - Modelo Cliente-Gordo

Nesse modelo o servidor é responsável somente pelo gerenciamento de dados. O software no cliente implementa a lógica da aplicação e as interações com o usuário do sistema.



❒ Figura 6.14 – Modelo Cliente-Servidor de duas camadas – Cliente-Gordo

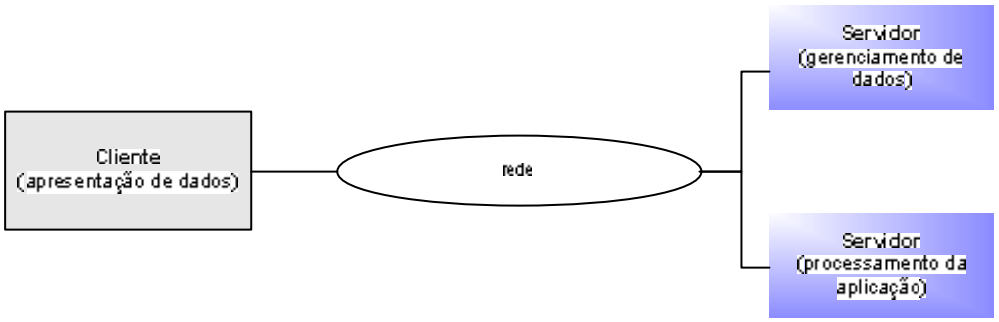
Vantagens	Desvantagens
Libera o servidor.	Qualquer mudança na aplicação tem

	que reinstalar em todas as maquinas.
Utiliza o potencial possivelmente ocioso do pc do usuário	

❒ Tabela 6.15 – Vantagens e desvantagens do modelo arquitetural – Cliente(magro)-Servidor

6.6.6.3 – Três camadas

No modelo cliente-servidor de três camadas, o servidor é dividido em dois, um para processamento de aplicações e outro para gerenciamento de dados.



❒ Figura 6.16 – Modelo Cliente-Servidor de duas camadas – Cliente-Gordo

Vantagens	Desvantagens
Divide a carga do servidor em dois servidores.	Necessidade de dois servidores potentes.
Mantém o processamento centralizado.	

❒ Tabela 6.17 – Vantagens e desvantagens do modelo arquitetural – Cliente(magro)-Servidor

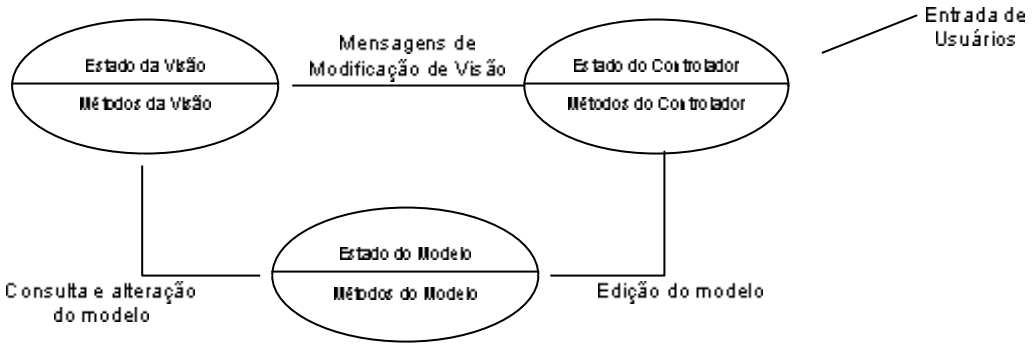
6.6.7 - MVC – MODEL VIEW CONTROL

O MVC(model – viewer - cntroller) tem as características do modelo orientado a objetos, com uma particularidade: Os dados a serem expostos estão encapsulados em um modelo de objeto. Cada modelo de objeto pode ter uma serie de objetos de visão separados, associados ele, em que cada visão é uma representação diferente de apresentação do modelo.

Cada visão tem um objeto controlador associado que manipula a entrada de dados dos usuários e a interação de dispositivos.

Encontrar a melhor maneira de apresentar as informações exige o conhecimento da experiência dos usuários das informações e da maneira pela qual eles utilizam o sistema.

A figura abaixo ilustra o modelo MVC apresentado:



Vantagens	Desvantagens
Os dados da aplicação podem ser apresentados de diferentes formas.	Dificuldade de tratamento de modificações nos dados do modelo da aplicação.

❒ Tabela 6.19 – Vantagens e desvantagens do MVC

Fatores importantes:

- O usuário esta interessado em informações precisas ou nas relações entre diferentes valores de dados?
- Com que rapidez os valores das informações são modificados? A mudança em um valor deve ser indicada imediatamente ao usuário?
- O usuário deve tomar alguma iniciativa em resposta a uma mudança nas informações?
- O usuário precisa interagir com as informações exibidas por meio de uma interface de manipulação direta?
- As informações a serem exibidas são textuais ou numéricas? Os valores relativos dos itens das informações são importantes?

7. Padrões de Programação

7.1 – INTRODUÇÃO

Durante minha formação acadêmica, cursei várias cadeiras, mas uma em especial mudou muito minha visão sobre programação. A cadeira ministrada pelo professor Arnt Von Staa, de Programação Modular(ou também chamada de Programação em Ponto Grande), provocou em mim o interesse em reuso e manter meus códigos documentados e inteligíveis para outros programadores e para mim mesmo.

A constante preocupação com padrões de codificação, fez com que os códigos produzidos por mim e meus colegas de convívio de faculdade, se tornasse cada vez mais característicos, diferentes dos de outros programadores.

Os padrões adotados aqui não tem como objetivo restringir o programador ou criar regras para dificultar a criação de códigos, mas sim, melhorar técnicas de programação visando vantagens como facilidade de manutenção, integridade, coerência, inteligibilidade, código claro e intuitivo...

Neste capítulo, descreverei brevemente os padrões de codificação e de documentação adotados. Não irei abordar nenhum conceito novo, nunca antes usado. Mas descreverei como utilizei os padrões existentes para alcançar maior legibilidade em meus códigos.

7.2 – PADRÕES ADOTADOS

7.2.1 - IDENTAÇÃO

- Use sempre espaços, nunca tabs, pois tabs são diferentes em cada editor de texto. O padrão adotado é de quatro espaços.
- Coloque as chaves nas linhas imediatamente seguintes a primeira linha de um bloco

```
//Em vez de:
function Date(){
   ...
}

//Faça:
function Date()
{
   ...
```

```
}
```

- Coloque espaços entre tokens e utilize parêntesis para evitar ambigüidade.

```
$fltTotal=2+3*4+1/2;                    //não indicado
$fltTotal = 2 + ( 3 * 4 ) + ( 1 / 2 );   //indicado
```

- Idente cada bloco internamente, isso aumenta a legibilidade do código.

```
$intLineCount = 0;
foreach( $arrDoc as $doc )
{
   $strDName    = $doc->Name();
   $strDAuthor  = $doc->Author();
   $strClass    = ( $intLineCount % 2 == 0 )? “Par” : “Impar”;
   $strLine     =  “<div class=” . $strClass .  “>”;
   $strLine     += $strDName . “-” . $strAuthor;
   $strLine     += “</div>”
   echo         ( strLine );
}
```

7.2.2 – NOMENCLATURA DE ARQUIVOS

- Cada classe está em um arquivo que contém apenas ela (salvo exceções como classes aninhadas e similares) .
- O nome da classe sempre começa com letra maiúscula, e nome do arquivo é o mesmo nome da classe.
- Em códigos orientados a ação, atribua um id para cada módulo, e.g. para um módulo de manipulação de listas atribua “LIS”.

7.2.3 – NOMENCLATURA DE CÓDIGO

- Procure dar nomes significativos às funções e variáveis, não declare uma função “foo” nem uma variável “x”.
- Ao nomear uma função/método inicie com letra maiúscula, e procure sempre nomear com ação+complemento. Para funções fora de uma classe, inicie com o nome com o id do módulo pertencente.

```
function  LIS_OrdenaLista( $lisLista ) ...
```

- Nomes de constantes em letras maiúscula.

- Usar prefixo de tipo de variáveis. Devem conter no início de seus nomes três letras que ajudam na identificação do tipo daquela variável.

```
$strNome          //para uma string
$arrNomes         //para uma array
$intNumUsuarios   //para um int
```

- Sempre abra e feche os blocos, mesmo que seja de apenas um comando.

```
//faça:
if( $bolPrint )
{
   echo( “Hello World” );
}

//em vez de:
if( $bolPrint ) echo( “Hello World” );
```

- Para reconhecer facilmente a visibilidade (private ou public) usar dois underscore “__” para variáveis privadas ou protegidas; um underscore “_” para variáveis encapsuladas (variáveis públicas mas que não é indicado a manipulação direta – manipulação via métodos); e nenhum underscore “” para as demais variáveis ao encapsulados (valores podem ser trocados diretamente acessando a variável).

7.2.4 – COMENTÁRIOS

- Comentar módulos, classes, funções, definições, e variáveis utilizando o padrão JavaDoc.
- Para blocos de códigos de pouca importância para documentação utilize o comentário “//” em vez de “/** ... */”

Detalhes sobre documentação no padrão JavaDoc serão discutidos no capítulo seguinte.

CAPÍTULO 8 – GERAÇÃO AUTOMÁTICA DE DOCUMENTAÇÃO

8.1 – INTRODUÇÃO

Neste projeto que desenvolvi no final de minha graduação, utilizei na base do projeto, PHP como linguagem principal. Em função disto, os exemplos e guias apresentados neste capítulo farão referência à geração automática de documentação de código em PHP.

A documentação principal de um projeto de desenvolvimento de software é o próprio código. Escrever uma boa documentação é essencial para o sucesso de qualquer projeto de software. Num ambiente de desenvolvimento de software, onde geralmente existem várias pessoas envolvidas num mesmo projeto, e às vezes mais de um desenvolvedor trabalhando em um mesmo código, em diferentes horários, manter coerente a documentação é muito difícil.

Gerada a documentação relacionada ao design e funcionamento do projeto em estágio de pré-desenvolvimento, quando iniciado de fato o desenvolvimento muitos detalhes que escapam da percepção do projetista fazem aumentar a distância entre o projeto inicial e o projeto final. O resultado desse processo pode ser crucial, desfalcando um projeto de documentação coerente, o que dificultaria depois num estagio de refatoramento, ou, no melhor cenário, penalizando um projeto com gastos excessivos de tempo do grupo envolvido para modificar a documentação original.

Em projetos open source, cada vez mais, os usuário finais do produto estão interessados em participar de alguma forma, utilizando e vendo o código, e até procurando e nos ajudando a resolver erros.

Uma documentação fundamentada no código seria uma solução interessante para manter um projeto consistente. Mas como seria possível gerar uma documentação que seria útil tanto para os projetistas, quanto como para os desenvolvedores, analistas, e todos interessados no funcionamento do produto final? A que custo isso seria gerado? Quanto da documentação poderia estar embutido num código? Como gerar diagramas e fluxogramas a partir deste conteúdo?

Neste parte veremos práticas e exemplos de geração de documentação a partir do código. Em seguida, veremos as regras fundamentais, e depois entraremos pelo “mundo” do diagramas e exploraremos algumas ADLs (Architecture Description Language – Linguagem de Descrição de Arquitetura).

8.2 – O PADRÃO JAVADOC/PHPDOC

Uma ferramenta para geração automática de documentação a partir do código bastante conhecida e difundida pelos usuários de Java é o JavaDoc. Hoje já existe também projetos para outras linguagens que se baseiam neste, por exemplo em PHP, existe o PHPDoc – PHPDocumentor (http://www.phpdoc.org) que utiliza tags e processos equivalentes ao JavaDoc.

O protocolo do JavaDoc se baseia em gerar a documentação, árvores de classes e descrição de variáveis e funções a partir de comentários inseridos no código. O output pode ser gerado inicialmente em HTML, PDF, entre outros formatos.

8.3 – PRÁTICAS DE BOM USO

Uma das melhores formas de gerar uma documentação boa é lendo o que você mesmo escreveu. Tente ler e usar, se for difícil a utilização, volte e reescreva a parte em dificuldade, remova o desnecessário e acrescente informações necessárias. Quando finalmente achar que está bom, passe o projeto para uma equipe ou usuário que não tenha conhecimento do projeto e anote sua opinião. Depois, quando revisitar o código verifique as anotações.

8.4 – SINTAXE

8.4.1 – DOCBLOCK

Um bloco de documentação (DocBlock) sempre começa com /** e termina com *, comentários iniciados por /* ou por // serão ignorados. O DocBlock deve vir imediatamente antes da declaração.

```
1   /**
2   * DocBlock de ME
3   */
4   define('ME', 2 );
5   /**
6   * DocBlock de Foo
7   */
8   function Foo( $param = ME )
9   {
10  }
```

Um DocBlock contém três partes:

- ShortDescription
- LongDescription (múltiplos espaços são convertidos para um espaço simples)
- Tags

```
1    /**
2     * Short description
3     *
4     * <p>Long description começa nesta linha e
5     * continua nesta outra linha até que haja uma linha em branco</p>
8     * Este texto é completamente ignorado, pois não esta entre tags p
9     * <p>Este é um novo parágrafo</p>
10     */
```

8.4.2 – TAGS HTML DENTRO DO DOCBLOCK

Tag	Descrição
b	bold
br	quebra de linha
i	itálico
kbd	keyboard
ul	lista não ordenada
ol	lista ordenada
li	item de lista
p	quando usado tem que fechar a tag, c.c. será considerado texto.
pré	preserva as quebras de linha e espaços, e assume que todas as tags são texto
code	para código php
samp	exemplos (non-php)
var	nome de variável

❒ Tabela 8.1 – Tags HTML dentro de um dockblock

8.4.4 – CONJUNTO DE TAGS PROPOSTAS

Adicional as tags do padrão JavaDoc, abaixo listo outras tags que posteriormente podem ajudar não só na geração de documentação da API das funções, mas também na geração de documentação arquitetural. È uma previsão ao que pode ser feito como geração de documentação. Introduzindo conceitos de ADL(Architecture Description Language) descritos seguindo o padrão JavaDoc.

Tag	Parâmetros	Entidade de destino
owner	1 – a quem pertence o código	arquivo fonte
version	1 – número da versão 2 – data da versão 3 – revisor da versão	arquivo fonte
license | copyright	1 – tipo de licença de uso do código	arquivo fonte
author	1 – id do autor 2 – nome do autor	arquivo fonte | classes | módulos | interfaces | funções | métodos
anotation	1 – anotação	*
todo	1 – tarefa a ser realizada	*
see	1 – indicação para outra parte de código	*
code	1 – código como comentários	*
example	1 – exemplo de uso	*
file	1 - nome do arquivo	arquivo fonte
package	1 – noma do pacote ao qual pertence	arquivo fonte
import	1 – nome do arquivo ou classe que importa	arquivo fonte
class	1 – nome da classe	classes
module	1 – id do módulo	módulos

	2 – nome do módulo	
interface	1 – nome da interface	interfaces
var | property	1 – nome 2 – descrição	variáveis
require	1 – nome 2 – tipo 3 – descrição	função | método
sets	1 – nome 2 – tipo 3 – descrição	função | método
param	1 – nome 2 – tipo 3 – descrição	função | método
use	1 – nome da classe ou função 2 – descrição	função | método
imlements	1 – nome da interface	classes
extends	1 – nome da classe	classes
adapter		classes
abstract | virtual		classes
static		classes | funções | métodos | variáveis
constructor		métodos
destructor		métodos
deprecated	1 – motivo ou indicação para um possível substituto	*
method | function	1 – nome da função ou método	métodos | funções
throws	1 – nome da exceção	métodos | funções |

	2 – descrição	classes
returns	1 – nome 2 – tipo 3 – descrição	métodos | funções
scenario	1 – título do cenários	métodos | funções | módulos | classes
actor	1 – nome do autor	métodos | funções | módulos | classes
lexicon	1 – nome do símbolo 2 – descrição de noção e impacto do símbolo	métodos | funções | módulos | classes
resource	1 – nome do recurso	métodos | funções | módulos | classes
exception	1 – descrição da exceção	métodos | funções | módulos | classes
geocontext	1 – descrição do contexto geográfico	métodos | funções | módulos | classes
tempcontext	1 – descrição do contexto temporal	métodos | funções | módulos | classes
precond	1 – descrição da precondição para que o cenário possa acontecer	métodos | funções | módulos | classes
goal	1 – descrição do objetivo do cenário	métodos | funções | módulos | classes
episode	1 – descrição de um episódio do cenário	métodos | funções | módulos | classes

❒ Tabela 8.2 – Tags propostas

Podemos observar que algumas das tags propostas tem mais que um parâmetro. Para facilitar a geração de documentação podemos adotar que só o ultimo parâmetro da tag pode ter mais de uma palavra. Outros parâmetros devem ter apenas uma palavra.

Podemos observar também que foi incluído neste conjunto de tags propostas tags relativas a descrição de cenários.

É importante lembrar que esta proposta não é o foco deste trabalho final de graduação. É simplesmente uma possibilidade futura que pode ser adicionada a este projeto.

CAPÍTULO 9 – ATIVIDADES REALIZADAS

Para o desenvolvimento deste projeto final de graduação procurei pesquisar diversas tecnologias, linguagens e técnicas de desenvolvimento. Estas pesquisas foram feitas a partir de livros, artigos, manuais divulgados na Internet e materiais fornecidos por professores nas cadeiras cursadas durante minha graduação na PUC. O objetivo dessas pesquisas era obter conhecimento técnico para, a partir delas, poder optar pela tecnologia que respondesse melhor às necessidades do projeto.

Como este projeto é voltado para web foram pesquisadas as mais usadas tecnologias para esse ambiente, entre elas o PHP, Java(Applet), JavaScript, Flash, ActionScript, HTML, XHTML, CSS, XML, DOM, SAX. Durante essa pesquisa pude testá-las e avaliá-las do ponto de vista teórico e prático.

Toda tecnologia precisa ser bem utilizada para que possamos extrair o máximo que ela pode nos oferecer. Por isso, além de estudar sobre cada tecnologia, também pesquisei sobre boas práticas e padrões de cada uma das que julguei mais indicadas para o projeto. Dessas pesquisas, reuni essas informações, e atento às recomendações encontradas e indicadas pelo W3C, para escrevi guias de utilizações simples, práticos e concisos sobre cada tecnologia.

O foco inicial é descrever uma arquitetura que desassocie o processamento do front-end em aplicações web do modelo da aplicação, permitindo assim que seja possível e fácil a disponibilização de diversos front-ends desenvolvidos em diferentes linguagens. Com isso, o cerne do projeto será em PHP, que ficará responsável pelo processamento principal, consequentemente, foi o mais estudado e testado e terá porcentagem maior neste documento.

Além dessas tecnologias e linguagens citadas, também trabalhei, e documentei essas experiências, com aplicações voltadas para web, como o Apache – servidor web, BulletProof - servidor FTP, MySQL - servidor de banco de dados. Ambos rodando nas plataformas Windows e Linux. Que em conjunto formam LAMP/WAMP (Linux/Windows, Apache, MySQL e PHP). Esta combinação é extremamente difundido no Brasil para a comunidade de desenvolvedores de software livre.

Todo o estudo, teste e metodologias, vantagens e desvantagens, e os motivos para as tomadas de decisão, guias de instalação e utilização, além de técnicas e boas práticas de uso serão apresentados nos capítulos a seguir.

È importante lembrar que a intenção principal aqui não é ensinar a manusear ferramentas ou programar ou algo similar. O foco principal aqui é a apresentação, para fins contextuais e documentais, do conteúdo da pesquisa realizada, bem como dos testes e experiências durante este projeto.

CAPÍTULO 10 – SERVIDOR WEB - APACHE

10.1 – VISÃO GERAL

O Apache HTTP Server é um projeto da Apache Software Foundation. Este projeto é um esforço de desenvolver e manter um servidor HTTP open-source que rode em modernos sistemas operacionais, incluindo UNIX e Windows NT. Outras preocupações latentes dos desenvolvedores deste projeto é com a segurança do servidor, a extensibilidade e eficiência.

O Apache foi o webserver mais popular no Internet desde abril 1996. Em um teste realizado em novembro 2005 pela Netcraft foi constatado que mais de 70% dos sites na Internet estão usando Apache.

10.2 – INSTALAÇÃO

- Baixe o instalador em: //httpd.apache.org/download.cgi e o execute.
- Escolha o local onde quer instalá-lo e siga deixando os valores defaut.
- Após a instalação, inicialize o servidor Apache.

10.3 – CONTROLE DE ACESSO ÀS PASTAS

O Apache permite que seja configurado diferentes regras de acesso para diferentes pastas. Isso pode ser configurado de duas formas distintas. Na primeira, para cada pasta exceção – pastas às quais queremos que tenha configuração diferente da default – precisamos inserir um bloco de diretivas no arquivo de configuração do Apache.

Uma outra forma é configurar o Apache para que cada pasta tenha possibilidade de escrever suas regras de acesso. Para isso, é necessário que o Apache seja notificado da seguinte forma:

- No arquivo de configuração do Apache – httpd.conf - modifique a linha:

```
AllowOverride None
```

para:

```
AllowOverride ALL
```

- Para cada pasta exceção crie dentro dela um arquivo .htaccess com conteúdo abaixo. Esse bloco de diretivas é idêntico para primeira opção de configuração, quando inserimos diretamente no arquivo de configuração.

```
<Location /folder-name>
        AuthType Basic
        AuthName zone-name
        AuthUserFile path-to-.passwd
        Require valid-user
</Location>
```

Para o arquivo que conterá os usuários e suas respectivas senhas - .passwd - você pode criar um para cada pasta exceção ou, se preferir, escolher um único local para um único arquivo de senhas e usuários que será referenciado por todos os .htaccess.

- Para criar um arquivo de usuários e senhas abra o prompt de comando, vá até a pasta principal do Apache e digite:

```
bin\htpasswd -c passwd <username>
```

- Para adicionar mais usuários digite:

```
bin\htpasswd passwd <username>
```

- Em seguida será pedida uma senha. Digite-a e aperte enter.

10.4 – DOMÍNIO VIRTUAL

O servidor tem um único endereço de IP, mas pode ter vários alias. Possibilitando que o servidor rode para dois, ou mais pontos na mesma maquina.

Existem dois casos possíveis utilizando domínio virtual, uma que é configurar os domínios virtuais no mesmo endereço IP e a outra forma é configurar utilizando endereços IP's diferentes. Abaixo seguem os dois formatos:

10.4.1 – DOMÍNIOS EM UM MESMO IP

- No arquivo de configuração procure pela linha:

VirtualHost example

- Logo abaixo insira o seguinte bloco

```
NameVirtualHost *:80

<VirtualHost *:80>
   ServerAdmin      webmaster@mestrecuca
```

```
   ServerName      www.mestrecuca
   DocumentRoot    /mestrecuca
   ErrorLog        logs/mestrecuca_log
   CustomLog       logs/mestrecuca_log common
</VirtualHost>
<VirtualHost *:80>
   ServerAdmin     webmaster@celevolution
   ServerName      www.celevolution
   DocumentRoot    /celevolution
   ErrorLog        logs/celevolution _log
   CustomLog       logs/celevolution _log common
</VirtualHost>

<VirtualHost *:80>
   ServerAdmin     webmaster@dingosoft
   ServerName      www.dingosoft
   DocumentRoot    /dingosoft
   ErrorLog        logs/dingosoft_log
   CustomLog       logs/dingosoft_log common
</VirtualHost>
```

O * pode ser subtituido pelo ip, mas para máquinas que têm IP dinâmico isso não é indicado.

10.4.2 – DOMÍNIOS EM IP'S DIFERENTES

- No arquivo de configuração procure pela linha:

VirtualHost example

- Logo abaixo insira o seguinte bloco

```
<VirtualHost ...40:80>
ServerAdmin      webmaster@mestrecuca
ServerName  www.mestrecuca
DocumentRoot     /mestrecuca
ErrorLog    logs/mestrecuca_log
CustomLog   logs/mestrecuca_log common
</VirtualHost>

<VirtualHost ...50:80>
ServerAdmin      webmaster@celevolution
ServerName  www.celevolution
DocumentRoot     /celevolution
ErrorLog    logs/celevolution _log
CustomLog   logs/celevolution _log common
</VirtualHost>

<VirtualHost ...60:80>
```

```
ServerAdmin          webmaster@dingosoft
ServerName  www.dingosoft
DocumentRoot         /dingosoft
ErrorLog       logs/dingosoft_log
CustomLog    logs/dingosoft_log common
</VirtualHost>
```

10.4.3 – USANDO PROXY

```
<VirtualHost *:*>
ProxyPreserveHost  On
ProxyPass              / http://139.82.24.196/
ProxyPassReverse   / http://139.82.24.196/
ServerName             hostname.example.com
</VirtualHost>
```

CAPÍTULO 11 – SUBVERSION REPOSITORY - SVN

11.1 – VISÃO GERAL

O subversion é um repositório que possui um controle de versão. Permite que se trabalhe com diversas versões de arquivos organizados em um diretório localizados local ou remotamente. Nele são mantidos versões antigas e os logs de quando e de quem manipulou os arquivos. É muito utilizado em composições colaborativas.

O objetivo do projeto do Subversion é construir um sistema de controle da versão para ser um substituto moderno do CVS – controlador de versões muito utilizado em comunidades open-source. O software é liberado sob uma licença Apache/BSD-style open-source – GPL.

Utiliza uma arquitetura cliente-servidor: um servidor armazena a(s) versão(ões) atuais do projeto e seu histórico, e os clientes se conectam a esse servidor para obter uma cópia completa do projeto, trabalhar nessa cópia e então devolver suas modificações.

11.2 – INSTALAÇÃO

Um dos servidores mais flexíveis para instalar o Subversion, de fácil instalação, que oferecem muitos benefícios é o Apache. O Subversion baseado no servidor Apache utiliza o protocolo WebDAV, que também é utilizado por outros programas. Possibilita que montemos um repositório como uma pasta na web e depois acessemos o como uma pasta do sistema de arquivo. Podemos também acessar o repositório com um browser, ou ainda podemos acessar via cliente SVN.

- Baixe os arquivos em http://subversion.tigris.org/ e execute o arquivo de instalação.
- Copie os seguintes arquivos para a pasta de módulos do Apache:
    - "...\httpd\mod-dav-svn.so"
    - "...\mod-authz-svn.so"
    - "...\bin\libdb42.so"
- Edite o arquivo de configuração do Apache (httpd.conf), descomente a linha, retirando o caracter "#" do inicio da linha que contém:

```
LoadModule dav_fs_module modules/mod_dav_fs.so.
```

- Adicione as linhas no final da sessão de carregamento de módulos:

```
LoadModule dav_svn_module modules/mod_dav_svn.so
LoadModule authz_svn_modules modules/authz_svn_modules.so
```

11.3 – CONFIGURAÇÃO

- Crie uma pasta que será a pasta principal dos repositórios. Para os exemplos a seguir será usado:

```
C:\WebArea\SVN
```

- Crie os repositórios que desejar criando pastas dentro da pasta raiz. Neste exemplo foi criado uma de nome "Work", assim:

```
C:\WebArea\SVN\Work
```

- Configure a pasta de repositório criada como um repositório digitando no prompt:

```
C:\Program Files\SVN\svnadmin create --fs-type fsfs C:\WebArea\SVN\Work
```

Esse processo pode ser feito usando um cliente SVN.

- No final do arquivo httpd.conf de configuração do Apache adicione as linhas:

```
<Location /svn>
        DAV svn
        SVNParentPath C:\WebArea\SVN
        AuthType Basic
        AuthName "Subversion Repository"
        AuthUserFile .passwd
        AuthzSVNAccessFile svnacessfile
        Require valid-user
</Location>
```

Isso configurará o Apache e deixará disponível o repositório em

```
http://MyServer/svn/
```

- Para que todos tenham acesso com permissão de leitura ao repositório, mas que só usuários autorizados tenham permissão de escrita troque a linha:

```
Require valid-user
```

por:

```
<LimitExcept GET PROPFIND OPITIONS REPORT>
        Require valid-user
</LimitExcept>
```

Para ter um controle melhor para cada usuário, com regras e grupos de usuários, é indicado o uso de um arquivo de controle de acesso.

11.4 – CONTROLE DE ACESSO

- Crie o arquivo .passwd com as senhas de acesso do usuários ao subversion. Siga os passos de criação do arquivo de usuários e senhas descrito no capítulo referente ao Apache. É necessário restartar o Apache para que funcione qualquer mudança efetuada.
- Crie um arquivo de controle de usuários e construa suas regras baseando-se no exemplo a seguir:

```
[groups]
developers          = user1, user2
docs                = user3, user4

#Allow everyone read access
[/]
*                   = r

#Allow all developers complete access
@developers         = rw

#Give the documentor user write access to "documentation" folder
[/Work/documentation]
@docs               = rw
```

- Salve esse arquivo no endereço setado para o SVNParentPath.

11.5 – INDEX DE REPOSITÓRIOS

- Para configurar um arquivo index.html que liste os repositórios no SVNParentPath insira as seguintes linhas no bloco <Location /svn>:

```
<Location /svn>
        RewriteEngine on
        RewriteRule ^/svn$ /svn_index.php [PT]
        RewriteRule ^/svn/$ /svn_index.php [PT]
        RewriteRule ^/svn/index.html$ /svn_index.php [PT]
<Location>
```

- Crie o arquivo que lista os repositórios e salve como svn_index.php e insira o seguinte conteúdo:

```
<html>
<head>
	<title>Subversion Repositories</title>
</head>
<body>
	<h2>My Subversion Repositories</h2>
	<p>
	<?php
	$SVNParentPath    = "C:/WebArea/SVN";
	$SVNParentURL     = "/svn";
	$home             = opendir( $SVNParentPath );
	if( !$home )
	{
		echo( "Can not open SVN home folder" );
	}
	while( $dir = readdir( $home ) )
	{
		$SVNDir           = $SVNParentPath . "/" . $dir;
		$SVNDBDir         = $SVNDir . "/db";
		$SVNFsTypeFile    = $SVNDBDir . "/fs-type";
		if( is_dir( $SVNDir ) && is_dir( $SVNDBDir ) )
		{
			echo( "<a href=\"" . $SVNParentURL . "/" . $dir . "\">" . $dir .
"</a> ";
			if( file_exists( $SVNFsTypeFile ) )
			{
				$handle = fopen( $SVNFsTypeFile , "r" );
				$buffer = fgets( $handle , 4096 );
				fclose( $handle );
				$buffer = chop( $buffer );
				if( strcmp( $buffer , "fsfs" ) == 0 )
				{
					echo( "(FSFS) <br /> \n" );
				}
				else
				{
					echo( "(BDB) <br /> \n" );
				}
			}
			else
			{
				echo( "(BDB) <br /> \n" );
			}
		}
		closedir( $home );
	}
```

```
        ?>
        <\p>
<\body>
<\html>
```

11.6 – CONTROLE DE SEGURANÇA COM SSL

O apache por default não prove suporte à SSL, mas facilmente é possível baixar e instalar o módulo referente.

- Baixe os arquivos em http://hunter.campbus.com/ e descomprima o arquivo zip.
- Copie o arquivo mod_ssl.so para a pasta de módulos do Apache.
- Copie o arquivo openssl.exe para a pasta bin também do Apache.
- Copie o arquivo de conf/ssl.conf para a pasta conf do Apache.
- Abra o arquivo ssl.conf e comente, colocando o caracter "#" no inicio das linhas a seguir:

```
DocumentRoot "C:/apache/htdocs"
ServerName www.example.com:443
ServerAdmin you@example.com
ErrorLog logs/error_log
TransferLog logs/access_log
```

- Troque a linha:

```
SSLCertificateFile conf/ssl.crt/server.crt
```

por:

```
SSLCertificateFile conf/ssl/my-server.cert
```

- Troque também a linha:

```
SSLCertificateKeyFile conf/ssl.key/server.key
```

por:

```
SSLCertificateKeyFile conf/ssl/my-serven.key
```

- E a linha:

```
SSLMutex file:logs/ssl_mutex
```

por:

```
SSLMutex default
```

- Delete as linhas:

```
<IfDefine SSL>
</IfDefine>
```

- No arquivo de configuração do Apache (httpd.conf) descomente a linha:

```
#LoadModulo ssl_module modules/mod_ssl.so
```

- Baixe o arquivo de configuração do Openssl em http://tud.at/programm/openssl.cnf e salve o arquivo em:

```
conf/openssl.cnf
```

- Crie um certificado, no prompt de comando, na pasta principal do Apache, digitando:

```
bin\openssl req -config\ conf\openssl.cnf -new -out my-server.csr
```

- Você será perguntado pela senha. Responda e depois prossiga e excute os comandos:

```
bin\openssl rsa -in privkey.em -out my-server.key
bin\openssl x509 -in my-server.csr -out my-server.cert -req -signkey my-server.key -days 4000
```

- Isso irá criar um certificado que expirará em 4000 dias. E finalmente entre com o comando:

```
bin/openssl x509 -in my-server.cert -out my-server.der.crt -outform DER
```

- Copie os arquivos gerados listados abaixo na pasta “...conf/ssl”
    - my-server.der.crt
    - my-server.csr
    - my-server.key
    - .rnd
    - privkey.pem
    - my-server.cert
- Restart o Apache e teste a nova configuração.

CAPÍTULO 12 – HIPERTEXT PREPROCESSOR - PHP

12.1 – INTRODUÇÃO

PHP é uma poderosa linguagem de programação interpretada para construção de sites dinâmicos e interativos. É uma linguagem server-side, i.e. processa as informações no lado do servidor, é open-source, multi-plataforma e bastante difundida no desenvolvimento web.

O nome PHP é um acrônimo recursivo para "Hypertext PreProcessor". A linguagem surgiu por volta de 1994, como um subconjunto de scripts Perl criados por Rasmus Lerdof. Em 1997, com adições dos programadores israelitas do Technion, o Instituto Israelita de Tecnologia, Zeev Suraki e Andi Gutmans, foi disponibilizado a versão três do PHP – a primeira versão estável e parecida com a atual. Hoje a versão atual é a cinco.

Por ser uma linguagem server-side, o PHP está apto a fazer qualquer coisa que um outro programa CGI pode fazer, como coletar dados de formulários, gerar páginas dinâmicas, enviar cookies, abrir seções...

A sintaxe é muito semelhante à Perl e C/C++. O código PHP pode ser embutido no código HTML. É freqüentemente usado junto com Apache (web server), mas tem suporte na maioria dos servidores web. Também suporta ISAPI e pode ser usado com o IIS da Microsoft em Windows. E pode ser instalado em diversos sistemas operacionais. Portanto, com o PHP temos a liberdade de escolher o sistema operacional e o servidor web.

Utilizando PHP temos a liberdade também de escolher o tipo de programação que iremos adotar. A partir da versão cinco, inclusive, PHP conta com um excelente suporte à orientação a objetos. Trata-se de uma linguagem híbrida, que dá suporte a orientação a objetos assim como à programação orientada a ações (programação procedimental). É extremamente modularizada, o que a torna ideal para instalação e uso em servidores web.

O PHP pode ser rodado de três formas diferentes, a partir dos binários CGI e CLI ou pelos módulos de servidor web. Os módulos de servidor têm desempenho significativamente melhor e funcionalidades adicionais comparadas com o binário CGI. A versão CLI é destinada para permitir usar o PHP para scripts da linha de comando.

12.2 – UTILIZAÇÃO

Um script PHP pode ser editado facilmente em qualquer editor de texto, como o notepad. Contudo existem vários editores que fornecem alguns recursos a mais, como por exemplo o Scite, que colore o código.

Todo código PHP deve ser salvos em arquivos ".php" e todo trecho de código PHP deve estar entre tags "<?php" e "?>" ou "<?" e "?>", ou ainda entre "<?=" e "?>" para códigos "in line", pois só assim o servidor web reconhecerá que é um script que deve ser interpretado.

O PHP pode estar embutido, aparecendo em conjunto com qualquer linguagem de marcação. Por exemplo, se estivermos usando HTML, podemos entremear as marcações com trechos em PHP. Ainda podemos alternar inúmeras vezes entre códigos e marcações. O código abaixo mostra um exemplo desse uso do PHP:

```
<html>
<body>
<?
$date  = date( "l dS of F Y h:i:s A" );
?>
<div> Rio de Janeiro <?= $date ?> </div>
<div>Bem vindo ao MestreCuca</div>
</body>
</html>
```

Contudo, com PHP não estamos limitados a gerar somente HTML ou simples textos. Mas também podemos gerar dinamicamente imagens, XHTML e outros formatos XML, arquivos PDF e animações Flash, tudo on the fly..

O PHP é extremamente útil em recursos de processamento de texto. Fornece suporte à expressões regulares, suporta os padrões SAX e DOM para processamento de XML, e também dá suporte a extensões XSL, para transformação de documentos XML. Prove em sua extensa biblioteca, módulos para abordagens de e-commerce, ferramentas de busca, conversão, tradução, entre outros recursos.

Além disso, o PHP lida com facilidade com servidores de banco de dados, como MySQL, PostgreSQL, Microsoft SQL Server e Oracle. Provê suporte aos protocolos: IMAP, SNMP, NNTP, POP3, HTTP, LDAP, XML-RPC, SOAP. É possível abrir sockets e interagir com outros protocolos. E as bibliotecas de terceiros expandem ainda mais estas funcionalidades.

Falando um pouco de comunicação, o PHP se comunica com outras linguagens de programação web. Além de se comunicar muito bem com Java, instancia objetos Java e os utiliza de forma transparente como objetos PHP local ou remotamente, utilizando extensões CORBA.

A intenção deste capítulo é introduzir e esclarecer características da linguagem, bem como mostrar as possibilidades de uso do PHP e um pouco de seu poder. Para o estudo da linguagem uma boa opção é a própria documentação disponibilizada em sua página oficial (http://www.php.net). No mesmo site podemos encontrar também a referência completa para as funções e os módulos de PHP que podem ser consultadas online de maneira simples e rápida.

12.3 – INSTALAÇÃO

Com a distribuição original é possível baixar kits de instalação do PHP, já em conjunto com o MySQL e Apache embarcados em Windows. Porém é indicado que a instalação seja feita manualmente. Isso permitirá um melhor entendimento do sistema e facilitará a atualização, manutenção e adição de novos módulos.

Neste tópico estão os passos de instalação manual do PHP e a configuração do servidor web – Apache, e do servidor de banco de dados – MySQL.

Grande parte dos manuais de instalação disponíveis na internet sugerem mover os arquivos .ini e dll para a pasta “system” do Windows. Porém isso pode tornar muito difícil futuras atualizações ou manutenções.

A melhor forma de fazer a instalação é manter todos os arquivos em uma só pasta e indicar ao sistema aonde estão localizados esses arquivos.

- primeiro faça o download dos arquivos em http://www.php.net/downloads.php
- extraia os arquivos na pasta desejada. Não é indicado o uso de diretórios que possuam espaços no caminho, e.g. C:/Arquivos de Programas/PHP5, pois alguns servidores podem não aceitar. Para os exemplo a seguir utilizaremos o diretório: C:/WebArea/PHP5

12.4 – CONFIGURAÇÃO DO PHP

Se você estiver instalando a versão 4 do PHP, antes de prosseguir, você deve manter todos os arquivos localizados nas pastas dll e sapi para a pasta principal do PHP. Para os exemplos a seguir utilizarei a notação da instalação do PHP5.

Os binários CGI e CLI, e os módulos de servidor web todos necessitam encontrar os arquivos dll para funcionarem corretamente. Para assegurar que esses arquivos serão encontrados temos três opções:

- copiar os arquivos para o diretório system do windows
- copiar os arquivos para o diretório do servidor web
- adicionar o diretório do PHP na variável de ambiente PATH.

Esta última opção é a mais indicada a ser adotada, para facilitar atualização e manutenção posteriores. Para isto:

- sete na variável de ambiente o caminho "C:/WebArea/PHP5/"

O passo seguinte é criar e editar um arquivo php.ini. No diretório principal do PHP:

- copie o arquivo php.ini-recomended e renomeie-o para php.ini.
- abra o arquivo com um editor de texto qualquer.
- sete a variável de ambiente que indicara aonde está localizado o arquivo de inicialização do PHP, PHPRC, com o caminho para o arquivo php.ini, i.e. "C:/WebArea/PHP5/".

Se você estiver usando NTFS no Windows NT, 2000, XP ou 2003, assegure-se que o usuário executando o servidor web tem permissões de leitura ao seu arquivo php.ini.

12.5 – EXTENSÕES DO PHP

Existem duas formas de carregar as extensões do PHP. Uma é a partir do arquivo de configuração php.ini – desta forma as dll's serão carregadas quando o PHP for iniciado. A outra forma é carregar dinamicamente as dll's com o comando dl().

- primeiro baixe as extensões no site no php.net.
- extraia os arquivos na pasta C:/WebArea/PHP5/extensions.

- No arquivo php.ini procure pela declaração extension_dir e sete o caminho para a pasta de extensões, não esqueça da última barra, i.e. extension_dir = "C:/WebArea/PHP5/extensions/"
- Habilite as extensões que desejar que o PHP carregue ao iniciar descomentando as respectivas linhas, retirando de seu inicio o caractere ";".

O bloco com as declarações de extensões está logo abaixo da declaração extension_dir.

12.6 – CONFIGURAÇÃO DO APACHE

Na pasta onde foi instalado o Apache abra o arquivo conf/http.conf num editor de texto.

- Localize a linha:

```
#LoadModule unique_id_module/mod_unique_id.so
```

- Adicione logo abaixo desta linha a seguinte linha:

```
LoadModule php5_module "C:/WebArea/PHP5/php5apache.dll"
```

- Localize a linha:

```
AddModule mod_setenvif.c
```

- Adicione logo abaixo:

```
AddModule mod_php5.c
```

- Localize:

```
AddType application/x-tar .tgz
```

- e adicione:

```
AddType application/x-httpd-php .php
AddType application/x-http-php-source .phps
```

- localize:

```
<IfModule mod_dir.c>
DirectoryIndex index.html
<IfModule>
```

- e adicione após index.html:

```
index.php main.php
```

- Para efetuar um teste, crie um arquivo chamado phpinfo.php com o conteúdo:

```
<?php
        phpinfo();
?>
```

- Reinicie o Apache e acesse o arquivo para conferir se as novas configurações foram corretamente setadas.

CAPÍTULO 13 – LINGUAGEM DE MARCAÇÃO

13.1 – VISÃO GERAL

Em informática, uma linguagem de marcação é um conjunto de regras sintáticas que aplicados a um texto ou a dados, adicionam informações particulares sobre esse conteúdo. Usada para descrever a aparência dos dados e a semântica do documento.

Sua finalidade fundamental é descrever documentos utilizando um padrão – formatá-los – para que seja possível o intercambio de informações.

Tradicionalmente as linguagens de marcação são descritas em formato ASCII.

13.2 – TIPOS DE ELEMENTOS DE MARCAÇÃO

13.2.1 – MARCAÇÃO ESTRUTURAL

Descreve o propósito do conteúdo, somente diz respeito a estrutura do documento, não indica a forma como será apresentada, por exemplo, em um texto descreve somente o que é um titulo e o que um texto normal, mas não indica como deverão ser desenhados na tela.

```
<h2>Minha Receitas</h2>
```

13.2.2 – MARCAÇÃO DE APRESENTAÇÃO

Marcações destinadas ao design do conteúdo. Descreve a aparência do conteúdo, posicionamento, tamanho...

```
<b>Bolo de chocolate</b>
```

O uso das marcações de apresentação é inapropriado. A melhor forma de tratar a apresentação de um documento é através de CSS e XSL. Falarei destas mais tarde, mas para ser breve: o CSS é responsável pelo tratamento do estilo da apresentação. E o XSL é responsável pela estrutura de apresentação. O CSS provê a separação da apresentação da estrutura + conteúdo.

13.2.3 – MARCAÇÃO HIPERTEXTUAL

Conecta o documento a outros conteúdos e a partes dele mesmo, cria elos entre conteúdos.

```
<a href="http://mestrecuca.org/">MestreCuca</a>
```

13.3 – STANDARD GENERALIZED MARKUP LANGUAGE – SGML

- Padrão internacional, definido em 1986, para formato de texto e documentos
- Popular em organizações que precisam criar e gerenciar grandes volumes de documentos
- Padrão adotado, entre outros, pela indústria aeroespacial, automotiva, de telecomunicações e de software.
- Compatível com padrões atuais e futuros
- Não proprietário - não se torna obsoleto
- Formato ASCII, liberta o conteúdo de produtos e softwares
- SGML oferece a possibilidade da criação de qualquer conjunto de tags
- O padrão comum (DTD - Document Type Definition) cobre o conjunto de tags a serem adotadas e a gramática de uma linguagem de marcação (markup language)

Problemas:

- Complexidade - torna difícil seu suporte por browsers web
- Inexistência de estilos amplamente difundidos
- Para uso na web é convertido para HTML perdendo muito da inteligência do documento original, impedindo a sua reutilização, intercâmbio e automação

13.4 – HYPERTEXT MARKUP LANGUAGE – HTML

Projetado inicialmente para formatação de documentos a serem compartilhados e intercambiados pela Web. É uma linguagem de marcação simples para criação de documentos hipertextos contendo imagens e outros elementos.

Originalmente era uma simplificação da SGML. Foi definida com regras sintáticas frouxas, que não exigiam do programador muita preocupação. Sua simplicidade e sintaxe simples, o padrão logo ficou popular. Com o tempo, evoluiu para aceitar outro tipo de conteúdo, como por exemplo, imagem, som e até vídeo. Se transformando na linguagem preferida para descrição de web sites.

Essa sintaxe frouxa levou os desenvolvedores de leitores HTML - browser's - a aceitarem padrões diferentes. Assim, páginas descritas para um browser não apareciam corretamente em outros. Criando então um ambiente de total instabilidade para troca de informações.

Razões da Popularidade do Padrão:

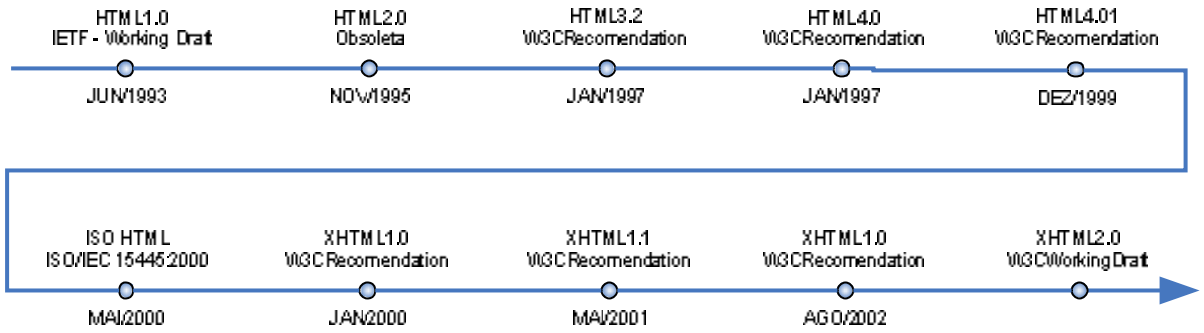
- Extremamente simples
- Estilo próprio para formatação de documentos
- Ligações entre hipertexto criadas facilmente
- Suporte a formulários
- Interação homem x máquina

Limitações:

- Estrutura limitada e simples
- Número fixo de tags
- Difícil reutilização da informação
- Inadequado para organização das informações
- Padrão modificado constantemente - adequação aos novos padrões requer um grande esforço
- Não oferece a funcionalidade requerida pelo comércio eletrônico

Simplicidade, rápida evolução e a enorme aceitação foram fatores que impulsionaram o "boom" da internet, no final dos anos 90. Mas também impulsionaram um crescimento desordenado.

Depois da apadrinhamento do padrão pelo W3C, a linguagem evolui para outros níveis que fortaleceram sua sintaxe e permitiram a definição de um padrão que funcionasse em qualquer browser. Contudo, até hoje programadores que seguem os padrões restritos são minoria.



❒ Figura 13.1 – Evolução da HTML

- HTML1.0 - http://www.w3.org/MarkUp/draft-ietf-iiir-html-01.txt
- HTML2.0 - http://www.ietf.org/rfc/rfc1866.txt
- HTML3.2 - http://www.w3.org/TR/REC-html32
- HTML4.0 – http://www.w3.org/TR/REC-html40-971218/
- HTML4.01 - http://www.w3.org/TR/html401
- ISO HTML - http://www.purl.org/NET/ISO+IEC.15445/15445.html
- XHTML1.0 - http://www.w3.org/TR/xhtml1/
- XHTML1.1 - http://www.w3.org/TR/xhtml11/
- XHTML2.0 - http://www.w3.org/TR/xhtml2/

XHTML é a uma evolução do HTML mais de sintaxe restrito, mais tarde falarei sobre XHTML em um capítulo onde será apresentada sua sintaxe, utilizações e boas práticas.

13.5 – DOCUMENT TYPE DEFINITION – DTD

13.5.1 – VISÃO GERAL

Document Type Definition, é um mecanismo para descrever a estrutura de um documento, nele são definidos os tipos encontrados no documento associado. Boa parte das linguagem de marcação com tipos restritos, se não todas, tem um DTD associado, por exemplo, HTML e XHTML.

Podemos também criar DTD's, e associá-los à documentos criados por nós mesmos. Por exemplo, ao criarmos um documento XML, podemos criar elementos simples, elementos aninhados, elementos de tipos diferentes, elementos vazios... E isso tudo pode ser pré-determinado com um DTD.

O DTD é útil durante a fase de criação do documento, mas em geral, não faz sentido distribuir o DTD com o documento XML, por exemplo. Com ele podemos verificar se um documento XML é válido ou não, em contrapartida, estaríamos enviando informações desnecessárias, logo gastando largura de banda, se enviássemos a um cliente que não faça o teste de validação

13.5.2 – POR QUE USAR DTD?

A DTD permite definir:

- quais elementos o documento pode ter
- quais atributos um elemento por ter
- o número de ocorrências dos elementos
- que sub-elementos um elemento pode ter

Problemas com DTD

- Sintaxe não XML!
- Poucos tipos de dados (PCDATA, ID, ...)
- Limitações na definição de cardinalidade (0,1,n)

13.5.3 – PRA QUE VALIDAR?

Poder validar um documento é uma característica muito útil. Um documento de entrada de um software validado é garantia de que está de acordo com nossa aplicação. Imagine quais problemas teria um simples programa que requer um arquivo de entrada e esse arquivo de entrada e feito de forma errada.

No contexto web, ter uma página, ou um conteúdo XML válido, por exemplo, é garantia de que qualquer browser saberá apresentar ler esse arquivo. Mas existem diversas outras boas razões para validar um documento:

- Dirigir a edição de um arquivo XML
- Simplificar a interpretação (por pessoas e máquinas)

13.5.4 – SINTAXE

13.5.4.1 – Elemento

È um mecanismo para descrever todo tipo de objetos que aparece no documento. Um elemento pode ser uma lista de elementos.

- O nome deve começar com um letra ou pelos caracteres "_" e ":";
- Não podem começar com a string "xml";
- Podemos usar "." e "-" para os demais caracteres;
- Não pode existir espaços no nome;
- Não usar ",", exceto para namespaces e em listas como conector;
- Podemos usar "|" para indicar que só um dos elementos podem aparecer;

```
<!ELEMENT entry (name, address*, tel*, email*)>
<!ELEMNT element_name element_content>
<!ELEMENT livro-de-receitas (receita+)>
<!ELEMENT receita ( titulo , tipo? , autor* , ingrediente* , modo_de_preparo )>
<!ELEMENT tipo ( entrada | principal | sobremesa )>
```

13.5.4.2 – Special Keywords

- #PCDATA - Parsed character data - Para elementos que contém texto, também é usado para elementos que não tem filhos. É um sub-elemto texto.

```
<!ELEMENT titulo (#PCDATA)>
```

- EMPTY - Para elementos vazios, sempre indica um elemento que não tem filhos. o elemento não pode ter qualquer outro sub-elemento ou texto em sua declaração.

```
<!ELEMENT modo_de_preparo EMPTY>
<!ELEMENT email EMPTY>
<!ATTLIST email
href CDATA #REQUIRED
preferred (true | false) "false"
>
```

- ANY - Pode conter qualquer outro elemento declarado na dtd.

13.5.4.3 – Indicadores de Ocorrência

Indica quando um elemento pode repetir, e quais elementos não precisam aparecer.

- "+" -> 1... -> indica que o elemento pode ser repetido.
- "*" -> 0... -> indica que o elemento pode aparecer 0 ou várias vezes.
- "?" -> 0..1 -> indica que o elemento pode aparecer 0 ou uma única vez.

```
<!ELEMENT receita ( titulo , tipo? , autor* , (ingrediente*) , modo_de_preparo* ) >
```

13.5.4.4 – Cardinalidade

- 1

```
<!ELEMENT entry (name, address, tel, email)>
```

- 0..*

```
<!ELEMENT entry (name, address*, tel*, email*)>
```

- 1..*

```
<!ELEMENT entry (name, address+, tel+, email+)>
```

- 0..1

```
<!ELEMENT entry (name, address?, tel?, email?)>
```

13.5.4.5 – Conectivos

- AND

```
<!ELEMENT entry (name, address*, tel*, email*)>
```

- OR

```
<!ELEMENT name (#PCDATA | fname | lname)*>
```

13.5.4.6 – Atributos

Atributos podem ter qualquer um dos valores abaixo:

- CDATA - para strings;
- ID - para identificador, único no documento;
- IDREF - é um valor de um ID que está em qualquer outro lugar do documento;

- IDREFS - é uma lista de IDREF separada pos espaços;
- ENTITY - é um nome de uma entidade externa;
- ENTITIES - é uma lista de ENTITY separada por espaços;
- NMTOKEN - é uma palavra sem espaços;
- NMTOKENS - é uma lista de NMTOKEN separada por espaços;
- Enumerated-type list – é uma lista fechada de NMTOKENS separada por “|”;

Porém a DTD oferece um valor default para atributos, quando não especificamos no documento. O valor default pode ter um dos quatro elementos:

- #REQUIRED – indica que o elemento deve ser fornecido no documento;

```
<!ATTLIST email href CDATA #REQUIRED>
```

- #IMPLIED – indica que se o elemento não for fornecido será usado o valor defaul;

```
<!ATTLIST email href CDATA #IMPLIED>
```

- #FIXED – indica que o valor deve ser o declarado na DTD; Um valor literal indica que o atributo terá esse valor se nenhum valor for fornecido;

```
<!ATTLIST email href CDATA #FIXED "nce@ufrj.br">
```

- Tipos enumerados

```
<!ATTLIST fax preferred (true | false) "false">
```

13.5.4.7 – Entidades

- Entidades Internas

```
DTD:
<!ENTITY jonas “Jonas Mala sem Alça”>
XML:
<raiz>
        &jonas;
</raiz>
```

Geralmente são usadas para associar um mnemonic:

```
<!ENTITY icirc “&#238”>
```

- Entidades externas: A declaração de entidades pode ser feita no DTD

```
<!ENTITY jonas SYSTEM "C:\lixo\jonas1.txt">
```

Ou pode também ser declarada em outra DTD:

```
<!ENTITY % carta SYSTEM "carta.dtd" >
%carta;
<!ELEMENT variasCartas (carta*) >
```

- Entidades de parâmetros - Existe um “%” antes do nome da entidade.

```
<!ENTITY % trat.at "Sr|Sra" >
<!ATTLIST destinatário tratamento %trat.at; >
<!ENTITY % id.at "ID #REQUIRED" >
<!ATTLIST emissor cod %id.at; >
```

13.5.4.8 – Indicação da DTD no XML

```
<?xml version="1.0"?>
<!DOCTYPE carta SYSTEM "carta.dtd" >
<carta>
...
</carta>
```

CAPÍTULO 14 – EXTENSIBLE MARKUP LANGUAGE – XML

14.1 – VISÃO GERAL

Na escala evolutiva das linguagens de marcação o XML é um marco, e seu surgimento foi provocado pela insatisfação com os formatos até então existentes para determinados fins. É uma recomendação do W3C (www.w3.org/XML) para descrever dados de forma padronizada para compartilhamento e distribuição de informações na web.

Projetada para definir documentos com qualquer tipo de dados estruturados em formato de arquivo universal, ASCII. Pode ser escrito em qualquer editor de texto e pode ser facilmente lido, preparado e parseado por qualquer programa. O que torna os dados escritos em XML independentes de plataforma, de aplicativos, fornecedores, implementações ...

É baseada em regras que asseguram sua consistência e seus dados são facilmente conversíveis. Linguagem de marcação extensível, capaz de gerar outras linguagens baseadas nela, por exemplo, XHTML, RDF, RSS, XSL, entre outras.

Comparando-a às outras linguagens de marcação que a antecede, ela possui algumas vantagens. Suporta praticamente todas as funcionalidades mais difundidas do padrão SGML, entretanto é mais simples por definição:

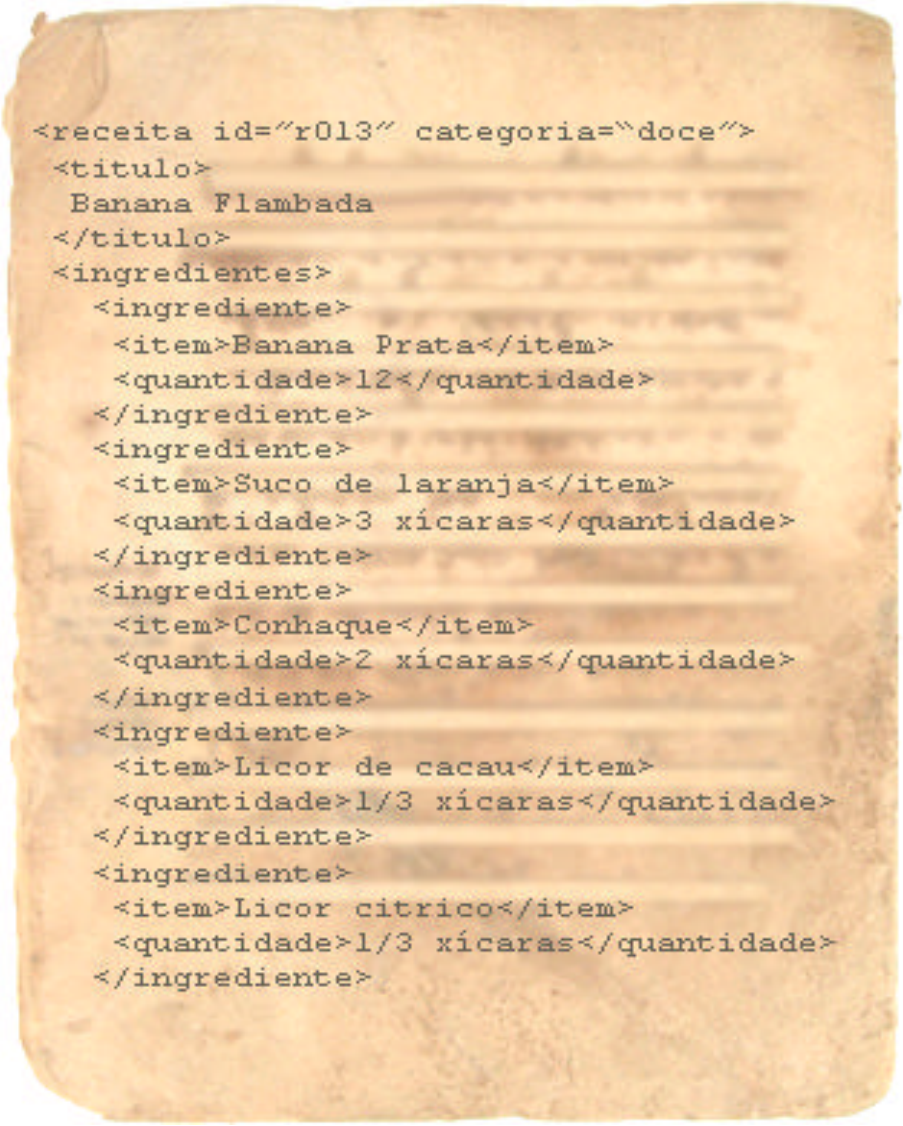
- Especificação SGML: 500 páginas X Especificação XML: 33 páginas

Para a interpretação de documentos XML existem várias APIs(Application Programming Interface), porém duas principais: DOM (Document Object Model) e SAX (Simple API for XML). DOM é um padrão definido pelo W3C que cria uma visão em árvore do documento XML, enquanto SAX, uma API de mais fácil utilização interpreta os dados seqüencialmente, sem ser necessário ter conhecimento total do arquivo XML. Essas APIs serão apresentadas e discutidas mais adiante no capítulo sobre Parsers e Tecnologias XML[15][16][17][18][19][20].

14.2 – CARACTERÍSTICAS

- Extensibilidade - permite um número ilimitado de tags.
- Dados hierarquicamente estruturados: conteúdo do documento = descrição da estrutura + dados. Estruturas podem ser aninhadas em nível de profundidade arbitrário.

- Formato inteligente – armazena dados e metadados em formato ASCII.
- Não é o modo mais eficiente de armazenar dados – taxa de overflow alta por ter que processar tags de marcação.
- Portabilidade – Apresentação separada do conteúdo. Tags descrevem os dados, e são definidas pelo criador do documento, cada uma delas indicativa, não de como algo deve ser exibido, e sim do que significa.
- Permite validação – permite criação de arquivos para validação de estrutura (DTD, XMLSchema), mas não é uma imposição.
- Interoperabilidade - com seu uso podemos interligar bancos de dados distintos, gerando a partir de um banco de dados arquivos XML que outros bancos saberão ler e coletar os dados.
- Inteligibilidade - Legibilidade tanto por humanos quanto por máquinas.



```
<receita id="r013" categoria="doce">
 <titulo>
  Banana Flambada
 </titulo>
 <ingredientes>
   <ingrediente>
    <item>Banana Prata</item>
    <quantidade>12</quantidade>
   </ingrediente>
   <ingrediente>
    <item>Suco de laranja</item>
    <quantidade>3 xícaras</quantidade>
   </ingrediente>
   <ingrediente>
    <item>Conhaque</item>
    <quantidade>2 xícaras</quantidade>
   </ingrediente>
   <ingrediente>
    <item>Licor de cacau</item>
    <quantidade>1/3 xícaras</quantidade>
   </ingrediente>
   <ingrediente>
    <item>Licor citrico</item>
    <quantidade>1/3 xícaras</quantidade>
   </ingrediente>
```

❒ Figura 14.1 – Documento encontrado por um arqueologista no século 23, que o catalogou como parte de um conjunto de receitas.

14.3 – POR QUE XML?

- Torna mais fácil a compreensão da semântica do documento para leitores humanos
- Melhoria quanto à recuperação: facilita a indexação de documentos
- Integra dado e meta-dado
- Já foi comprado por muitos!

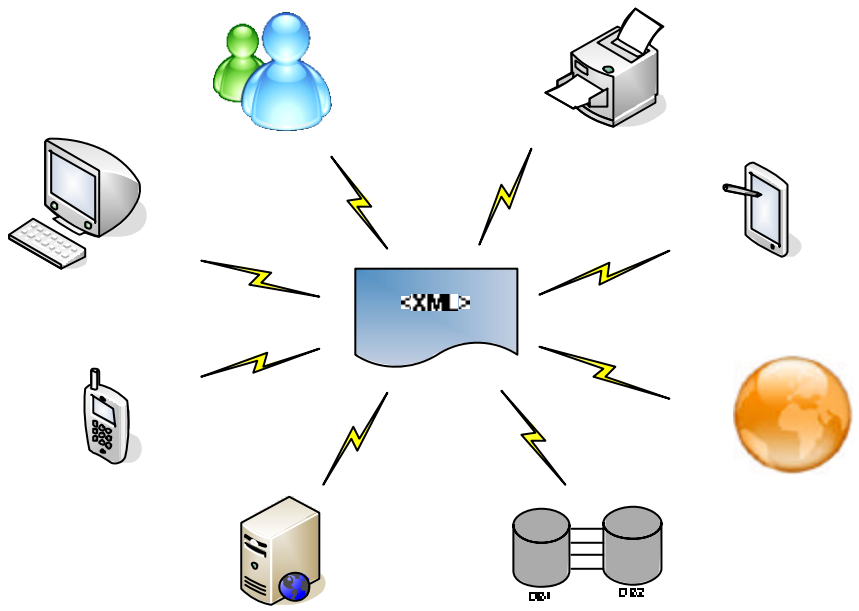
14.4 – CONSEQÜÊNCIAS DO USO DE XML

Poderia listar enumeras conseqüências do uso de XML em nossas aplicações. Mas creio que as principais estão ligadas diretamente às características mencionadas. A mais notória é que

seu uso possibilita que qualquer aplicativo possa interpretar os dados de um documento XML, inclusive aplicativos de diferentes finalidades, i.e., para um mesmo arquivo XML podemos ter aplicações distintas onde cada uma só coletaria os dados relevantes a ela. Um bom exemplo são os canais de notícias em RSS, que pode ser lido tanto em browser's através de transformações XSL, ou em agrupadores de notícias, como o Outlook e o Thunderbird.

O uso de XML possibilitou o desenvolvimento de aplicações flexíveis para a web, tornando a internet o meio preferido para aplicações de negócio.

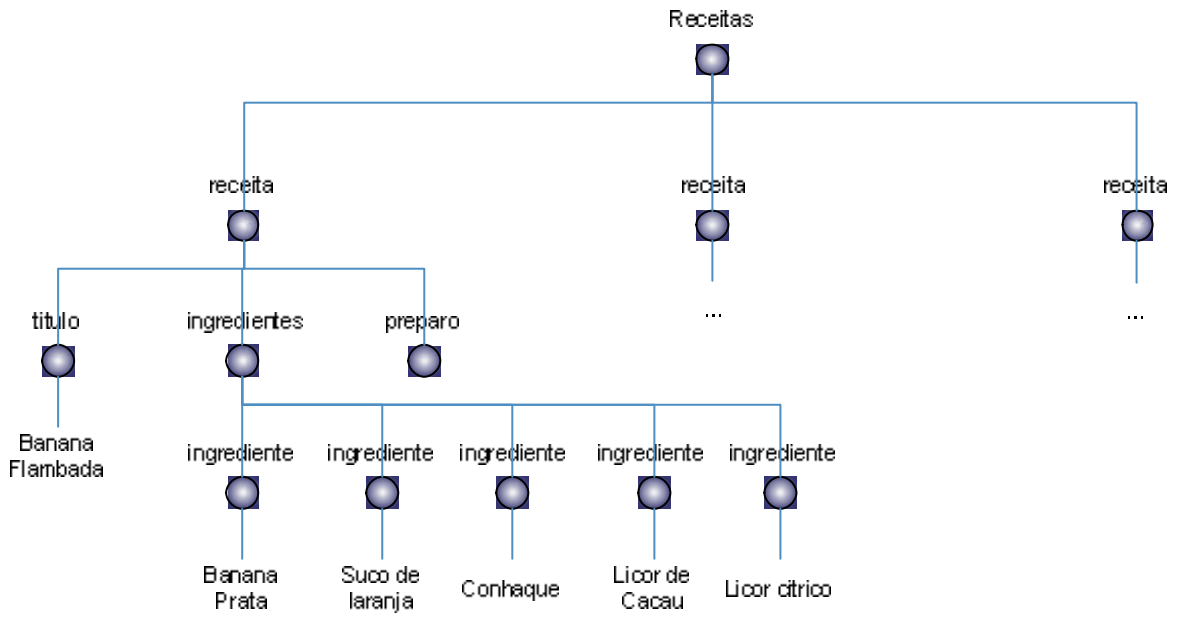
Uma outra vantagem que segue em conseqüência é que nossos dados não ficam presos a plataformas específicas e não é necessária a fidelidade a um aplicativo simplesmente porque seus dados foram armazenados em formato proprietário. Por exemplo, poderíamos em uma empresa, mudar todo seu sistema e aplicativos sem ter que se preocupar em como portar o banco de dados, bastaria que os novos aplicativos soubessem ler e interpretar dados XML.



Certamente, é necessário levar em conta o propósito da aplicação. Existem aplicações em que o armazenamento binário terá grandes vantagens, em outras XML levará vantagens. Mas ainda sim, de um modo geral, no contexto web, com as inovações que vem sendo feitas, considero as desvantagens conhecidas da utilização de XML são pequenas frente às vantagens.

14.5 – REPRESENTAÇÃO DO XML

```
<?xml version="1.0" encoding="UTF-8"?>
<receitas>
    <receita id=”r013” categoria=”doce”>
        <titulo>Banana Flambada</titulo>
        <ingredientes>
                    <ingrediente>
                        <item>Banana Prata</item>
                        <quantidade>12 unidades</quantidade>
                    </ingrediente>
            ...
    </receita>
</receitas>
```



❒ Figura 14.3 – Exemplo de representação de árvore de um documento XML

14.5 – SINTAXE BÁSICA DE XML

14.5.1 – REGRAS BÁSICAS

- Delimita os elementos do documento através de tags.
- Todas as tags tem um start-tag e um end-tag
- Todo documento tem somente um elemento raiz

14.5.2 – INSTRUÇÕES DE PROCESSAMENTO

São marcações específicas para algumas ferramentas (por exemplo: um tradutor de XML para PDF). Em especial, há uma PI reservada que determina algumas propriedades do documento XML, e.g. versão e codificação de caracteres

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheet href="receitas.xsl" type="text/xsl" media="explorer"?>
<?xml-stylesheet href="mestrecuca.css" tipo="texto/css"?>
```

14.5.3 – ELEMENTOS

```
<autor>Vinicius de Moraes</autor>                          CORRETO
<autor id=”001” sexo=”M” nome=”Vinicius de Moraes”/>   CORRETO
<autor>Vinicius de Moraes                                  INCORRETO
```

14.5.4 – NOMES DE ELEMENTOS

- Espaços não são permitidos
- Case sensitive
- Primeira letra ['_', 'a'..'z', 'A'..'Z']
- Demais letras: ['_', 'a'..'z', 'A'..'Z', '0'..'9', '.', '-']

14.5.5 – ATRIBUTOS

Todo atributo está em um start-tag, tem um nome único na tag e um valor entre aspas(“ou‘). Podemos adicionar informação a um elemento na forma de atributo.

```
<autor id=”001” sexo=”M”>Vinicius de Moraes</autor>
```

14.5.6 – ELEMENTOS VAZIOS

Elementos vazios são, em geral, inseridos no documento por força de seus atributos.

```
<email href=”mailto:mvm@mestrecuca.com”/>
<email href=”mailto:mvm@mestrecuca.com”><email/>
```

14.5.7 – REFERÊNCIA ENTRE ELEMENTOS

Elementos podem ter IDs e referenciar outros IDs.

```
<autor id="001">
	Vinicius de Moraes
</autor>
<comentario>
	<data>20061224</data>
	<autor id-em-autor=”001” />
	<texto>...</texto>
</comentario>
```

14.5.8 – ELEMENTO RAIZ

Todos os elementos em um documento devem ter um ancestral comum. Formando árvore de dados hierárquicos aninhados com um único nó pai inicial.

14.5.9 – DECLARAÇÃO XML

A declaração XML é, se presente, obrigatoriamente a primeira linha no documento. Por ela é informada a versão do XML, linguagem em que foi escrita, e com que codificação.

```
<?xml version="1.0" encoding="UTF-8"?>
```

14.5.10 – COMENTÁRIOS

```
<!-- isto e' um comentario -->
```

14.5.11 – ENTIDADES

Existem dois tipos de entidades, as que são pré-definidas e as que são definidas pelo usuário.

```
<copyright>&MestreCucaCopyright;</copyright>
```

Podemos definir essas entidades em arquivo DTD:

```
<!ENTITY MestreCucaCopyright SYSTEM
 "http://www.rodrigobuas.no-ip.com/mestrcuca/copyright.xml">
```

Existem várias entidades pré-definidas. Abaixo estão listadas algumas:

- < <

- & &
- > >
- ' '
- " "

14.5.12 – CDATA

Documentos XML podem ter dados que não são avaliados ou que não queremos que sejam avaliados. É usado para escrever textos contendo caracteres que poderiam ser reconhecidos como marcações ou inserir elementos, e.g., colocar imagens bitmap no documento XML, texto ASCII formatado, passar instruções de instalação ou uso...

Existe uma restrição para esse tipo de dados, não permite a ocorrência apenas da sequência "]]> ".

```
<receita>
	<imagem>
		<![CDATA[
			**********<aqui> <não> tem <sintaxe>
		]]>
	</imagem>
…
```

14.5.13 – NAMESPACE

Os namespaces permitem dar um contexto para os nomes dos elementos e atributos, e.g., texto no namespace “comentario” e texto no namespace “preparo”.

```
<comentario:texto> ... </comentario:texto>
<preparo:texto> ... </preparo:texto>
```

14.5.14 – DECLARAÇÃO DE DTD

Para definir o que pode conter em um XML, usamos XMLSchema ou DTD. Esses documentos definem:

- as tags de cada documento
- quais tags podem conter outras tags
- o número e sequência das tags
- os atributos que as tags podem ter e seus valores

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mestrecuca SYSTEM "http://www.rodrigobuas.no-
ip.com/mestrecuca.dtd">
```

Também podemos definir internamente, no próprio XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE receitas      [
            <!ELEMENT receita (titulo, autor, ingrediente, preparo )>
            <!ELEMENT titulo (#PCDATA)>
            …
                        ]
>
<receitas>
      …
</receitas>
```

14.6 – XML SCHEMAS

É uma alternativa para descrever a estrutura dos dados em XML. Assim como DTD, tem o propósito de descrever como devem ser escritos os blocos de dados no documento. Como ele é possível fazer validação no XML.

Um XMLSchema pode definir:

- Elementos que podem aparecer no documento
- Atributos que podem aparecer no documento
- A hierarquia dos dados – qual elemento é filho de quem, a ordem e a quantidade dos elementos filhos
- Se um elemento pode ser vazio ou se pode conter um texto
- O tipo de dado dos elementos e dos atributos

Substitui as DTD com algumas vantagens:

- Nova forma de representar vocabulários XML
- É escrita em XML, não é necessário aprender outra linguagem
- Inclui tipos de dados
- Herança
- Aceita vocabulários múltiplos – namespaces

```
<?xml version="1.0"?>
<schema>
```

```
   <element name="receita">
      <type>
         <element name=”autor” type=”string” minOccurs=”1” maxOccurs=”1” />
         <element name=”titulo” type=”string” minOccurs=”1” maxOccurs=”1” />
         <element name=”data_criacao” type=”date” minOccurs=”1” maxOccurs=”1”
/>
         <element name=”ingredientes”>
            <type>
               <element name=”item” type=”string” />
               <element name=”quantidade” type=”number” />
            </type>
         </element>
         <element name=”prepare” type=”string” minOccurs=”1” maxOccurs=”1” />
         <element name=”obs” type=”string” minOccurs=”1” maxOccurs=”1” />
         <element name=”img” type=”image” minOccurs=”0” maxOccurs=”*” />
         <element name=”comentarios”>
            <type>
               <element name=”autor” type=”string” />
               <element name=”texto” type=”string” />
            </type>
         </element>

      </type>
   </element>
</schema>
```

CAPÍTULO 15 – CASCADING STYLE SHEET – CSS

15.1 - VISÃO GERAL

O uso de estilos em uma linguagem de marcação é um método para caracterizar com cada elemento será apresentado. Em conjunto, as regras de estilo definem o layout do documento. Folhas de estilo apareceram nos anos 70 utilizada juntamente com SGML. Cada fabricante de browser definia sua própria linguagem para customização da aparência de documentos web.

Seu uso foi muito difundido paralelamente ao uso de HTML. Com o crescimento da HTML, para atender a demanda dos desenvolvedores web, mais e mais propriedades foram sendo inseridas ao conjunto de propriedades que o CSS suportava. Porém, o modo como eram utilizados, totalmente entrelaçadas, cada vez mais foram aumentando a complexibilidade dos códigos e causando dificuldades no desenvolvimento.

Um problema muito comum era quando numa equipe de desenvolvimento de conteúdos web, onde continham programadores e designers, eventualmente um acabava interferindo e atrapalhando o outro. Para resolver este e outros problemas foi criada a CSS.

A CSS é usada para descrever a apresentação de um documento escrito com uma linguagem de marcação. Foi criada para substituir o uso de linguagens de marcação com fins de visualização. Proposta inicialmente por Hâkon Wium Lie em 1994, e em seguida por Bert Bos em 1995. Um projeto mantido pelo W3C desde dezembro de 1996. Separa sua evolução em níveis, iniciada no CSS level 1, em seguida, em 1998 foi publicada como recomendação W3C a CSS level 2, que adiciona propriedades de posicionamento – relativo, absoluto e fixo, propriedade de fontes – sombreamento, entre outras propriedades. Atualmente a CSS level 3 ainda esta em desenvolvimento.

CSS é um simples mecanismo para acrescentar estilo a um documento. Propõe a separação do estilo de apresentação para a estrutura dos dados do documento. Essa separação possibilita, entre outras coisas, mostrar o mesmo documento com estilos diferentes para modos de renderização diferentes, permite também trocar rapidamente o modo de visualização – looking fill.

Em uma folha CSS podemos definir classes, cores, fontes, posicionamento, visibilidade, espaçamento, bordas, entre outras características de apresentação. Consiste em uma ou mais

regras - lista de regras - descritas com sintaxe simples, que controlam o modo como cada elemento será exibido.

15.2 – CARACTERISTICAS

- Um estilo CSS define como os elementos de uma linguagem de marcação serão apresentados
- Os estilos são descritos em uma folha de estilos
- Estilos podem ser cascateados
- Sintaxe simples
- Criação rápida de folhas de estilo, podem ser facilmente reaproveitadas de projetos anteriores
- Folhas de estilos são salvos em arquivos “.css”
- Possibilitam o controle de layout de uma aplicação web de forma centralizada, facilitando e agilizando a atualização
- Usuários podem visualizar o conteúdo de forma personalizada

15.3 – CONSEQÜÊNCIAS

- Torna o conteúdo mais acessível
- Possibilita que uma página possa ser apresentada de diversas formas
- Reduz a largura de banda utilizada, pois evita a repetição de estilo em cada página.
- Economiza espaço de armazenamento em servidor.
- Torna o código de marcação mais enxuto, menos complexo e mais fácil de entender e de fazer manutenção.
- Permite que alterações globais sejam feitas em minutos.

15.4 – TIPOS DE ESTILOS

Os arquivos CSS podem ser providos de diferentes fontes em diferentes locais. Quando carregados, formam uma folha de estilos virtual combinando em cascata os tipos de estilos na seguinte ordem: browser default + estilos externos + estilos embutidos + estilos inline + estilos de usuários. Onde os últimos são herdam e sobrescrevem propriedades dos primeiros.

15.4.1 – ESTILOS DEFAULT

Estilos determinados pelos browsers que visualizarão o conteúdo. Por default, o browser seta como cada elemento será apresentado sem mesmo ter sido setado algum estilo.

15.4.2 – ESTILOS EXTERNOS

A lista de regras é descrita em um arquivo CSS externamente ao documento com o conteúdo, em um arquivo texto de extensão ".css". O link com um documento XHTML pode ser feito da seguinte forma:

```
<link rel="StyleSheet" href="MyStyleSheet.css" type="text/css" media="all" />
```

Ou pela seguinte declaração:

```
<style type="text/css" media="all" src="MyStyleSheet.css" />
```

Para aplicar um estilo à uma arquivo XML declaramos da seguinte forma:

```
<?xml-stylesheet type="text/css" href="MyStyleSheet.css" ?>
```

15.4.3 – ESTILOS EMBUTIDOS

As declarações CSS estão dentro do documento de marcação, mais especificamente no cabeçalho do documento, como ilustrado a seguir:

```
...
<head>
<title>MyPage</title>
<style type="text/css">
<!—
  //declaração da lista de estilos
-->
</style>
</head>
<body>
...
```

15.4.4 - ESTILOS INLINE

As regras CSS podem ser aplicadas a um elemento individual usando o atributo style diretamente em um elemento, e.g.:

```
<table style="//declaração do estilo">
```

15.4.5 – ESTILOS DE USUÁRIOS

Folhas de estilos setadas diretamente pelo usuário em seu browser. Onde são setadas as opções de visualização que mais se agrada o usuário. Permite que o usuário personalize nosso conteúdo disponibilizado da forma que melhor achar. Vários browser permitem que essas regras sejam setadas facilmente.

A folha de estilos do usuário é a mais específica, i.e., sobrescreve qualquer regra antes setada por outra folha de estilos.

15.5 – SINTAXE

Uma folha CSS é um conjunto de regras, onde cada regras consiste em duas partes: um seletor e uma declaração. Em seguida, veremos cada uma delas.

15.5.1 – SELETORES

No CSS, o "C" refere-se a cascading. O nome é devido ao fato de que CSS implementa herança de regras. Os valores setados nas propriedades são herdados de pai para os elementos filho. A herança é indicada no seletor da regra. A seguir, veremos os tipos existentes de seletores e como podemos trabalhar com eles.

15.5.1.1 – Seletor Simples

Usado para setar propriedades de um elemento específico de uma linguagem de marcação. Nestes casos, os seletores recebem o nome do elemento. Veja o exemplo a seguir:

```
table{ /* declaração do estilo */ }
```

No exemplo as propriedades setadas na declaração do estilo são referentes a todos os elementos table.

15.5.1.2 – Seletores Agrupados

Usado para compartilhar propriedades estilísticas entre diversos elementos. Aplica uma única regra à vários elementos dispostos em uma lista de elementos delimitados por uma ",".

```
table, p, li { /* declaração do estilo */ }
```

Neste exemplo, os elementos table, p e li receberão as mesmas características declaradas. Podemos também usar expressões para agrupar ou delimitar o grupo de elementos correspondentes. Vejamos:

Para todos os elementos podemos usar o seletor “*”:

```
* { /* decalaração do estilo */ }
```

Para elementos adjacentes, podemos usar expressões do tipo:

```
h1 + p { /* declaração do estilo para elementos p precedidos de h1 */ }
```

Para elementos filhos:

```
table > p
{
  /* declaração do estilo somente para os elementos p que estiverem dentro de table
*/
}
```

Para elementos que possuem um atributo com um valor específico:

```
input[type=“text”]
{
  /* declaração do estilo para somente para elementos input que possuam o atributo
type setado como text */
}
```

15.5.1.3 – Seletores Contextuais

Também chamados de descendentes. O estilo determinado pelas declarações dependem do contexto de que um elemento está inserido.

No exemplo a seguir, todos os elementos da lista que estiverem no contexto strong, ou seja, que aparecerem dentro de um elemento strong, receberão o estilo declarado. Todos os elementos que forem uma lista mas que não estiverem no contexto strong não receberão o estilo.

```
li:strong{ /* declaração do estilo */ }
```

Também podemos usar esse tipo de seletor como pseudo-classes/emento:

```
a:hover
{
```

```
  /*declaração do estilo de uma âncora no momento que o foco do mouse está no elemento*/
}
```

ou

```
p:first-line { /* declaração do estilo para primeira linha de um parágrafo */ }
```

ou

```
p:lang(php) { /* declaração do estilo para escritos em linguagem “php” */ }
```

15.5.1.4 – Seletores de Id

São referenciados nos elementos pelo atributo “id”. Com este tipo de seletor podemos criar uma única regra e atribuir esta regra a diversos elementos. Por exemplo:

```
#infobar { /* declaração de estilo para os elementos de id igual a “infobar” */ }
```

E poderíamos atribuí-los à um elemento da seguinte forma:

```
<div id= “sidebar”> ... </div>
```

Note que os seletores de id são precedidos do caractere “#”. Os id não podem iniciar por números. Aplicados desta forma também são chamados de seletores autônomos. Podemos também usar seletores contextuais em conjunto com seletores de id:

```
#elementid:hover{ /* declaração do estilo */ }
```

Cascateando os tipo de seletores já vistos poderíamos fazer combinações como no exemplo a seguir:

```
#infobar p { /* declaração do estilo unicamente para parágrafos que possuírem id igual a “infobar”. Outros elementos que receberem o id “infobar” não receberão este estilo */ }
```

15.5.1.5 – Seletor de Classes

Os seletores de classes são declarados com os nomes das classes precedidos de um “.” no arquivo “.css”. São referenciados nos elementos pela propriedade “class”. Vejamos o seguinte exemplo:

```
.MyClass
{
```

```
 /* declaração do estilo para elementos que possuem o atributo class como
“MyClass” */
}

<div class=“MyClass”> ... </div>
```

O nome da classe não pode ser iniciado por um número. Do mesmo modo como foi feito com os seletores de id, podemos cascatear seletores de classes com outros tipos de seletores. Por exemplo:

```
.MyClass td
{
 /* declaração do estilo de elementos td de atributo class igual a “MyClass” */
}
```

15.5.2 – DECLARAÇÃO

Uma declaração ou um bloco de declarações são delimitados por chaves, “{” e “}”. Cada declaração possui duas partes: uma propriedade seguida do caractere “:” e por um valor. Toda declaração deve terminar um o caractere “;”, como o modelo a seguir:

```
{ propriedade : valor ; }
```

Por exemplo:

```
p { color : #ff0000 ; }
p
{
 color : #ff0000 ;
 backgorund : transparent;
}
```

O motivo deste capítulo e introduzir o conceito de estilos no projeto. Cobrir todas as técnicas e usos de CSS certamente seria conteúdo para um trabalho inteiro. Todas as propriedades da API de CSS estão disponíveis para consulta em http://www.w3schools.com/css.

15.6 – PROPRIEDADES ABREVIADAS

Nas declarações de estilo podemos fazer algumas abreviações, declarar propriedades de forma mais simplificada. Por exemplo, para declararmos a margem de um elemento explicitamente podemos fazer da seguinte forma:

```
P
{
```

```
  margin-top    : 10px ;
  margin-right  : 0 ;
  margin-bottom : 10px ;
  margin-left   : 0 ;
}
```

Ou então poderíamos declarar:

```
p
{
  margin : 10px 0 10px 0 ;
}
```

Ou simplesmente:

```
p
{
  margin : 10px 0 ;
}
```

Como no exemplo da propriedade margens, funcionam similarmente outras propriedades. Algumas delas estão na tabela abaixo. A lista completa pode ser encontrada também no site http://www.w3schools.com/css.

Propriedade	Valores
background	background-color , background-image , background-repeat , background-attachment , background-position
font	font-style , font-variant , font-weight , font-size/line-height , font-family , caption , icon , menu , message-box , small-caption , status-bar
border	border-width , border-style , border-color
margin	margin-top , margin-right , margin-bottom , margin-left
padding	padding-top , padding-right , padding-bottom , padding-left
list-style	list-style-type , list-style-position , list-style-image

❒ Tabela 15.1 – Algumas propriedades de CSS

15.7 – PSEUDO-CLASSES E PSEUDO-ELEMENTOS

Já foi introduzido nos exemplos anteriores o conceito de pseudo-classe e pseudo-elementos. São elementos que adicionam estilos a elementos devido a determinadas características de cada um. Resumidamente, adicionam efeitos especiais aos elementos. São descritos seguindo os seguintes modelos:

```
seletor:pseudo-classe { /* declaração do estilo */ }
seletor:pseudo-elemento { /* declaração do estilo */ }
```

Ou usado juntamente com classes:

```
seletor.classe:pseudo-classe { /* declaração do estilo */ }
seletor.classe:pseudo-elemento { /* declaração do estilo */ }
```

Os Abaixo está a lista das pseudo-classes:

- :active
- :focus
- :hover
- :link
- :visited
- :first-child
- :lang

Em seguida encontra-se a lista dos pseudo-elementos:

- :first-letter
- :first-line
- :before
- :after

Para ver a lista completa, com as características de cada pseudo-classe e pseudo-elemento, e quais propriedades se aplicam a cada um, pode ser encontrada nos sites indicados sobre CSS neste documento.

Poderíamos utilizar pseudo-classes para colorir links. Vamos primeiro ver os exemplos para poder depois tecer comentários sobre eles:

```
a:link
{
  text-decoration : none ;
  color : #333399 ;
}
a:visited
{
  text-decoration : none ;
  color : #990000 ;
}
a:hover
{
  text-decoration : underline ;
  color : #ff6800 ;
}
a:active
{
  text-decoration : none ;
  color : #f76910 ;
}
```

Note que as declarações dos estados do link seguiram uma ordem: link, visited, hover e active. Usualmente, devido a história de erros de browsers antigos, é indicado seguirmos essa ordem.

15.8 – HIDE E BLOCK

Usualmente criamos uma classe hide para manipular posteriormente a visibilidade de elemento. A declaramos da seguinte forma:

```
.hide { display : none ; }
```

A propriedade display indica o modo como serão apresentados os elementos. O valor none, indica que o elemento que receber a classe hide não será mostrado. Também é muito útil declarar para certos elementos que sejam apresentados em blocos atribuindo o valor block à propriedade display. Por exemplo:

```
img
{
  display : block ;
  border : 0 ;
}
```

CAPÍTULO 16 – PARSER'S XML

16.1 – VISÃO GERAL

Um parser de XML tem como input uma string serializada e executa determinadas operações nele. Primeiramente verifica se está bem formatada, analisando os dados sintaticamente (conferindo Tags e codificação). Alguns parsers podem executar também uma etapa de validação, em geral partindo do Document type definition (DTD) ou do XMLScheme para verificar se a estrutura e o índice estão como especificado. Finalmente, no output fornece o acesso aos dados contidos no documento XML através de APIs.

16.2 – DOCUMENT OBECT MODEL – DOM

16.2.1 – INTRODUÇÃO

Neste parte será apresentada a interface de DOM, um dos padrões mais usados para manipulação e processamento de XML. É importante salientar que o motivo deste capítulo é meramente para documentar o estudo que realizei durante esse projeto, e assim, apresentar a teoria que embasou o desenvolvimento.

Recomendação da W3C, Document Object Model – DOM é uma API que possibilita programas e scripts acessar dinamicamente o conteúdo, estrutura e o estilo do documento XML. Em conteúdos web, o documento original pode ser processado e os resultados daquele que processo podem ser incorporados na página apresentada sem ser necessário um pedido ao servidor.

Fornece um conjunto de objetos e interfaces que representam o conteúdo e a estrutura de documentos XML, sem perda de informações significativas, em forma de uma árvore de nós.

DOM é ótimo para processamento de XML, faz a leitura de um documento inteiro na memória, armazenado todos os dados em nós, assim, permite acesso muito rápido aos nós da árvore de forma aleatória. Contudo, carregar toda árvore na memória pode ser uma desvantagem, pois os recursos podem se tornar insuficientes, reduzindo freqüentemente a velocidade, em alguns casos impossibilitando o uso de uma aplicação.

16.2.2 - CARACTERÍSTICAS

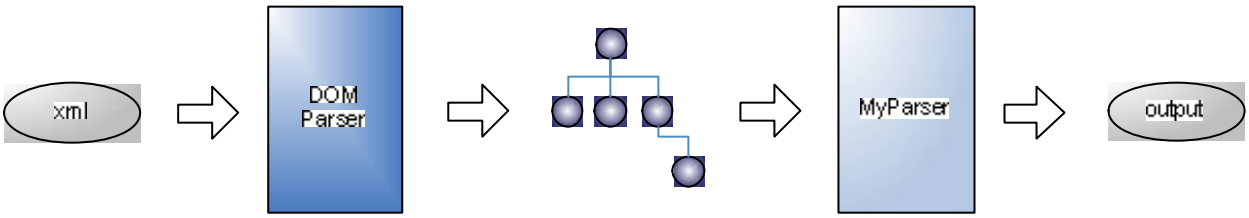
- Representa um documento XML bem formado na forma de uma árvore

- Monta uma estrutura em árvore na memória, modelo de objeto na forma como estão organizados os dados do documento
- Parsers baseados em DOM conseguem percorrer estas estruturas
- Existe em várias linguagens: Java, C, C++, Python, Perl, etc

Parsers Baseados em DOM:

- msxml (Microsoft) – Micrisoft XML parser
- JAXP (Sun Microsystem) – Java API for XML Processing
- 4DOM – parser para a linguagem Pyton
- XML4J (IBM) – XML Parser for Java (-> Xerces2)
- Xerces2 (Apache) – Xerces2 Java Parser
- XML::DOM – modulo de Pearl

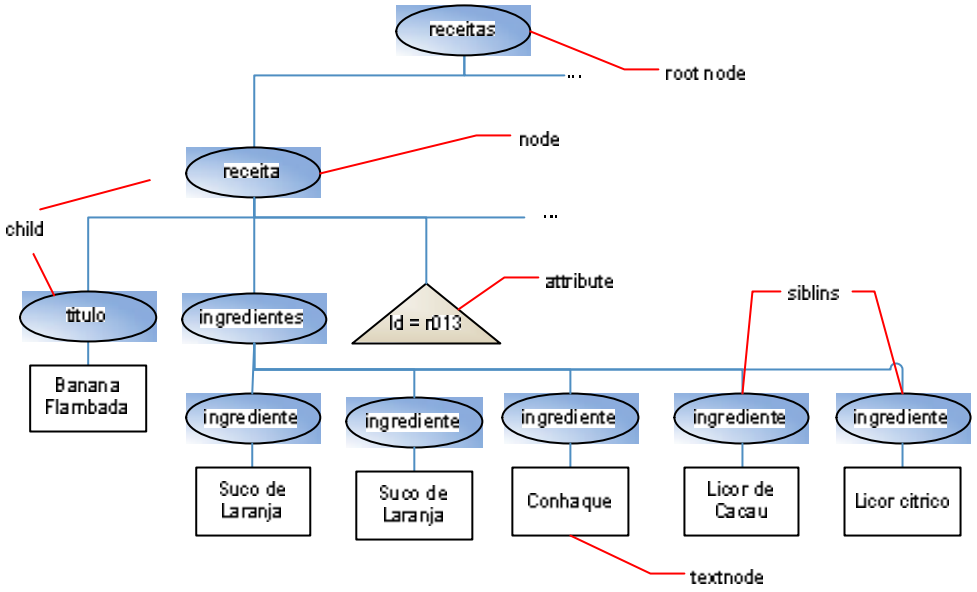
16.2.3 – FUNCIONAMENTO



❒ Figura 16.1 – Funcionamento DOM.

A partir de um documento XML, DOM analisa gramaticalmente o conteúdo do documento, em seguida, cria na memória a árvore do documento XML em forma de objeto. Deixando a disposição do usuário o acesso aos nós através de sua API. Também é possível criar uma árvore vazia, e através da API, construir os nós e popular o XML.

Em seguida encontra-se a representação de DOM, em forma de arvore de dados. Note que as elipses representam os nós, os triângulos os atributos e os retângulos os nós de texto.



❒ Figura 16.2 – Representação DOM de um documento XML.

16.2.4 – INTERFACE DOM

Entidade	Descrição
Document	representa o nó de nível mais alto em um documento XML. Provê acesso a todos os nós do documento, incluindo o elemento raiz
Node	representa um nó do documento XML
List Node	uma lista somente de leitura de objetos Node
Element	representa um nó elemento, herda do objeto Node
Attr	representa um nó atributo, herda do objeto Node
CharacterData	estende Node com um conjunto de atributos e métodos para acessar character date em DOM
Text	representa um nó texto, texto contido em um Element ou Attr. Herda de CharacterData
Comment	representa um comentário, herda de CharacterData
ProcessingInstruction	representa um nó de instruções de processamento. Herda de Node
DocumentType	DTD associado à árvore de nós
CDATASection	representa uma seção CDATA. Herda de Text

❒ Tabela 16.3 – Entidades DOM

A API de DOM, para manipulação de cada entidade, pode ser encontrada em http://www.w3school.org/DOM .

16.2.5 – UTILIZAÇÃO

Mostrarei resumidamente a utilização de DOM com um exemplo. Neste exemplo, utilizando um script PHP, a partir do XML fonte abaixo, irei percorrer a árvore DOM referente mostrando como saída os dados contidos no XML.

```
<?xml version=“1.0”?>
<usuarios>
	<usuario user_id=“001” status=“online”>
		<nome>Vinicius de Moraes</nome>
		<senha>vm</senha>
	</usuario>
	<usuario user_id=“002” status=“offline”>
		<nome>Gilberto Gil</nome>
		<senha>gg</senha>
	</usuario>
	<usuario user_id=“003” status=“online”>
		<nome>Bill Gates</nome>
		<senha>bg</senha>
	</usuario>
	<usuario user_id=“004” status=“online”>
		<nome>Albert Einstein</nome>
		<senha>ae</senha>
	</usuario>
	<usuario user_id=“005” status=“offline”>
		<nome>Ray Charles</nome>
		<senha>rc</senha>
	</usuario>
	<usuario user_id=“006” status=“online”>
		<nome>Jô Soares</nome>
		<senha>js</senha>
	</usuario>
</usuarios>
```

Abaixo segue o código, em PHP, que percorrerá a estrutura do documento XML.

```
<?php
function PrintXML( $domXML )
{
   $ndeNode = $domXML->firstChild;
   while( !is_null( $ndeNode ) )
   {
```

```
      switch( $ndeNode->nodeType )
      {
         case XML_TEXT_NODE:
            if( !( trim( $ndeNode->nodeValue ) == "" ) )
            {
               echo($ndeNode->nodeValue );
            }
            break;
         case XML_ELEMENT_NODE:
            echo( $ndeNode->nodeName );
            if( $ndeNode->hasAttributes() )
            {
               $att = $ndeNode->attributes;
               foreach( $attributes as $index => $att )
               {
                  echo( $att->name . “=” . $att->value;
               }
            }
            break;

      }
      if ( $ndeNode->hasChildNodes() )
      {
         PrintXML( $ndeNode );
      }
      $ndeNode = $ndeNode->nextSibling;
   }
}
?>
```

16.3 – SIMPLE API FOR XML PARSING – SAX

16.3.1 – INTRODUÇÃO

Veremos agora alguns aspectos de SAX, uma outra API para manipulação de XML que também é muito utilizada. Abordarei resumidamente sobre suas características, suas vantagens e suas limitações.

Não é um padrão de W3C, mas é um API tão conhecida e utilizada quanto DOM. Esta API provê uma armação baseada em eventos para fazer parsing dos dados XML, cresceu pelo motivo do método DOM ser extremamente complexo em determinadas aplicações, tornando-se inadequado.

Diferentemente de DOM, SAX não constrói uma representação da árvore de um XML inteiro na memória, em vez disso, dispara uma série de eventos enquanto lê o conteúdo do

documento. Desmonta os dados em partes utilizáveis e os lê nó à nó. O processamento é feito passando uma só vez no documento, sendo que o programa não pode ir para lugares aleatórios do documento.

O modelo de processamento do SAX é menos flexível e limita as possibilidades de utilização. Em compensação utiliza muito pouca memória e aumenta muito a velocidade de processamento. Por isso SAX é indicado para leitura de grandes arquivos, sendo ideal para manipular dados da XML no servidor.

16.3.2 – CARACTERÍSTICAS

Um programa controlado por eventos pode:

- Procurar em um documento um elemento que possua uma palavra-chave no seu conteúdo.
- Imprimir conteúdo formatado na ordem em que aparece.
- Modificar um documento XML fazendo pequenas mudanças, como corrigindo a ortografia ou re-nomeando elementos.
- Ler dados para montar uma representação interna ou estrutura de dados complexa. Em outras palavras, a API simples pode ser usada como base para API mais complexa, como DOM.

Um programa controlado por eventos não pode:

- Reordenar os elementos em um documento.
- Resolver referências cruzadas entre elementos.
- Verificar links de ID-IDREF
- Validar um documento XML

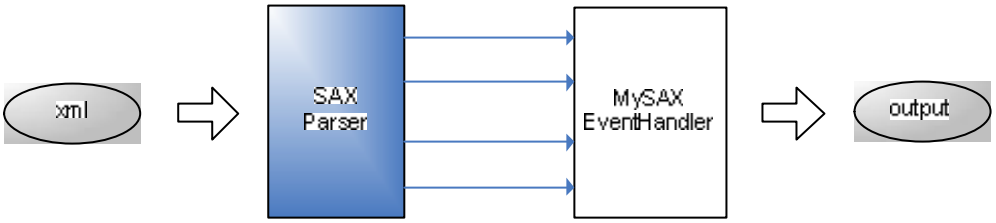
16.3.3 – FUNCIONAMENTO

Em um sistema baseado em eventos, o arquivo é analisado uma vez informando a ocorrência de forma linear. No momento da ocorrência do evento, o programa invoca o método adequado para manipular o evento. No SAX cada controlador de eventos é submetido ao processamento de um evento.

Os eventos são empurrados para os alimentadores do evento, que fornecem o acesso aos índices do documento. Há três tipos básicos de alimentadores do evento:

- ContentHandler, receberá notificação de eventos relacionados ao documento, como inicio e final de elementos.
- DTDHandler, receberá notificação de DTD do XML
- ErrorHandler, receberá notificação de erros do parsing.

A figura abaixo mostra como o SAX parser relata eventos através de um mecanismo de callback. O parser lê o original de entrada e empurra cada evento para MyContentHandler ao processar o documento XML.



❒ Figura 16.4 – Processo de leitura de um documento XML através de enventos com o SAX parser

16.3.4 – SAX API

Evento	Descrição
Attributes	Interface da lista de atributos XML
ContentHandler	Interface para acessar o conteúdo do documento XML, é o mais usado e muitas vezes é a única interface que realmente precisa ser implementada.
DTDHandler	Recebe notificações de eventos relacionados a DTD do documento XML
ErrorHandler	Interface para acessar os erros do parsing.
EntityResolver	Se a aplicação precisa redirecionar URLs no documento esta interface deverá ser implementada. Interface básica para resolver entidades.

Locator	Interface para associar um evento SAX a um posição no documento.
XMLFilter	Interface para filtros XML.
XMLReader	Interface para ler um documento XML usando callbacks

❒ Tabela 16.4 – API SAX

16.3.5 – UTILIZAÇÃO

Novamente, mostrarei a utilização do parser com um exemplo em PHP. Utilizando o mesmo XML usado no exemplo de DOM, farei o mesmo procedimento utilizando a API SAX.

Primeiro, devemos construir um handler, que deve ser passado como uma array que indicará os nomes da funções chamadas para cada evento. Não é necessário no handler-array conter todos os elementos que um handler sax pode aceitar. O array pode ser da seguinte forma:

```
<?php

$handlers = array(   "document"   => array( "start_doc" , "end_doc" ),
                     "element"    => array( "start_element" , "end_element" ),
                     "namespace" => array( "start_namespace" , "end_namespace" ),
                     "comment"    => "comment",
                     "pi"         => "pi",
                     "character"  => "characters"
                );
?>
```

As declarações das funções que responderão aos hendlers devem seguir o seguinte padrão:

- start_doc (resource processor)
- end_doc (resource processor)
- start_element (resource processor, string name, array attributes)
- end_element (resource processor, string name)
- start_namespace (resource processor, string prefix, string uri)
- end_namespace (resource processor, string prefix)
- comment (resource processor, string contents)
- pi (resource processor, string target, string contents)
- characters (resource processor, string contents)

Abaixo segue uma simplificação de uma possível implementação das funções listadas acima.

```
function start_doc()
{
  //start reading the document
}

function end_doc()
{
  // end reading the document
}

function start_element( $parser , $name , $attributes )
{
  echo( $name );
  foreach( $attributes as $index => $att )
  {
     echo( $att->name . "=" . $att->value;
  }
}

function end_element( $parser , $name )
{
  // end reading the element
}

function characters( $parser , $data )
{
  echo ( $data );
}
```

Depois para fazer o processamento simplesmente precisamos chamar o parser, indicando a ele o handler e o XML a ser processado.

```
xml_parse( $xml , $handlers );
```

16.4 – DOM VERSUS SAX

É muito importante ter uma maneira eficiente de analisar gramaticalmente dados de XML, especialmente nas aplicações com grandes volumes de dados. Mas analisar de forma não adequada pode resultar em tempos excessivos do uso do processador e de memória.

Diversos tipos de parsers de XML estão disponíveis, DOM e SAX são os padrões mais usados e difundidos para XML. Cada uma das técnicas dependendo da utilização têm vantagens e desvantagens. Mas qual é o mais apropriado? Em que situação o uso de um pode ser melhor que outro? Em seguida irei fazer considerações sobre algumas características de cada um.

Como apresentado anteriormente, utilizando DOM podemos alcançar qualquer nó da árvore em ordem aleatória, pois este cria uma árvore inteira na memória representando os dados contidos no XML. Isso possibilita a manipulação, edição, adição e remoção dos nós.

Embora seja uma vantagem poder navegar livremente pelos nós da árvore, dependendo do tamanho do conteúdo XML, colocá-lo na memória pode ser custoso. Primeiramente, antes de criar a árvore, o XML tem que ser analisado gramaticalmente, e geralmente também tem que ser validado. Para que isto seja possível, é necessário ter o conteúdo inteiro a disposição. Outra questão relevante é o espaço de memória consumido, pois construir a árvore inteira na memória de um documento XML grande pode ser bastante caro. Em terceiro lugar, o tipo de nó é genérico em DOM, o que é ruim para fazer algum tipo de bind.

A verificação e a validação podem ser custosas, mas por outro lado podem ser extremamente valiosas quando, por exemplo, uma aplicação precisa garantir a consistência dos dados envolvidos. O uso de DOM é mais indicado para determinados tipos de aplicações do que outras. DOM é apropriado quando a aplicação necessita ter acesso aleatório aos nós do XML. Um bom exemplo é o uso em aplicações que necessitam visualizar e atualizar os dados, tais como editores de XML.

Em comparação com o DOM, SAX pode oferecer melhor performance. O modelo SAX consome pouca memória, porque o documento não precisa ser carregado inteiramente na memória de uma só vez. Também não é necessário criar objetos para todos os nós, como é feito no DOM. Uma dos destaques de SAX é o paralelismo que ele oferece, pois podemos receber os dados em paralelo utilizando vários receptores. Outra vantagem é no processo de validação, que utilizando SAX, gera um ganho significativo em eficiência em função do pouco uso de memória.

Entretanto, como o parser SAX não carrega o documento XML inteiro ele não é capaz de analisar sintaticamente o documento, pois analisar sintaticamente só é possível em um documento inteiro. SAX não possui a meta-informação tal como a hierarquia dos nós DOM, desse modo, não sabemos exatamente onde estamos no documento, a menos que isso seja controlado na aplicação, mas assim, a complexibilidade de um programa pode crescer exponencialmente em relação à estrutura dos dados contidos no XML.

SAX é muito indicado para aplicações que necessitam ler o conteúdo em uma única passagem pelo documento. SAX 2,0 tem um mecanismo interno de filtro que torna fácil de mexer num subconjunto do documento XML ou de fazer uma transformação simples.

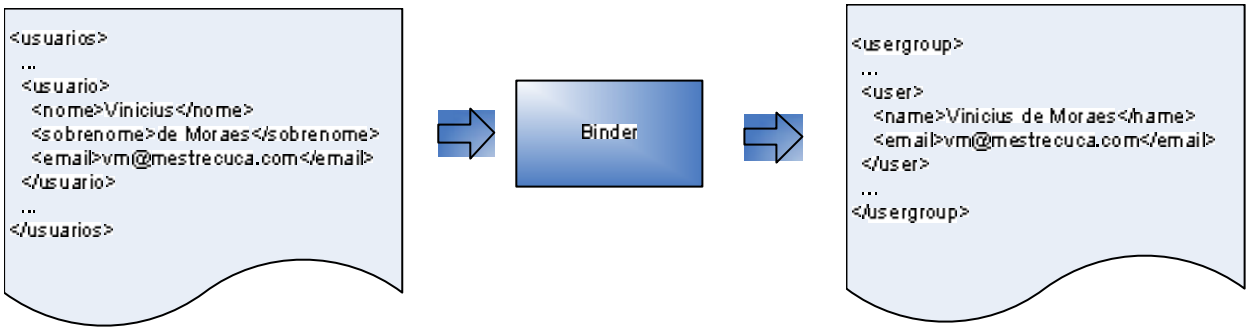
Parser	Vantagens	Desvantagens	Indicações
DOM	Fácil de usar	Todo conteúdo deve ser carregado de uma só vez	É indicado para aplicações que necessitam visualizar e atualizar os dados, tais como editores de XML
	Permite acesso aleatório	Consumo alto de memória	Não é indicado em aplicações read-only
		Tipo de nó genérico – dificuldade em fazer bind's	
SAX	Permite paralelismo	Não dá suporte para navegação no documento	É indicado para aplicações read-only
	Consome pouca memória	Não permite acesso aleatório	Não é indicado para aplicações que desejam manipular dados
			É indicado em aplicações para tráfego e encapsulamento de dados

❒ Tabela 16.5 – Comparação DOM e SAX

CAPÍTULO 17 – TRANSFORMADORES XML

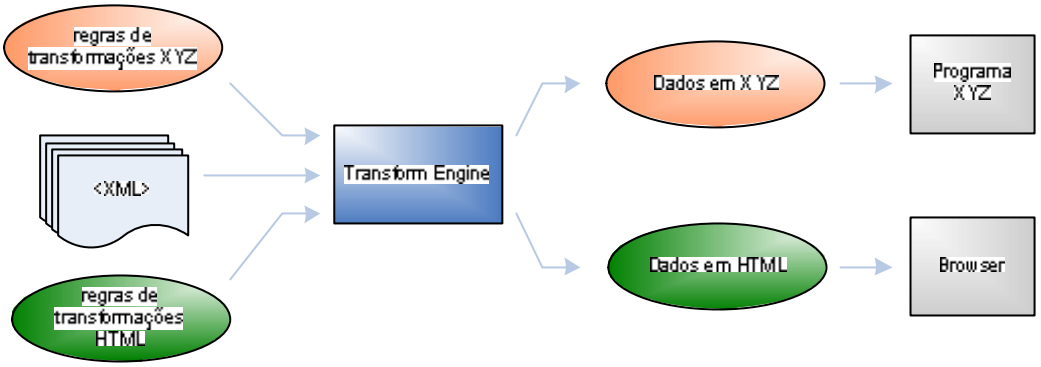
17.1 – VISÃO GERAL

Com o padrão XML podemos descrever dados estruturados em nossa aplicação de uma maneira bem simples. Essa característica abre um leque enorme de possibilidade com XML. Primeiramente, possibilita a separação entre os dados de todo o resto da aplicação. Consequentemente, podemos trocar dados entre diferentes plataformas, integrar dados de diferentes fontes e aplicações, e entre outras coisas, podemos publicar os dados em diversos formatos. A figura abaixo ilustra um exemplo de um bind realizado entre duas aplicações. Uma simples tradução de vocabulários.



❒ Figura 17.1 – Exemplo de Bind utilizando XML

Existem várias maneiras de se visualizar documentos XML. Podemos usar editores de textos, visualizar o conteúdo através de browsers, e entre várias outras possibilidades podemos usar uma folha de estilo para transformar o arquivo XML em algo que o browser apresente de uma forma diferente de tags abrindo e fechando, e ainda transformar esses dados em algo para ser lido por outro software.



❒ Figura 17.2 – Transformação de XML

17.2 – XSL – XML STYLE LANGUAGE

XML não utiliza tags pré-definidas, podemos usar qualquer nome para as tags. O significado dessas tags não são bem entendidas, e.g., um elemento <table> pode significar uma tabela HTML, um móvel ou qualquer outra coisa. E o browser não saberá como apresentar isso. XSL descreve como um documento XML deve ser apresentado.

XSL é mais que um StyleSheet Language, consiste em três partes:

- XSLT – uma linguagem para transformar documentos XML.
- XPath – uma linguagem para navegar em um documento XML.
- XSL-FO – uma linguagem para formatar documentos XML.

Características:

- Compatível com CSS (Cascading Style Sheet).
- Capacidade de reordenação da informação sem consultar o servidor Web.
- Suporte aos formatos impressos e online.

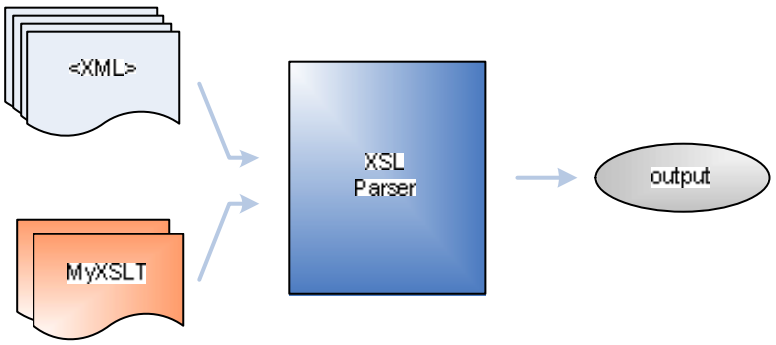
17.3 – XLST – EXTENSIBLE STYLESHEET LANGUAGE TRANFORMATIONS

O modo mais comum de criação de documentos ou páginas web é utilizando CSS e XHTML. Com a combinação desses dois webstandard's alcançamos grandes avanços, como a separação entre conteúdo e o estilo de apresentação. Porém com CSS temos algumas limitações:

- CSS não pode mudar a ordem que cada elemento aparece no documento.
- CSS não pode filtrar nem ordenar documentos baseados em outros dados
- CSS não pode calcular nenhum valor a partir de outro
- CSS não pode combinar múltiplos documentos

O XSLT é uma recomendação oficial do W3C e foi criada para ser a linguagem mais flexível e poderosa para transformação de documentos. Permite a completa separação entre conteúdo e design da aplicação. Exatamente a utopia que estávamos procurando. Transfere para o CSS toda a parte referente ao estilo de apresentação. E ao invés de misturar marcação de estrutura de apresentação com os dados, os separa em XSLT e XML.

Provê flexibilidade ao XML, e permite que transforme documentos XML em outros formatos. O processo de transformação de um documento no XSLT envolve a conversão de um formato ou estrutura para outro. Na folha XSLT são escritas as regras de transformação que manipulam sua estrutura. Como XSLT é XML, o parser carrega o XML com os dados e a XSL para a memória, então transforma o XML para outro formato e entrega os dados para o browser.



❒ Figura 17.3 – Funcionamento do processo de transformação

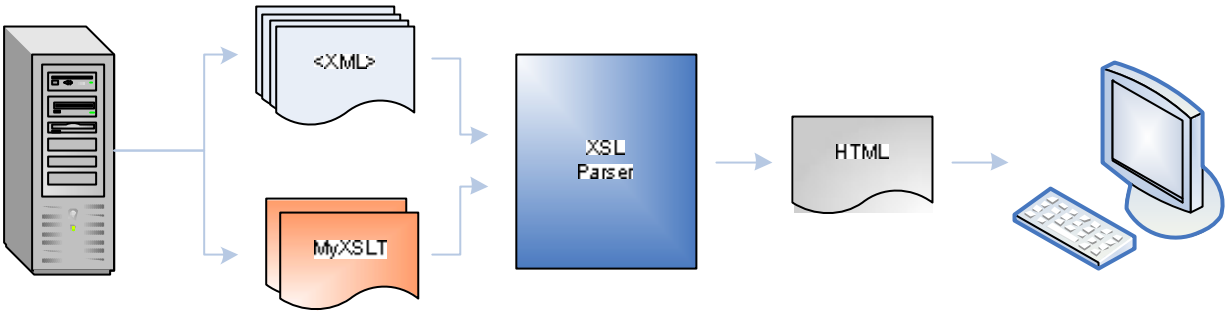
17.4.1 – CARACTERÍSTICAS

- Uma linguagem de programação para transformar documentos XML em outros formatos, e.g., HTML/XHTML, PDF, RTF, TXT, CVS...
- Adiciona e remove elementos
- Provê interoperabilidade entre sistemas
- Permite conversão de formatos
- Utiliza XPath query's

- Assume que os dados podem ser estruturados como uma árvore
- Possui recursos de recursividade

17.4.2 – SERVER-SIDE PROCESSING

A adequação a uma nova tecnologia pode demorar muito tempo. Apesar de a maioria dos browser's já suportarem esse padrão de transformação, ainda existem alguns que não. Para resolver esse problema podemos fazer a transformação no lado do servidor.



❒ Figura 17.4 – Processamento de Transformações no lado do servidor

O carregamento do XML e XSL e o processamento são todos feitos no servidor. Podemos ter algumas vantagens mantendo o processamento no servidor, tais como acessibilidade completa para qualquer browser(depende do formato que foi transformado), cacheamento de query's, e controle maior dos dados.

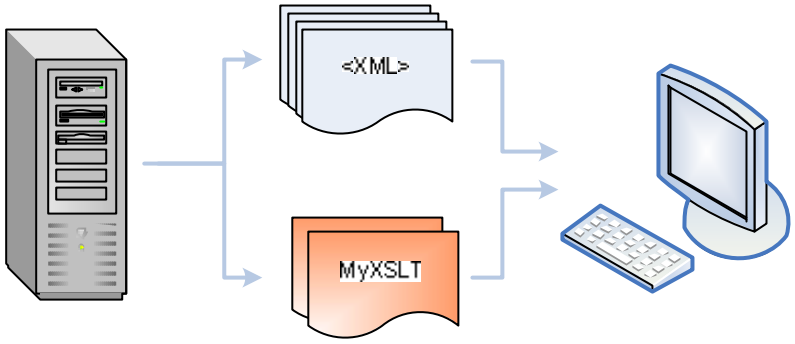
17.4.3 – CLIENT-SIDE PROCESSING

Imagine uma aplicação que mostra uma lista com a população da China, que contém bilhões de entradas. A construção desta lista pode ser demorada e pesada para o servidor. Agora imagine que uma vez já visualizada a lista o usuário deseje que a lista seja mostrada em uma ordem diferente. Normalmente, a aplicação enviaria um pedido ao servidor para que o mesmo retornasse a mesma lista ordenada de forma diferente. O servidor então teria que recalcular a lista e reordená-la. Piorando ainda mais a situação, imagine que a população da China esta fazendo uso dessa aplicação.

Esse exemplo ilustra que um mau uso do processamento do servidor. Pois dado que o usuário já possui a lista, para executar uma simples reordenação, ou um filtro qualquer (e.g. indivíduos na terceira idade), não haveria necessidade de pedir isso ao servidor.

O principal motivo de executar esse tipo de operação no servidor é o uso do processador do servidor, que na maioria dos casos é mais poderoso. Porém cada usuário tem um processador em sua maquina capaz de executar essas ações individualmente sem estourar a memória ou derrubar um sistema inteiro.

O transporte desse processamento para o cliente pode evitar overflow do servidor e tornar aplicações mais ágeis. Os mais usados browser já suportam o padrão XSLT, e se adaptam bem ao conteúdo dinâmico.



❒ Figura 17.5 - Processamento de Transformações no lado do cliente

17.4.2 – EXEMPLO

```
//Stylesheet - XSLT
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
            xmlns:xsl=http://www.w3.org/1999/XSL/Transform
            xmlns="http://www.w3.org/TR/REC-html40">

<xsl:output media-type=”html”
         doctype-public=”-//W3C//DTD HTML 4.01 Transitional//EN”
         omit-xml-declaration=”yes” />

<xsl:template match="/receitas">
<table>
   <thead>
      <th>Titulo</th>
      <th>Autor</th>
      <th>Data de criação</th>
   </thead>
   <xsl:for-each select=”receita”>
      <tr>
         <td><xsl:value-of select="titulo"/></td>
      </tr>
      <tr>
```

```
        <td><xsl:value-of select="autor"/></td>
      </tr>
      <tr>
        <td><xsl:value-of select="data_criacao"/></td>
      </tr>
   </xsl:for-each>
</table>
</xsl:template>
</xsl:stylesheet>

//Server-Side Processing
<?php
   $xml                = new DOMDocument('1.0', 'utf-8');
   $xml->load          ( “mestrecuca.xml” );
   $xsl                = new DOMDocument;
   $xsl->load          ( “v1.xsl” );
   $proc               = new XSLTProcessor;
   $proc->importStyleSheet  ( $xsl );
   $strOutput          = $proc->transformToXML( $xml );
   echo                ( $strOutput );
?>
```

Para o processamento na ponta do cliente é necessário inserir uma declaração XSLT na declaração do XML do documento contendo os dados.

```
<?xml-stylesheet type=“text/xsl” href=“NomeDoArquivo.xsl”>
```

17.5 – XPATH

XPath é uma linguagem para encontrar informações e navegar em um documento XML. XQuery e Xpointer, outros padrões para XML, ambos são construídos em expressões de XPath. Este padrão será mais discutido adiante neste documento.

17.5.1 – O QUE É XPATH?

XPath é uma linguagem para encontrar informações e navegar em um documento XML. XQuery e Xpointer, outros padrões para XML, ambos são construídos em expressões de XPath.

- XPath é uma sintaxe para definir partes de um documento XML
- XPath usa path expression para navegar em documentos XML
- XPath contém uma biblioteca de funções padrão
- XPath é o elemento principal na XSLT

- XPath é um padrão W3C
- Similar ao endereçamento do sistema de arquivos

Xpath utiliza “path expressions” para selecionar nós em um documento XML. Esses “path expressions” se assemelham muito aos “paths” de um sistema de arquivos tradicional.

Um fator importante a se mencionado aqui é que no XPath, ao invés de usar índices de array começando do zero, BASE-0, utiliza índices começando do 1, BASE-1.

17.5.2 – SINTAXE

17.5.2.1 – Expressões

Expressão	Descrição
Nodename	Seleciona todos os nós no nó
/	Seleciona do nó raiz
//	Seleciona nós do documento a partir do nó corrente que combinam com a seleção, não importando onde eles estão
.	Seleciona o nó corrente
..	Seleciona o nó pai do nó corrente
@	Seleciona os atributos

❒ Tabela 17.6 – Expressões utilizadas no XPath

Exemplos:

```
/* seleciona todos os filhos do nó receitas */
receitas

/* seleciona o nó raiz do elemento receitas */
/receitas

/* seleciona todos os elementos receita que são filhos do elemento receita */
receitas/receita

/* seleciona todos os elementos independente de onde eles estão */
//receita
```

```
/* seleciona todos os elementos receita que são descendentes do elemento receita,
não importando aonde eles estão dentro do elemento receitas */
receitas//receita

/* seleciona todos os atributos de nome lang */
//@lang
```

17.5.2.2 – Predicados

Os predicados são usados para encontrar um nó específico ou um nó que contenha um valor específico. Os predicados sempre aparecem entre colchetes.

Path Expression	Resultado
/receitas/receita[1]	Seleciona a primeira receita filha do elemento receitas
/receitas/receita[last()]	Seleciona a última receita filha do elemento receitas
/receitas/receita[last()-1]	Seleciona a penúltima receita filha do elemento receitas
/receitas/receita[position()<3]	Seleciona as duas primeiras receitas filhas do elemento receitas
//title[@lang]	Seleciona todos os elementos que tenham o atributo lang
//title[@lang='pt']	Seleciona todos os elementos que tenham o atributo lang igual a “pt”

❒ Tabela 17.7 – Expressões XPath com predicados

17.5.2.3 – Wildcards

Wildcard	Description
*	Seleciona todos os elementos no nó
@*	Seleciona todos os atributos no nó
node()	Seleciona todos os nós de todos os tipos

❒ Tabela 17.8 – XPath Wildcards

Exemplo:

```
/* seleciona todos os nós filhos do elemento */
/receitas/*

/* seleciona todos os elementos do documento */
//*

/* seleciona todos os elementos titulo que tem qualquer atributo */
//titulo[@*]
```

17.5.2.4 – Several Paths

Usando o operador | na "pathexpression" podemos selecionar diversos "paths".

Exemplos:

```
/* seleciona todos os nós E titulo e preco de todos os elementos receita */
//receita/titulo | //receita/preco

/* seleciona todos os elementos titulo E preco do documento */
//titulo | //preço

/* Seleciona todos os elementos titulo do elemento receita do elemento receitas E
todos os elementos preco do documento */
/receitas/receita/titulo | //preco
```

17.5.2.5 – XPath Axes

AxisName	Result
ancestor	Seleciona todos os ancestrais (nó pai, avô, etc.) do nó corrente
ancestor-or-self	Seleciona todos os ancestrais (nó pai, avô, etc.) do nó corrente e o próprio nó corrente
attribute	Seleciona todos os atributos do nó corrente
child	Seleciona todos os nós filhos do nó corrente
descendant	Seleciona todos os nós decendentes do nó corrente
descendant-or-self	Seleciona todos os nós decendentes do nó corrente e o próprio nó corrente
following	Seleciona tudo no documento após o fechamento da tag do nó corrente
following-sibling	Seleciona todos os nós irmãos do nó corrente

namespace	Seleciona todos os nós do mesmo namespace do nó corrente
parent	Seleciona o nó pai do nó corrente
preceding	Seleciona tudo no documento antes da abertura da tag do nó corrente
preceding-sibling	Seleciona todos os irmão antes do nó corrente
self	Seleciona o nó corrente

❒ Tabela 17.9 – XPath Axes

17.5.2.6 - Location Path Expression

Um path expression pode ser absoluto ou relativo. O absoluto começa com "/" e a relativa não.

Em ambos os casos cada path consiste em um ou mais passos, que são separados por uma barra "/". Cada passo é executado no conjunto de nós selecionados do passo anterior. Um passo consiste em:

- Um axis (define a árvore de relacionamento entre os nós selecionados e o nó corrente)
- Um nó-teste (identifica um nó dentro de um axis)
- Um ou mais predicados

Exemplos:

```
/* seleciona todos os elementos receita que são filhos do nó corrente  */
child::receita

/* seleciona o atributo lang do nó corrente */
attribute::lang

/* seleciona todos os filhos do nó corrente */
Child::*

/* seleciona todos os atributos do nó corrente */
Attribute::*

/* seleciona todos os nós filhos do nó corrente de tipo text */
Child::text()

/* seleciona todos os nós filhos do nó corrente */
```

```
Child::node()

/* seleciona todos os elementos receita descendentes do nó corrente */
Descendant::receita

/* seleciona todos os elementos receita antecessores do nó corrente */
Ancestor::receita

/* seleciona todos os elementos receita antecessores do nó corrente, e o corrente também se for um nó receita */
Ancestor-or-self::receita

/* seleciona todos os elementos preco netos do nó corrente */
Child::*/child::receita
```

17.5.2.7 – XPath Operators

Os seguintes operadores podem ser usados nas path expressions:

Operator	Example	Return value
|	//book | //cd	Retorna um nó que tenha um elemento book e um cd
+	6 + 4	10
-	6 - 4	2
*	6 * 4	24
div	8 div 4	2
=	quantidade=13	true – se a quantidade = 13 false – caso contrário
!=	quantidade!=13	true – se a quantidade for diferente de 13 false – caso contrário
<	quantidade<13	true – se a quantidade for menor que 13 false – caso contrário
<=	quantidade<=13	true – se a quantidade for menor ou igual a 13 false – caso contrário

>	quantidade>13	true – se a quantidade for maior que 13 false – caso contrário
>=	Quantidade>=13	true – se a quantidade for maior ou igual a 13 false – caso contrário
or	quantidade=7 or quantidade=13	true – se a quantidade for igual a 13 ou igual a 7 false – caso contrário
and	quantidade>=7 and quantidade<=13	true – se a quantidade estiver entre 7 e 13 false – caso contrário

❒ Tabela 17.10 – Operadores utilizados em XPath

17.5.2.8 – XPath Functions

A API das funções XPath é extensa, e não é necessária integralmente aqui. Mas pode ser facilmente consultada em (http://www.w3chool.org/xpath). Abaixo estão alguns exemplos para ilustrar o uso das funções.

```
concat( “Titulo da receita: ” , /receitas/receita/titulo )
count( /receitas/receita )
position(/receitas/receita)=1
```

Na primeira linha do exemplo acima é concatenado uma string fixa ao título da receita. Na linha seguinte, a função count retorna o número de receitas encontradas. E na última, é testado se a posição atual é igual a 1.

17.6 – XLL – EXTENSIBLE LINKING LANGUAGE:

- Compatível com endereçamento baseado em URL´s
- Links bi-direcionais
- Links indiretos

17.7 – XLINK

Permite a declaração de elos, sem associar semântica ou forma de apresentação. Tipos de elos:

- simples: referência a um único recurso, como em HTML
- estendido: permitem referência a mais de um recurso de destino, entre outras facilidades

Exemplo de elo simples (semelhante à HTML):

```
<receita xml:link="SIMPLE"
        href="http://www.rodrigobuas.no-ip.com/mesrtecuca/receitas.xml">
        …
</receita>
```

17.8 – XPOINTER

Aplica-se quando a referência é para um documento XML e permite localizar um elemento dentro de um documento XML de destino.

```
<receita xml:link="SIMPLE"
        Href="receitas.xml#root().child(1).child(3)">
…
</receitas>
```

Para mais informações sobre este tópico ver em http://www.w3schools.com/xlink/default.asp.

CAPÍTULO 18 – AJAX

18.1 – INTRODUÇÃO

Neste capitulo, fala-se de Ajax, mas o que seria Ajax? Seria mais um ingrediente para a nossa sopa de letrinhas? Ou um novo super-herói? Um time de futebol da Holanda? Um personagem da mitologia Grega? Sim seria a resposta à todas as perguntas anteriores, mas o Ajax que irei apresentar aqui eu diria que é a consolidação de tudo que foi estudado e apresentado até aqui, ou de quase tudo.

A pesquisa realizada neste capítulo se baseia em informações distribuídas pela web, com algumas fontes principais[5][7][23].

18.2 – VISÃO GERAL

O nome é uma abreviação de Asynchronous JavaScript And XML. Não é uma linguagem de programação, não é nenhuma ferramenta que pode ser baixada, nem uma biblioteca a ser instalada. E sim, é um modo de pensar sobre arquitetura de web softwares, técnicas de desenvolvimento para criação de aplicações web interativas. Representa uma mudança radical no desenvolvimento web. É o uso sistemático de diversas tecnologias em conjunto com modelo de solicitações assíncronas de informações.

Com Ajax, as aplicações web podem ser mais rápidas, mais interativas e mais amigáveis. Muitas aplicações web vêm utilizando as técnicas e usufruindo das possibilidades adjuntas ao uso de Ajax. A exemplo são as novas versões dos softwares da família google, como o GoogleMaps e GMail, e o A9.com, o motores de busca de um dos mais famosos site de compras, Amazon.

Ajax utiliza em combinação diversas tecnologias client-side, que rodam no lado do cliente – frequentemente os browser's. E mantém a comunicação com o servidor de forma assíncrona. Incorpora:

- XHTML
- XSLT
- XML
- DOM
- XMLhttpRequest

- JavaScript

Resumidamente, Ajax é o conjunto de várias tecnologias que juntas provêem comunicação com o servidor de forma assíncrona. As tecnologias utilizadas são as já conhecidas, a diferença está no modo como as aplicamos.

Na internet existe uma comparação muito boa, de autor desconhecido, que ilustra a diferença que Ajax tem em relação à web antecedente: “Ela é claramente uma descendência da aplicação clássica baseada em páginas, mas a similaridade não é mais forte do que entre um cavalinho de madeira e uma moderna bicicleta de passeio. Sustentando essas diferenças em mente nos ajudará a criar aplicações web verdadeiramente convincentes”.

As técnicas Ajax agilizam nossas aplicações web. Engordam o lado do cliente e aliviam a carga no lado do servidor. Veremos mais adiante o funcionamento e como os novos conceitos apresentados estarão inseridos neste projeto.

A maior mudança que Ajax introduz no desenvolvimento web é similar à inspiração deste trabalho. Não é técnica, e sim arquitetural. Extinguirá as limitações web e nos dará liberdade para desenvolvermos em um ambiente rico e poderoso com inúmeras possibilidades.

18.3 – BENEFÍCIOS

Web softwares possuem grandes vantagens em relação a softwares tradicionais. São mais leves pois aproveitam do processamento de softwares na maquina do cliente, como os browsers. Não precisam se preocupar com processo de redes e fluxo de dados, pois a internet e os protocolos associados a ela provêem um meio pronto pra isso, abstraindo topologia e outros conceitos de redes. Contam com processamento server-side, que possibilita a centralização do modelo da aplicação. O desenvolvimento de aplicações web são simplificados com linguagens de script. São fáceis de instalar, de dar suporte, atualizar e a distribuição é direta ao público mundial.

Em compensação web softwares não se comparam a softwares desktop se compararmos as interações com usuários. Agilidade e rapidez na resposta e na apresentação da resposta à uma ação disparada pelo usuário são características fortes de um software tradicional.

As mudanças propostas pelo uso das técnicas Ajax destroem as desvantagens que um web software tinha para os softwares tradicionais. E deixa intacta as vantagens que já possuía.

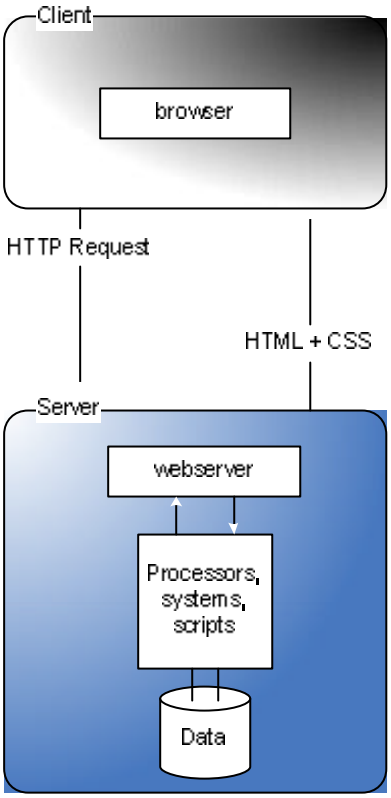
18.4 - WEB ASSÍNCRONA

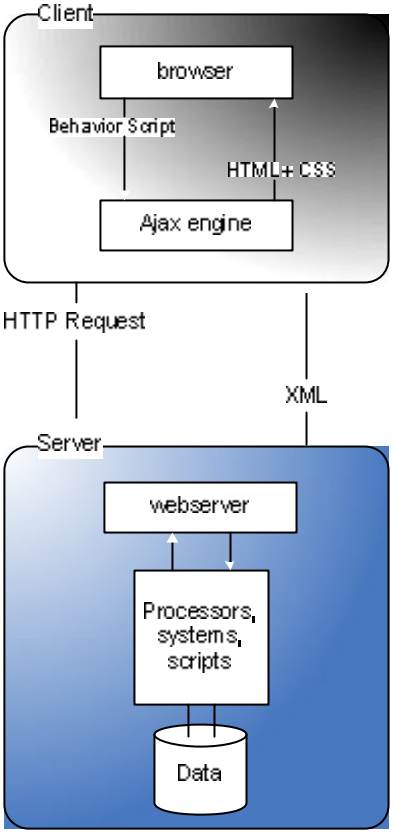
Neste documento já foi visto e discutido o funcionamento da web. É iniciado no cliente através da interface gráfica quando é disparada uma solicitação ao servidor. O servidor então processa as informações e devolve a página ao cliente.

Esse modelo de processamento veio do principio da web, e veio se arrastando até os dias de hoje. Com o tempo, veio se adaptando as necessidades contemporâneas e incorporando as tecnologias que foram surgindo. Mas todas elas continuam da mesma forma, enviam uma requisição e esperam ate a resposta chegar.

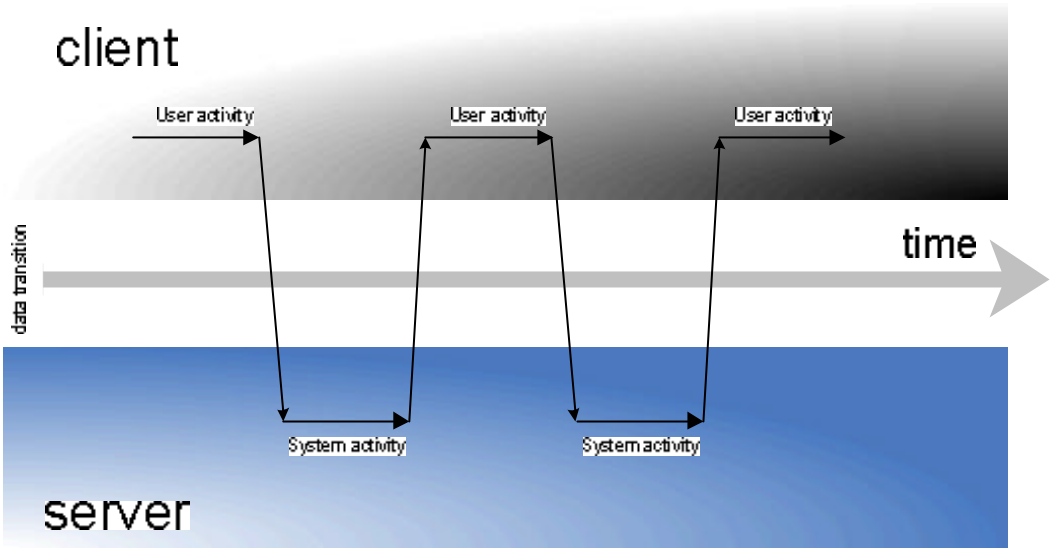
Se pensarmos um pouquinho, veremos que algumas coisas não fazem muito sentido nesse modelo. Uma delas é que uma vez que o usuário já carregou todos os elementos da interface gráfica, por que ele os vê sumirem e depois aparecerem toda vez que é feita uma viagem ao servidor? Em uma aplicação web comum, o usuário esta normalmente está aguardando a página ser recarregada, e frequentemente, dos mesmos dados que continham antes.

A ilustração abaixo [23] mostra a comparação entre o modelo tradicional de web softwares e o modelo Ajax. Repare como o cliente fica mais “gordo”, deixando de ser um dumb terminal e tendo agora mais poder. Enquanto o servidor não necessariamente emagrece. A principal diferença no servidor é ele deixa de receber pedidos desnecessários que só o sobrecarregam.

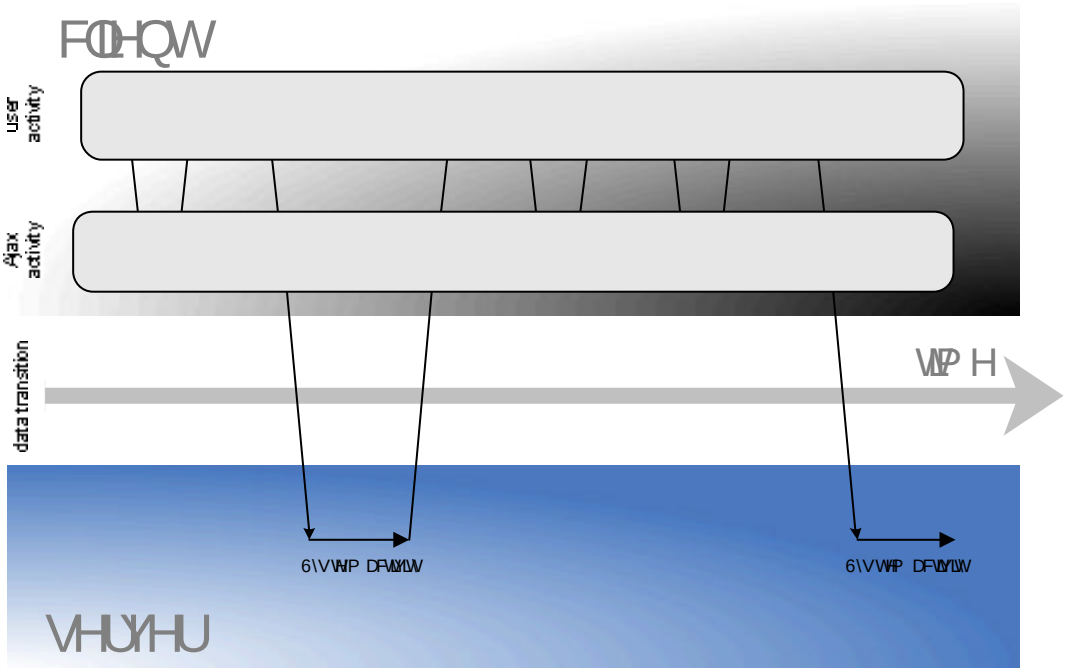

Client
browser
HTTP Request
HTML + CSS
Server
webserver
Processors, systems, scripts
Data


Client
browser
Behavior Script
HTML+ CSS
Ajax engine
HTTP Request
XML
Server
webserver
Processors, systems, scripts
Data

usuário ocorra de forma que o usuário nunca fique com a página me branco esperando que o servidor responda algo.



❒ Figura 18.2 – Análise temporal de atividades cliente/servidor em aplicações web síncronas



❒ Figura 18.3 – Análise temporal de atividades cliente/servidor em aplicações web assíncronas

A engine Ajax intercepta todas as ações dos usuários que normalmente iriam direto para o servidor. Essas ações são avaliadas se realmente é necessário uma viagem ao servidor. Se necessário, ela repassa o pedido para o servidor. Uma ação que é necessária a ida ao servidor é uma requisição de novos dados, submeter dados para processamento ou até carregar novas

informações de interface gráfica. E ações que não são necessárias são ações de navegação, ordenação de uma lista já carregada, filtragem, validação...

A dosagem de quantidade de ações para o servidor e ações para cliente pode ser feita como o desenvolvedor achar melhor. Mas neste trabalho irei propor uma equação balanceada do uso em conjunto de cliente e servidor.

Essas técnicas já são estudadas há anos. Tiveram inicio nos estudos relacionados a Scripts Remotos realizados pela Microsoft, e nos estudos que levaram ao JavaApplet.

18.6 – PORTABILIDADE

Hoje em dia, Ajax tem suporte na maioria dos browser's existentes. Mas a engine Ajax é baseada em JavaScript, o que a torna cross-plataform. Abaixo segue uma lista de browser's que dão suporte a Ajax:

- Internet Explorer 5.0, e browsers baseados
- Opera a partir da versão 8.0 incluindo o Opera Mobile Browser
- Mozilla, Mozilla Firefox, SeaMonkey, Camino, Flock, Epiphany, Galeon e Netscape apartir da 7.1
- Konqueror a partir da versão 3.2, Apple Safari a partir da versão 1.2, e Nokia's Web Browser

Note que Ajax roda na ponta do cliente através de JavaScript, o que obriga ao usuário deixar habilitado em seu browser a opção de JavaScript. Cabe a nós desenvolvedores julgar se vamos impor ao usuário algum requisito, ou se vamos dar uma opção a mais para os usuários de nossas aplicações.

CAPÍTULO 19 – FERRAMENTAS

19.1 – INTRODUÇÃO

Para um processo de desenvolvimento é imprescindível o uso de ferramentas. Existem ferramentas de todos os tipos, das que agilizam o processo de desenvolvimento como um todo, às que aumentam a qualidade do produto final, além das que são extremamente necessárias. Para este projeto foram usadas ferramentas de todos os tipos.

19.1 – ECLIPSE E PHPECLIPSE

Uma ferramenta fundamental que utilizei em tempo integral neste trabalho foi o Eclipse, que também é muito usado na comunidade Java. Usado em conjunto com o plugin para php – PHPEclipse, disponibilizado pelo próprio projeto Eclipse em seu site oficial (http://www.eclipse.org/).

Com ele, é possível criar projetos em PHP, com classes, pacotes e scripts. Também é possível testar e editar localmente. Ele ainda mostra, de forma simples e clara, variáveis e métodos das classes, o que é uma vantagem enorme quando se trata de um projeto grande.

19.2 – DREAMWEAVER

Outra ferramenta muito boa, principalmente na edição dos códigos referentes as partes estruturais, como HTML, XHTML, XML, CSS, entre outros.

Ele possui uma serie de vantagens em ralação à editores de textos comumente utilizados, pois além de colorir o código, ele fornece propriedades de tags HTML, bem como as propriedades CSS, e ainda provê suporte FTP, que permite que os arquivos sejam editados diretamente no servidor.

Essas vantagens permitem que o processo de desenvolvimento web, na fase de construção e edição da parte estrutural e visual, tenham seu tempo consideravelmente reduzidos.

19.3 – FLASH

Um ponto importante deste projeto é ter a interface gráfica de uma aplicação web implementada em diferentes linguagens ou tecnologias. O Flash, da Macromedia, cada vez

mais vem ganhando mercado e espaço no desenvolvimento web. Em função disso e por sua versatilidade, foi escolhido como uma das tecnologias utilizadas neste projeto.

O Flash teve início como um simples plug-in, chamado FutureSplash. A Macromedia logo reconheceu seu potencial e o comprou, juntamente com a ferramenta de criação e re-nomeou como Flash.

Ele possui enumeras vantagens frente a linguagens de marcação. Consistência visual e estrutural, facilidade de criação e manipulação de elementos de interface-gráfica, apoio nativo de linguagem com padrão ECMA'Script e compilação do produto final são características que agregam valores importantíssimos, além do fato de que projetos em Flash vêm agradando muito os usuários finais principalmente na interface-gráfica.

Possui também desvantagens frente a tecnologias .NET e JavaApplet. A principal delas é que o Flash não foi criado para o propósito de desenvolver softwares pesados. Consequentemente a construção de forms não é completamente estruturada.

19.4 – PHPMyAdmim

O PHPMyAdmin é uma ferramenta escrita em PHP para administração de banco de dados MySQL. Possui uma interface web que possibilita administração via browser. Remotamente, não necessita de acesso à maquina servidor.

Com ele podemos criar tabelas, remover, editar, e executar consultas SQL. É Disponibilizado em diversas linguagens.

Para mais informações sobre essa ferramenta acesse: http://www.phpmyadmin.net/.

19.6 – Uso Geral

Entre outros softwares que são necessários estão os clientes de FTP; o software de gerenciamento de usuários, permissões e tabelas do banco de dados; servidor de email; e o software de controle de versões - Subversion (SVN), repositório de arquivos, e TortoiseSVN como cliente.

Por fim foram usados ferramentas do Office, como por Exemplo o Project, para gerenciamento do projeto, assim como o Visio, para criação dos modelos, e o Word para a

criação deste documento e de outros. Além de conversores para PDF e XHTML para publicação e exposição do projeto e dos códigos produzidos.

CAPÍTULO 20 – WEBENGINE

20.1 – MOTIVAÇÃO

Acredito que a linha de desenvolvimento de web softwares é um nicho muito grande, expansível e ainda pode ser considerado pouco explorado no mercado. Sinto que ainda não temos disponíveis ferramentas suficientemente simples e eficientes que atendam as necessidades desse nicho.

A implementação de uma aplicação web que esteja adequada ao meio, baseada em uma arquitetura que atenda suas necessidades, e que se enquadre ao paradigma web, poderiam ter um tempo inicial de desenvolvimento muito grande, já que seria necessário desenvolver além da aplicação em si, também uma infra-estrutura que responda à arquitetura.

Certamente, para diversos tipos de softwares rodando em um mesmo ambiente, facilmente poderíamos encontrar padrões, em níveis mais básicos, que serviriam como base para muitos casos. A identificação e a implementação prévia desses padrões poderiam reduzir o custo, tempo/pessoal, do desenvolvimento de projetos.

A proposta deste projeto é primeiramente a identificação de padrões e a descrição de uma arquitetura que atenda essas necessidades básicas. A partir desta, a proposta é construir, de uma forma genérica, um "framework" que forneça a implementação básica necessária da arquitetura. Essa engine deverá ser centralizadora de processamento, modularizada para poder ser facilmente estendia ou atualizada e conterá classes fornecedoras de diversos serviços.

Muitas aplicações web, hoje em dia, ainda sofrem com problemas de compatibilidade e acessibilidade em função da não total adoção dos padrões, tanto do lado dos fornecedores de browser como os próprios desenvolvedores web. Devido a este fato, adicionalmente a engine terá que estar atenta aos padrões web recomendados pelos órgãos responsáveis, bem como fornecer base para utilizá-los. Deverá ser extremamente flexível e multi-plataforma.

Terá como um de seus principais objetivos desassociar inteiramente o processamento principal da aplicação das demais entidades envolvidas, ou seja, dados, estrutura de marcação... Para que seja possível, em uma mesma aplicação, acessarmos seus dados através de interfaces gráficas – front-end - desenvolvidas em diferentes linguagens de programação.

Adicionalmente, a engine deverá ser de fácil utilização e compreensão, consistente, bem documentada e que dê suporte à serviços comuns em aplicações web de diferentes finalidades. Deverá também estar em constante crescimento, possibilitando que novos serviços ou funcionalidades ou classes sejam inseridas no sistema.

20.2 – OBJETIVO DO PROJETO

- Prover base estrutural para o desenvolvimento web.
- Desassociar o processamento da interface gráfica com o usuário.
- Permitir que uma mesma aplicação tenha diversos frot-ends de tecnologias diferentes.
- Promover o desenvolvimento multi-plataforma.
- Promover e prover a reutilização de componentes para web, criando códigos reutilizaveis.

Os objetivos desse projeto final de graduação são simplificados indicando duas metas principais:

- Descrição de uma arquitetura de softwares web que atenda aos objetivos.
- Implementação de um framework baseado na descrição da arquitetura proposta

❒ Figura 20.1 – WebEngine – Framework baseado na arquitetura proposta.

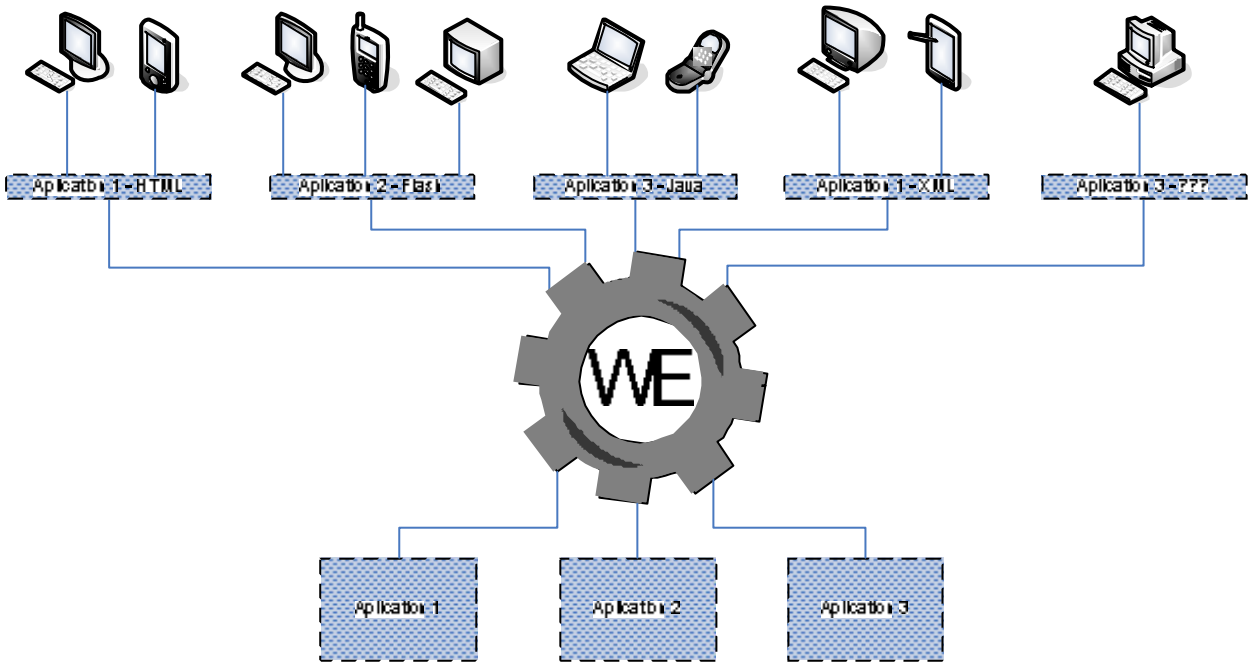
20.3 – PROJETO CONCEITUAL

Este projeto consiste na descrição de uma arquitetura de software, bem como na implementação de um framework baseado na arquitetura descrita.

A partir dos estudos realizados sobre web, suas vantagens e desvantagens frente à softwares tradicionais, de estudos sobre tecnologias web e de estudos de arquitetura e padrão de software, neste projeto, irei descrever uma arquitetura voltada para softwares web.

A arquitetura descrita tem como alvo os softwares de pequeno a médio porte emergidos na web. Esta então, deverá estar atenta aos novos paradigmas web descritos anteriormente neste documento. Deverá aproveitar as vantagens que o meio oferece e possuir características que minimizam as desvantagens.

Os pontos fortes dessa arquitetura são desacoplamento e centralização do processamento principal de uma aplicação. De modo que possamos separar a interface gráfica do modelo da aplicação, permitindo assim que uma mesma aplicação possa ter vários "front-ends" implementados com diferentes tecnologias. Consequentemente terá que prever portas de comunicação para diferentes linguagens.



❒ Figura 20.2 – Pontos fortes do framework – centralização do processamento e desacoplamento da interface gráfica.

Outros pontos relevantes na arquitetura é ser multi-plataforma, modularizada, flexível e de fácil manutenção e evolução. Para tanto, o framework será implementado utilizando tecnologia server-side, e utilizando técnicas de orientação a objetos. Adicionalmente, será implementado seguindo um padrão de codificação, também descritos anteriormente neste documento. Também será descrito na arquitetura um modelo de manipulação de dados.

A escolha da linguagem para desenvolvimento do framework foi o PHP, pois tem algumas vantagens sobre seus concorrentes, como por exemplo, ASP. Outras vantagens já foram antes

apresentadas neste documento, mas reforçando, uma delas é que ele é "open-source", outra é que o PHP é uma das linguagens mais difundidas no Brasil e no mundo para web. É também uma linguagem extremamente poderosa, que esta em constante atualização. Possui uma extensa biblioteca de serviços web, tem bom desempenho, além de ter sintaxe semelhante a C, conhecida por grande parte dos programadores. Outro fator relevante é que PHP é uma linguagem híbrida – tem suporte à orientação a objetos, e orientação a ações – pode possuir elementos procedimentais trabalhando em conjunto com classes de objetos.

20.4 – CARACTERÍSTICAS DA ENGINE

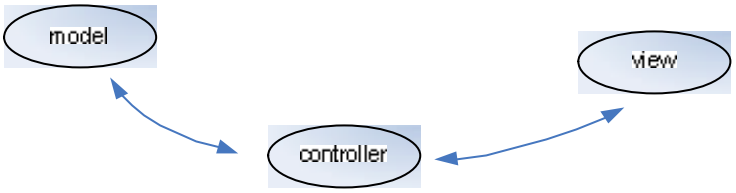
- Biblioteca orientada à objetos.
- Baixo nível de acoplamento.
- Multi-plataforma.
- Projeto modularizado.
- Linha de processamento principal pré-definida(pipeline de processamento).
- Modelo de aplicação consistente.
- Porta de comunicação para diferentes front-ends.

20.5 – PROJETO ARQUITETURAL

A arquitetura proposta da engine é baseada em dois modelos de arquiteturas básicas conhecidas, Cliente-Servidor, onde cada cliente é visto como um subsistema (applet). O servidor é onde está o processamento principal da aplicação. Cada applet é esquematizado com uma variação da arquitetura MVC (Model – View – Controller)[3], onde na verdade, para centralizar o processamento principal da aplicação, o modelo se encontra no servidor.

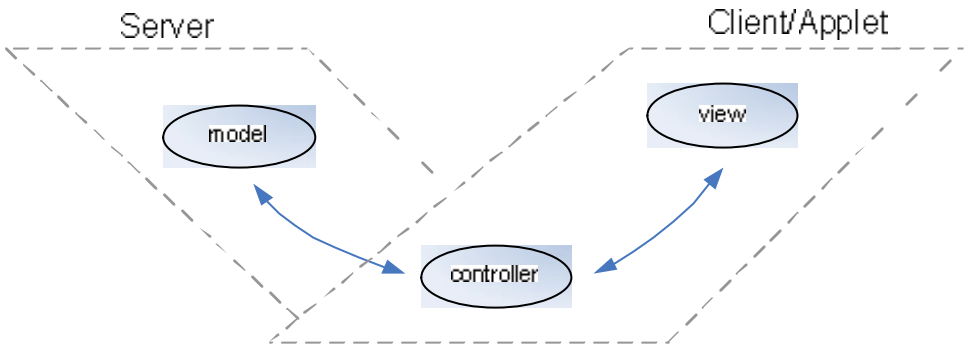
A adoção do MVC inicialmente atende alguns requisitos desejados neste projeto. Sua idéia principal é a separação entre o modelo da aplicação e a interface gráfica com o usuário. O controlador tem o papel de manter os dados que são mostrados através da interface gráfica consistente com a base de dados do modelo da aplicação.

Existem várias formas de implementar o MVC, uma delas é a utilizada como base para a "engine" e esta representada na figura abaixo.



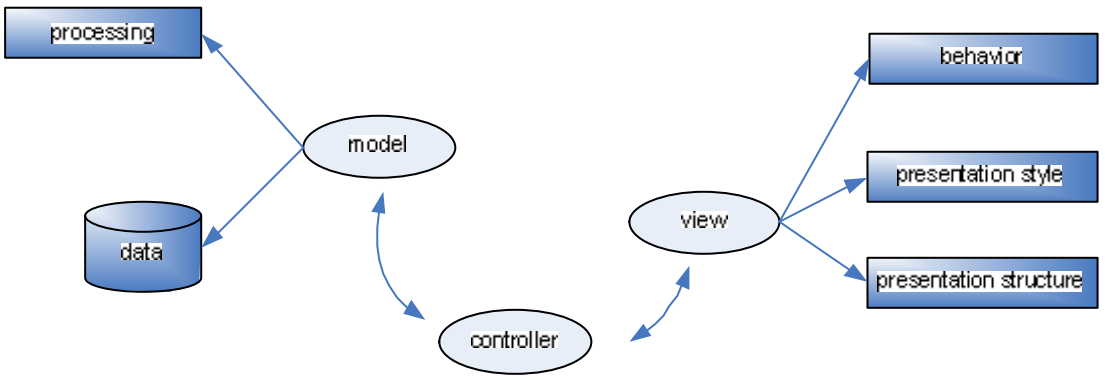
❒ Figura 20.3 - MVC padrão – A figura ilustra uma interpretação do pattern MVC.

A figura abaixo ilustra a arquitetura adotada localizando as entidades do pattern por sua distribuição geográfica.



❒ Figura 20.4 - MVC adaptado à distribuição geográfica.

Adaptando esse padrão aos padrões estruturais web, descritos anteriormente neste documento, adicionei alguns detalhes a essa representação, como mostra a figura abaixo:



❒ Figura 20.5 – MVC adaptado aos padrões estruturais da web.

Primeiramente, separei o modelo da aplicação em duas partes, processamento e os dados. Onde os dados são todas as informações relevantes do software e podem estar armazenados em um servidor de banco de dados qualquer ou em arquivos XML ou qualquer outro tipo de armazenamento.

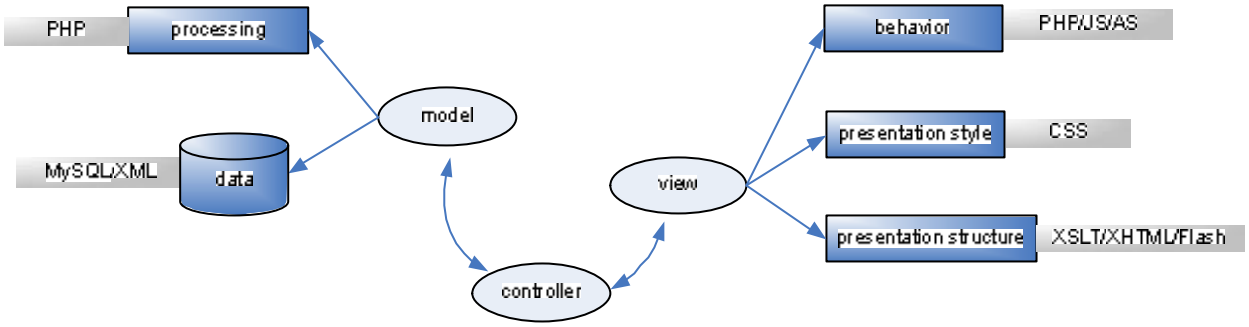
O processamento é o centro do software, a essência do modelo da aplicação. Para que seja desassociado de apresentação e estrutura, ele é responsável por somente como deve ser processada cada informação. A melhor opção para mantê-lo multi-plataforma e sendo um concentrador do modelo da aplicação é tê-lo rodando em um servidor. Assim, o processamento é implementado em uma só linguagem e em um só local. O fato de estar em um único local, oferecendo serviços a clientes, facilita atualizações e evoluções posteriores no software.

Para atender aos padrões estruturais da web o view foi divido em três partes, estrutura e estilo. Que respectivamente estão relacionadas à estrutura que serão apresentados os dados da aplicação e o estilo da apresentação.

Enquanto o comportamento é responsável pela parte do modelo da aplicação que responderá às ações disparadas por usuários. Diferente do processamento o comportamento está mais ligado à cada interface gráfica.

Como mencionado no capítulo de “Websites Compatíveis e Acessíveis”, para cada tipo de interface gráfica, implementadas com diferentes tecnologias, possivelmente seria necessária uma nova implementação das partes de comportamento, apresentação e, ocasionalmente da estrutura.

Baseado nos testes de tecnologias apresentados neste documento e nas experiências aqui também documentadas, optei por tecnologias que mais atenderiam às necessidades do projeto. Elas estão ilustradas na figura abaixo separadas por item arquitetural.



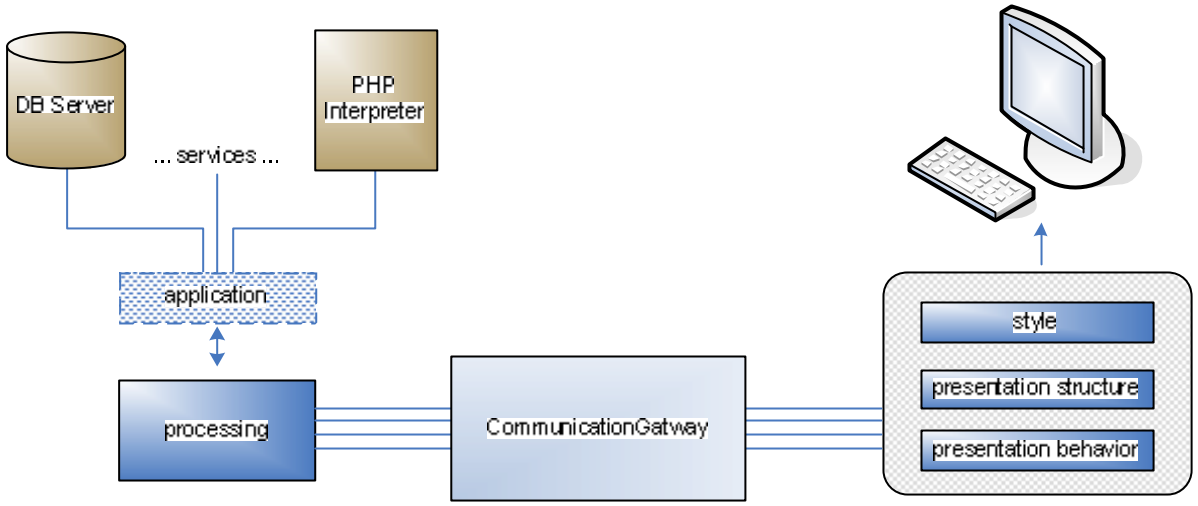
❒ Figura 20.6 – Tecnologias utilizadas no projeto ilustradas por item arquitetural.

Pelos motivos já aqui comentados, no processamento utilizei PHP rodando no servidor. Parte do comportamento também foi implementada em PHP(comentarei sobre este fator mais adiante). Outras partes do comportamento em JavaScript, e outras em ActionScript.

O comportamento desenvolvido em JavaScritp está ligado ao front-end estruturado em XHTML. Enquanto que o ActionScript esta atrelado ao front-end com estrutura em Flash. O CSS atende às necessidades de ambas as estruturas utilizadas para a parte de estilo da apresentação.

Para o banco de dados optei por usar MySQL e XML em conjunto. Onde os dados estão armazenados em um servidor de banco de dados MySQL. E para o transporte dos dados usei XML, opcionalmente é disponilibilizado o transporte dos dados em outros formatos, como por exemplo um string serializada.

O que até então foi discutido aqui foi uma visão de arquitetura pontual. Em seguida, irei apresentar uma visão macro da arquitetura do projeto, que mostra detalhes da arquitetura já descrita. Mais perto do ambiente onde será desenvolvida. A visão Cliente-Servidor é ilustrada na figura abaixo.



❒ Figura 20.7 – Visão cliente-servidor simplificada da arquitetura do projeto.

A figura acima insere o modelo apresentado no ambiente web. Repare que aqui foi inserida uma nova entidade, representada por "portão de comunicação" ou "communication gatway". Este é uma abstração que engloba partes relacionadas ao processamento e ao comportamento da aplicação. Basicamente, é a parte do motor que contem as interfaces de comunicação entre os diversos front-ends e o processamento da aplicação.

As implementações dessas interfaces perduram para outras aplicações, i.e., uma vez implementada para um front-end, poderá ser reutilizada em outra aplicação, pois trata-se somente da comunicação do motor. Neste projeto, as interfaces de comunicação foram implementadas para front-ends em HTML/XHTML e para Flash, mas ainda existem possibilidades interessantes para Java e .Net.

20.6 – O Conceito Skin

As características da arquitetura adotadas possibilitam que o processamento fique separado do modo como são apresentados os dados. Mas como seria possível realizar essa separação visto que no ambiente que estamos inseridos o comum é ter o código do modelo da aplicação entremeado à interface gráfica (e.g., código PHP entre as marcações HTML)? Para solucionar este impasse, proponho neste tópico o uso do Skins.

Para facilitar o entendimento e simplificar a explicação deste conceito, aqui farei, na maior parte, referencia a linguagens de marcação, porém este conceito também é aplicável às outras tecnologias de interface gráfica diferente.

Para chegar ao nível de desacoplamento desejado é necessário que não haja código misturado às marcações de estrutura e à apresentação. Para tanto, foi introduzido neste projeto um novo conceito, o conceito “skin”.

Existem diversas tentativas diferentes de implementação de modelos que separam o processamento da apresentação. Muitas, inclusive criam uma outra linguagem, com sintaxe diferente. Outras, não separam inteiramente. A forma que proponho, apresentada neste trabalho difere das demais pois além desse benefício, trás consigo outros:

- Permite que o desenvolvedor construa a interface gráfica como desejar, e.g. construindo tabelas, listas ...
- Concentra todo o código de processamento da aplicação.
- Cria um API simples, para acesso a dados e variáveis, mas não obriga ao desenvolvedor a aprender e utilizar outra sintaxe – outra linguagem.
- A configuração é rápida, simples e direta.
- Permite que o desenvolvedor utilize diversas tecnologias para construir a interface gráfica(e.g. JavaScript, PHP, XSL)

No desenvolvimento web é normal construir páginas com dados variáveis – páginas dinâmicas. Contudo, o desacoplamento poderia ficar comprometido se construíssemos partes da apresentação no modelo da aplicação – processamento principal. A “skin” é responsável somente pela estrutura da aplicação, contendo apenas marcações estruturais e acessos à variáveis. Para ilustrar isso vamos analisar o exemplo a seguir.

Imagine que dentro das classes e métodos de processamento eu construa dinamicamente partes da interface gráfica, o que é muito comum, como por exemplo construir tags de estrutura. Isso fortaleceria o acoplamento.

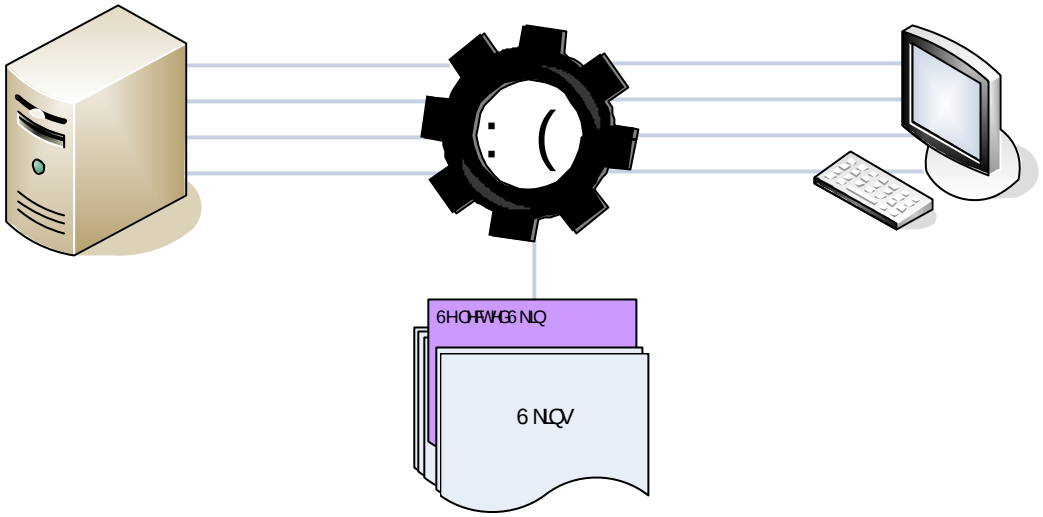
Para solucionar este problema, sem restringir o potencial do PHP foi previsto na arquitetura apresentada que as skins terão uma área de pré-processamento. Nesta área, o desenvolvedor poderia colocar códigos da linguagem que desejar, cuja única proposta é acessar variáveis para construir a interface com usuário - código comportamental. Esses códigos são responsáveis estritamente pela construção de marcações estruturais e ficam agrupados em um bloco separado no início de cada skin. Este detalhe mantém o grau de acoplamento desejado. Observe o exemplo a seguir:

```
<?php
/*******
* Skin – área de pré-processamento
*******/
$strVar             = Environment::GetVar( “strMyVar” );
$arrVars            = Environment::GetVar( “arrVars” );
$strLista           = “<ul>”;
foreach( #arrVars as $strVarAux )
{
        $strLista       .= “<li>” . $strVarAux . “</li>”
}
$strLista           = “</ul>”;
/* Final do bloco de pré-processamento da skin */
?>
<html>
…
        <p>Variável 1: <?= $strVar ?> </p>
        <p>Array de variáveis: <?= $strLista ?> </p>
…
</html>
```

O exemplo anterior ilustra a utilização de skins. Fica claro neste exemplo, a separação da modelo da aplicação e da interface gráfica com usuário. Repare que não existe processamento do modelo na skin. No código, não há nenhuma referencia a banco de dados, nem resolução de tabelas de relacionamentos, nem muito menos como essas informações forma processadas. O único processamento existente na skin é a recuperação das variáveis desejadas na skin e a construção de uma lista de variáveis, que esta diretamente ligada à interface gráfica.

Inserido na engine, esse conceito é implementado de forma bem simples, como podemos ver no exemplo anterior. Ainda é possível o uso de templates, que chamei de skeleton para acompanhar o conceito skin.

Ainda resta saber como a engine seleciona a skin para ser apresentada para o usuário. Mais adiante veremos detalhes do funcionamento da engine. Mas por hora, basta saber que em toda execução será indicado qual skin deverá ser apresentada. Ficando a cargo da engine selecionar a skin do conjunto de skins disponíveis.



❒ Figura 20.8 – O conceito Skin.

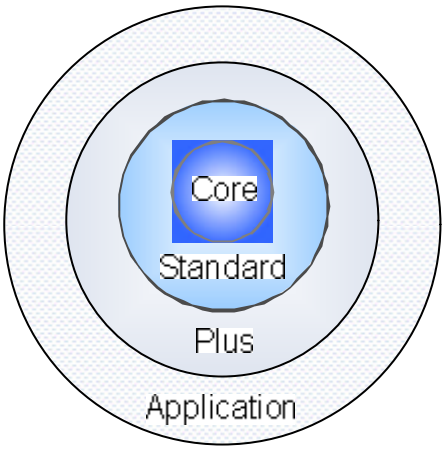
20.7 – BIBLIOTECA PADRÃO

Em conseqüência da arquitetura adotada, e a partir dos padrões encontrados em aplicações web, desenvolvi uma biblioteca totalmente orientada a objetos e a dividi em duas partes: Biblioteca de Processamento (Processing Library) e Biblioteca de Comportamento (Behavior Library).

É importante lembrar que a biblioteca de comportamento é desenvolvida para cada linguagem que se deseja implementar o front-end. Podendo até ter de ser implementada diferentemente para cada aplicação, uma vez que nesta parte do código pode conter scripts de construção de tabelas, listas ou símbolos. Contudo, a Biblioteca de Processamento é, por definição, inteiramente reutilizável.

A biblioteca de processamento da engine foi desenvolvida em camadas, porém camadas não restritivas. Apenas com restrições abstratas para manter a consistência do design da arquitetura. Diferentemente do modelo em camadas costumeiro, cada camada superior tem acesso direto aos serviços de todas as camadas inferiores. Ela é subdividida em quatro partes: “Core Library”, “Standard Library”, “Plus Library” e “Application Library”. A figura abaixo ilustra essa divisão, onde a camada mais superior usufrui dos serviços das mais inferiores.

Na figura, a biblioteca “Behavior” não esta representada porque sua arquitetura depende de cada aplicação.



❒ Figura 20.9 – Divisão da biblioteca padrão da engine.

A “Core Library” contém as classes bases, fundamentais para o funcionamento da engine, e que provêem serviços básicos, e.g. sistema de arquivos, interface de acesso a banco de dados e entidades de funcionamento interno do motor, como por exemplo, o centro de comando de execuções, selecionador de skins, coletor de respostas e interfaces de comunicação ...

“Standard Library”, como o nome indica, contém classes relacionadas a padrões difundidos na web, como por exemplo, manipulação de XML através de DOM, criação de arquivos PDF, classe de serviço de email e FTP, entre outras.

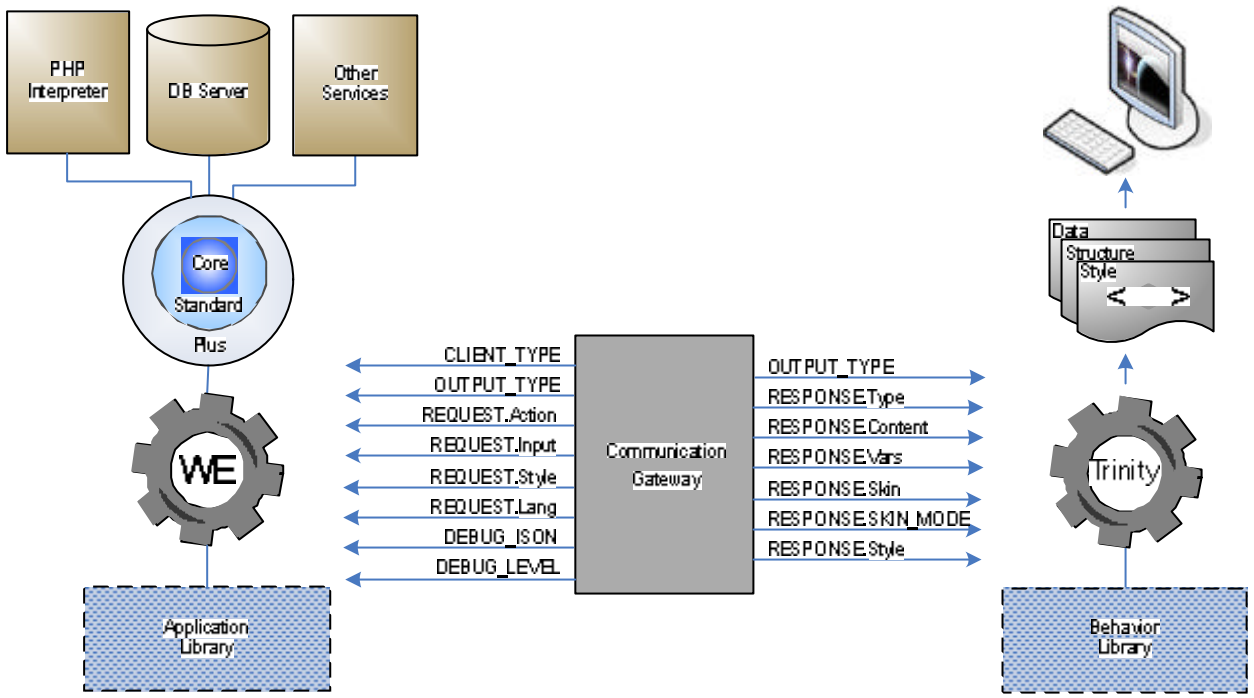
A “Plus Library”, engloba classes que atendem à serviços comumente utilizados por aplicativos web, e.g., controle e cadastramento de usuários, sistema multi-linguagem, referência cruzada, classe de serviços estatísticos, hipermídia, entre outras.

A “application library”, o nível mais alto das bibliotecas, pode fazer uso de todos os recursos das demais. Como o nome já diz, ela é restrita à aplicação, e deve ser implementada para cada uma. Agrupa classes relacionadas ao modelo da aplicação.

Um pouco mais adiante veremos detalhes das entidades principais da biblioteca principal da engine. Mas antes veremos o funcionamento global da engine.

20.8 – FUNCIONAMENTO

Neste tópico, serão mostrados detalhes do funcionamento da engine. Ainda dispostos como uma visão cliente-servidor, serão apresentadas as peças que compõem a engine e como se interligam.



❒ Figura 20.10 – Funcionamento simplificado da engine. Entradas e Saídas.

Na figura 26.10, é ilustrado, em um nível de abstração elevado, o funcionamento da engine. No lado do servidor podemos visualizar as ligações a alguns serviços que a engine utiliza. Para simplificar, não estão representados todos os serviços possíveis, como servidor de email, FTP, entre outros.

Abaixo da engine, encontra-se representada a biblioteca da aplicação. Ao centro da ilustração existe uma porta de comunicação que tem a responsabilidade de transferir os dados para o cliente. É importante ressaltar que para cada tipo de front-end diferente deverá existir uma implementação desta interface. Previamente já foi implementada para clientes tipo HTML, Flash e para Java.

No lado do cliente existe uma entidade que pode ou não existir. Ela implementa técnicas AJAX antes descritas neste documento[Capítulo 18]. A não existência desta entidade implica em não ter as vantagens dessa técnica nas aplicações. Mas não interfere no funcionamento da engine. Devido a sua relevância, receberá um tópico exclusivo neste capítulo. Por hora, basta saber que é chamada de Trinity e se existir, repassará as informações recebidas do servidor ao cliente.

Podemos ver na ilustração que na comunicação são determinadas informações a serem passadas – dados de entrada e saída. Veremos a seguir o detalhamento das entradas e saídas da engine. Provavelmente o leitor não estará familiarizado com todos os nomes de variáveis, tipos e classes mencionados na tabela abaixo. Mas estes serão explicados mais adiante. O foco agora é entender o que entra e o que sai da engine.

Entradas	Descrição
CLIENT_TYPE	Indica o tipo do cliente que esta sendo realizada a comunicação. Os valores possíveis desta entrada são pré-definidos pela engine. Cada um deles corresponde a uma implementação da interface de comunicação. WE_CLIENT_TRANS – TransparentGate – para clientes que enxergam objetos PHP. Em geral é usado para clientes tipo HTML. WE_CLIENT_XML – XMLGate – para clientes XML que fazem uso de Skins escrita como folhas de estilo em XSL. WE_CLIENT_FLASH – FLAGate – para clientes em Flash. WE_CLIENT_JAVA – JavaGate – para clientes em Java.
OUTPUT_TYPE	Em conjunto com o tipo de cliente informado, determinam como será efetuada a comunicação. Indica como os dados repassados e como estarão dispostos na saída da engine. Os possíveis valores para este campo são pré-definidos pela engine. WE_OUT_TYPE_XML – 10 – indica que os dados estarão dispostos na saída da engine no formato XML. Geralmente e usado para clientes XML. WE_OUT_TYPE_PHP – 11 – indica que os dados não sofrerão conversão. Geralmente é usado em clientes que enxergam objetos PHP e utilizam o TransparentGate. WE_OUT_TYPE_SERIAL – 12 – indica que os dados estarão dispostos de forma serializada. Uma opção genérica de comunicação que permite a comunicação com qualquer tipo de tecnologia utilizada para o front-end. Mas força que este cliente saiba fazer o processo reverso – unserializar.
REQUEST.Action	Indica a ação que será executada pela engine. Esta ação deve estar cadastrada nas configurações da aplicação da mesma forma como serão chamadas. Caso contrário, o centro de comandos da engine(ExecCmd) não a reconhecerá.
REQUEST.Input	Variáveis de entrada para a ação.

REQUEST.Style	Indicam qual estilo de apresentação deve ser apresentado. Não precisam ser enviados em toda requisição. Basta indicá-lo uma única vez e a engine armazenará a opção escolhida. Caso deseje trocar o estilo de apresentação, basta indicar o novo estilo. Este estilo deve estar cadastrado nas configurações da aplicação da mesma forma como serão indicados. Caso contrário, a engine não a reconhecerá o estilo e manterá o anterior.
REQUEST.Lang	Indicam qual linguagem esta a aplicação. Não precisam ser enviados em toda requisição. Basta indicá-lo uma única vez e a engine armazenará a opção escolhida. Caso deseje trocar de linguagem, basta indicar uma nova. Esta linguagem deve estar devidamente cadastrado nas configurações da aplicação. Caso contrário, a engine poderá não conseguir carregar o arquivo de resource.
DEBUG_ISON	Indica se o debug está ligado ou não. Em geral é indicado somente em fase de desenvolvimento da aplicação ou para detecção de erros.
DEBUG_LEVEL	Indica o nível do debug. Quando o debug está desligado, funciona para criar logs da aplicação. Mas cuidado, o seu uso reduz a desempenho da engine pois a todo momento cria arquivos de log.

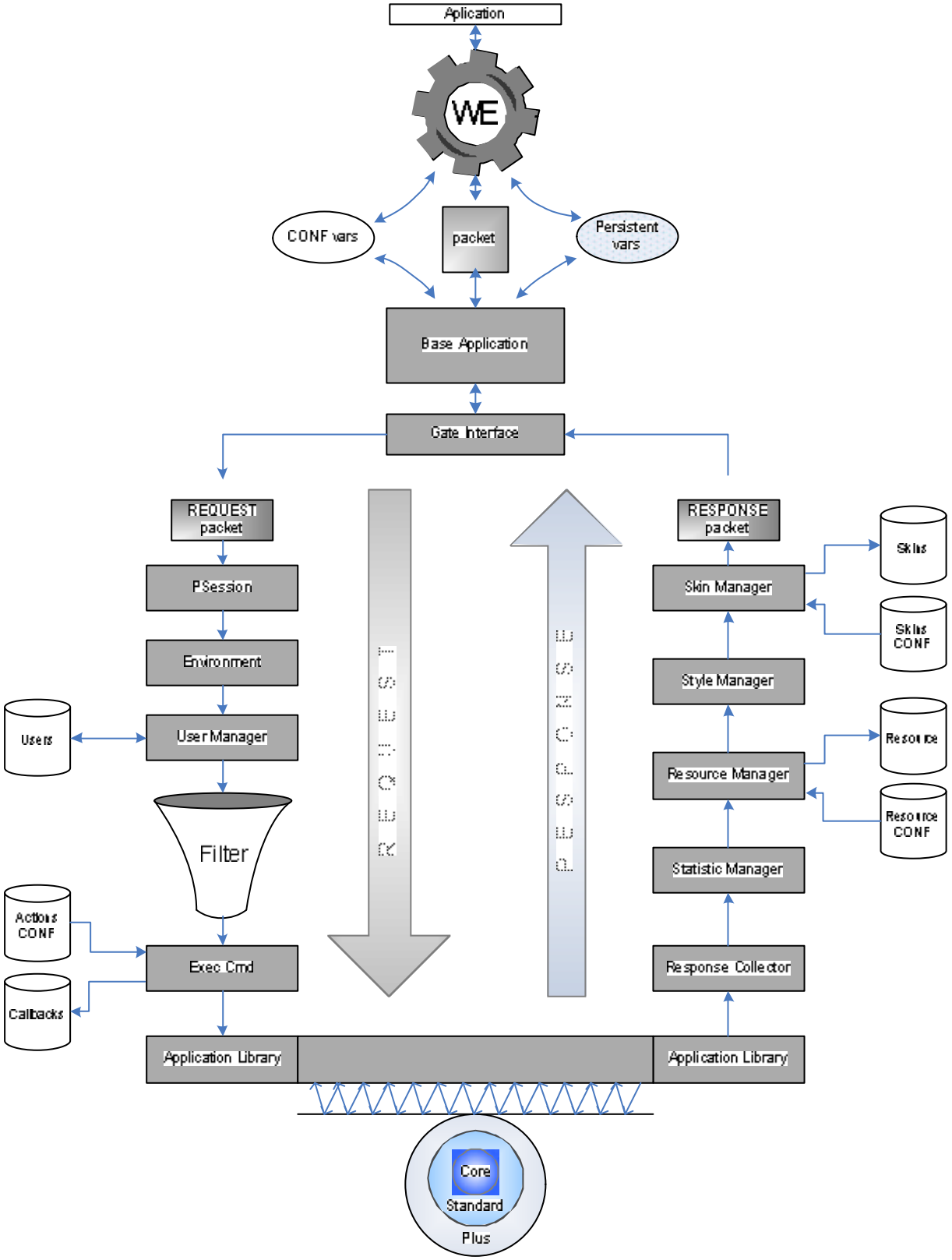
❒ Tabela 20.11 – Entradas da engine - REQUEST.

Saídas	Descrição
OUTPUT_TYPE	Indica o tipo do output. O mesmo que foi requisitado na ação.
RESPONSE.Type	Indica o tipo de resposta obtido da ação requerida.
RESPONSE.Content	Conteúdo da resposta. Varia conforme a ação requerida.
RESPONSE.Vars	Variáveis persistentes de sessão.
RESPONSE.Skin	Em forma de string. Indica qual Skin deverá ser aberta. O modo como é processada depende do Gate de comunicação. Por exemplo, de uma forma genérica é indicada somente. Mas para front-end tipo HTML, a Skin é selecionada no banco de Skins para ser apresentada.
RESPONSE.SKIN_MODE	Indica o modo como a Skin deverá ser apresentada. Para este resposta existem tipos pré-definidos:

	WE_SKIN_MODE_EDIT – abre a Skin em modo de edição. WE_SKIN_MODE_VIEW – abre a Skin em modo de visualização. Mas é uma simples indicação. O tratamento cabe ao front-end. Os usuários da engine que não desejarem implementar essa funcionalidade podem simplesmente ignorar este parâmetro e construir Skins diferentes para edição e para visualização.
RESPONSE.Style	Em forma de string. Indica qual Style deverá ser aberta. O modo como é processada depende do front-end. De uma forma genérica é indicada somente.

❒ Tabela 20.12 – Saídas da engine - RESPONSE.

A seguir, veremos a mesma arquitetura, avançando um pouco mais no nível de abstração. O diagrama a seguir ilustra o funcionamento da engine. Nesta representação nota-se algumas das entidades já discutidas neste capítulo. Além de outras adicionais que aqui aparecem meramente para facilitar o entendimento do motor. Mas que na verdade estão embutidas nas já apresentadas anteriormente.



❒ Figura 20.13 – Arquitetura geral da WebEngine com entidades graficamente estendidas.

O modelo é caracterizado por um pipeline de execução, que é divido em duas etapas, uma para cada direção de tráfego de dados. A primeira, “REQUEST”, tem seu inicio no cliente - usuário da aplicação - e termina no servidor – engine. E a segunda acontece na direção reversa, servidor-cliente, “RESPONSE”. Ambas serão detalhadas a seguir.

O tráfego de dados no sentido “REQUEST” é iniciado quando um usuário faz uma requisição ao servidor. Esta pode ser dada por um botão ou um outro disparador qualquer. O sistema, então, reúne as informações necessárias, que são os dados relevantes do ambiente de execução. Nesses dados, estão as variáveis de input, tecnologia de front-end e a ação que deve ser executada, REQUEST.Action.

Essas informações são repassadas para a interface de comunicação relacionada à tecnologia de front-end. Onde são traduzidas para um pacote de REQUEST entendido pela engine. O modo de envio deste pacote depende da tecnologia utilizada no front-end e da forma que foi implementada essa interface. Por exemplo, quando o front-end é em alguma linguagem de marcação, a comunicação é feita diretamente com PHP, passando objetos PHP, enquanto que para um front-end Flash, os dados são serializados antes de serem enviados.

Em seguida, são realizados os processos de inicialização e recuperação do ambiente e sessão que a aplicação está rodando. Após a inicialização, é feito, quando configurado na aplicação, o controle de usuários, passando à frente a permissão, além dos dados já existentes no pacote.

A próxima entidade na seqüência do pipeline é uma brecha planejada para que o desenvolvedor da aplicação tivesse a possibilidade de agrupar códigos que seriam normais a qualquer ação requisitada. Chamada de filtro (Filter), o arquivo correspondente é indicado à engine quando chamada. Essa entidade dá maior liberdade ao desenvolvedor.

Após a execução do filtro, o centro de execução de comandos (ExecCmd) assumirá o controle da engine. Ele controla as ações permitidas na aplicação. Todas as ações e as permissões são configuradas no arquivo de configuração. Cada ação está relacionada á uma callback, que será executada caso a permissão tenha sido fornecida.

```
/* Exemplo de configuração de callback*/
$_CONF[WE_EXEC_CMDS]['NOME_DA_AÇÃO']  = 'CALLBACK_NAME';
```

Essas callbacks, no design deste projeto, estão agrupadas na “application library” e farão uso direto das classes do modelo da aplicação. Toda callback registrada deve atender a um padrão.

Onde cada uma deve sempre retornar um array contendo três valores: ResponseType, ResponseContent, ResponseSkin, que informam, respectivamente, o tipo de resposta, o conteúdo da resposta, a skin que deverá ser apresentada ao usuário.

Finalizando o processo REQUEST ao final da callback, é iniciado o processo reverso, RESPONSE, cuja primeira providência é a reunião dessas respostas recebidas da callback. Formado o pacote de retorno ao cliente da engine, o pacote servirá ainda de informações para alguns processos.

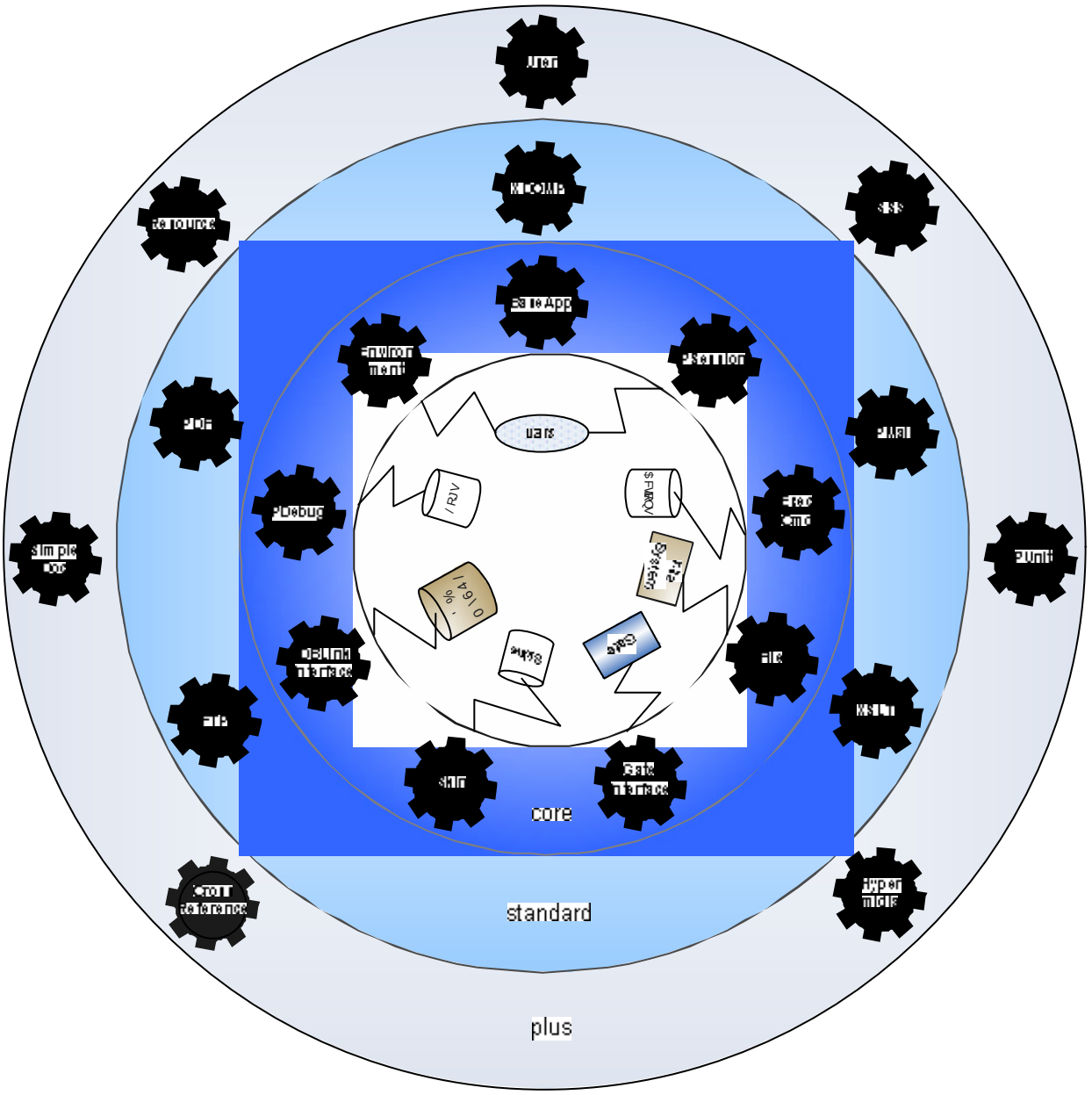
Em seguida, são salvas as informações estatísticas. Na implementação realizada neste projeto, os dados estatísticos são salvos em formato XML e agrupados mensalmente. Logo após esse processo, é carregado o arquivo resource correspondente. Este é destinado a conter dados invariáveis que podem ser utilizados na construção da interface gráfica com o usuário.

Após o carregamento do arquivo resource também são adicionados ao pacote de resposta, informações de estilo de apresentação. Também são processados as informações da skin – ver sobre o conceito skin e como são processadas.

Finalizando o processo RESPONSE, as informações de resposta são convertidas pela interface de comunicação relacionada a tecnologia correspondente à aplicação. Em seguida esse pacote é retornado à aplicação.

20.9 – CLASSES WEBENGINE

A figura abaixo ilustra as bibliotecas da engine, com suas respectivas classes. Para melhor entendimento, também são ilustrados os serviços fornecidos aos quais as classes estão ligadas e suas funções dentro da engine.

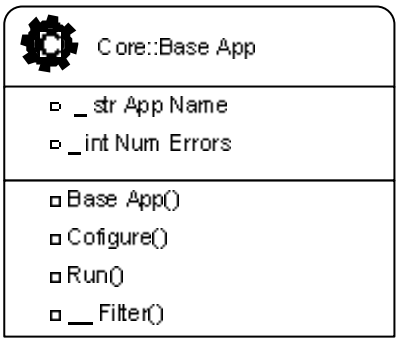


❒ Figura 20.14 – Classes e bibliotecas da WebEngine.

20.10 – API DA BIBLIOTECA

20.10.1 – CORE::BASEAPP

Esta classe é responsável pelo processamento principal da engine. Nela é determinada a ordem de processamento de qualquer aplicação que é baseada na engine. Esta classe também é responsável pela configuração da engine em função das definições efetuadas no arquivo de configuração.

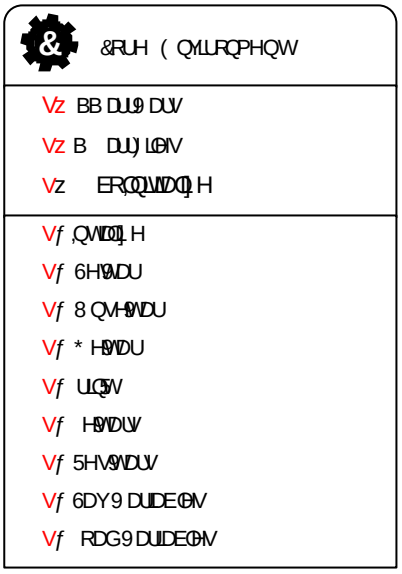


Método	Parâmetros	Descrição
Base App (construtor)	str Application Name str Filter File str Ini Conf File str Conf File	Construtor da classe. Armazena informações, aciona a configuração e inicializa a aplicação. Parâmetros: nome da aplicação ; caminho do arquivo filtro da aplicação ; caminho do arquivo de configuração (.ini) ; caminho do arquivo de configuração (.php)
Configure		Lê os arquivos de configuração e prepara a aplicação para rodar.
Run		Inicializa o processamento da aplicação.
__ Filter		Processa informações do arquivo filtro.

❒ Tabela 20.15 – WebEngine API – BaseApp

20.10.2 – CORE::ENVIRONMENT

Classe responsável por todas as variáveis voláteis do sistema. Com ela, é possível armazenar temporariamente, carregar e transportar variáveis. Trata-se de uma classe estática (singleton).

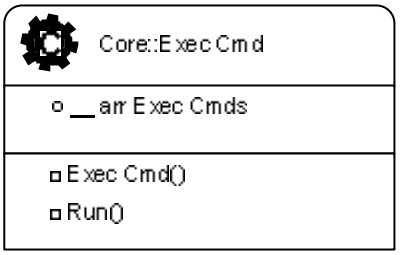


Método	Parâmetros	Descrição
Initialize	arr Input	Inicializa o ambiente de variáveis. Parâmetro: variáveis de input.
SetVar	str Var mix Value	Salva uma variável temporariamente. Parâmetros: nome da variável ; valor da variável
UnsetVar	str Var	Elimina uma variável. Parâmetro: nome da variável.
Get Var	str Var	Recupera uma variável do ambiente. Parâmetro: nome da variável
Print R		Imprime as variáveis de ambiente na tela.
GetVars		Retorna uma array com as variáveis setadas.
ResetVars		Elimina todas as variáveis.
SaveVariables		Slava as variáveis em um arquivo ini
LoadVariables		Carrega variáveis de um arquivo ini

❒ Tabela 20.16 – WebEngine API – Environment

20.10.3 – CORE::EXEC CMD

Classe de controle de execuções. Nela então relacionadas as ações e callbacks da aplicação. É nela que são realizadas as chamadas de ações.

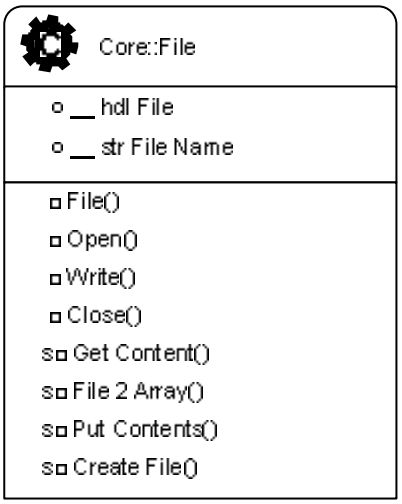


Método	Parâmetros	Descrição
Exec Cmd (constructor)	arr Exec Cmds	Construtor da classe. Parâmetro: array contendo as ações e suas respectivas callbacks.
Run	str Exec Cmd mix Input	Executa a chamada da ação (str Exec Cmd) passando como entrada o valor de mix Input.

❒ Tabela 20.17 – WebEngine API – ExecCmd

20.10.4 – CORE::FILE

A classe File provê acesso e à manipulação de arquivos.



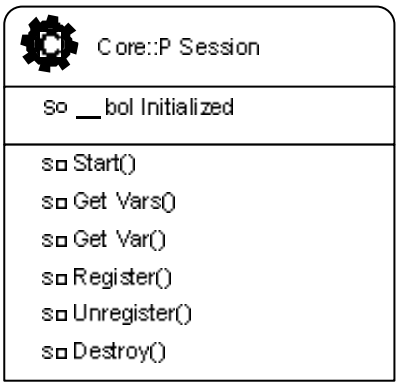
Método	Parâmetros	Descrição
File (constructor)	str File Name str Mode int Include Path Context	Construtor da classe. Automaticamente abre o arquivo chamando o método Open
Open	str File Name str Mode int Include Path Context	Abre o arquivo strFileName no strMode.
Write	str Data int Length	Escreve dados em um arquivo aberto
Close		Fecha um arquivo aberto
Get Content	str File Name int Include Path Context	Retorna o conteúdo de um arquivo
File2Array	str File Name int Include Path Context	Retorna uma array, onde cada elemento na array é uma linha do arquivo
PutContents	str File Name str Data	Carrega para dentro do arquivo strFileName os dados str Data

	int Flag Context	
Create File	str File Name str Data int Flags Context	Cria um arquivo strFileName com conteúdo str Data

❒ Tabela 20.18 – WebEngine API – File

20.10.5 – CORE::PSESSION

Nesta classe encontram-se métodos de manipulação de variáveis persistentes na aplicação. Nela também estão os tratamentos de sessões. É mais um singleton pertencente a engine.

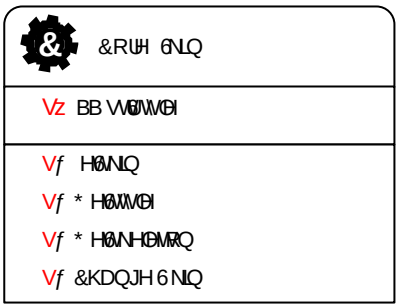


Método	Parâmetros	Descrição
Start		Inicializa a sessão e a classe
Get Vars		Retorna uma array contendo as variáveis persistentes da aplicação
Get Var	str Var	Retorna o valor de uma variável
Register	str Var mix Value	Registra uma variável persistente na aplicação
Unregister	str Var	Apaga o registro de uma variável persistente antes registrada
Destroy		Finaliza a sessão

❒ Tabela 20.19 – WebEngine API – PSession

20.10.6 – CORE::SKIN

Classe para manipulação e controle de Skins.

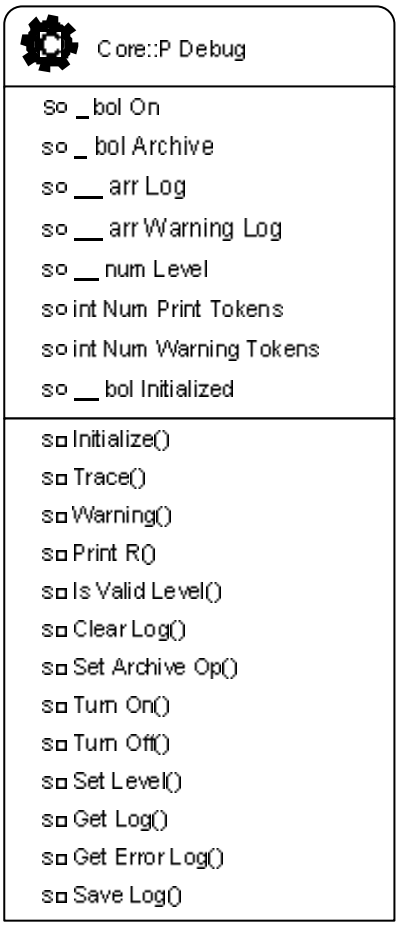


Método	Parâmetros	Descrição
Get Skin	str Skin bol Auto Include	Seleciona a skin strSkin no banco de skins e retorna o campinho da skin. Caso bolAutoInclude seja verdadeiro automaticamente a apresenta ao usuário.
Change Skin	str Skin str Error	Troca a skin de apresentação
Get Style		Retorna o caminho do arquivo de style para ser incluído
Get Skeleton	str Skeleton bol Auto Include	Retorna o caminho do arquivo template(skeleton) para ser incluido. Caso bolAutoInclude é verdadeiro, o arquivo é automaticamente incluído.

❒ Tabela 20.20 – WebEngine API – Skin

20.10.7 – CORE::PDEBUG

Esta classe tem como objetivo acompanhar o processamento da aplicação. Auxilia na detecção de erros e gera automaticamente logs de execução periodicamente ou quando requisitado. Classe estática – padrão singleton.



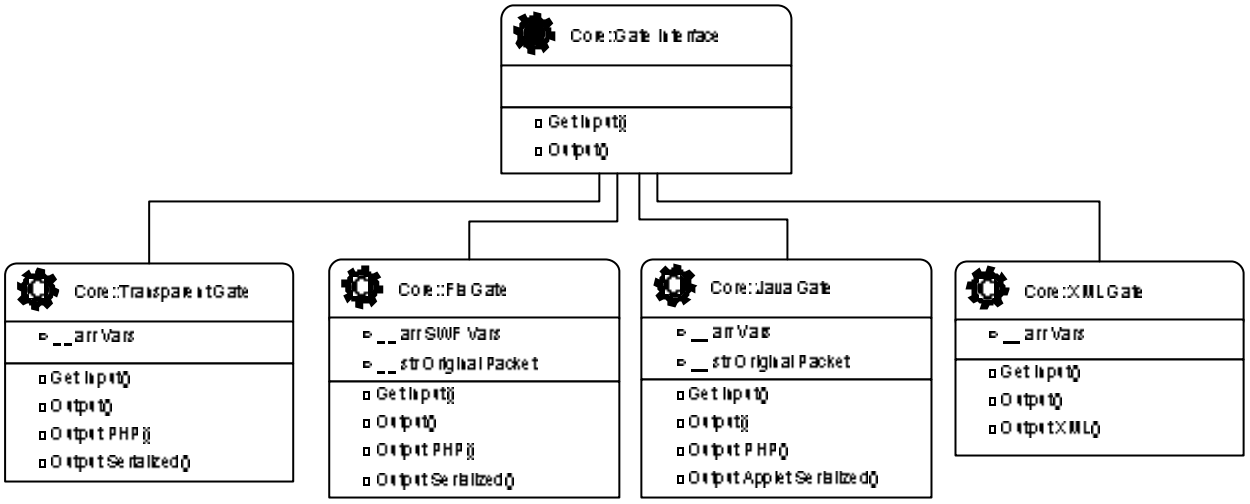
Método	Parâmetros	Descrição
Initialize	bol On bol Arc num Level	Inicializa as variáveis da classe. Parâmetros: indicação de ligado ou desligado; indicação para arquivar e depois criar logs; nível de que está ligado o debug.
Trace	str Origin mix Packet num Level	Imprime e/ou armazena informação caso o level de parâmetro esteja ligado.
Warning	str Origin str Message	Imprime e/ou armazena um aviso de atenção.
Print R	mix Packet num Level str Ident	Imprime uma variável de forma a ser lida por pessoas caso o level indicado esteja ligado.

IsValidLevel	int Num Level	Indica se o nível intNumLevel é está ligado.
Clear Log		Limpa o log.
SetArchiveOp	bol Op	Muda a opção de arquivar logs.
Turn On		Liga o debug
Turn Off		Desliga o debug
Set Level	int Level	Salva o nível de debug
Get Log		Retorna a array contendo um log
Get Error Log	str File	Retorna uma array contendo os avisos de erros(warning). Caso strFile não seja nulo será criado automaticamente um arquivo de log
Save Log	str File	Salva log no arquivo strFile.

❒ Tabela 20.21 – WebEngine API – PDebug

20.10.8 – CORE::GATE

A classe gate é a classe responsável pela comunicação com o front-end da aplicação. Deve ser implementada para cada tecnologia diferente.

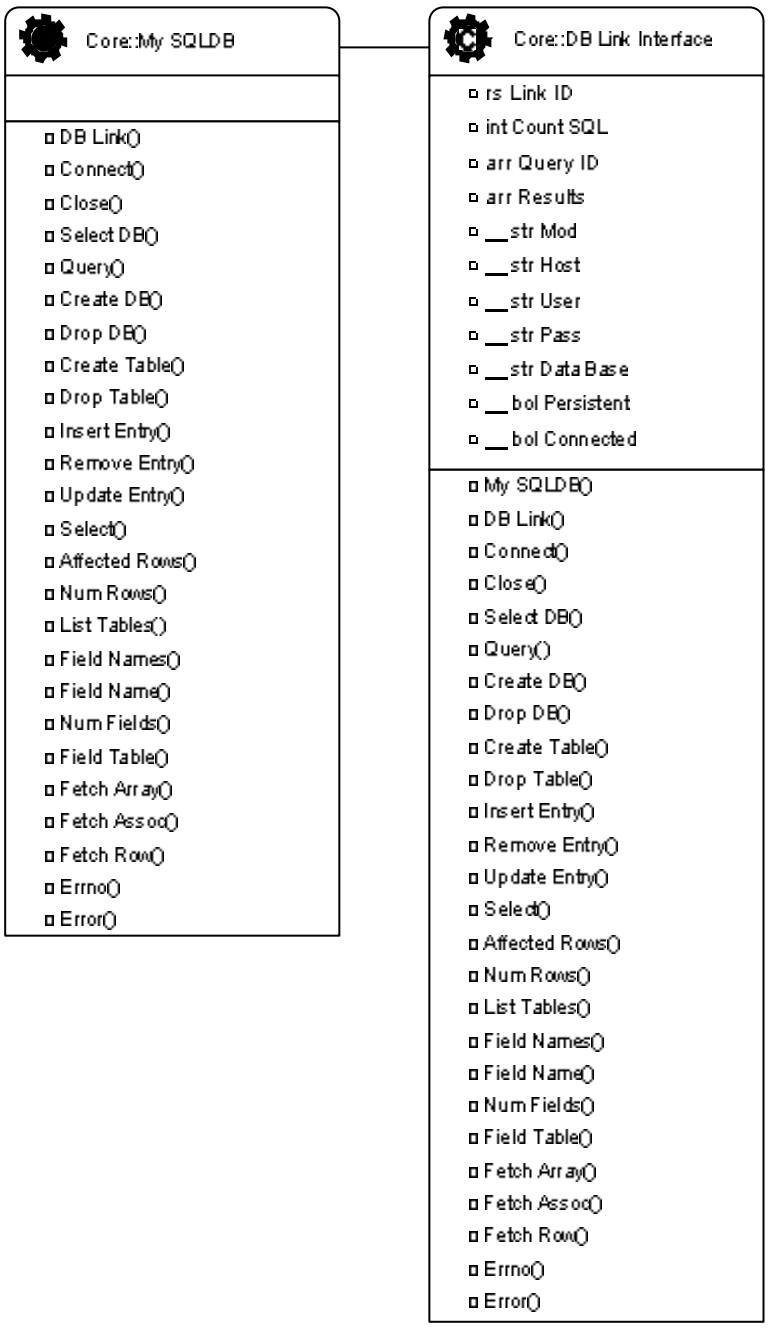


Método	Parâmetros	Descrição
Get Input		Recupera os dados e variáveis vindas do front-end
Output	Packet	Prepara e repassa os dados de resposta para o front-end

❒ Tabela 20.22 – WebEngine API – Gate

20.10.9 – CORE::DBLINK

Uma entidade muito importante para aplicações web que possibilitam acesso ao banco de dados é a “DBLink”, que possui uma interface clara e simples para estabelecer conexões com a base de dados. Sua versatilidade, possibilita que troquemos o servidor de banco de dados sem ter que alterar toda a aplicação, somente sendo necessária uma nova implementação de sua interface para o novo servidor de dados.



Método	Parâmetros	Descrição
DBLink	str Mod str Host str User str Pass str Data Base bol Persistent	Cria uma conexão com o banco de dados. Chama automaticamente o método Connect

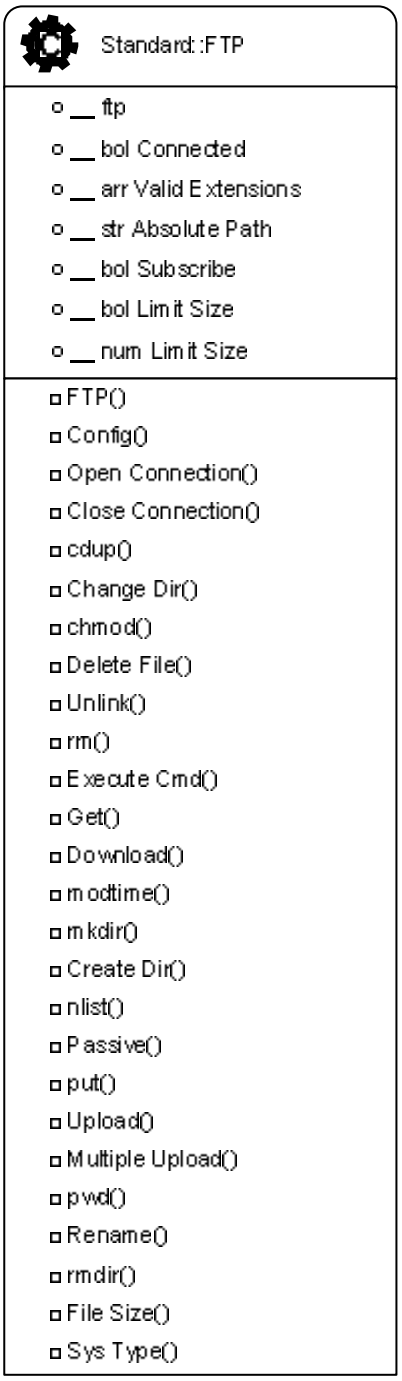
Connect		Efetua a conexão com o banco de dados
Close		Fecha a conexão com banco de dados
SelectDB	str Data Base	Seleciona a base da dados onde serão efetuadas as operações seguintes.
Query	str Query bol Store	Executa uma consulta (query) no banco de dados. Caso bol Store seja verdadeiro a consulta é armazenada
CreateDB	str DB Name	Cria uma base da dados nova no banco de dados.
Drop DB	str DB Name	Destrói uma base de dados existente no banco de dados
Create Table	str Table Name arr Fields arr Primary Key arr Unique str Table Type	Cria um tabela no banco de dados
DropTable	str Table Name	Destrói uma tabela do banco de dados
InsertEntry	str Table Name arr Infos	Insere dados na tabela. Os dados passados na array estão associados as suas chaves, i.e. field => data
RemoveEntry	str Table Name arr Info int Limit	Retira uma entrada da tabela strTableName que contenha as informações iguais a arrInfo(field => data). O intLimit indica o número máximo de linhas que podem ser afetadas.
UpdateEntry	str Table Name arr Fields arr Cond int Limit	Atualiza uma entrada na tabela. Os dados arrFields(field => data) são os dados a serem atualizados na entrada cuja informações são iguais a arrCond(field => data).
Select	str Tabel Name str Select Exp arr Where arr Having str Group By int Group Dir str Order By	Seleciona informações no banco de dados

	int Order Dir int Limit	
AffectedRows		Retorna o número de linhas afetadas na última operação. Retorna -1 caso tenha havido falha na última operação. Somente para operações INSERT, UPDATE e DELETE
ListTables	str Data Base	Retorna uma array contendo as tabelas da base de dados indicada.
FieldsName	str Table Name	Retorna uma array contendo os nomes dos campos de uma tabela
FieldName	Index	Retorna o nome de um campo específico de uma consulta. Indicado pelo index.
NumFields		Retorna o número de campos retornados na última consulta.
FieldTable	Index	Retorna o nome da tabela que contém o campo indicado.
FetchArray		Retorna informações de uma linha do resultado de uma consulta em forma de array numérica e array associativa.
FetchAssoc		Retorna informações de uma linha do resultado de uma consulta em forma de array indexada pelos nomes dos campos da tabela.
FetchObject		Retorna informações de uma linha do resultado de uma consulta em forma de um objeto cujas propriedades recebem o nome dos campos na tabela.
FetchRow		Retorna informações de uma linha do resultado de uma consulta em forma de array numérica.
Errono		Retorna o número do erro ocorrido em uma operação.
Error		Retorna a descrição do erro ocorrido em uma operação.

❒ Tabela 20.23 – WebEngine API – DBLink

20.10.10 – STANDARD::FTP

Classe que provê acesso FTP. Foi colocada dentro da biblioteca de padrões pois segue o padrão de acesso FTP.



Método	Parâmetros	Descrição
FTP	str Absolute Path bol Subscribe arr Extensions num Limit	Inicializa variáveis da classe e chama o método que configura a conexão.

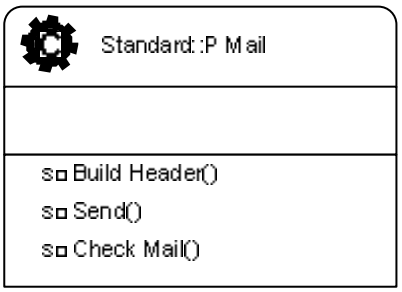
Config	str Absolute Path bol Subscribe arr Extensions num Sixe Limit	Configura a conexão.
OpenConnection	url	Abre conexão com a url informada
CloseConnection		Encerra a conexão FTP.
Cdup		
ChangeDir	dir	Abre a pasta dir no servidor ftp
Chmod	perm file	Troca o modo do arquivo
DeleteFile | Unlink | rm	file	Remove o arquivo no servidor FTP
Execute Cmd	command	Executa um comando
Get | Download	remote local mode	Carrega um arquivo do servidor.
Modtime	file	Retorna o tempo da última modificação do arquivo(file)
Mkdir | CreateDir	dir	Cria um diretório no servidor
nlist	dir	Lista os arquivos do diretório indicado
Passive	pasv	Seleciona o modo ativo. Caso pasv seja verdadeiro seleciona o modo passivo.
Put | Upload	remote local mode	Envia um arquivo para o servidor
MultipleUpload	arr Files local	Envia uma série de arquivos para o servidor

	mode	
Pwd		
Rename	old new	Renomeia um arquivo no servidor
Rmdir	dir	Remove um diretório no servidor
FileSize	file	Retorna o tamanho do arquivo no servidor
SysType		Retorna o tipo do sistema

❒ Tabela 20.24 – WebEngine API – FTP

20.10.11 – STANDARD::PMAIL

Esta classe provê suporte a envio de e-mails. Possui uma simples interface para métodos estáticos.



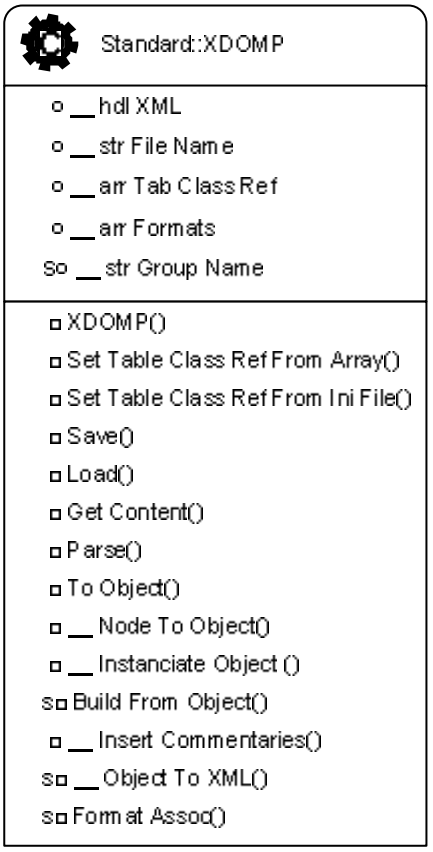
Método	Parâmetros	Descrição
Build Header	str Mail Reply To str Mail From arr Mails Cc arr Mails Bcc	Constrói o cabeçalho do email a ser enviado.
Send	arr Mails To str Title str Msg str Mail Reply To str Mail From arr Mails Cc	Envia email para a lista de e-mails em arrMailTo, arrMailsCc e arrMailsBcc

	arr Mails Bcc	
CheckMail	str Mail	Checka se o email está em um formato válido.

❒ Tabela 20.25 – WebEngine API – PMail

20.10.12 – STANDARD::XDOMP

Classe para manipulação de XML. Utiliza o padrão DOM para manipulação de XML.



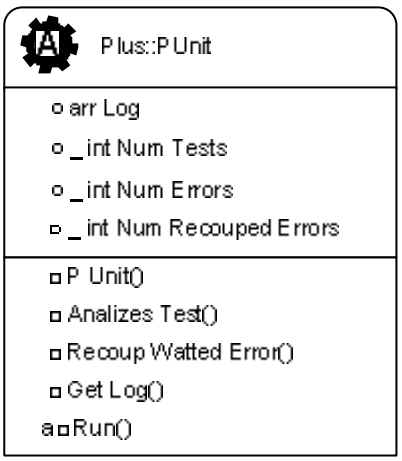
Método	Parâmetros	Descrição
XDOMP (constructor)	strEncoding bolWhiteSpace	Instancia um novo objeto XML vazio
SetTableClassRef FromArray	arrTable	Armazena informações para transformação de objetos em XML e vice-versa. As informações são passadas pela arrTable no formato (nome da classe no XML => nome da classe em PHP)

SetTableClassRef FromIniFile	strIniFileName	Armazena informações para transformação de objetos em XML e vice-versa. As informações são passadas pela pelo arquivo ini strIniFileName no formato (nome da classe no XML = nome da classe em PHP)
Save	bolSaveInFile strFileName	Salva o xml. Caso não seja verdadeiro bolSaveInFile, o XML só será salvo na memória, caso contrário será salvo no arquivo strFileName.
Load	strXML	Carrega o XML de um arquivo ou de uma string.
GetContent		Salva o XML na memória e retorna o conteúdo do XML como uma string.
Parse	strXML	Parseia e valida o XML.
ToObject	objResp	Transforma um XML em um objeto e o retorna em objResp.
BuildFromObject	xdpObj obj strComment strXSLFile	Constrói um objeto XDOMP a partir de um objeto obj e o retorna em xdpObj.
FormatAssoc	ass	Formata uma array associativa em uma array numérica que é entendida pelo transformador de XML do XDOMP

❒ Tabela 20.26 – WebEngine API – XDOMP

20.10.13 – PLUS::PUNIT

Essa classe é uma simples implementação similar ao padrão JUnit. Com ela é possível criar testes de unidades automatizados facilmente. Trata-se de uma classe abstrata. Para usá-la basta criar uma classe de testes de sua classe que estenda PUnit. Depois crie uma função de testes para cada função em sua classe. Em seguida crie o script de testes em um função Run utilizando as funções de testes construídas.

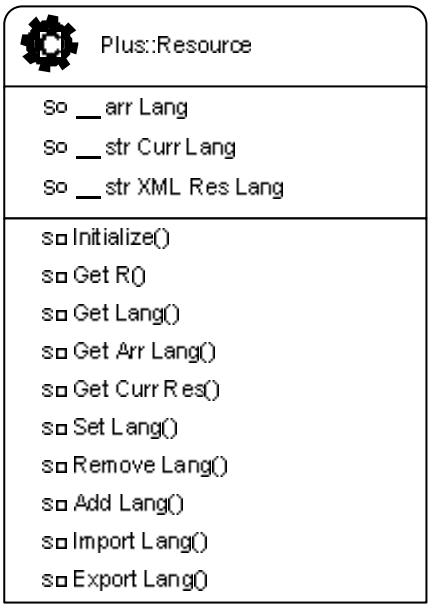


Método	Parâmetros	Descrição
PUnit (construtor)		Inicializa variáveis da classe.
AnalizesTest	strFunctionName arrParams mixExpectedValue mixRecievedValue strCaseTest fltTExec	Analisa um teste executado. Deve ser chamado em toda função de teste.
RecoupWattedError		Recupera um erro forçado pelo programador. Decrementa a contagem de erros.
GetLog		Retorna o log das análises dos testes
Run		Executa o script de testes

❒ Tabela 20.27 – WebEngine API – PUnit

20.10.14 – PLUS::RESOURCE

A classe resource fornece métodos para acesso diversos tipos de recursos, como arquivos de diferentes finalidade (estilo, script, imagem) e texto. Separa as fontes de recurso por linguagens. Segue o padrão singleton, logo, não precisa ser instanciada e seus métodos podem ser acessados diretamente após a classe ser inicializada.



Método	Parâmetros	Descrição
Initialize		Inicializa as variáveis da classe.
GetR	type strLabel	Retorna um recurso do tipo type armazenado no label strLabel
GetLang		Retorna a linguagem corrente
GetArrLang		Retorna a array de labels da linguagem corrente
GetCurrRes		Retorna o xml de entrada da função ImportLang dá ultima linguagem incluída.
SetLang	strLang	Seleciona uma nova linguagem strLang como corrente
RemoveLang	strLang	Retira a referência para a linguagem do conjunto de linguagens disponível.
AddLAng	strLang objResLang	Insere no conjunto de linguagens disponível as uma nova linguagem strLang com os recursos em objResLang
ImportLang	strLang xmlResLang	Insere no conjunto de linguagens disponível as uma nova linguagem strLang com os recursos em xmlResLang
ExportLang	strLang	Exporta a lista de recursos da linguagem strLang

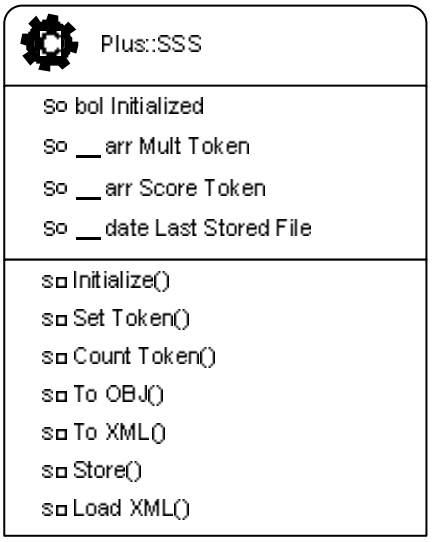
	strXMLLang	para XML e o retorna em strXMLLang

❒ Tabela 20.28 – WebEngine API – Resource

20.10.15 – PLUS::SSS

A classe SSS(Simple Score Statistic) foi idealizada para armazenar informações estatísticas de uma aplicação. Seu uso é fácil, simples e rápido. Seu uso de forma incorreta pode criar arquivos enormes estatísticos que não informam muita aos interessados. Por exemplo, usar informações que são contabilizadas não acarreta nenhum aumento no arquivo, logo este recurso pode ser usado várias vezes. Em contrapartida, armazenar informações como nome ou qualquer outro tipo que seja diferente a cada acesso pode aumentar muito o arquivo gerado.

Esta classe também segue o padrão singleton e seus métodos estáticos podem ser facilmente acessados de qualquer lugar.



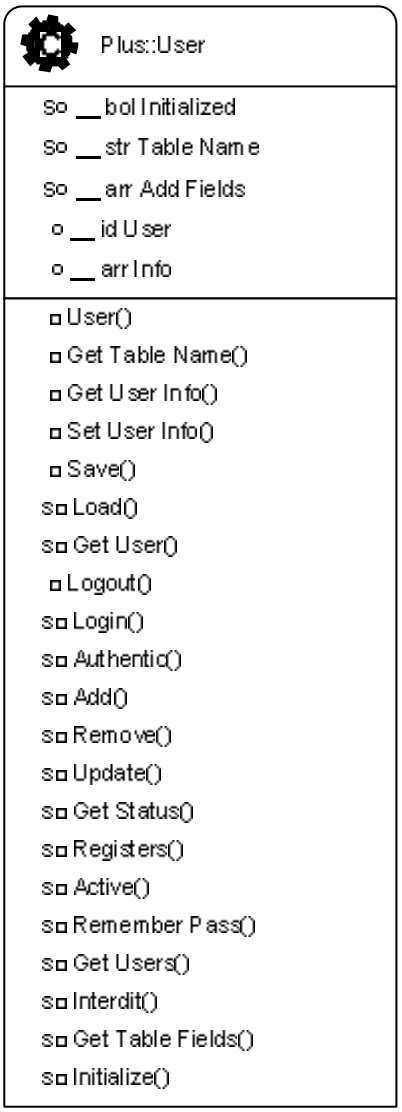
Método	Parâmetros	Descrição
Initialize		Inicializa as variáveis da classe.
SetToken	strToken strValue	Armazena informação. Cuidado ao usar esta função. Deve ser utilizada como último recurso para obter informações estatísticas. Seu uso não é indicado mais que uma vez em toda aplicação.
CountToken	strToken	Contabiliza uma passagem pelo token.

ToObj	obj	Retorna em obj o objeto com as informações estatísticas.
ToXMl	xml	Retorna em xml um XML as informações estatísticas
Store	strFile	Armazena no formato XML as informações estatísticas
LoadXML	strXML	Carrega de um arquivo ou de uma string XML informações estatísticas.

❒ Tabela 20.29 – WebEngine API – SSS

20.10.16 – PLUS::USER

A classe User foi criada para manipulação e autenticação de usuários da aplicação. Com ela é possível adicionar, remover e alterar dados de um usuário da aplicação no banco de dados. As informações dos usuários da aplicação podem ser configuradas e facilmente alteradas. Ela permite também fazer o controle de usuário, como bloquear ou liberar.



Método	Parâmetros	Descrição
User (construtor)	strTableName arrFields arrUserInfo	Inicializa variáveis da classe e onde são passadas as configurações dos usuários da aplicação. Parâmetros: strTableName é o nome da tabela no banco de dados arrFields são os campos adicionais do usuário na aplicação arrUserInfo é um parâmetro opcional que se informado é instanciado um novo usuário com as informações contidas na array
GetTableName		Retorna o nome da tabela de usuários da aplicação

		no banco de dados
GetUSerInfo	strKey	Retorna uma informação do usuário se existir
SetUserInfo	strKey value	Armazena uma informação do usuário
Save		Salva as informações do usuário no banco de dados
Load	user strUsername arrUserInfo	Carrega um usuário strUserName do banco de usuários e o retorna em user
GetUSer	user intUserId	Carrega um usuário intUserId do banco de usuários e o retorna em user
Logout		Efetua logout do usuário
Login	user strUsername strPassword bolExclusive	Método estático que efetua login de um usuário e o retorna em user
Authentic	status strUsername strPassword	Autentica o usuário no sistema e retorna seu status em status
AddUser	arrUserInfo	Adiciona um usuário ao sistema com as informações de arrUserInfo
Remove	strUserName strPassword	Remove um usuário do sistema. É necessário informar a senha do usuário para removê-lo.
Update	strUserName strPassword arrUserInfo	Atualiza informações do usuário strUserName com as informações em arrUserInfo
GetStatus	strUserName	Retorna o status do usuário strUserName
Register	strUserName strEmail	Opção para cadastramento de usuários. Funciona como um convite para o usuário se cadastrar.
Active	strUserName	Ativação de convite de usuário

	strEmail strActiveCode	
RememberPass	strUserName	Envia um email para o usuário cadastrado relembrando sua senha.
GetUsers		Retorna os usuários do sistema
Interdit	strUserName	Interdita um usuário no sistema
GetTableFields		Retorna uma array contendo os campos de informações adicionais configurados na aplicação
Initialize	strTableName arrFields arrUserInfo	Inicializa e configura as variáveis da classe

❒ Tabela 20.30 – WebEngine API – User

20.11 – TRINITY – MODELO AJAX

Foi dito neste capítulo que a entidade chamada de Trinity é opcional na arquitetura da WebEngine, ou seja, existindo ou não, a engine manterá seu funcionamento padrão.

A proposta desta entidade, a qual batizei de Trinity, é inserir na arquitetura descrita neste projeto técnicas AJAX. Essas técnicas e os conceitos envolvidos a respeito foram apresentados anteriormente[Capítulo 23]. Resumidamente, possibilita a comunicação assíncrona com a WebEngine – lado do servidor.

A descrição minuciosa desta entidade está fora do escopo deste projeto, bem como sua implementação. Podendo ser vistos como um tema completo de um outro projeto de graduação. Creio que a simples exposição da idéia cobre um tópico relevante do meu projeto em relação a teoria e aos últimos avanços relacionados as tecnologias aqui utilizadas. A seguir farei uma sugestão de protocolo que poderá ser implementado para que a Trinity atenda seu propósito.

```
Recebe requisição do usuário
Armazena temporariamente o REQUEST.Target
Armazena temporariamente o REQUEST.Reaction
Verifica se é necessária a viagem ao servidor
        Caso não seja necessário
                Executa a Reaction informada
```

```
            Insere resposta da Reaction no Target
      Caso contrário
            Constrói pacote de requisição para a WebEngine
            Executa requisição (HTTPReuqest)

            Recebe pacote de resposta
            Verifica resposta pelo RESPONSE.Type
            Caso não seja positiva
                  Informa ao usuário o erro
            Caso positiva
                  Executa Reaction informada.
                  Inserre resposta da Reaction no Target
```

Uma requisição do usuário conterá os itens padrões da comunicação com a WebEngine(REQUEST.ACTION, REQUEST.INPUT...) e adicionalmente REQUEST.Target e REQUEST.Reaction. O REQUEST.Target indica o local onde seriam inseridos os dados de resposta. Por exemplo, se estivermos usando tecnologias de front-end tipo HTML, o target poderia ser um id de uma tag.

O REQUEST.Reaction indica o procedimento – nome da função - a ser executada quando obter resposta. Uma reação a reação. Geralmente essa Reaction deve estar ligada a códigos de comportamento. Por exemplo, poderia ser uma função que a partir do RESPONSE.Content (dados em XML) e do RESPONSE.Skin (código estrutural em XSL) retornasse a resposta da transformação XML/XSL. Note que neste exemplo a resposta da WebEngine veio por um portão de comunicação XML – output em XML e skins em XSL.

Outro exemplo poderia ser para comunicações pelo portão direto (TransparentGate). Onde a reaction teria que criar listas ou tabelas a partir dos objetos contidos no RESPONSE.Content.

Reforçando mais uma vez. Essa descrição é importante para este projeto porque completa o ciclo de comunicação da WebEngine. Mas este projeto final de graduação não comporta a implementação, nem a descrição detalhada desta entidade.

20.12 – ADD-ON'S

Add-on's são módulos adicionais a engine. Que agregam funcionalidades utilizadas diretamente pelos usuários finais dela.

Quando comecei a pensar neste projeto de graduação meu esboço inicial era criar uma base reutilizável para agilizar o desenvolvimento de aplicações web. Essa base teria uma parte

central e vários módulos adicionais. Esses módulos teriam como objetivo auxiliar o desenvolvedor e/ou auxiliar a administração do aplicativo.

A base central se transformou na engine proposta neste trabalho. Devido a sua complexibilidade, ela pos si só já bastava como conteúdo de um projeto final de graduação. Entretanto, no meu ponto de vista, acho que pode ser interessante a exposição das idéias dos módulos adicionais, pois revelam a extensibilidade e o potencial da engine.

A idéia desses módulos adicionais é seguir o modelo do PHPMyAdmin. Que basta baixar a última versão, colocá-lo em uma pasta na web, junto à aplicação, configurar e utilizar via web.

Cada um desses módulos tem um foco e um público alvo de uso. Por exemplo, o PUnit e o SimpleDoc seriam de grande valia para os desenvolvedores. Enquanto que o Smith, UserAdmin, XLang e MyStatistics seriam direcionados aos administradores de softwares ou páginas na web.

Em papel tenho esboços detalhados de como seriam alguns módulos, mas creio que os detalhes não são relevantes neste momento, uma vez que fogem do escopo deste projeto. Contudo, em seguida farei uma breve exposição de algumas idéias de módulos adicionais.

20.12.1 - PUNIT - MÓDULO PARA TESTE AUTOMATIZADO

O desenvolvedor tem disponível uma classe abstrata (PUnit), com alguns métodos auxiliares. Para criar testes, cria uma classe de testes que estende PUnit. Nela cria funções que testam outras funções de alguma entidade que se deseja testar a partir de entradas e saídas informadas. Depois, cria um script de testes utilizando essas funções criadas.

Este módulo seria responsável por acionar os testes, armazenar resultados e gerar relatórios de teste.

20.12.2 - SIMPLEDOC – MÓDULO PARA GERAÇÃO AUTOMÁTICA DE DOCUMENTAÇÃO

A idéia é baseada na geração automática de documentação JavaDoc, porem com algumas diferenças. A geração usando JavaDoc serve para Java, pois é feita lendo diretamente o código e comentários contidos no código.

A diferença seria que o script de geração de documentação só faria leitura dos comentários. Que seguiriam o mesmo padrão JavaDoc, porém com tags configuráveis.

Outra característica desejada para este módulo, está relacionada a descrição de arquitetura. Hoje em dia existem diversos programas que de um código documentado com uma sintaxe pré-definida consegue extrair modelos UML de hierarquia de classes, e de relacionamento de classes, entre outros. No SimpleDoc, a idéia é que a sintaxe de descrição também seguisse o padrão de comentários JavaDoc.

Outra característica é que essa documentação gerada ficaria online e poderia ser gerada e configurada via web também.

20.12.3 – USERADMIN – ADMINISTRADOR DE USUÁRIOS

O UserAdmin seria basicamente um administrador de usuários do sistema. Onde o administrador poderia fazer bloqueios a usuários, definir permissões, adição e remoção de usuários, e seria informado sobre acessos, duração...

20.12.3 – XLANG – ADMINISTRADOR DE LINGUAGENS

O XLang facilitaria a vida dos administradores de sites ou softwares na web. Utiliza os serviços de resource da WebEngine e possibilita via web a adição e de novas linguagens na aplicação, edição de labels e figuras.

Assim, o dono do negócio, ou administrador do site, depois de ter o software em uma língua, precisaria apenas contratar tradutores. Esses tradutores poderiam abstrair totalmente o programa e através de uma simples interface traduziriam o software a partir de outra linguagem que já existisse.

20.12.3 – MYSTATISTICS – MÓDULO ESTATÍSTICO

Esse módulo utilizaria recursos da classe SSS da WebEngine. Com ele, o administrador, ou dono do negócio poderia ver online como, quando e quem está acessando seu site ou aplicativo web.

A idéia é não só apresentar tabelas, mas também gerar gráficos e relatórios.

20.12.3 – SMITH – ADMINISTRADOR DE AGENTES

Esse módulo é um dos meus preferidos. É responsável por executar rotinas agendadas, sem que sejam necessários disparos de usuários.

Essas rotinas podem ser desde envio de emails aos usuários da aplicação, até geração de relatórios de outros módulos, como o estatístico. Ou mais ainda, rodar rotinas relacionadas ao modelo da aplicação a que esta dando apoio.

20.13 – DISPOSIÇÃO DE ARQUIVOS

Pelas experiências que tive durante o desenvolvimento. E da experiência que passei quando outras pessoas, alunos de PES, utilizavam a engine mesmo antes de ficar pronta, me levaram a adotar a seguinte disposição de arquivos. Repare que o primeiro nível abaixo da pasta da engine é para controle de versões. Desse modo, o sujeito que desejar usá-la pode utilizar a versão WE001, WE002... A pasta dev, como o nome já indica é a versão que está em desenvolvimento.

Dentro da pasta de uma versão podemos encontrar o arquivo WebEngine.php, que é o arquivo que deve ser incluído em uma aplicação que deseje usar a WebEngine. Também podemos encontrar pastas relacionadas às tecnologias de interface-gráfica com o usuário. Essa é uma maneira de acumular portões de comunicação e código genérico para a biblioteca de comportamento.

Além dessas pastas, podemos encontrar a pasta do PHP, onde se encontram os códigos relativos a engine. Dentro dela podemos ver as pastas das bibliotecas.

À direita, na figura abaixo, disponibilizei uma sugestão de como deve ser hierarquizada uma aplicação baseada na engine. Esta hierarquia estará online. Logo, é preciso ter muito cuidado para não deixar vulnerável qualquer dado relevante de uma aplicação ou do próprio servidor.

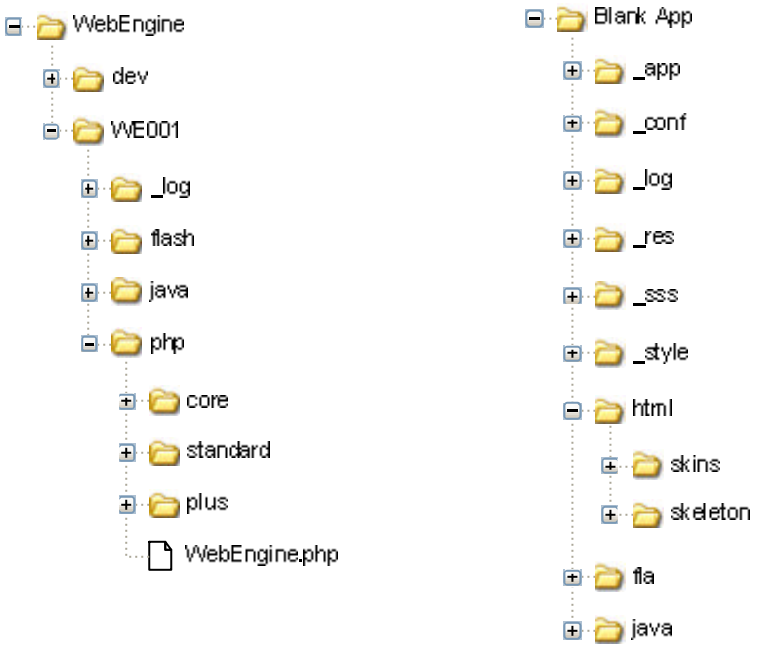
Pensando nesta peculiaridade, determinei, para o padrão que adotei que as pastas que não devem ser vista por um usuário comum na web devem receber um "_" no início do nome. Desse modo fica mais fácil o tratamento de segurança. Por exemplo, podemos configurar para que o servidor web não disponibilize essas pastas, ou até mesmo, exija uma senha de administrador.

Todos os nomes das pastas podem ser configurados. Mas, o padrão que proponho é o que tem os nome na figura abaixo. Intuitivamente podemos perceber que a pasta "_conf" receberá os arquivos de configuração, a pasta "_log" os arquivos de log, e a pasta "_app" os arquivos referentes a biblioteca da aplicação.

As outras pastas, "_res", "_sss", "_style" são destinadas a armazenamento de arquivos de resource, estatísticos e de estilo de apresentação respectivamente. As pastas subseqüentes são pastas destinadas a cada tecnologia de front-end.

Por exemplo, para o MestreCuca, poderíamos acessar a aplicação em diferentes interfaces gráficas:

- www.mestrecuca.no-ip.com/html
- www.mestrecuca.no-ip.com/fla
- www.mestrecuca.no-ip.com/java



❒ Figura 20.31 – Hierarquia de arquivos adotado na engine e uma sugestão para aplicações.

20.14 – VANTAGENS E DESVANTAGENS

As principais vantagens do uso da engine são:

- Reutilização – uso verbatim.

- Fácil evolução.
- Fácil manutenção.
- Fácil detecção de erros – geração de logs e estatística.
- Facilita a execução de testes automatizados.
- Rapidez no desenvolvimento de novas aplicações.

E as desvantagens mais visíveis são:

- Quebra de antigos paradigmas sobre desenvolvimento web.
- Entendimento prévio da arquitetura.
- Adaptação ao modelo adotado.
- Familiarização com a API da biblioteca padrão.

20.15 – TESTES DE UNIDADE

Foram realizados diversos testes de unidades na engine e em parte dela. Estes foram realizados utilizando o arcabouço construído, que chamei PUnit. Os testes realizados foram testes chamados de testes de caixa cinza[Capítulo 7].

Nos testes foram testados os métodos das principais classes do framework desenvolvido. Verificando parâmetros de entradas e saídas obtidas de forma automatizada. Em cada teste adicionalmente é informado qual caso de teste aquele procedimento esta executando.

Ao final de cada teste é exibido um pequeno relatório que contém informações do teste, tais como, quantos testes foram executados, quantos erros foram encontrados e quantos desses eram esperados.

Abaixo segue uma tabela contendo os resultados obtidos dos testes executados na engine.

Classe	Testes executados	Erros Encontrados	Erros Esperados
MySQLDB(BDLink)	36	14	14
Environment	60	12	12
XDOMP	34	11	11
Resource	19	6	6

SSS	17	1	1
PMail	3	1	1

❒ Tabela 20.32 – Resultado dos testes executados na engine.

20.16 – TESTES DE EFICÁCIA

Durante o desenvolvimento deste projeto, participei, desta vez não como aluno, mas sim como auxiliar do professor Júlio em mais um processo de evolução do C&L com os alunos de PES(Princípios de Engenharia de Software). Nesta interação, a engine desenvolvida por mim foi utilizada como base para o desenvolvimento da nova versão do C&L pelos alunos.

Essa experiência foi muito importante pra mim, mas principalmente foi importante para a engine. Pois pude testá-la com usuários pertencentes ao público alvo da engine – os desenvolvedores.

Em contato com os alunos que a utilizaram, pude perceber as maiores dificuldades encontradas por eles. Ao final do período, realizei entrevistas informais onde ouvi opiniões, críticas e sugestões. Entre elas destaco as seguintes:

- Inicialmente os alunos não entenderam como usar a engine, que ainda não possuía documentação detalhada.
- Depois de uma aula sobre utilização da engine os alunos mudaram de opinião. Acharam simples de utilizar, sem mesmo entender todo o funcionamento interno da engine.
- Achara que o desenvolvimento foi rápido.
- Gostaram da forma de geração de log – facilidade de encontrar erros.
- Acharam o código bem documentado
- Não sabiam a API da biblioteca da engine

É importante ressaltar que os alunos utilizaram a engine bem no início dela. Quando a documentação e a API não estava à disposição para consultas. Apenas existia um documento descrevendo o funcionamento geral e algumas dicas de utilização.

20.17 – O FUTURO DA WEBENGINE

A WebEngine foi projetada para estar constantemente em evolução, sempre podendo ser acrescentadas novas classes, serviços e funcionalidades. No papel, foram pensados diversas funcionalidade para serem implementadas. Contudo, o tempo de desenvolvimento deste projeto não permitiu que fosse coberto todos os itens idealizados.

Abaixo segue uma lista de partes do projeto que forma idealizadas, e algumas até projetadas no papel, que seriam de grande valia para o motor.

- Portões de comunicação para Java, VB, e .Net
- Controle de acesso a dados e controle de versão
- Suporte a templates para skins
- Adicionar funcionalidades à classe de Hipermídia
- Threads – filas de ações
- Add on's

Poderia ser pensada uma infinidade de possibilidades. Este fato de existir muito a pensar e desenvolver na engine fortalece sua essência, de estar em constante crescimento. Espero que esse seja um fator que impulsione ainda mais outros interessados em ajudar e participar do desenvolvimento e assim dar continuidade e prosseguimento a este projeto.

20.18 – INSTALAÇÃO

Tendo instalados e configurados o PHP, o servidor HTTP e o servidor de banco de dados, a instalação da engine é bastante simples.

Basta baixar o pacote contendo a engine e extrair o conteúdo para a pasta desejada no servidor. Em seguida, configure suas preferências no arquivo de configuração e depois basta fazer uma inclusão da WebEngine no seu projeto para começar a usá-la.

20.19 – EXEMPLO MESTRECUCA

20.19.1 – VISÃO GERAL

O exemplo do MestreCuca foi exposto no início deste documento. Tem o objetivo de apresentar de forma clara os conceitos estudados e propostos neste projeto final de graduação.

Durante todo este documento fiz referências a esse exemplo, utilizando nomes de personagens e elementos como receita, ingredientes, etc. A seguir apresentarei ainda alguns exemplos de uso de algumas funcionalidade e pontos chaves da engine proposta. Começarei mostrando exemplos de uso de callbacks, em seguida veremos um exemplo com skins e skeleton, e posteriormente veremos exemplos de algumas funcionalidades extras.

20.19.2 - CALLBACKS

```
function MC_AddReceita()
{
	$strReceita		= Environment::GetVar( "strReceitaTitulo" );
	$strPreparo		= Environment::GetVar( "strPreparo" );
	$arrIngradientes	= Environment::GetVar( "arrIngradientes " );

	$ResponseType		= MestreCuca::AddReceita( intIdUser, strReceita , 
strPreparo, arrIngredientes );

		switch( $ResponseType )
	{
		case( MC_ERROR_DB ):
			$ResponseContent	= "Erro no banco de dados";
			$ResponseSkin		= "MC_ERROR";
		break;
		case( MC_ERROR_DB ):
			$ResponseContent	= "Já existe uma receita com esse nome";
			$ResponseSkin		= "MC_ERROR";
		break;
		case( MC_OK ):
			$ResponseContent	= "Receita adicionada";
			$ResponseSkin		= "RECEITA";

		break;
		default:
			$ResponseContent	= "Erro desconhecido";
			$ResponseSkin		= "MC_ERROR";
	}
	Return		( array( $ResponseType , $ResponseContent , $ResponseSkin ) 
);
}
```

No exemplo acima foi mostrado a callback de adição de novas receitas. Repare que para q execução desta callback são necessários três parâmetros: strReceitaTítulo, strPreparo e arrIngredientes. Após a chamada do método da classe relacionada a ação requisitada a resposta é tratada e a callback retorna ao sistema os parâmetros de resposta: RESPONSE.Skin, RESPONSE.Content e RESPONSE.Type.

20.19.3 - SKINS

```
<?php
/*******
* Skin – área de pré-processamento
*******/
$strTituloReceita    = Environment::GetVar( “strTituloReceita” );
$arrIngradientes     = Environment::GetVar( “arrIngredientes” );
$strPreparo          = Environment::GetVar( “strPreparo” );
$strListaIngred      = “<ul>”;
foreach( #arrIngredientes as $strIngrediente => $intQuantidade )
{
        $strListaIng  .= “<li>” . $intQuantidade . “ - ” . $strIngrediente . “</li>”
}
$strListaIngred      = “</ul>”;
/* Final do bloco de pré-processamento da skin */
?>
<html>
<head>
        <title>MestreCuca</title>
</head>
<body>
        <h1>MestreCuca</h1>
        <h3>Receita:<?= $strTituloReceita ?> </h3>
        <p>Ingredientes:</p>
        <?= $strListaIng ?>
        <p>Preparo:</p>
        <?= $strPreparo ?>
</body>
</html>
```

O exemplo anterior mostra a construção da skin de visualização de detalhes de uma receita do MestreCuca. Fica claro neste exemplo, a separação da modelo da aplicação e da interface gráfica com usuário.

CAPÍTULO 21 – O PROJETO C&L

21.1 – INTRODUÇÃO

Este capítulo tem como objetivo mostrar o projeto que executei em paralelo. Não é o foco do meu projeto final de graduação, mas faz parte. E além disso, ele serve como um exemplo do funcionamento da engine desenvolvida.

Diferentemente do outro exemplo usado neste documento, o C&L é um websoftware de porte maior, com muitos detalhes, possui um banco de dados com várias tabelas, entre outras coisas que o torna mais complexo. Será um excelente teste para a engine e o seu desenvolvimento foi uma escola para mim, me motivando a aprender e pesquisar sobre diversos assuntos relacionados a ele.

21.2 – VISÃO GERAL

A descrição de cenários e léxicos é uma técnica, já discutida antes neste documento, utilizada na engenharia de requisitos que auxilia o entendimento de uma situação específica de uma aplicação de software.

Existem várias propostas para representação de cenários, desde a mais informal até representações formais. Uma delas, foi descrita em um artigo[3], cujo um dos autores é o professor Julio César Leite. Em conseqüência deste artigo e de estudos oriundos, foi iniciado o projeto C&L.

Acredita-se que no mercado não existe, atualmente, nenhuma ferramenta de software livre que trate de edição de cenários e léxicos de acordo com suas regras e especificações, forçando os possíveis usuários desse tipo de ferramenta a usar ferramentas mais genéricas, como por exemplo, o Word e o Visio, ambos da Microsoft.

O C&L tenta preencher esta lacuna oferecendo um ambiente diferenciado de edição. Trata-se de um ambiente colaborativo que auxilia a descrição de cenários e léxicos em linguagem natural semi-estruturada.

Criada nos moldes da filosofia de desenvolvimento de software livre, o sistema vem sendo utilizado e evoluído, principalmente, por alunos da cadeira de Princípio de Engenharia de

Software – PES - oferecida no curso de Engenharia de Computação pela Pontifícia Universidade Católica do Rio de Janeiro.

Uma característica interessante no desenvolvimento da aplicação é que a descrição dos requisitos esta embutida no código. Muitas vezes, a documentação existente está sendo atualizada enquanto uma funcionalidade nova é adicionada. Em conseqüência, podemos ter uma documentação um pouco desatualizada. Contudo, se no próprio código estiver contida a documentação, os dois irão evoluir lado a lado. No C&L, a documentação é gerada automaticamente a partir do código.

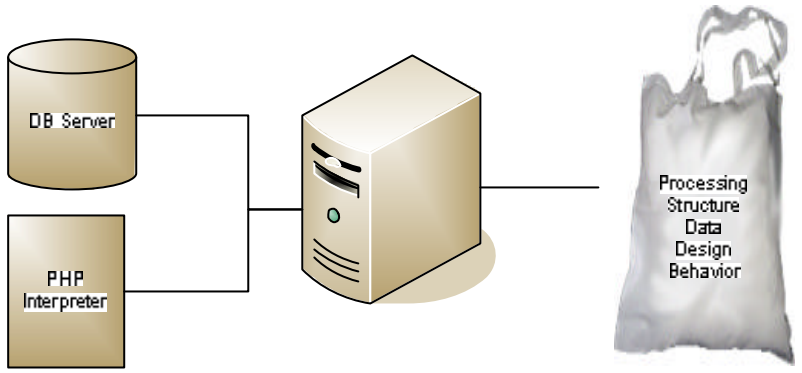
21.3 – O ESTADO DA ARTE

Há alguns anos, quando cursei a cadeira de PES, tive meu primeiro contato com o C&L. Quando participei do processo evolutivo do projeto e tive a oportunidade de conhecer detalhes de sua implementação.

A constante evolução é muito boa para o ciclo de vida de um software[8], desde que seja possível manter as mudanças e analisar se as modificações eram relevantes e realmente agregaram ao sistema. Por outro lado, se essa evolução é feita de forma errada, ou se não foi apropriadamente testada, a complexibilidade da aplicação pode aumentar e impedir evoluções futuras.

Acredito que a constante evolução do C&L por pessoas diferentes, de diferentes grupos de diferentes períodos, com níveis de conhecimento diferente, trabalhando em cima de uma base estrutural fortemente acoplada, foram fatores causadores de um desenvolvimento desordenado.

O projeto tinha uma arquitetura inicial, contudo, dentro de um mesmo arquivo de código era possível encontrar linhas de código referente à estrutura, apresentação, processamento, e comportamento tudo junto se entremeando. Além disso, existia muita marcação fora dos padrões estipulados pelo W3C. A figura a seguir ilustra esse fato.



❒ Figura 21.1 – Arquitetura encontrada no C&L.

A estrutura utilizada na maior parte das versões anteriores no desenvolvimento do C&L consiste em um servidor da base de dados e uma aplicação cliente se comunicando diretamente. Podemos identificar alguns pontos críticos no design adotado:

- Não modularizado
- Fortemente acoplado
- Difícil evolução e manutenção
- Difícil detecção de erro
- Replicação de código encontrada em diversos arquivos e ate no mesmo arquivo
- Linhas de código mortas

Esse, e outros fatores, contribuíram para que se torne difícil, e até inviável, novas evoluções em um futuro não distante. Esse fato foi o grande motivador para a reconstrução do projeto.

21.4 – PROCESSO EVOLUTIVO

Antes de iniciar o processo de desenvolvimento da nova versão do C&L, chamada de C&LEvolution, era necessário saber quais funcionalidades eram realmente necessárias, quais teriam que ser mantidas e quais teriam que ser reformuladas.

Inicialmente, tinha a minha disposição quatro fontes para basear o processo de re-engenharia: os requisitos antes descritos, o artigo que proporcionou o projeto, versões anteriores do C&L, além de estar em contato direto com o idealizador do projeto.

O processo de engenharia reversa, na área de software, é caracterizado por analisar como foi feito e entender detalhes da implementação, a partir de um produto final, geralmente um arquivo binário. Quando comecei esse processo no C&L, em parceria com um colega de

turma, Thiago Crispino, não imaginei que seria tão trabalhoso. Quanto mais fundo íamos nas pesquisas, nós encontrávamos mais pontos relevantes que influenciavam diretamente as decisões de design.

Adotamos o método de re-engenharia descrito por Leite[8], que tem os seguintes sub-processos: recuperação, especificação, re-design e re-implementação.

Iniciando o processo de recuperação, ao reunir a documentação existente pude verificar que esta estava ultrapassada em relação à implementação atual. Entretanto, ainda continha a idéia central dos requisitos funcionais e não-funcionais do projeto inicial.

Após o entendimento destes, o próximo passo seria isolar e agrupar os códigos por seções relacionadas às funcionalidades. Em seguida, avaliar quais códigos seriam reaproveitados e quais eram obsoletos

A tarefa de ler e entender códigos de outros programadores pode ser extremamente difícil. Quando tive que analisar os detalhes das versões anteriores me deparei com códigos incompletos e muitas vezes obscuros. O código ainda continha uma infinidade de estilo de programação, o que dificulta mais ainda sua compreensão.

Apesar deste projeto se tratar de um projeto open-source, o forte acoplamento e a pouca modularidade deixa antigas versões com grau baixo de inteligibilidade. Mediante os fatos, optei por executar um processo reverso, estudando detalhes da aplicação sem me ater aos detalhes da implementação. E depois, poder re-escrever os requisitos do projeto e iniciar a fase de desenvolvimento.

O processo de engenharia reversa realizado teve as seguintes etapas:

- Utilização e pequenos testes em versões anteriores – avaliação ignorante.
- Leitura e entendimento dos requisitos funcionais e não-funcionais da aplicação
- Utilização e testes, baseado nos requisitos, da versão mais estável – avaliação esperta
- Separação de código útil correlacionando-o a funcionalidades
- Estudo das entidades existentes e seus relacionamentos
- Listagem dos pontos críticos do sistema
- Pesquisa de possibilidades técnicas e tecnológicas para solucionar pontos críticos
- Sucessivas reuniões com a idealizador do projeto

Ao final dessas etapas, tinha uma base sólida de conhecimento sobre a aplicação. Em seguida, o processo de re-engenharia do sistema foi iniciado. Neste, segui as etapas listadas abaixo:

- Descrição da proposta do projeto
- Nova descrição dos requisitos funcionais e não-funcionais do projeto – cenários
- Relacionamento de Cenários
- Descrição dos UID's
- Descrição do modelo conceitual
- Descrição do modelo navegacional
- Descrição do modelo entidade-relacionamento – MER
- Desenvolvimento das classes bases do sistema – classes de entidades
- Desenvolvimento das callbacks do sistema
- Criação de skins
- Desenvolvimento de funcionalidades comportamentais
- Criação de folhas de estilo

Em paralelo a este processo foi desenvolvido uma engine que serviria de base para a aplicação, a WebEngine, que já foi apresentada anteriormente neste documento. Os documentos gerados como resultados desse processo serão apresentados mais adiante nos tópicos seguintes.

21.5 – PROPOSTA

Para seguir a linha de desenvolvimento de software livre, foi utilizado o conjunto AMP(Apache, MySQL e PHP), todos livres, entretanto é possível optar por rodar em Linux ou Windows.

A motivação para evoluir o C&L se baseia no desejo de desenvolver um software de apoio à descrição de requisitos, planejamento e desenvolvimento de softwares rodando na web, que seja um software open-source modelo e um referencial de ferramenta de engenharia de requisitos para representação e edição de cenários.

A proposta dessa nova versão do C&L é desenvolvê-lo baseando-o na WebEngine. Evoluindo a versão atual em diversos aspectos:

- Aplicação modularizada

- Código orientado a objetos
- Baixo acoplamento
- Processamento separado da interface gráfica com usuário
- Possibilidade do uso de diferentes front-ends
- Facilidade de futuras evoluções

Além das funcionalidades básicas, o C&LEvolution conta também com funcionalidades oriundas de versões antigas, com o conceito "view code", que disponibiliza todo seu código fonte diretamente na aplicação. Também nesta versão, foram incluídas diversas funcionalidades que versões anteriores não possuíam:

- Importação e exportação de projeto utilizando XML
- Atrelação de documentos – upload de arquivos
- Inserção de usecases
- Relatórios em PDF

21.6 – CENÁRIOS E LÉXICOS

Título	Acessar ao Sistema
Objetivo	Permitir que o usuário acesse ao Sistema de Cenários e Léxicos
Contexto	Pré-Condição: Usuário ter acessado à Internet via browser
Atores	Usuário, Sistema, Suporte.
Recursos	URL de acesso ao sistema
Episódios	O usuário digita a URL de acesso ao sistema no seu computador. O sistema apresenta uma tela principal. O sistema é apresentado para o usuário, disponibilizando lhe as opções: - LOGAR NO SISTEMA, caso seja cadastrado e deseja se logar. - LEMBRAR SENHA, caso seja cadastrado e esqueceu sua senha, e deseja se logar.

	- CADASTRAR USUÁRIO, caso não seja cadastrado e deseja se cadastrar. - INFO, caso o usuário deseje conhecer informações sobre o sistema. - ENTRAR EM CONTATO, caso o usuário deseje entrar em contato com o suporte do sistema. - EXIBIR AJUDA, no qual o usuário terá a possibilidade ler a ajuda do sistema.
Léxico	Usuário::sujeito: pessoa que utiliza o sistema. Sistema::objeto: objeto de aplicação Suporte::sujeito: pessoa responsável pela manutenção e atendimento do sistema. URL de acesso::objeto: conjunto de caracteres único identificadores do sitio da aplicação.

❒ Tabela 21.2 – C&L - Acessar ao Sistema

Título	Cadastrar Usuário
Objetivo	Permitir o usuário que não esteja cadastrado se cadastrar no sistema.
Contexto	Usuário não está cadastrado no sistema e deseja se cadastrar. Usuário está na tela inicial do sistema. Pré-Condições: Acessar ao sistema
Atores	Usuário, Sistema.
Recursos	Dados cadastrais do usuário.
Exceção	Usuário já cadastrado no sistema. Caso o username já exista, o sistema retornará uma mensagem de erro na tela informando que este username já existe. Confirmação de senha não confere com a senha. Neste caso o sistema retornará uma mensagem de erro na tela informando que a confirmação não confere. Confirmação de email não confere com a email. Neste caso o sistema retornará uma mensagem de erro na tela informando que a confirmação não confere.
Episódios	Usuário seleciona a opção de CADASTRAR USUÁRIO. O sistema retorna para o usuário uma interface com campos para entrada de username, senha, e confirmação de senha, nome, email, e confirmação do email, país, telefone de fixo e cel .

	Usuário então preenche o bloco de informações e clica em enviar. O Sistema cadastra o usuário no banco de dados, devolve uma mensagem de confirmação de cadastro e o redireciona para a interface inicial do sistema.
Léxico	Dados cadastrais do usuário::objeto: elementos de informação ao sistema sobre o usuário. Sitio:: objeto: local na web onde está hospedado a aplicação Username::objeto: conjunto de caracteres único identificador do usuário. Interface::objeto: elemento visual responsável pela comunicação e controle da aplicação pelo usuário.

❒ Tabela 21.3 – C&L - Cadastrar Usuário

Título	Entrar em contato com suporte
Objetivo	Permitir que o usuário possa tirar dúvidas, dar sugestões e/ou entrar em contato com suporte do sistema.
Contexto	Usuário está no sistema e deseja se comunicar com a equipe de suporte da aplicação. Usuário está na tela inicial do sistema. Pré-Condições: Acessar ao sistema.
Atores	Usuário
Recursos	Username, sistema.
Exceção	
Episódios	Usuário seleciona a opção de CONTACTAR SUPORTE. A aplicação redireciona o usuário para a tela correspondente, com campos para entrada dos dados email remetente, assunto, e o corpo da mensagem. Caso o usuário selecione a opção de desistir de se comunicar com a equipe o sistema o redireciona para a página inicial da aplicação. Caso contrário, o usuário preenche os campos. Usuário seleciona a opção de enviar. O sistema então retorna ao usuário uma mensagem de confirmação de envio de mensagem.

Léxico	

❒ Tabela 21.4 – C&L - Entrar em contato com suporte

Título	Recuperar senha
Objetivo	Permitir o usuário cadastrado, que esqueceu sua senha, recupere a senha do sistema
Contexto	Usuário esqueceu sua senha Pré-Condição: Acessar ao sistema
Atores	Usuário, Sistema
Recursos	Banco de dados
Exceção	Caso não exista nenhum username cadastrado igual ao informado pelo usuário, sistema exibe mensagem de erro na tela informando que username é inexistente.
Episódios	O usuário seleciona a opção LEMBRAR SENHA. O sistema apresenta uma interface pedindo ao usuário que digite o seu username na caixa de texto, e apresenta duas opções: RECUPERAR ou VOLTAR. Caso o usuário selecione voltar o sistema o redirecionará para a tela inicial do sistema. Caso contrário, o usuário digita o seu username e clica no botão de RECUPERAR. Sistema consulta no banco de dados qual o email e senha do username informado. Sistema envia a senha para o email cadastrado correspondente ao username que foi informado pelo usuário.
Léxico	

❒ Tabela 21.5 – C&L - Recuperar senha

Título	Exibir informações do sistema
Objetivo	Permitir que os usuários obtenham conhecimento sobre a aplicação.
Contexto	Usuário deseja ver as informações da aplicação. Pré-Condições: Acessar ao sistema
Atores	Usuário, Sistema
Recursos	Sistema
Exceção	
Episódios	Usuário seleciona a opção INFO, para visualizar informações do sistema. A aplicação exibirá as informações do sistema.
Léxico	

❒ Tabela 21.6 – C&L - Exibir informações do sistema

Título	Exibir ajuda
Objetivo	Permitir ao Usuário ver a ajuda do sistema.
Contexto	Usuário deseja ler a ajuda do sistema. Pré-Condição: Acessar ao sistema.
Atores	Usuário, Sistema
Recursos	

Exceção	
Episódios	O usuário escolhe a opção EXIBIR AJUDA. O sistema exibirá o conteúdo da ajuda descrevendo o funcionamento do sistema.
Léxico	

❒ Tabela 21.7 – C&L - Exibir ajuda

Título	Logar no sistema
Objetivo	Permitir ao usuário entrar no sistema.
Contexto	Usuário sabe a sua senha Usuário deseja entrar no sistema com seu perfil Pré-Condição: Acessar ao sistema
Atores	Usuário, Sistema
Recursos	Banco de dados
Exceção	Caso o username não seja encontrado no banco de dados o sistema retorna uma mensagem de erro para o usuário o informando. Caso a senha não confere com a senha relacionada ao usuário no banco de dados o sistema retorna uma mensagem de erro para o usuário o informando o erro.
Episódios	O usuário fornece para o sistema um username e uma senha. O sistema autentica esta senha para este usuário, consultando o banco de dados, carrega o perfil do usuário. O sistema redireciona para a tela inicial de usuário logado. O sistema fornece ao usuário as opções: - ALTERAR CADASTRO, no qual o usuário terá a possibilidade de realizar alterações nos seus dados cadastrais. - SAIR do sistema, no qual o usuário terá a possibilidade de sair da sessão e se logar novamente.

	- BUSCA, caso o usuário deseje fazer uma busca. - PROJETOS, caso o usuário deseje ver seus projetos. - CRIAR PROJETO, para criar novos projetos. - PUBLICAÇÔES, caso o usuário deseje ver publicações. - RETIRAR CADASTRO, no qual o usuário se desassocia do sistema
Léxico	Perfil::objeto: conjunto de informações sobre o usuário

❒ Tabela 21.8 – C&L - Logar no sistema

Título	Alterar cadastro
Objetivo	Permitir ao usuário realizar alteração nos seus dados cadastrais
Contexto	Usuário deseja alterar seus dados cadastrais Pré-Condição: Logar no sistema
Atores	Usuário, Sistema
Recursos	
Exceção	Confirmação de senha não confere com a senha. Neste caso o sistema retornará uma mensagem de erro na tela informando que a confirmação não confere. Confirmação de email não confere com a email. Neste caso o sistema retornará uma mensagem de erro na tela informando que a confirmação não confere.
Episódios	O usuário seleciona a opção ALTERAR CADASTRO. O sistema fornecerá para o usuário uma tela com um formulário que contém campos com possibilidade de alteração para os seguintes dados do usuário: username, nome, email, confirmação de email, senha e confirmação da senha, pais, telefone e celular. O usuário altera os dados desejados. O usuário salva as alterações.
Léxico	

❒ Tabela 21.9 – C&L - Alterar cadastro

Título	Sair do sistema
Objetivo	Permitir ao usuário sair do sistema, mantendo a integridade do que foi realizado.
Contexto	Usuário deseja sair da aplicação e manter a integridade do que foi realizado. Pré-Condição: Logar no sistema.
Atores	Usuário, Sistema
Recursos	
Exceção	
Episódios	O usuário seleciona a opção de SAIR. O sistema fecha a sessão do usuário, mantendo a integridade do que foi realizado e o redireciona para a interface de login.
Léxico	

❒ Tabela 21.10 – C&L - Sair do sistema

Título	Efetuar busca
Objetivo	Permitir ao usuário realizar uma busca de informações sobre os dados existentes no sistema.
Contexto	Usuário deseja efetuar uma busca no sistema. Pré-condição: Logar no sistema.
Atores	Usuário, Sistema

Recursos	Interface
Exceção	
Episódios	Usuário seleciona a opção de efetuar BUSCA no sistema. A aplicação o redireciona para a interface de busca, apresentando campos para preencher a palavra chave na qual será feita a busca e o escopo no qual será executada a busca. O escopo pode ser escolhido entre as seguintes opções: todos, projetos públicos, projetos relacionados ao usuário, projeto corrente. A aplicação indica ao usuário o resultado da busca, listando para cada objeto encontrado o nome da entidade de sistema, o tipo da entidade, o projeto relacionado, o autor do projeto, uma pequena descrição, e um link para o objeto caso isso seja possível.
Léxico	Escopo::objeto: alvo, mira, parte de um todo onde se deseja atuar.

❒ Tabela 21.11 – C&L - Efetuar Busca

Título	Retirar cadastro
Objetivo	Permitir ao usuário desassociar-se do sistema.
Contexto	Usuário está cadastrado no sistema e deseja se desassociar. Pré-condição: Logar no sistema.
Atores	Usuário, Sistema
Recursos	
Exceção	
Episódios	O usuário seleciona a opção de se RETIRAR CADASTRO. A aplicação então o redireciona para a tela correspondente, exibindo uma pergunta de confirmação.

	O usuário confirma que deseja retirar o cadastro ou cancelar desvinculação. Caso o usuário tenha selecionado a opção de se desassociar-se a aplicação efetua o desligamento e redireciona o usuário para a tela de login.
Léxico	Desassociar::verbo: separar, desligar, desunir o que estava associado, desvincular

❒ Tabela 21.12 – C&L - Retirar cadastro

Título	Obter lista de projetos
Objetivo	Permitir ao usuário obter a lista de projetos relacionados a ele.
Contexto	Usuário deseja ver lista de projetos que participa. Pré-condição: Logar no sistema.
Atores	Usuário, Sistema
Recursos	Publicações, perfil do usuário, projetos.
Exceção	Caso o usuário não participe de nenhum projeto a lista de projetos é vazia.
Episódios	Usuário seleciona a opção PROJETOS. O sistema retorna para o usuário uma lista contendo os projetos que participa, apresentando as seguintes informações do projeto: Nome do projeto, Gerente e Data de criação.
Léxico	Projeto::objeto: entidade relacionada ao sistema que engloba outras entidades da aplicação.

❒ Tabela 21.13 – C&L - Obter lista projetos

Título	Selecionar projeto
Objetivo	Selecionar o projeto um projeto na lista de projetos

Contexto	Usuário deseja selecionar um projeto. Pré-condição: Obter lista de projetos.
Atores	Usuário, Sistema
Recursos	Projeto corrente, Lista de projetos, Opções de ações do projeto.
Exceção	Usuário não é o gerente do projeto selecionado, então ele não tem a opção de associar usuário ao projeto, nem a opção de verificar alterações, e também não tem a opção de excluir projeto.
Episódios	Na lista de projetos, usuário clica no nome do projeto. O sistema apresenta os dados do projeto selecionado, marca projeto como projeto corrente, e apresenta as opções: - INCLUIR DOCUMENTAÇÃO EXTRA, para anexar um documento extra ao projeto. - Obter LISTA DE DOCUMENTAÇÂO ANEXADA - ASSOCIAR USUÁRIO, para o usuário gerente associar outro usuário ao projeto. - EXCLUIR PROJETO, para o usuário gerente excluir o projeto corrente. - Obter LISTA DE ALTERAÇÕES, caso o usuário deseje ver alterações realizadas no projeto por outros usuários envolvidos no projeto. - PUBLICAR PROJETO, para publicar o projeto no sistema - Obter LISTA DE SÍMBOLOS - Obter LISTA DE CENÁRIOS - CRIAR SÍMBOLO - CRIAR CENÁRIO
Léxico	

❒ Tabela 21.14 – C&L - Selecionar projeto

Título	Criar novo projeto
Objetivo	Criar um novo projeto em branco tendo o usuário criador como gerente.

Contexto	Usuário deseja criar um projeto em branco novo para gerenciar. Pré-condição: Acessar ao sistema.
Atores	Usuário, gerente, sistema.
Recursos	Opções de ações do projeto.
Exceção	
Episódios	Usuário seleciona a opção de CRIAR PROJETO. Sistema o redireciona para uma tela que possui um formulário com os campos para preenchimento de nome do projeto, e uma pequena descrição do mesmo. Usuário preenche os campos e envia as informações para o sistema. Sistema processa as informações, seta o novo projeto como projeto corrente e apresenta as opções de ações sobre o projeto.
Léxico	Gerente::pessoa: pessoa responsável pelo projeto.

❒ Tabela 21.15 – C&L - Criar novo projeto

Título	Excluir projeto
Objetivo	Excluir um projeto do banco de dados.
Contexto	Usuário é gerente do projeto corrente e deseja excluir este projeto de sua lista de projetos. Pré-condição: Selecionar projeto.
Atores	Usuário, Gerente, Sistema.
Recursos	Projeto
Exceção	Usuário não é gerente do projeto, então ele não pode efetuar a exclusão do projeto.

Episódios	Usuário seleciona a opção de excluir projeto. Sistema pede confirmação de ação. Caso o usuário confirme o sistema exclui os dados do projeto do banco de dados e confirma a ação executada para o usuário.
Léxico	

❒ Tabela 21.16 – C&L - Excluir projeto

Título	Obter lista de publicações
Objetivo	Permitir ao usuário verificar as publicações efetuadas na aplicação.
Contexto	Usuário deseja ver publicações feitas no sistema. Pré-condição: Logar no sistema.
Atores	Usuário, Sistema
Recursos	Publicações, perfil do usuário, projetos.
Exceção	Usuário não é o usuário gerente do projeto, então ele não terá a opção de apagar publicações do projeto.
Episódios	Usuário seleciona a opção PUBLICAÇÔES. O sistema exibe lista de todos os projetos publicados, oferecendo as seguintes opções: - Ver publicações sobre todos os projetos. - Ver publicações de todos os projetos relacionados ao seu perfil. - Ver publicações sobre de um determinado autor. Listando para ambos os casos o nome do projeto, autor, tipo do arquivo, data de criação, VISUALIZAR.
Léxico	Publicação::objeto: publicação de um projeto em formato integro. Projeto corrente::objeto: projeto indicado pelo usuário ao qual serão refletidas ações futuras.

❒ Tabela 21.17 – C&L - Obter lista de publicações

Título	Realizar publicação
Objetivo	Realizar uma publicação de um projeto
Contexto	Usuário deseja publicar um projeto seu. Pré-condição: Selecionar projeto.
Atores	Usuário, Sistema
Recursos	Projeto, Publicação, Projeto corrente.
Exceção	Somente o gerente do projeto pode publicar um projeto.
Episódios	Usuário seleciona a opção de PUBLICAR PROJETO. A aplicação oferece ao usuário dois modos para realizar a publicação, uma em formato PDF e outra em formato XHTML. O usuário seleciona a opção desejada. O sistema, para os dois formatos, processará a publicação sobre o projeto corrente e confirmará para o usuário a publicação.
Léxico	

❒ Tabela 21.18 – C&L - Realizar publicação

Título	Visualizar publicação
Objetivo	Visualizar conteúdo do documento publicado de um projeto.
Contexto	Usuário deseja visualizar documento de projeto publicado. Pré-condição: Obter lista de publicações.

Atores	Usuário, Sistema
Recursos	Lista de publicações
Exceção	
Episódios	O usuário seleciona a opção de visualizar publicação. O sistema exibe o documento publicado.
Léxico	

❒ Tabela 21.19 – C&L - Visualizar publicação

Título	Excluir publicação
Objetivo	Excluir uma publicação existente no sistema.
Contexto	Usuário deseja excluir documento de projeto publicado. Pré-condição: Obter lista de publicações.
Atores	Usuário, Sistema
Recursos	Lista de publicações
Exceção	Usuário não é o usuário gerente do projeto, então ele não terá a opção de apagar alguma publicação daquele projeto. Usuário não confirma exclusão.
Episódios	O usuário seleciona a opção de EXCLUIR publicação. O sistema pedirá confirmação. Caso positivo o sistema excluirá a publicação existente.
Léxico	

❒ Tabela 21.20 – C&L - Excluir publicação

Título	Obter lista de documentos anexados
Objetivo	Permitir ao usuário visualizar os documentos anexados no projeto.
Contexto	Usuário deseja ver documentos anexados ao projeto. Pré-condição: Selecionar projeto.
Atores	Usuário, Sistema
Recursos	Publicações, perfil do usuário, projetos.
Exceção	Usuário não é o usuário gerente do projeto, então ele não terá a opção de apagar documento anexado ao projeto.
Episódios	Usuário seleciona a opção LISTA DE DOCUMENTOS ANEXADOS. O sistema exibe lista de todos os documentos anexados oferecendo as seguintes opções: EXCLUIR ANEXO e BAIXAR ANEXO.
Léxico	Publicação::objeto: publicação de um projeto em formato integro. Projeto corrente::objeto: projeto indicado pelo usuário ao qual serão refletidas ações futuras.

❒ Tabela 21.21 – C&L - Obter lista de documentos anexados

Título	Incluir documento extra
Objetivo	Usuário deseja anexar documento extra ao projeto.
Contexto	Usuário possui um documento digital e deseja associar este documento ao projeto. Pré-condição: Selecionar projeto.
Atores	Usuário, sistema

Recursos	Projeto corrente
Exceção	O nome do arquivo de documentação não pode ser o mesmo de algum já existente.
Episódios	Usuário seleciona a opção INCLUIR DOCUMENTAÇÃO EXTRA. Sistema apresenta para o usuário a lista de documentações extras já incluídas e um campo para adicionar novas documentações extras. O usuário preenche o campo com o endereço completo em seu computador do arquivo digital, o nome que o arquivo receberá no sistema e o tipo do arquivo, e seleciona a opção de enviar o arquivo. O sistema carrega o arquivo para o banco de dados e o exibe na lista junto com os outros arquivos já existentes.
Léxico	

❒ Tabela 21.22 – C&L - Incluir documento extra

Título	Excluir documento extra
Objetivo	Excluir um documento anexado ao sistema.
Contexto	Usuário deseja excluir documento anexado. Pré-condição: Obter lista de publicações.
Atores	Usuário, Sistema
Recursos	Lista de documentos anexados
Exceção	Usuário não é o usuário gerente do projeto, então ele não terá a opção de apagar nenhum documento extra daquele projeto. Usuário não confirma exclusão.
Episódios	O usuário seleciona a opção de EXCLUIR ANEXO. O sistema pedirá confirmação. Caso positivo o sistema excluirá o documento existente.

Léxico	

❒ Tabela 21.23 – C&L - Excluir documento extra

Título	Associar usuário
Objetivo	Permitir um usuário outro usuário participar de um projeto.
Contexto	Usuário gerente do projeto deseja adicionar usuário do sistema para participar de seu projeto. Pré-condição: Selecionar projeto.
Atores	Usuário, Gerente, Sistema
Recursos	Lista de usuários do sistema, Lista de usuários do projeto, banco de dados.
Exceção	Usuário não é o gerente do projeto, então ele não pode executar a opção de associar usuários ao projeto corrente.
Episódios	Usuário seleciona a opção de ASSOCIAR USUARIO ao projeto. Sistema o redireciona para tela correspondente que terá uma lista de usuários cadastrados no sistema e uma lista de usuários associados ao projeto. O usuário então tem a opção de mover usuários de uma lista para a outra, incluindo e excluindo usuários da lista de participantes do projeto. Usuário envia alterações para o sistema. Sistema salva alterações no banco de dados.
Léxico	

❒ Tabela 21.24 – C&L - Associar usuário

Título	Obter lista de alterações

Objetivo	Obter lista de alterações efetuadas no projeto corrente por usuários participantes do projeto.
Contexto	Usuário gerente do projeto deseja ver a lista de alterações efetuadas no projeto por usuários participantes. Pré-condição: Selecionar projeto.
Atores	Usuário, sistema, gerente.
Recursos	Lista de alterações
Exceção	Usuário não é o gerente do projeto, então ele não possui a opção de APROVAR ALTERAÇÔES e RECUSAR ALTERAÇÕES.
Episódios	Usuário seleciona a opção de OBTER LISTA DE ALTERAÇÕES realizadas. O sistema apresenta para o usuário uma lista com as modificações, inclusões ou exclusões feitas por outros usuários participantes do projeto. E apresenta para cada alteração o nome do usuário que a fez, a data em que foi feita, em que entidade do projeto foi feita, e as seguintes opções: - VISUALIZAR ALTERAÇÂO, que mostrará ao usuário as modificações feitas em uma determinada entidade do sistema. - ACEITAR ALTERAÇÃO, que irá salvar as alterações, criando então uma nova versão para a entidade e apagando a anterior. - RECUSAR ALTERAÇÃO, que apagará as alterações e manterá a versão atual da entidade.
Léxico	Entidade do sistema::objeto: partes do sistema, e.g. cenário, léxico...

❒ Tabela 21.25 – C&L - Obter lista de alterações

Título	Visualizar alteração
Objetivo	Visualizar uma alteração da lista de alterações efetuadas por usuários participantes do projeto sobre o projeto corrente.
Contexto	Usuário gerente do projeto deseja visualizar detalhes de uma alteração da lista de alterações efetuadas no projeto por usuários participantes. Pré-condição: Obter lista de alterações.

Atores	Usuário, sistema, gerente.
Recursos	Lista de alterações
Exceção	
Episódios	Usuário seleciona a opção de VISUALIZAR ALTERAÇÂO. O sistema redireciona o usuário para uma tela correspondente que terá os seguintes itens: - ORIGINAL, onde o usuário verá a entidade do sistema antes da alteração. - ALTERADO, onde o usuário verá a entidade do sistema com as modificações ressaltadas. O usuário gerente do projeto então pode optar por RECUSAR ou ACEITAR a alteração.
Léxico	

❒ Tabela 21.26 – C&L - Visualizar alteração

Título	Criar novo cenário
Objetivo	Adicionar novo cenário ao projeto.
Contexto	Usuário deseja adicionar novo cenário a um projeto. Pré-condição: Selecionar projeto.
Atores	Usuário, Sistema
Recursos	Dados a serem cadastrados
Exceção	Caso o usuário não possua nenhum projeto setado como corrente ele não possui esta opção. Cenário cadastrado, caso o usuário tente adicionar um cenário já existente no projeto o usuário receberá uma mensagem de notificação.

Episódios	Usuário seleciona a opção de CRIAR CENÁRIO. O sistema fornecerá para o usuário uma tela com um formulário com os seguintes campos: - Nome Cenário - Objetivo – Contexto::Localização geográfica – Contexto::Locação temporal – Contexto::pré-condição – Atores – Recursos – Exceção - Episódios O usuário então preenche os campos e os envia para o sistema. O sistema verifica a integridade dos dados e verifica também se já não existe algum cenário com o mesmo nome no projeto. Caso aceito o sistema cadastra o novo cenário e retorna uma mensagem de confirmação para o usuário.
Léxico	

❒ Tabela 21.27 – C&L - Criar novo cenário

Título	Criar novo símbolo
Objetivo	Adicionar novo símbolo ao projeto.
Contexto	Usuário deseja adicionar novo léxico a um projeto. Pré-condição: Selecionar projeto.
Atores	Usuário, Sistema
Recursos	Dados a serem cadastrados
Exceção	Caso o usuário não possua nenhum projeto setado como corrente ele não possui esta opção. Caso o usuário tente adicionar um símbolo do léxico já existente no projeto o usuário receberá uma mensagem de notificação de erro.

Episódios	Usuário seleciona a opção de CRIAR LÉXICO. O sistema fornecerá para o usuário uma tela com um formulário com os seguintes campos: - Nome do símbolo - Lista de sinônimos – Noção – Impacto O usuário então preenche os campos e os envia para o sistema. O sistema verifica a integridade dos dados e verifica também se já não existe algum léxico com o mesmo nome no projeto. Caso aceito o sistema cadastra o novo símbolo do léxico e retorna uma mensagem de confirmação para o usuário.
Léxico	

❒ Tabela 21.28 – C&L - Criar novo símbolo

Título	Ver lista de cenários
Objetivo	Visualizar todos os cenários do projeto em uma lista ordenada.
Contexto	Usuário possui um projeto com cenários e deseja ver uma lista dos cenários deste projeto. Pré-condição: Selecionar projeto.
Atores	Usuário, Sistema
Recursos	Lista de cenários
Exceção	Caso o usuário não possua nenhum projeto setado como corrente ele não possuirá esta opção. Não existe nenhum cenário no projeto então o usuário verá uma lista vazia.
Episódios	Usuário seleciona a opção de ver LISTA DE CENÁRIOS. Sistema executa a busca de cenários relacionados ao projeto corrente, redireciona o usuário para uma tela com a lista de cenários encontrada, listando para cada item da lista o nome do cenário, o autor, a data de criação, as opções VER CENÁRIO, EDITAR CENÁRIO, e EXCLUIR CENÁRIO.

Léxico	

❒ Tabela 21.29 – C&L - Ver lista de cenários

Título	Ver lista de símbolos
Objetivo	Visualizar todos os símbolos do projeto em uma lista ordenada.
Contexto	Usuário possui um projeto com símbolos e deseja ver uma lista dos símbolos deste projeto. Pré-condição: Selecionar projeto.
Atores	Usuário, Sistema
Recursos	Lista de símbolos
Exceção	Usuário não possui nenhum projeto setado como corrente, neste caso o usuário não possui esta opção. Não existe nenhum símbolos no projeto então o usuário verá uma lista vazia.
Episódios	Usuário seleciona a opção de ver LISTA DE SÍMBOLOS. Sistema executa a busca de símbolos relacionados ao projeto corrente, redireciona o usuário para uma tela com a lista de símbolos encontrada, listando para cada item da lista o nome do símbolo, o autor, a data de criação, e as opções VER SÍMBOLO, EDITAR SÍMBOLO e EXCLUIR SÍMBOLO.
Léxico	

❒ Tabela 21.30 – C&L - Ver lista de símbolos

Título	Visualizar cenário
Objetivo	Visualizar detalhes do conteúdo do cenário.

Contexto	Usuário deseja visualizar detalhes de um cenário cadastrado em um de seus projetos. Pré-condição: Ver lista de cenários.
Atores	Usuário, Sistema
Recursos	Lista de cenários, projeto corrente.
Exceção	Usuário não possui nenhum projeto setado como corrente, neste caso o usuário não possui esta opção. Não existe nenhum cenário cadastrado no projeto então o usuário não terá esta opção.
Episódios	Usuário seleciona a opção de VER CENÁRIO para ver detalhes do cenário na lista de cenários. O sistema o redireciona para tela correspondente mostrando todas as informações sobre o cenário e as opções EXCLUIR CENÁRIO, EDITAR CENÁRIO, ASSOCIAR ARQUIVO e ver LISTA DE ARQUIVOS ASSOCIADOS.
Léxico	

❒ Tabela 21.31 – C&L - Visualizar cenário

Título	Visualizar símbolo
Objetivo	Visualizar detalhes do conteúdo do símbolo.
Contexto	Usuário deseja visualizar detalhes de um símbolo cadastrado em um de seus projetos. Pré-condição: Ver lista de símbolos.
Atores	Usuário, Sistema
Recursos	Lista de símbolos, projeto corrente.

Exceção	Usuário não possui nenhum projeto setado como corrente, neste caso o usuário não possui esta opção. Não existe nenhum símbolo cadastrado no projeto então o usuário não terá esta opção.
Episódios	Usuário seleciona a opção de VER SÍMBOLO, para ver detalhes do símbolo na lista de símbolos. O sistema o redireciona para tela correspondente mostrando todas as informações sobre o símbolo e as opções EXCLUIR SÍMBOLO, EDITAR SÍMBOLO, ASSOCIAR ARQUIVO e ver LISTA DE ARQUIVOS ASSOCIADOS.
Léxico	

❒ Tabela 21.32 – C&L - Visualizar símbolo

Título	Excluir cenário
Objetivo	Excluir um cenário do projeto corrente.
Contexto	Usuário deseja excluir um cenário existente no projeto. Pré-condição: Ver lista de cenários ou Visualizar cenário.
Atores	Usuário, Sistema
Recursos	Cenário, projeto
Exceção	Não existe nenhum cenário na lista de cenários. Caso o usuário não seja o gerente do projeto, o cenário será colocado em uma área de espera, e só será excluído realmente do projeto quando o gerente autorizar. Caso o gerente do projeto reprove a exclusão o cenário volta para o banco de cenários. Usuário não confirma a operação.
Episódios	O usuário seleciona a opção EXCLUIR CENÁRIO. O sistema então pedirá confirmação da ação. O usuário confirma a exclusão. O sistema então processa a exclusão.

Léxico	

❒ Tabela 21.33 – C&L - Excluir cenário

Título	Excluir símbolo
Objetivo	Excluir um símbolo do projeto corrente.
Contexto	Usuário deseja excluir um símbolo existente no projeto. Pré-condição: Ver lista de símbolos ou Visualizar símbolo.
Atores	Usuário, Sistema
Recursos	Léxicos, projeto
Exceção	Não existe nenhum símbolo cadastrado no projeto então o usuário não terá esta opção. Caso o usuário não seja o gerente do projeto, o léxico será colocado numa área de espera, e só será excluído realmente do projeto quando o gerente autorizar. Caso o gerente do projeto reprove a exclusão o léxico volta para o banco de léxicos. Usuário não confirma a operação.
Episódios	O seleciona a opção de EXCLUIR SÍMBOLO. O sistema então pedirá confirmação da ação. O usuário confirma a exclusão. O sistema então processa a exclusão.
Léxico	Léxico::objeto: “símbolo”

❒ Tabela 21.35 – C&L - Excluir símbolo

Título	Associar arquivo

Objetivo	Associar arquivo anexado à um símbolo ou à um cenário.
Contexto	Usuário deseja associar um arquivo à um símbolo ou cenário. Pré-condição: Visualizar cenário ou Visualizar símbolo.
Atores	Usuário, Sistema
Recursos	Arquivo anexado
Exceção	
Episódios	Usuário seleciona a opção ASSOCIAR ARQUIVO à uma entidade do sistema. Sistema o redireciona para a tela correspondente com um formulário que contém os campos a serem preenchidos: nome do arquivo (de uma lista de arquivos anexados) e descrição do que é o arquivo para esta entidade. O usuário preenche os dados e envia para a aplicação. O sistema cria a associação e salva no banco de dados.
Léxico	Arquivo::objeto: documento digital.

❒ Tabela 21.35 – C&L - Associar arquivo

Título	Ver lista de arquivos associados
Objetivo	Visualizar todos os arquivos associados a uma entidade do sistema.
Contexto	Pré-condição: Visualizas cenário ou Visualizar símbolo.
Atores	Usuário, Sistema
Recursos	Lista de modelos e diagramas

Exceção	Não existe nenhum arquivo associado à entidade.
Episódios	Usuário seleciona a opção de ver LISA DE ARQUIVOS ASSOCIADOS. Sistema apresenta para o usuário para uma lista de arquivos associados à entidade, indicando para cada item da lista o nome do arquivo, descrição do que é o arquivo para esta entidade, informações adicionais do arquivo(documento extra), e as opções de DESASSOCIAR ARQUIVO e VER ARQUIVO.
Léxico	

❒ Tabela 21.36 – C&L - Ver lista de arquivos associados

Título	Desassociar arquivos
Objetivo	Desassociar um arquivo de uma entidade.
Contexto	Usuário deseja desassociar um arquivo de uma entidade do sistema. Pré-condição: Ver lista de arquivos associados.
Atores	Usuário, Sistema
Recursos	Arquivo anexado, projeto, liste de arquivos associados a entidade
Exceção	Não existe arquivo associado à entidade então o usuário não terá esta opção. Caso o usuário não seja o gerente do projeto, as modificações ficarão aguardando aprovação do gerente do projeto. Usuário não confirma operação.
Episódios	Na lista de arquivos associados à entidade o usuário seleciona a opção DESASSOCIAR ARQUIVO. O sistema então pedirá confirmação da ação. Usuário confirma operação. O sistema desassocia o arquivo anexado da entidade.
Léxico	

❒ Tabela 21.37 – C&L - Desassociar arquivo

Título	Visualizar árvore de arquivos
Objetivo	Possibilitar ao usuário visualizar árvore de arquivos que estão relacionados ao seu perfil no sistema.
Contexto	Usuário deseja ver arquivos e entidades relacionados ao seu perfil no sistema. Pré-condição: Logar no sistema.
Atores	Usuário, Sistema
Recursos	Árvore de arquivos
Exceção	Usuário não possui nenhum projeto relacionado ao seu perfil, neste caso o usuário receberá terá uma árvore vazia.
Episódios	Usuário seleciona a opção de visualizar árvore de arquivos. A aplicação verifica e relaciona os arquivos que possuem ligação com o usuário e em seguida exibe as informações para o usuário em uma lista de projetos, onde para cada projeto terá uma coleção de documentações anexadas, conjunto de símbolos e conjunto de cenários. Onde cada cenário terá uma lista de símbolos que faz referência.
Léxico	Árvore de arquivos::objeto: conjunto hierárquico de arquivos.

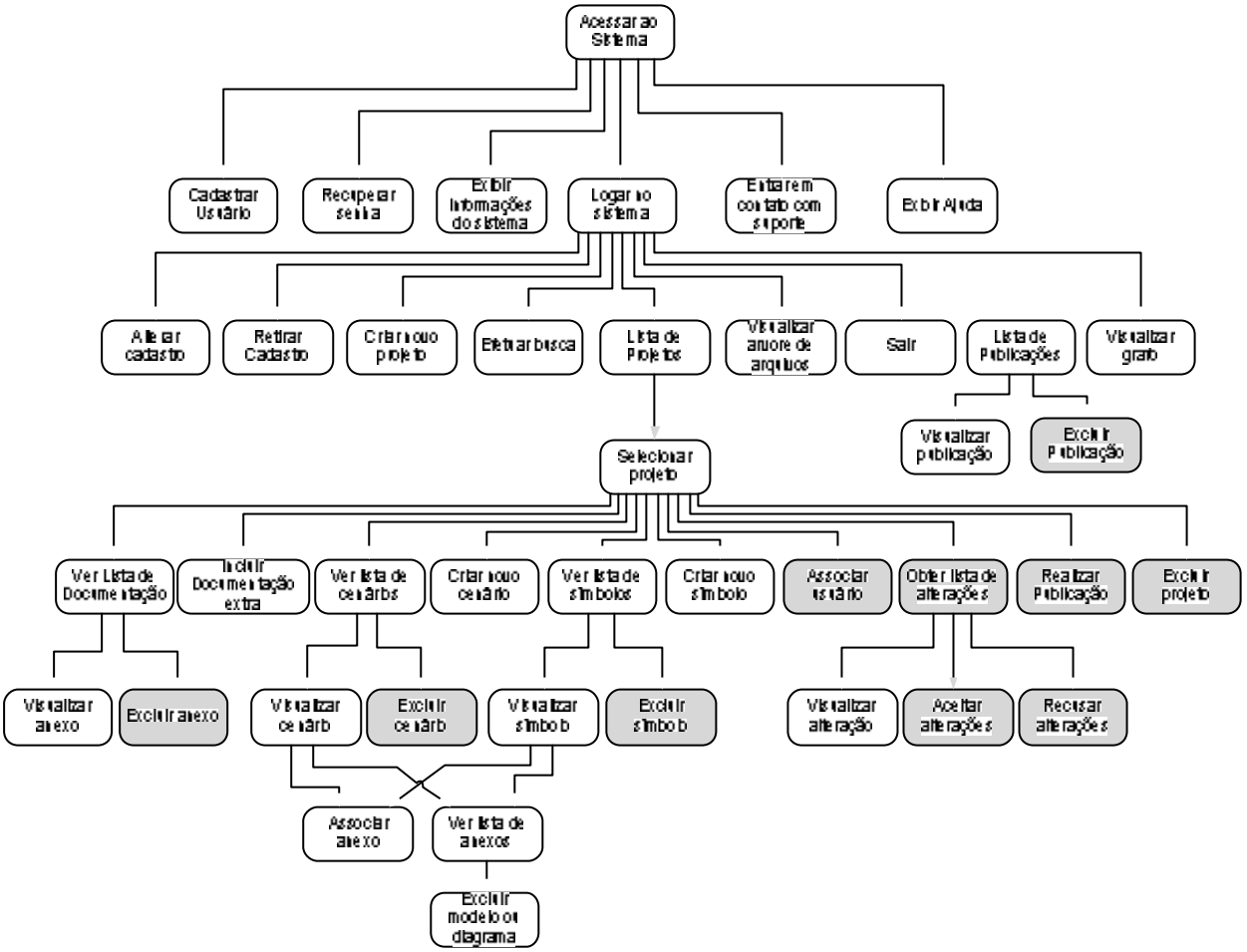
❒ Tabela 21.38 – C&L - Visualizar árvore de arquivos

Título	Visualizar grafo
Objetivo	Visualizar grafo de relacionamento entre entidades do sistema do mesmo projeto.
Contexto	Usuário deseja visualizar grafo de relacionamento entre entidades do sistema do projeto corrente. Pré-condição: Selecionar projeto.

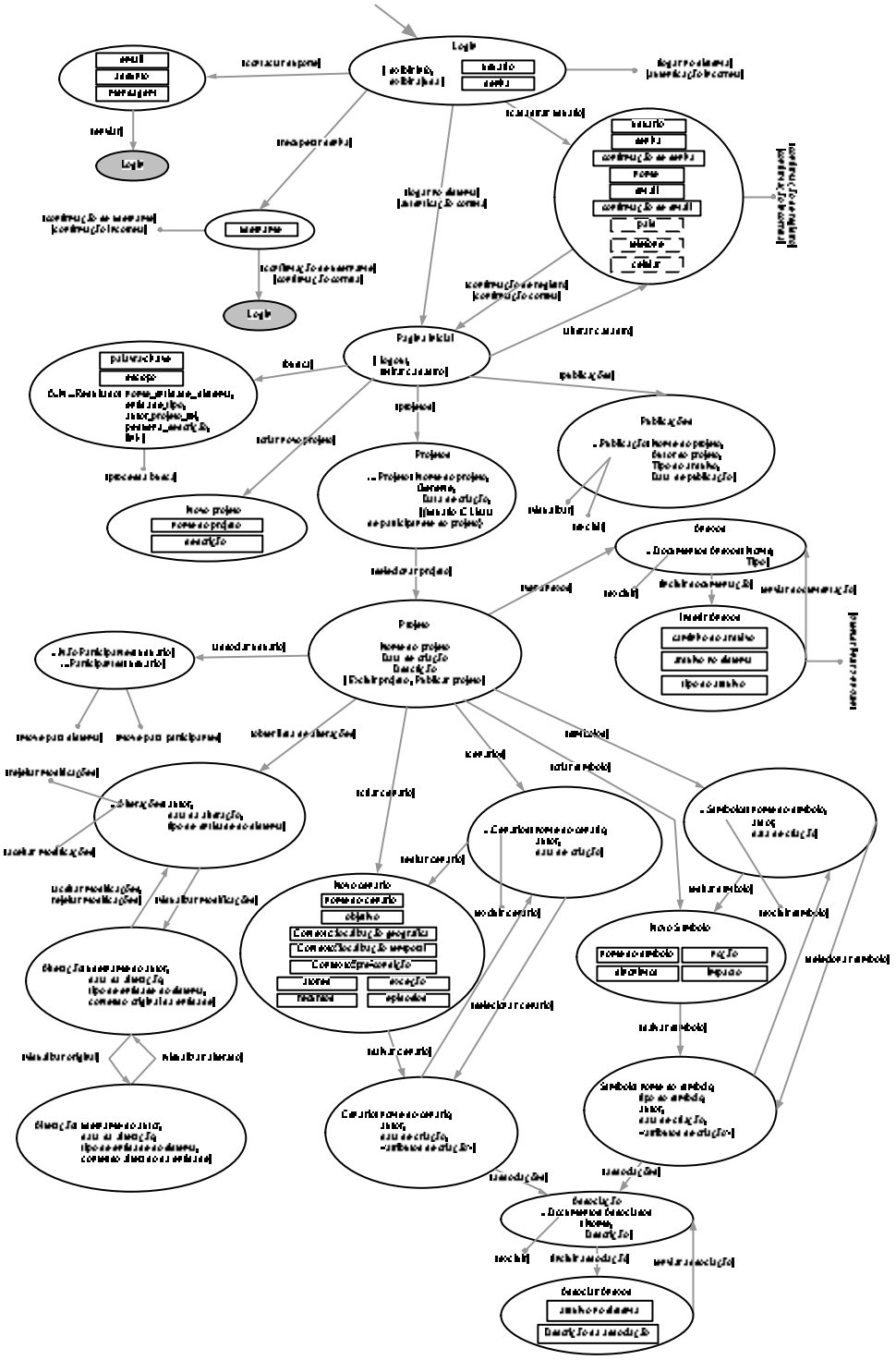
Atores	Usuário, Sistema
Recursos	Projeto corrente
Exceção	Usuário não possui nenhum projeto selecionado como projeto corrente, neste caso o usuário não possui esta opção.
Episódios	Usuário seleciona a opção de visualizar grafo de relacionamento. O sistema o redireciona para tela correspondente mostrando o grafo.
Léxico	

❒ Tabela 21.39 – C&L - Visualizar grafo

21.6 – RELACIONAMENTO DE CENÁRIOS



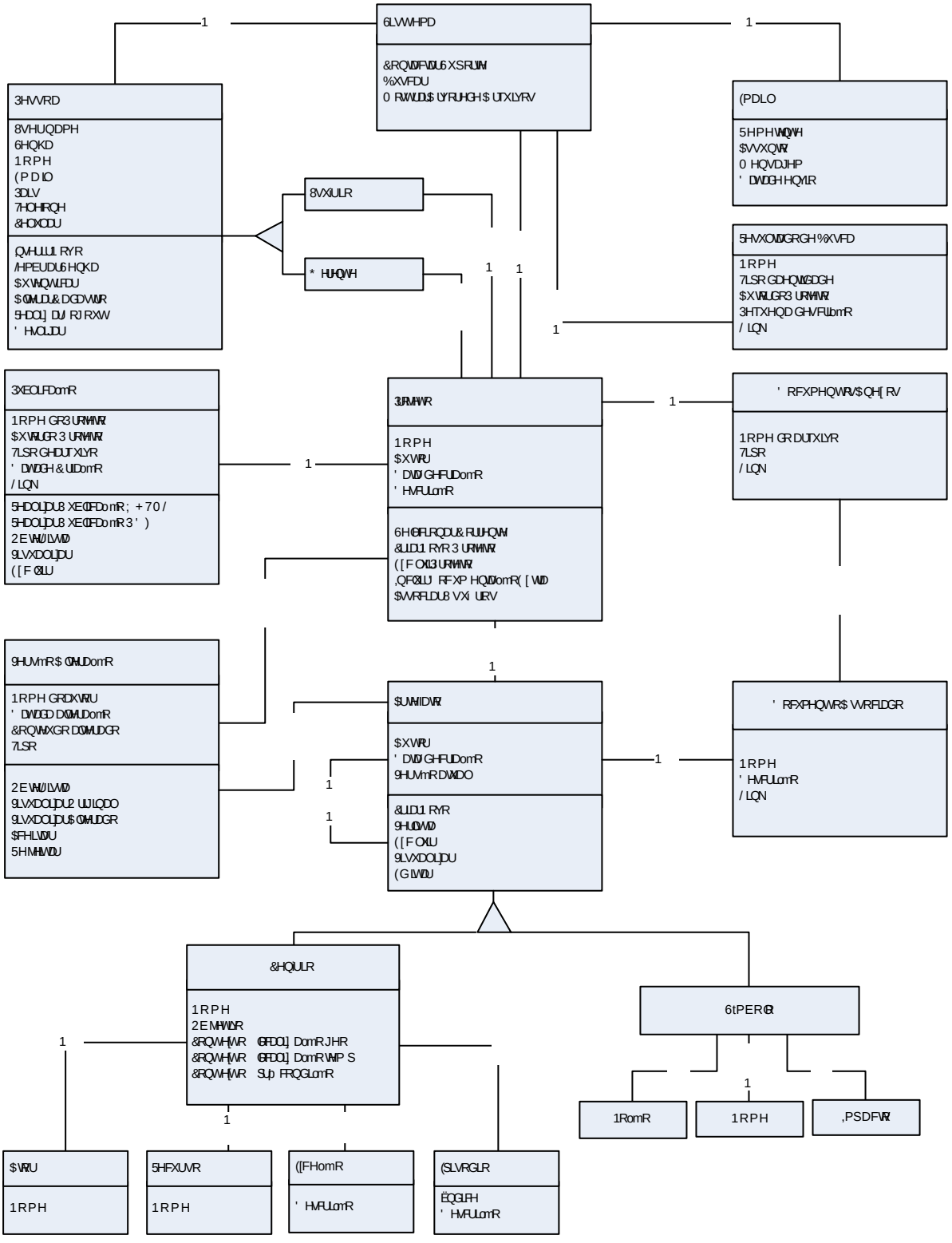
21.8 – UID'S – USER INTERATION DIAGRAM



❒ Figura 21.41 – UID – User Interation Diagram

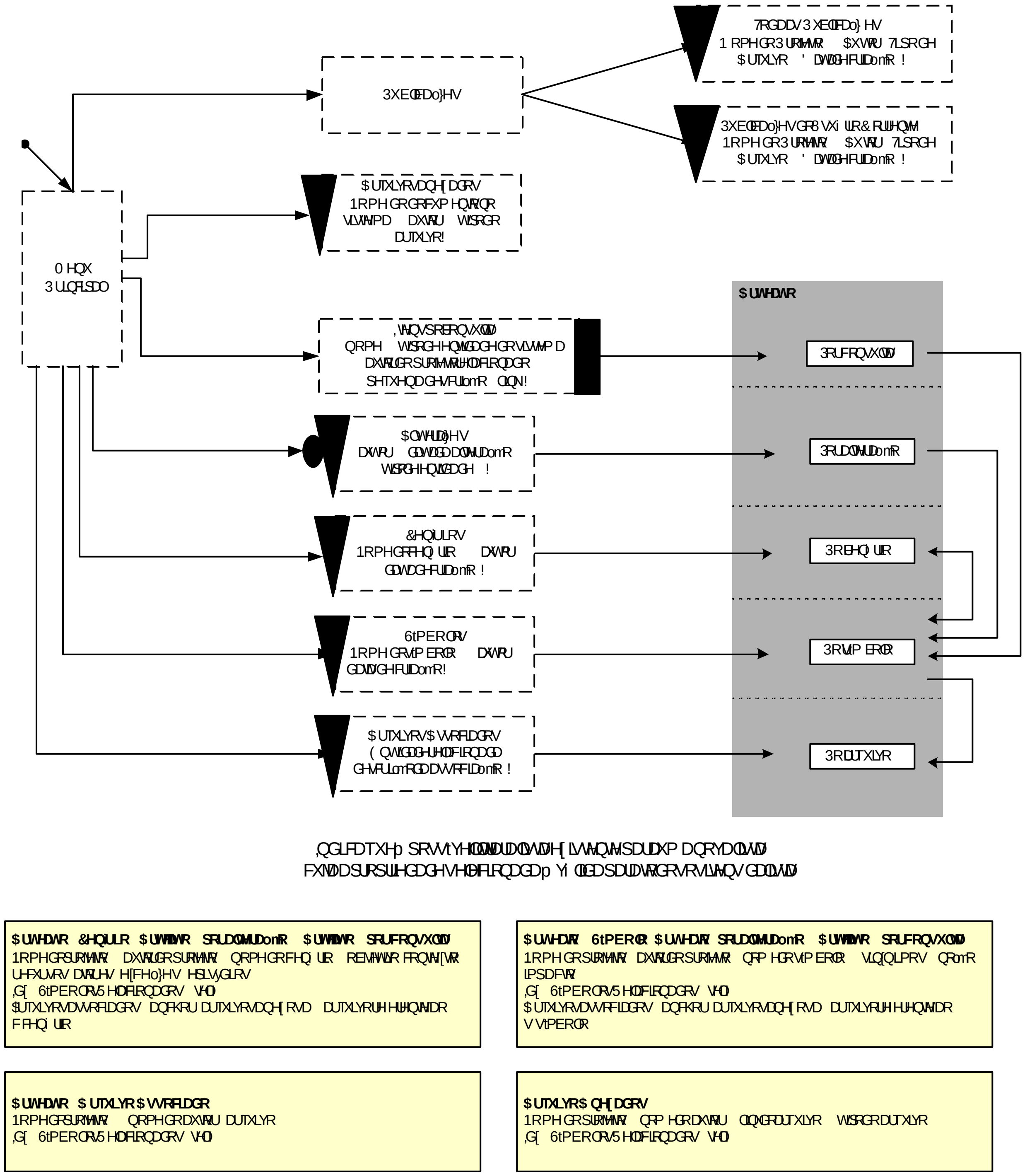
Para ver o arquivo em detalhes procure pela imagem em anexo: FIG21-41.vsd.

21.9 – MODELO CONCEITUAL



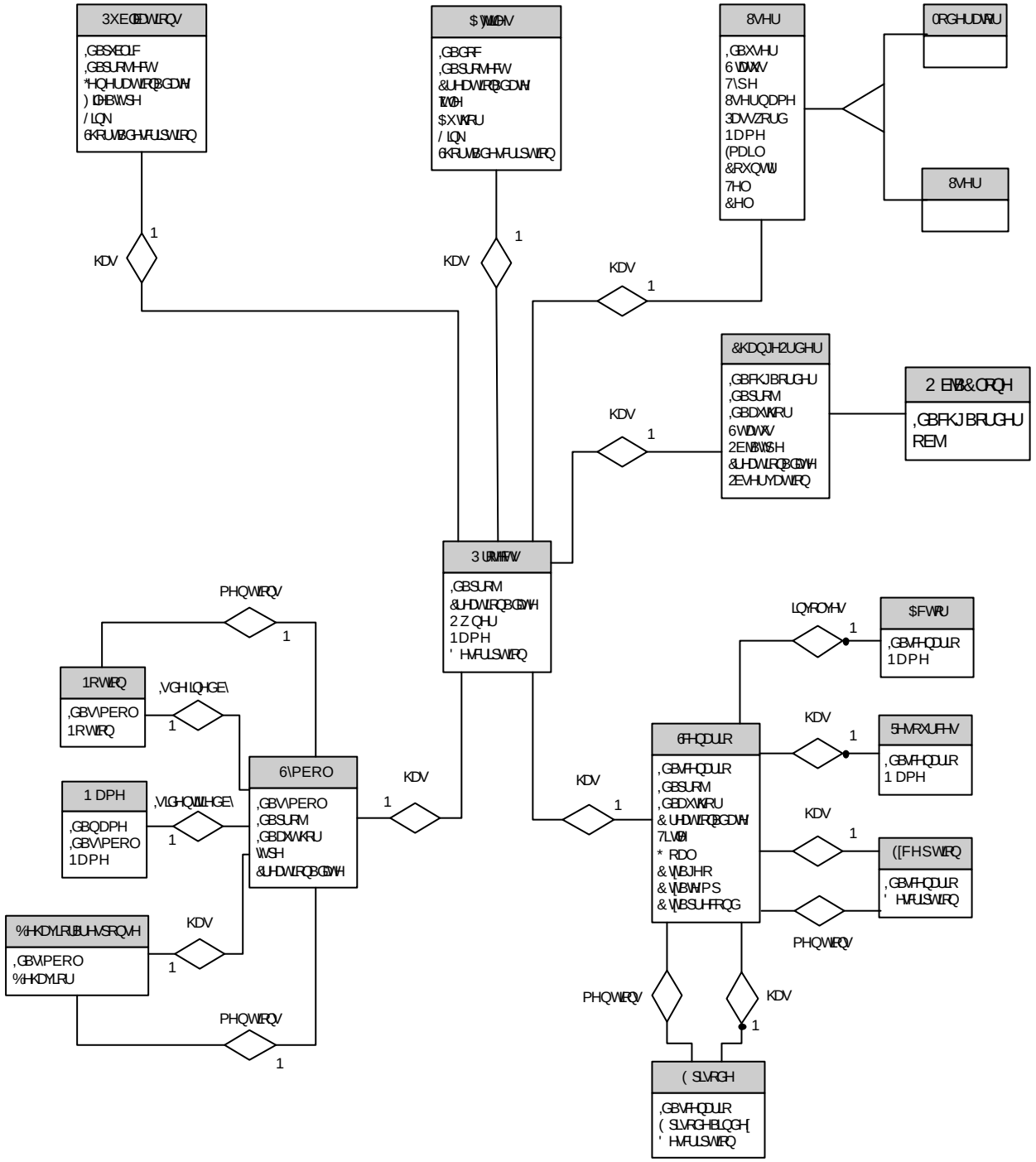
❒ Figura 21.42 – Modelo conceitual

21.10 – Modelo Navegacional



❐ Figura 21.43 – Modelo navegacional

21.11 – Modelo Entidade Relacionamento



❒ Figura 21.44 – Modelo entidade relacionamento

21.12 – O PROCESSO CONTINUA

O tempo que pude me dedicar a esse projeto não permitiu implementar todas as funcionalidades que eu havia planejado. Contudo, isso pode se tornar um ponto de partida

para motivar outros desenvolvedores a continuar a evoluir o C&L. Abaixo estão listadas algumas possibilidades:

- Tornar pública as informações de cenários e léxicos
- Permitir mais de um administrador por projeto de diferentes níveis de permissão
- Disponibilizar mecanismo de comunicação
- Mecanismo de inspeção de cenários e léxicos
- Servidor em três camadas – inserir camada de gerenciamento de dados e controle de acesso
- Disponibilizar editor de texto
- Disponibilizar editor gráfico - UML

CRONOGRAMA SIMPLIFICADO

Para o cronograma apresentado a seguir listei as atividades, planejadas, realizadas e não realizadas. Para cada atividade fora do comum, foi escrito uma observação. Utilizei a seguinte notação:

P	Planejado - Foi planejado para o projeto
A	Adicional - Foi realizado sem planejamento prévio
R	Realizado – foi realizado como planejado
F	Fração – realizada parcialmente. Estudado e analisado, mas não documentado.
I	Incompleto – Ainda esta em desenvolvimento
N	Não realizado – Não foi realizado como planejado

❒ Tabela 22.1 – Legenda do Cronograma

Grupo	Item	Estado	Observação
Estudo			
	Desenvolvimento web	PR	
	Padrões utilizados na web	PR	
	Engenharia de software	PR	
	Arquitetura de software	PR	
	Modelos arquiteturais	PR	
	Teste de Software	AF	Não foi planejado, foi estudado, mas por falta de tempo não foi

			documentado
	XML	AR	Foi adicionado este tópico porque as pesquisas iniciais e me levaram a esse caminho.
	DOM, SAX, XSL	AR	Foi adicionado este tópico porque as pesquisas iniciais e me levaram a esse caminho.
	Ajax	AR	Foi adicionado este tópico porque as pesquisas iniciais e me levaram a esse caminho.
Documentação WebEngine			
	Projeto Conceitual	PR	
	Projeto Arquitetural	PR	
	Conceitos adicionais	PR	
	Explicação geral da biblioteca	PR	
	API da Biblioteca	PR	
	Add-on's da engine	AR	Foi exposto somente as idéias adicionais.
	Análise de vantagens e desvantagens	PR	
	Testes	PR	
	Exemplos	PR	
	Conclusão	PR	
	Revisão da documentação gerada	PR	
Documentação C&L			
	Estado da Arte	PR	
	Proposta de evolução	PR	
	Cenários e Relacionamentos	PR	
	UID's	PR	

	Modelo Conceitual	PR	
	Modelo Navegacional	PR	
	Modelo Entidade Relacionamento	PR	
	Análise final	PR	
	Revisão da Documentação gerada	PR	
Desenvolvimento WebEngine			
	Planejamento	PR	
	Core::BaseApp	PR	
	Core::ExecCmd	PR	
	Core::Environment	PR	
	Core::PSession	PR	
	Core::PDebug	PR	
	Core::GateInterface	PR	
	Core::Gates	AR	Alguns gates opcionais foram criados, como para Java e para XML
	Core::Skin	PR	
	Core::DBLink	PR	
	Core::MySQLDB	PR	
	Standard::XDOMP	PR	
	Standard::PMail	PR	
	Standard::PDF	PF	Foi planejado e iniciado. Mas as pesquisas posteriores ao seu desenvolvimento e as modificações feitas na engine o tornaram obsoleto.

	Standard::FTP	PI	Em fase de testes
	Plus::PUnit	AR	Foi adicionada essa classe de testes em função dos estudos realizados e por decisão de projeto a respeito de testes.
	Plus::Resource	AR	Foi adicionado esse recurso na engine por decisão de projeto.
	Plus::SSS	AR	Foi adicionado esse recurso na engine por decisão de projeto.
	Plus::User	PR	
	Plus::SimpleDoc	AI	Foi idealizado após estudos, mas a falta de tempo não permitiu sua conclusão.
	Plus::Hypermidia	AF	Foi adicionado ao projeto para agregar valor a engine. Foi estudado e projetado em papel, mas a falta de tempo não permitiu sua conclusão.
	Plus::Image	AN	Foi adicionado ao projeto para agregar valor a engine. Mas a falta de tempo não permitiu sua desenvolvimento.
	Plus::CrossRef	AF	Foi idealizado para agregar valor a engine e suprir necessidade do C&L, foi estudado e planejado, mas por falta de tempo não foi concluído.
Desenvolvimento C&L Evolution			
	Project Class	PR	
	Scenario Class	PR	
	Lexicon Class	PR	
	C&L Filter	PR	
	Callbacks	PR	
	Skins	PR	

	Testes	PR	

❒ Tabela 22.2 – Cronograma simplificado

CONSIDERAÇÕES FINAIS

Para este trabalho final de graduação, me dediquei a fazer estudos relacionados a web softwares e arquitetura de sofwares. Procurei, durante o processo de desenvolvimento do mesmo, pesquisar as mais avançadas tecnologias e metodologias relacionadas ao projeto. Procurei ser detalhista quanto à documentação, documentando além dos resultados também as pesquisas realizadas.

Apesar de ser um tema já explorado por outros trabalhos, e bem extenso no que diz respeito a fontes de pesquisa, tentei manter o projeto com um único foco: descrever e implementar uma arquitetura base para softwares web que possibilite que desenvolvedores construam aplicações web com diferentes fron-ends. Procurei também deixar pontes para o uso de outros avanços tecnológicos ou futuras possibilidades, como por exemplo, o uso da engine com metodologias AJAX. Procurei ser prático e apresentei possibilidades futuras de utilização da engine, como por exemplo, apresentando as idéias de add-ons para a engine.

Para a descrição deste relatório e para a apresentação à banca examinadora tentei também ser didático e simplista, sem diminuir o grau de profundidade nos detalhes e conceitos aos quais me referi. Procurei utilizar exemplos que trazem os conceitos para perto da realidade, para assim facilitar o entendimento e prender a atenção do leitor.

Após idealizar este relatório, o produzi de maneira que fosse possível ser lido de diferentes formas. Podendo ser lido como uma seqüência de tópicos que ao final levariam à uma solução de problemas expostos, e como fonte de consulta de assuntos relacionados a softwares e tecnologias web.

É importante ser considerado também que o desenvolvimento de softwares web é uma área relativamente nova. Os estudos relacionados a essa área também são novos e ainda não consolidados completamente. Em conseqüência disso, este projeto teve algumas fases de retrabalho. Esse fato foi importante para que eu ganhasse mais experiência e ao final propusesse uma solução viável, mutável e prática.

Chegando ao final de minha graduação, ao final de mais uma etapa de minha vida, olhando para o que foi proposto e o que foi realizado, faço uma análise geral e considero que depois de muito esforço fiz um trabalho ao qual me orgulho. Considero também que este projeto pode ir

muito mais além do que seu estado atual, mas considerando o tempo de realização do mesmo, creio que os resultados alcançados são satisfatórios.

Ainda há muito a ser feito, mas creio que este projeto renderá frutos, adeptos e projetos subseqüentes que contribuirão para sua evolução e o engrandecerão. Considero este projeto como um ponta-pé inicial.

REFERÊNCIAS

[1] Sommerville, I.- Engenharia de Software – Editora Pearson

[2] Silva M.C. – Identificação de Estilos de Arquitetura: um processo dirigido por conhecimento.

[3] J.C.S.P.Leite, G.D.S.Hadad, J.H. Doorn, G.N.Kaplan – A Scenario Construction Process

[4] E.Gamma , R.Helm, R. Johnson, J. Vlissides – Design Patterns

[5] J.Zeldman – Projetando Web Sites Compatíveis – Editora Elsevier

[6] J.G.R. Araújo "NewsGeneration – o Desenvolvimento de Aplicações Web" in RNP – Rede Nacional de Ensino e Pesquisa, October 1997.

[7] W3C - W3Schools - http://www.w3schools.com/

[8] E.J. Chikofsky and J.H., Cross II - "Reverse Engineering and Design Recovery: A Taxonomy" in IEEE Software, pp 13-17, IEEE Computer Society, January 1990.

[9] J.C.S.P. Leite - "Working Results on Software Re-Engineering" in Software Engineering Notes pp 39-44, ACM SIGSOFT, March 1996.

[10] J.C.S.P. Leite – Understanding the Word "Analysis" in the Context of Requeriments Enginnering.

[11] M.Sayão , J.C.S.P.Leite – Rastreabilidade de Requisitos – Monografia em ciência da computação no 20/05 PUC-Rio

[12] F.A.C.Pinheiro – Chapter 5 of Perspective on Software Requerements.

[13] L.F.G.Soares, G.Lemos, S.Colcher – Redes de Computadores – segunda edição – editora campus

[14] Tanembaum – Redes

[15] R.Q.Almeida , [UNICAMP] - XML

[16] J.Knopman , [NCE/UFRJ] – Linguagem XML

[17] J.F.Hübner , [FURB / DSC] – XML

[18] M.A.C.Nova , [PUC-RIO] - Módulo1a-XML Básico

[19] A.S.S.Dangui , Perspectivas sobre o Desenvolvimento de Aplicações Baseadas em Java e XML

[20] XML.ORG - http://xml.org

[21] XML.COM - http://www.xml.com

[22] Reengenharia e Engenharia Reversa - http://www.dei.unicap.br/~almir/seminarios/2004.2/ts04/reengenharia/

[23] J.J.Garrett - Ajax – A New Approach to Web Aplications – http://www.adaptativepath.com

[24] Rocha, Maldonado, Weber Prentice-Hall – Qualidade de Software: teoria e prática

[25] J.C.S.P.Leite, L.F.Silva – Desenvolvimento de Software Livre e o Software C&L

[26] Universidade Nacional del Centro de la Província de Buenos Aires – LEL de LEL y Escenarios

[27] Wikipedia – www.wikipedia.com