Rio de Janeiro, quinta-feira, 27 de outubro de 2005

PUC
RIO

C&L EVOLUTION
DOCUMENTAÇÃO

RODRIGO BUÁS LOPES – rodrigobuas@gmail.com

THIAGO CRISPINO SANTOS – thiagocs_@terra.com

ORIENTADOR: JULIO CESAR SAMPAIO DO PRADO LEITE – julio@inf.puc-rio.br

INTRODUÇÃO

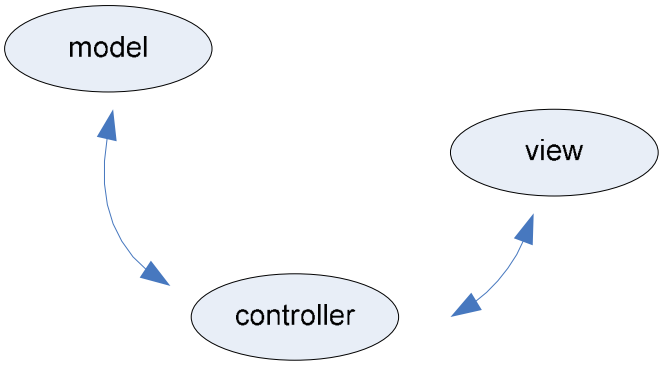
Esse documento explica de forma simplificada o uso da engine, que chamamos de WebEngine©, usada na base do modelo da plicação C&L.

Ilustraremos neste documento a micro-arquitetura e a macro-arquitetura da engine. Também será explicitado o funcionamento e como podemos utilizar da melhor forma a engine.

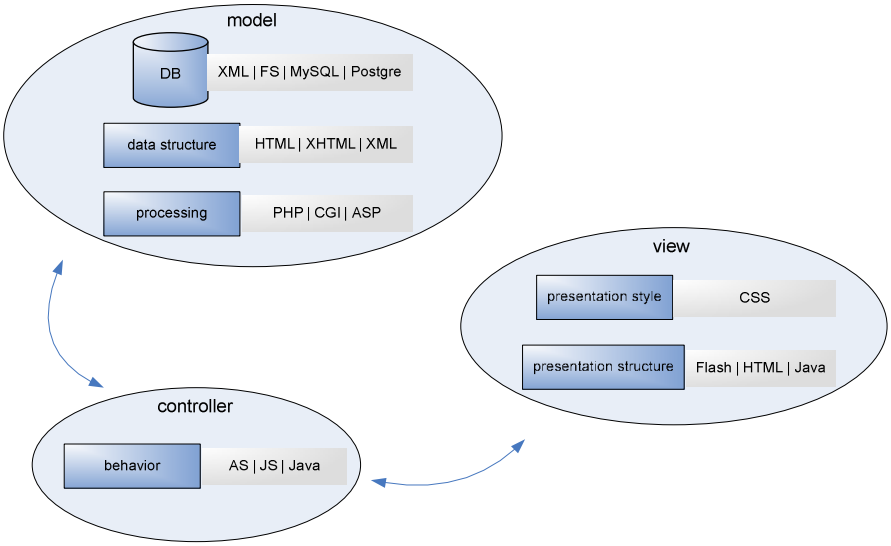
Em seguida, mostraremos a estrutura e disposição das pastas na rede. E por fim faremos alguns comentários sobre o bom uso desse motor.

ARQUITETURA

A arquitetura da engine é baseada na visão de MVC ilustrada abaixo.



Adicionado à visão deste framework e adaptada aos requisitos de funcionamento do motor evoluímos para a seguinte arquitetura.

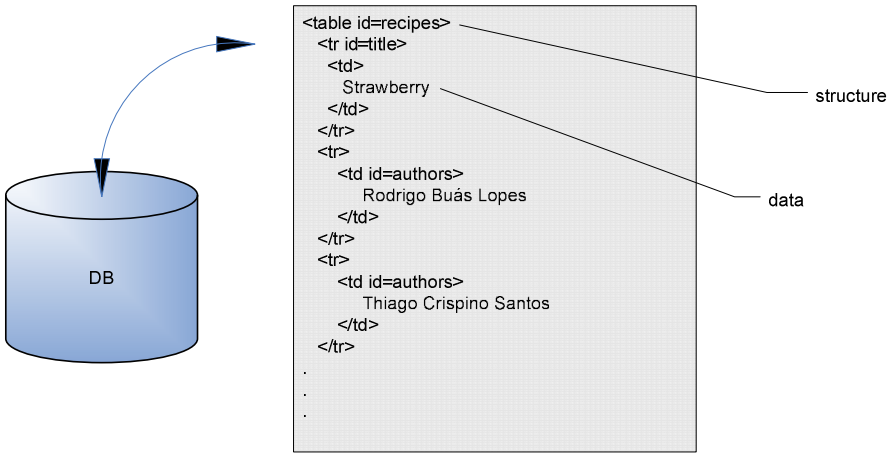


Nesta figura está indicado também para cada parte algumas da possibilidades de tecnologias que poderiam ser adotadas.

Como indicado na figura acima, o modelo foi dividido em base de dados, estrutura dos dados e processamento. O processamento é o centro do software, e é executado sempre no lado do servidor da aplicação.

Adotamos para a parte de processamento do motor o PHP – Hipertext Preprocessor. Para o banco de dados usamos MySQL. E para a estrutura de dos dados inseridos na base de dados escolhemos XML.

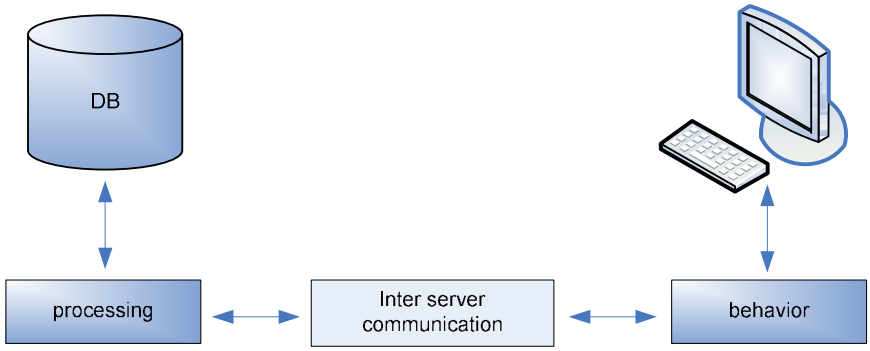
A base de dados é uma coleção de dados, enquanto que a estrutura dos dados é a forma como estão armazenados. A figura abaixo ilustra a diferença entre o banco de dados e a estrutura armazenada dentro dele.



O "view" foi dividido em duas partes, "presentation style", que normalmente é usado CSS (Cascade Style Sheet) para esta parte; e "presentation structure", que pode ser implementado em diversas tecnologias.

O controlador, que aqui chamaremos de "behavior", contém a parte de processamento que é unicamente dedicada ao fort-end da aplicação. Implicitamente, esta parte terá que ser desenvolvida para cada frot-end de tecnologia diferente, enquanto que o modelo é único.

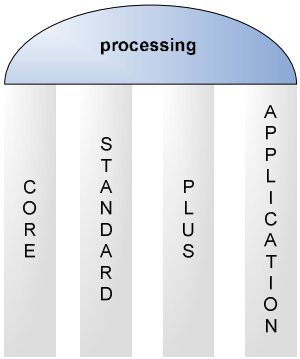
DISTRIBUIÇÃO DA ARQUITETURA

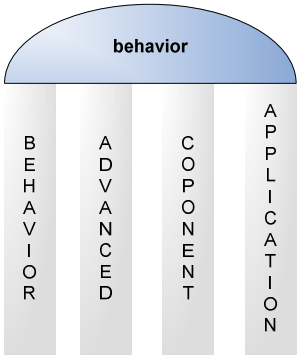


O diagrama acima mostra as entidades estruturais similares as existentes no ambiente web.

A entidade *InterServer* é uma abstração dos componentes de processamento e de comportamento. Essencialmente, é a parte da biblioteca que contém as interfaces a serem implementadas para cada *frontend* diferente.

Como resultado da arquitetura adotada, construímos a biblioteca padrão utilizada pelo *engine*, que é dividida em duas partes, processamento (processing) e comportamento (*behaviour*), inteiramente orientada a objetos.





A biblioteca *behavior* deve ser reimplementada para cada *frontend* construído em diferentes linguagens. A biblioteca *processing*, por sua vez, é totalmente reutilizável.

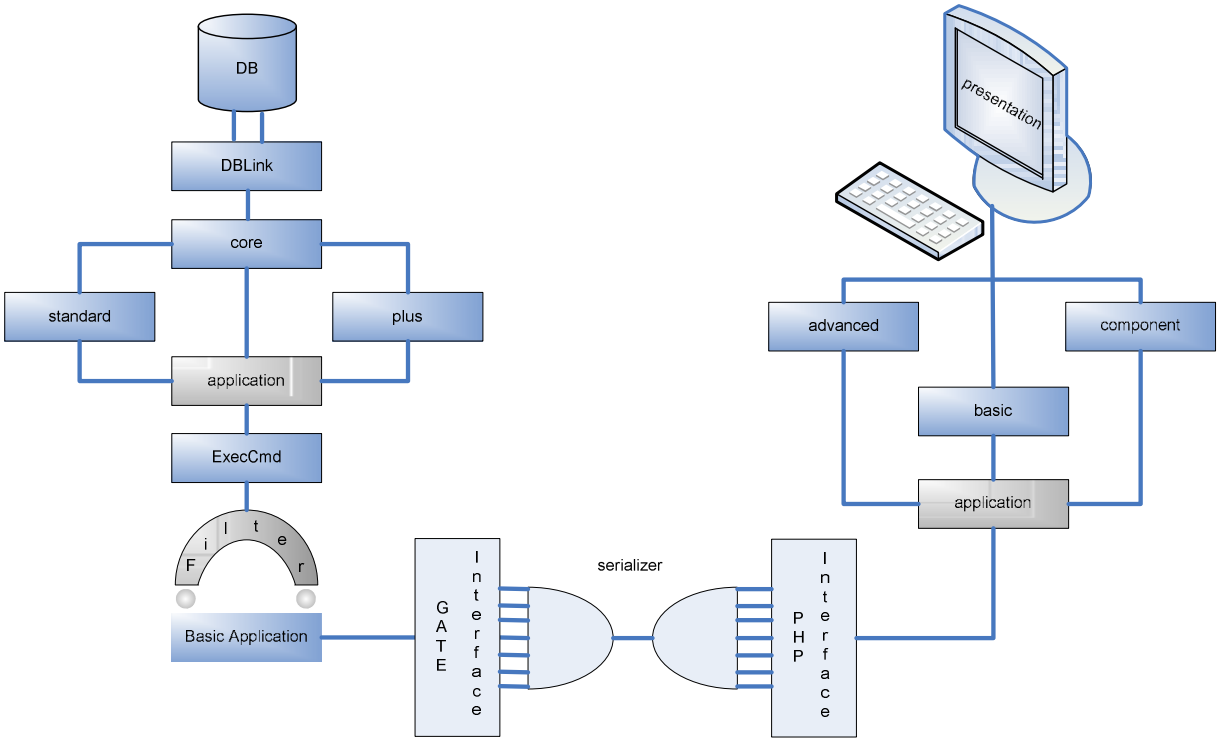
Existe uma subdivisão interna de ambas partes, mostrada no diagrama a seguir. A parte de *processing* inclui o *core*, que contém as classes base, fundamentais para o funcionamento do *engine*, tal como serviços básicos (*i.e.* Sistema de arquivos); o *standard*, que contém as classes que suportam os serviços complementares (*i.e.* XML, PDF, email); o *plus*, onde se encontram os serviços comumente utilizados por aplicativos *web* (*i.e.* Classes de usuário). A ultima subparte da

biblioteca é a *application*, que contém as *callbacks* de usuário. Essa subparte é implementada para cada aplicação e será detalhada adiante.

O *behavior* é dividido em quatro partes: *basic, advanced, components* e *application*. Tal divisão é similar à do *processing*.

FUNCIONALIDADE

O diagrama abaixo ilustra o funcionalidade do *engine*. Nota-se algumas das entidades discutidas, além de adicionais que estão, na verdade, incluídas nas anteriores. A separação é meramente para facilitar o entendimento da funcionalidade.



O modelo é caracterizado por uma execução em *pipeline* em ambas direções, vindo da interface com usuário até o sistema e seu processamento, chamada *requisition*, e o caminho reverso, vindo do sistema até o usuário, chamada *response*.

A *requisition* é iniciada quando uma tarefa é requisitada pelo usuário. O sistema, então, agrega a informação necessária (*input*) e a ação a ser executada (*Action*). Essa informação é direcionada para o mecanismo de comunicação de PHP, chamado *PHPGate*, que irá serializá-la e envia-la por HTTP.

No lado do servidor, o *gate* receberá a informação, enviando-a para executar o procedimento padrão, até que o filtro especifico da aplicação seja chamado.

O filtro irá executar o procedimento, singularmente descrito para cada aplicação, e escutar qual ação foi requisitada pelo usuário, enviando-a para o *ExecCmd* (uma entidade em *core* que relaciona a ação com a *callback* correspondente).

A biblioteca de aplicação, como o nome sugere, está relacionada com a aplicação propriamente e é implementada para cada uma. Pode fazer uso das classes *core*, *standard* e *plus* e é, essencialmente, uma série de *callbacks* relacionada a ações.

Essas *callbacks* possuem um padrão a ser seguido: cada *callback* deve retornar um *array* contendo três valores: *ResponseType*, *ResponseContent* and *ResponseSkin*.

A classe *BDLink* encontrada em *core*, é descrita aqui devido sua importância e versatilidade. A interface da classe deve ser implementada, de forma que seja possível a troca o banco de dados sem alterar o resto do software.

Temos duas implementações, uma para *MySQL*, outra fazendo uso do *SQLite* do PHP, que suporta uma versão do banco de dados em arquivos.

Continuando com a execução, a *callback* executa o procedimento correspondente, utilizando as bibliotecas, consulta a base de dados e começa a *response*.

Então, o pacote de responsta é criado contendo o *response*-> *ResponseType*, que indica a resposta de procedimento.; response->*ResponseContent*, que são os dados relevantes da resposta; e o *response*->*ResponseSkin*, que define qual skin deve ser mostrada para o usuário.

Quando o procedimento acaba, a aplicação entrega os dados pelo Gate aberto, que por sua vez serializa toda a informação e envia para o front-end.

Assim que o pacote é recebido pelo front-end, o processo inverso começa e os dados são unserializados, e a skin apropriada é requerida. Neste ponto o behavior assume o controle.

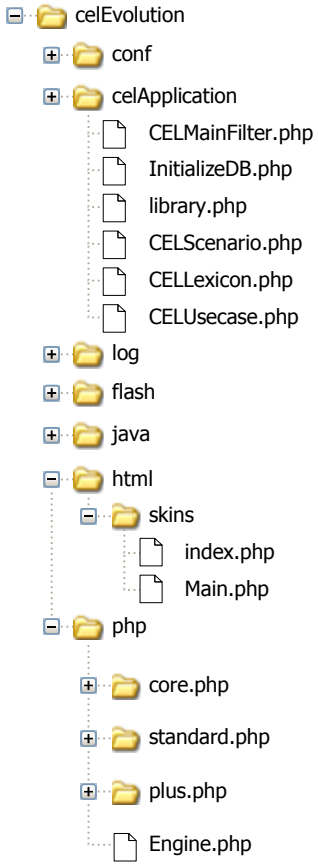
PALAVRAS FINAIS

Essa engine esta em desenvolvimento, em caráter beta, o que indica que ainda não esta inteiramente estável e completa. E esta sujeita a modificações.

Qualquer pessoa que esteja usando esse motor, seja ela usuário ou desenvolvedor, e detectar alguma falha ou erro, entre em contato pelo email rodrigobuas@grad.inf.puc-rio.br ou thiagocs@grad.inf.puc-rio.br .

Diagrama de Classes

Organização online das pastas



Observe a distribuição das pastas na figura acima e repare que toda callback deverá ser armazenada na pasta celApplication. Repare também que cada front-end de tecnologia diferente terá sua própria pasta, assim sendo, para o usuário, entraríamos na aplicação da seguinte forma: http://host/celEvolution/html/ para utilizar a versão em html, : http://host/celEvolution/flash/ para a versão em flash, e assim por diante.

Dicas de uso

- Primeiramente, prefira sempre os serviços fornecidos pela engine, caso este exista, evitando usar funções de bibliotecas fornecidas pelo PHP.
- Para resgatar e salvar variáveis que desejamos que sejam vistas pelo front-end aconselhamos que use a classe Environment, usando os methodos GetVar e SetVar. Toda variável que for salva globalmente de qualquer outra forma, e.g. usando GLOBALS do php, não serão visíveis no front-end.
- São indicadas duas formas de registrar cada callback dentro da pasta celApplication. Modularizado, sendo assim, só precisamos do nome da callback. Ou OO (em uma classe que contenha somente métodos estáticos). Desta ultima forma, teremos que passar para a engine a classe e o nome.

CALLBACK_NAME;

CLASS_NAME::CALLBACK_NAME;

- Registre toda ação no arquivo de configuração, referenciando a assinatura de cada callback previamente definida da seguinte forma:

$_CONF['EXEC_CMDS']['NOME_DA_AÇÃO'] = 'CALLBACK_NAME';

- Toda callback deve receber um único parâmetro, que podemos adotar sempre como um pacote de variáveis, ou pode também não receber parâmetro nenhum se não é necessário nenhum parâmetro para a callback.
- Toda callback deve retornar uma array que contenha, nesta ordem, ResponseType , ResponseContent , ResponseSkin.
- Toda skin deve ser registrada no arquivo de configuração da seguinte forma:

$_CONF['SKINS']['NOME_DA_SKIN'] = 'SKIN_NAME';

- Deve ser registrado também o path da pasta onde estão as skins.
- Para acompanhar o processamento da callback, aconselhamos o uso da classe PDebug,
- Utilize os níveis de debug de 0 a 5 (para callbacks), sendo o nével mais detalhado o nível 5.
- Ative ou desative o debug passando como uma das variáves no acesso debugison=1 para ativar e debugison=0 para desativar.
- Mude o nível do debug passando como variável o debuglevel=LEVEL