UNIVERSIDADE SALVADOR – UNIFACS
MESTRADO EM SISTEMAS E COMPUTAÇÃO

DANIEL DE OLIVEIRA SANTOS NETO

MODELO DE COMPONENTE DE CONTEÚDO DIGITAL PARA PHP APLICADO AO EVACMS

Salvador
2009

DANIEL DE OLIVEIRA SANTOS NETO

MODELO DE COMPONENTE DE CONTEÚDO DIGITAL PARA PHP APLICADO AO EVACMS

Dissertação apresentada ao Mestrado em Sistemas e Computação da Universidade Salvador – UNIFACS, como requisito parcial para obtenção do título de Mestre.

Orientador: Prof. Dr. André Santanchè

Salvador
2009

Ficha Catalográfica
(Elaborada pelo Sistema de Bibliotecas da Universidade Salvador - UNIFACS)

Santos Neto, Daniel de Oliveira

Modelo de componente de conteúdo digital para PHP aplicado ao EVACMS. / Daniel de Oliveira Santos Neto. – Salvador, 2009.
96 p. : il.

Dissertação apresentada ao Curso de Mestrado em Sistemas e Computação da Universidade Salvador – UNIFACS, como requisito parcial para a obtenção do grau de Mestre.
Orientador a: Prof. Dr. André Santanchè.

1. PHP (Linguagem de programação de computador). I. Santanchè, André, orient. II. Universidade Salvador – Unifacs. III. Título.

CDD: 005.133

DANIEL DE OLIVEIRA SANTOS NETO

MODELO DE COMPONENTE DE CONTEÚDO DIGITAL PARA PHP APLICADO AO EVACMS

Dissertação apresentada ao Mestrado em Sistemas e Computação da Universidade Salvador - UNIFACS, como requisito parcial para obtenção do título de Mestre.

André Santanchè - Orientador
Doutor em Ciência da Computação pela Universidade Estadual de Campinas, Brasil
Universidade Salvador - Unifacs

Jose Maria Nazar David - Avaliador
Doutor em Engenharia de Sistemas e Computação pela Universidade Federal do Rio de Janeiro - UFRJ, Brasil
Universidade Salvador - Unifacs

Ricardo da Silva Torres - Avaliador
Doutor em Doutorado Em Ciência de Computação pela Universidade Estadual de Campinas - UNICAMP, Brasil
Universidade Estadual de Campinas - UNICAMP

Salvador, 22 de Abril de 2009.

À toda minha família e amigos, dedico este trabalho, parte tão significativa de minha vida, pelo apoio, compreensão e estímulo que sempre me dedicaram, sem os quais nada seria possível.

AGRADECIMENTOS

Ao meu professor e orientador, Doutor André Santanchè, pela confiança, atenção e cuidado dedicados na leitura e orientação, tão importantes para encontrar o rumo desse trabalho. A Deus. À minha mãe, minhas irmãs, minha noiva Leila, toda minha família e amigos pelo apoio que tenho recebido durante toda a minha vida. Ao meu coordenador e amigo Luciano Pena e professores do curso de Redes de Computadores. Aos colegas de pesquisa, também orientandos de André Santanchè, pela grande troca de conhecimento compartilhada. Aos colegas de turma de mestrado 2007 e 2008 que me apoiaram estes anos no caminho rumo ao Mestrado.

RESUMO

O desenvolvimento de aplicações Web tem se tornado uma tarefa cada vez mais complexa, exigindo estratégias eficientes para reuso sistematizado de código. Neste contexto, linguagens para a construção de páginas dinâmicas (e.g., PHP, ASP, JSP etc.) são amplamente utilizadas. Entretanto, apesar do uso de componentes de software ser uma importante estratégia de reuso, não existem modelos de componentes amplamente aceitos especializados em linguagens para a construção de páginas dinâmicas. Este cenário motivou o tema principal desta pesquisa, que consiste na definição de um modelo de componentes apto às linguagens para a construção de páginas dinâmicas e mais especificamente a linguagem PHP. O modelo proposto é uma derivação de outro já existente chamado DCC – *Digital Content Component*. Tal adaptação foi validada em um cenário prático: o *framework* de gerenciamento de conteúdo EVAcms. A partir do uso dos DCCs PHP, o EVAcms adquiriu dois diferenciais em relação a outros *frameworks*: um sistema robusto de extensão baseado em componentes e a capacidade de inserir componentes do EVA como subsistemas dentro de outras aplicações na Web.

Palavras Chaves: Componente de Software. DCC. PHP. EVAcms. CMS. Web.

ABSTRACT

The development of Web applications is becoming an increasingly complex task, requiring efficient strategies for systematic reuse of code. In this context, languages for building dynamic pages (eg, PHP, ASP, JSP etc..) are widely used. However, even though the use of software components is an important strategy for reuse, there are no models of components widely accepted in specialized languages for building dynamic pages. This scenario motivated the main theme of this research, which is the definition of a component type apt to languages for building dynamic pages and more specifically the PHP language. The proposed model is a derivation of an existing model named DCC – Digital Content Component. This adaptation was validated in a practical scenario: the content management framework EVAcms. Through PHP DCCs, EVAcms acquired two differentials compared to other frameworks: a robust extension system based on components and the ability to insert EVA components inside other applications as subsystems in composite Web applications.

Keywords: Software Component. DCC. PHP. EVAcms. CMS. Web.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Dentre os principais objetivos do reuso de software, estão a produtividade, a redução de custos e a melhoria a qualidade do produto final. Atualmente o reuso sistematizado tornou-se uma necessidade para atender às demandas do mercado que cresce aceleradamente, aumentando a competição e necessidades de qualidade.

A crescente popularização da Internet, bem como o crescimento do número de serviços e informações disponibilizadas por esse meio, nos leva a repensar a forma como nossos sistemas de informação são desenvolvidos no sentido de reaproveitamento e compartilhamento de serviços. Linguagens de programação estabelecidas na Internet muitas vezes não dispõem de modelos eficientes para reuso de software. Este é o caso da linguagem PHP que, dentre as estratégias de reuso disponíveis, demonstra uma carência de um modelo de componentes de *software* robusto e flexível.

A necessidade do mercado de redução de custos e aumento da agilidade no desenvolvimento de soluções de *software* para Web estimulou Alessandro do Valle Nunes e Daniel de Oliveira Santos Neto ao desenvolvimento, no início de 2004, de um *framework* personalizado, para suprir a carência da época de suporte ao desenvolvimento de soluções para o gerenciamento de conteúdo na *Web*. O projeto foi concretizado, chamado de EVA *framework e* publicado em agosto de 2004 e vem sendo amplamente utilizado até os dias de hoje para as mais diversas aplicações, os componentes padrões deste projeto davam ao *framework* características de um CMS, nascendo assim o EVAcms.

A palavra *framework* citada no nome EVA remete ao desenvolvimento aplicações Web. Ao longo do tempo as indústrias começaram a isolar os recursos utilizados na maioria dos aplicativos em *frameworks* de desenvolvimento. A ideia principal é permitir que os recursos comuns a quase todos os sistemas sejam reutilizados, desde pequenos sites até grandes portais. Neste caso, para cada novo aplicativo desenvolvido, é necessária apenas a implementação dos seus requisitos particulares.

Durante todos estes anos, o EVA vem passado por diversos *upgrades* e vem sendo aplicado nas mais diversas aplicações. No entanto, tal como acontece com vários outros *frameworks* PHP, seu modelo de componentes até então era limitado às extensões do sistema, além de não implementar o conjunto completo características de um componente de *software*, que propiciariam maior reusabilidade do código. Daí surgiu a necessidade de a ampliar as possibilidades do EVA e

principalmente alcançar um modelo de componentes mais completo do que o desenvolvido na época de sua concepção.

No contexto desta pesquisa o sistema EVA representa na verdade o protótipo de um típico sistema de gerenciamento de conteúdo para Web. Desse modo, apesar da implementação e validação do modelo ter acontecido dentro do EVA, foi desenvolvido um modelo genérico o suficiente para ser estendido para outros sistemas PHP, em especial de gerenciamento de conteúdo.

Para tal ampliação o EVA necessita de um modelo de componentes apto à linguagem PHP. Após pesquisas em modelos já existentes, verificou-se que a linguagem PHP não dispõe de um modelo amplamente adotado de componentes, que siga os requisitos mínimos necessários, conforme será detalhado no Capítulo 2 Por esse motivo, o modelo de componentes denominado *Digital Content Component* (DCC) foi adotado e adaptado para o contexto PHP. O modelo DCC é capaz de encapsular tanto conteúdo em geral quanto código executável (SANTANCHÈ, 2006), sendo flexível o suficiente para atender diferentes contextos, como por exemplo o contexto das aplicações Web, mais especificamente o PHP e CMS.

A evolução e disseminação da Educação à Distância (EAD) para meios contemporâneos, como por exemplo a Web, favoreceu o aparecimento de ferramentas hoje conhecidas como Learning Management Systems (LMS), ou seja, ferramentas especializadas em EAD, que por outro lado não são tão especializadas em conteúdo, como acontece com o CMS. Porém a união do potencial destas duas especialidades, LMS e CMS, desperta a necessidade de construção de sistemas Web de lógica híbrida. Componentes de software se apresentam como uma estratégia interessante nesta direção. Sistemas construídos a partir da composição de componentes tornam possível a integração de funcionalidades em sistemas híbridos.

Esta dissertação se estrutura em capítulos que fazem análises e comparações entre soluções de desenvolvimento de CMSs e modelos de componentes de software. Em cada contexto apresenta vantagens e desvantagens advindos de cada solução, com a finalidade de fundamentar a solução proposta neste trabalho, que combina um modelo de componentes – os DCCs – com um ambiente CMS – o EVA. As duas principais contribuições desta pesquisa são: um modelo de componentes apto a linguagens de produção dinâmicas de páginas, em especial o PHP e a adaptação deste modelo de componentes para o contexto de gerenciamento de conteúdo.

Esta dissertação está organizada como segue: nos Capítulos 2 a 5 apresentamos a fundamentação teórica e nos Capítulos 6 a 8 é detalhada a nossa contribuição. No Capítulo 2 apresentamos o conceito de componente de *software* e suas características. No Capítulo 3 é apresentado o nosso modelo base de componentes, o *Digital Content Component*. No Capítulo 4 são analisados e comparados modelos de componentes em PHP. No Capítulo 5 são resumidas as

principais características de um Sistema de Gerenciamento de Conteúdo (CMS – *Content Management System*). No Capítulo 6 apresentamos a primeira parte da nossa pesquisa, na qual os DCCs são adaptados para o contexto PHP. No Capítulo 7 é apresentado o sistema de gerenciamento de conteúdo no qual o modelo DCC/PHP foi implantado – o EVAcms. No Capítulo 8 apresentamos detalhes da implantação dos DCCs em PHP no EVAcms e seus diferenciais. No Capítulo 9 são apresentadas as considerações finais e trabalhos futuros.

2 COMPONENTES

Pesquisas apontam que de um modo geral 40% de todo o código pode ser reutilizado de um sistema para outro. No caso específico de aplicações comerciais, 60% do *design* e do código podem ser reutilizados em aplicações comerciais (EZRAN; MORISIO; TULLY, 2002).

Uma empresa nórdica chamada CelsiusTech relata que ao focar o desenvolvimento baseado em reuso (mais especificamente na abordagem de famílias de sistemas) obteve as seguintes melhorias (BROWNSWORD; CLEMENTS, 1996):

a) Inversão da participação do *software* no custo total do produto final: de 65% para 20%;
b) Redução da mão de obra de desenvolvimento de *software* (dos projetos de uma família de produtos) de 200 pessoas para menos de 50;
c) O tempo de entrega (*time-to-market*) passou de alguns anos para meses;
d) Constatou-se que 70% a 80% dos sistemas de *software* resultantes eram compostos de componentes reutilizados;
e) Aumento da qualidade dos sistemas desenvolvidos e da satisfação dos clientes;
f) Mais de 50 aplicações já foram derivadas da linha de produto, cujo sistema contém quase 1,5 milhões de linhas de código em linguagem Ada.

Tais números demonstram que é notória a vantagem de se adotar o reuso com eficácia e como estratégia principal para o desenvolvimento de *software.* Nesse contexto, surge a noção de componente de *software*. Mesmo sendo um termo antigo, cuja primeira definição popular data de 1968 (MCILROY, 1998), até hoje ainda não há consenso sobre a definição mais adequada para este termo (HOPKINS, 2000). Embora seja objeto de estudo constante de muitas publicações na área de *software*, há diferentes perspectivas sobre o que é um componente, ou sobre o que é o desenvolvimento baseado em componentes ou, ainda, como reunir estes componentes para formar um sistema (BASS e outros, 2001).

Pode-se dizer, usando uma comparação popular, que componentes são os “blocos de lego” da Engenharia de *Software* (CAPRETZ; CAPRETZ; LI, 2001) ou, de maneira mais formal, componente de *software* define-se como sendo uma implementação em *software* que pode ser reutilizada em aplicações diferentes sem modificação e que é acessada via API (BASS e outros, 2001).

Em artigo que sintetiza um debate de especialistas sobre o tema (BROY e outros, 1998), é possível se identificar diferentes definições sobre componente. Usualmente elas não são conflitantes, mas cada uma delas insere especificidades que não aparecem nas demais. Escolhemos, neste estudo, definir componente de *software* a partir da compilação de um conjunto de suas características que consideramos fundamentais. Tal compilação deriva de outro trabalho anterior produzido por Santanchè (SANTANCHÈ; MEDEIROS; PASTORELLO, 2007), o qual refinamos a partir de definições de outros trabalhos (BASS e outros, 2001)(BROY e outros, 1998) (SZYPERSKI, 2002). Um componente possui as seguintes características:

Trata-se de uma **unidade projetada para composição** – do latim *componens*, derivado de *componere*, que quer dizer "colocar junto".

Utiliza **interfaces** para publicar sua funcionalidade e como meio de comunicação com outros componentes. A interface é um aspecto fundamental dos componentes que será aprofundada neste texto. Elas são por um lado o meio privilegiado por onde os componentes se comunicam com o meio externo. Por outro lado, representam uma síntese da funcionalidade dos componentes, funcionando como um contrato de como o componente se comportará em cada contexto.

Componentes são passíveis de **composição hierárquica**, ou seja, subcomponentes podem ser compostos para formar componentes em um nível mais alto.

Contém **código executável** que implementa funcionalidades declaradas na interface. Alguns autores como Szyperski (SZYPERSKI, 2002) originalmente utilizavam o termo "código binário" para se referir à implementação de um componente. Na versão atualizada de seu livro, ele comenta que substitui o termo binário por executável, em face às múltiplas possibilidades de se representar um executável, tal como as modernas linguagens de *script*, que são justamente o nosso foco neste documento, dado que o PHP é uma linguagem de *script*.

Tem **acesso privilegiado** à sua funcionalidade **a partir da interface**. Esta característica é usualmente denominada *black-box,* na qual entidades externas ao componente desconhecem detalhes da sua implementação e têm acesso exclusivo aos seus serviços a partir de suas interfaces públicas.

Definem um **formato** (e.g., pacote) para **compartilhamento de distribuição**.

Este conjunto de caraterísticas delineia o que aqui passaremos a chamar de componente de *software*, entretanto, tal como acontece nas demais definições de componentes, neste conjunto de características ainda está incluso um vasto conjunto de modelos de componentes, que abrange desde aqueles usados por linguagens de programação (e.g., JavaBeans e EJB (SUN, 2009), COM (MICROSOFT, 2009a), XPCOM (PARRISH, 2001) etc.), até aqueles usados por *plugins* para a extensão de sistemas para Web. Este último caso é de especial interesse nesse nosso estudo, dado

que trataremos exatamente da aplicação de um modelo de componentes em um *framework* para a construção de CMSs na Web. Dentre os aspectos apontados, o que mais se diferencia entre modelos de componentes é a interface.

2.1 TECNOLOGIAS DE COMPONENTES

As tecnologias de componentes de software modernas podem ser divididas em dois grandes grupos: aquelas que atuam localmente e aquelas voltadas a ambientes distribuídos. Dentre as tecnologias para componentes locais destacam-se os *Javabeans* da Sun (SUN, 2009), o COM da Microsoft (MICROSOFT, 2009a) e o XPCOM (MOZILLA, 2009), que deriva do COM e foi desenvolvida pela Mozilla, e o OSGi (OSGI, 2009), que apesar de oferecer suporte à execução local de componentes, também possibilita a integração destes componentes em ambientes distribuídos. Dentre as tecnologias de componentes distribuídos se destacam o Enterprise Java Beans (EJB) (SUN, 2009) da Sun, o COM+ da Microsoft (MICROSOFT, 2009a) e o CORBA Component Model (CCM) (IBM, 2009).

JavaBeans: Representam a tecnologia de componentes do Java. A tecnologia *JavaBeans* faz uso de um padrão de programação em Java a partir da qual é possível se construir componentes reusáveis, com as seguintes características: (i) podem ser customizados externamente a partir de propriedades; (ii) utilizam a estratégia de eventos para possibilitar a conexão dinâmica de componentes; (iii) permitem a introspecção para a detecção automática de características dos componentes; (iv) dispõem de recursos de persistência.

***Enterprise JavaBeans*:** A especificação *Enterprise JavaBeans* é um modelo que define uma arquitetura para o desenvolvimento e implantação de objetos distribuídos e transacionais. Estes objetos são componentes *server-side* chamados de *Enterprise Beans*, que por sua vez são hospedado em um *container* para *Enterprise JavaBeans*, provendo serviços remotos para clientes em toda rede. O modelo de programação *Enterprise JavaBeans* fornece aos desenvolvedores de *Beans* e fornecedores de servidores *Enterprise JavaBeans* um conjunto de contratos e especificações que definem uma plataforma comum para o desenvolvimento. O objetivo desses contratos é garantir portabilidade entre fornecedores, criando um amplo conjunto de funcionalidades (SUN, 2009).

COM: *Component Object Model* é um modelo de componentes desenvolvido pela Microsoft com enfoque na comunicação de componentes que estejam na mesma máquina, mesmo que estejam implementados em linguagens diferentes. Para isso o COM estabelece um padrão de compatibilidade binária de interface, independente da linguagem de programação. COM é usado por desenvolvedores para criar componentes de *software* reutilizáveis aproveitando os serviços do Microsoft Windows. Esta arquitetura de componente faz parte da família de tecnologias COM que inclui o COM +, DCOM (*Distributed* COM) e controles ActiveX (MICROSOFT, 2009a). O modelo COM com será melhor detalhado no Capítulo 2.2 .

COM+: Representa a *COM-based services and technologies* lançado pela primeira vez no Windows 2000. COM+ integra os serviços do DCOM (*Distributed* COM), permitindo que este modelo funcione em plataformas distribuídas e une a tecnologia de componentes COM e o padrão Microsoft *Transaction Server* (MTS), permitindo que o COM+ processe automaticamente as tarefas de programação como *pooling* de recursos, aplicações desconectadas, publicação e subscrição de eventos e transações distribuídas (MICROSOFT, 2009a).

XPCOM: *Cross Platform Component Object Module* é uma tecnologia de componentes derivada do COM. O XPCOM possui um modelo e define funcionalidades equivalentes ao COM, entretanto, não está limitado a plataforma Windows. Este modelo não somente apóia o desenvolvimento de componentes de *software*, como também fornece grande parte das funcionalidades para uma plataforma de desenvolvimento, tais como, gerenciamento dos componentes, abstração de arquivos, troca de mensagens entre objetos e gerenciamento de memória (MOZILLA, 2009).
O objetivo do XPCOM é permitir que diferentes peças de software sejam desenvolvidas de forma independente uma da outra, a fim de permitir a interoperabilidade entre os componentes internos de um aplicativo. Usando XPCOM, os desenvolvedores criam componentes que podem ser reutilizados em diferentes aplicações ou que podem ser substituídos para alterar a funcionalidade das aplicações já existentes (MOZILLA, 2009). Mais detalhes deste componente será apresentado no Capítulo 2.2 .

OSGi: É uma tecnologia mantida pela OSGi Aliance (Organização formada em 1999 por um grupo de empresas como IBM, Ericsson, Siemens, Oracle, etc). Esta tecnologia define um sistema de módulos dinâmicos para linguagem Java e também especifica um *framework* para o gerenciamento destes módulos, que é capaz de gerenciar seus componentes desde sua instalação e atualização até a sua desinstalação. Este gerenciamento é feito em tempo de execução. Os módulos do OSGi são

chamados de *bundles*.

O framework OSGi é responsável por manter a consistência dos serviços ao controlar a relação de dependência entre os módulos instalados, minimizando e gerenciando o acoplamento. A tecnologia OSGi fornece uma arquitetura orientada a serviços que permite que esses componentes descubram-se dinamicamente, em tempo de execução, através das informações das interfaces publicadas em um serviço chamado *service registry local* (OSGI, 2009).

CCM: CORBA *Component Model* é parte da especificação CORBA 3.0 que em suas versões anteriores não incluíam suporte a componentes distribuídos. Sua concepção era baseada na lógica dos objetos. O CCM é um modelo *server-side* de componentes para a construção e implementação de aplicações CORBA. Este novo modelo é muito semelhante ao *Enterprise Java Beans* (EJB), porém sobre uma perspectiva independente de linguagem. O CCM estende o modelo de objetos CORBA, definindo funcionalidades e serviços em um ambiente padrão que permitem aos desenvolvedores de aplicação implementar, gerenciar, configurar e implantar os componentes que integram o CORBA *Services*. Estes serviços do lado do servidor incluem transações, segurança, persistência e eventos (IBM, 2009).

Dois aspectos fundamentais das tecnologias apresentadas acima estão relacionadas ao mecanismo de interfaces e à identificação de componentes e suas interfaces, que serão detalhados a seguir.

2.2 INTERFACES DE COMPONENTES

Podemos resumir uma interface como um grupo relacionado de métodos através do qual clientes e objetos comunicam seus componentes. A ideia é fazer com que a única maneira de comunicação entre estes seja através dos métodos declarados em sua interface (MICROSOFT, 2009b).

Nos modelos COM e XPCOM as interfaces são definidas utilizando uma linguagem IDL (*Interface Definition Language*), e seus modelos determinam certas regras que têm que ser seguidas ao se escrever uma interface. Por exemplo, todas as interfaces devem derivar de uma interface base chamada `IUnknown` (denominada `ISupports` no XPCOM) e para eliminar qualquer possibilidade de colisão que poderia ocorrer com o uso de nomes como identificadores dos objetos, cada interface deve ser associada com um GUID (*Globally Unique Identifier*) (MICROSOFT,

2009b).

A interface `IUnknown/ISupports` é composta por três importantes métodos: QueryInterface, AddRef e Release. Estes três métodos são necessários para a implementação de operações básicas do componente e todos os componentes são obrigados a implementar estes métodos (MICROSOFT, 2009b). O método QueryInterface é usado para requisitar uma interface específica a um componente através de seu identificador único. Os métodos AddRef e Release são usados para que clientes registrem que estão fazendo uso do componente (AddRef) ou que não estão mais fazendo uso (Release). A partir do controle de quantos clientes estão fazendo uso do componente é possível controlar o seu ciclo de vida.

Os modelos mais populares de interfaces estão diretamente relacionados com a forma como se criam interfaces em linguagens de programação. Como está ilustrado na Figura 1, nesse contexto uma interface é definida por um identificador e um conjunto de operações; cada operação é definida por um nome e uma lista de parâmetros; cada parâmetro é definido por seu nome, tipo e alguma indicação se consiste em um parâmetro de entrada/saída. Esta mesma estruturação básica influenciou outros contextos, tal como a definição de interfaces em WSDL para *Web services* (ALONSO e outros, 2004) e linguagens de IDL para tecnologias de componentes (MICROSOFT, 2009a) (PARRISH, 2001).

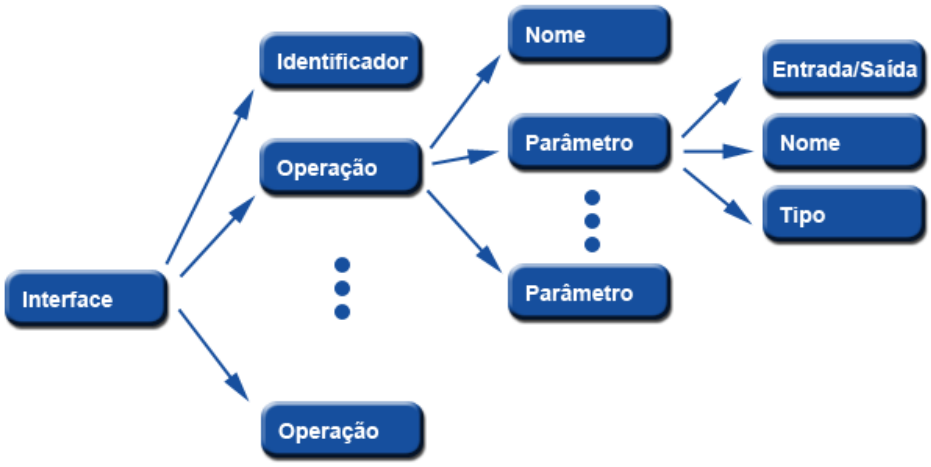

Figura 1 - Diagrama ilustrando os elementos de uma interface

No contexto de linguagens e *frameworks* para Web e, mais especificamente, no contexto de *frameworks* PHP, existe uma prática do uso de interfaces na construção de componentes (*plugins*), que segue uma abordagem diferente. A partir de *frameworks*/ambientes analisados, constatou-se um padrão de relacionamento entre o ambiente e o *plugin*, o qual denominaremos neste estudo de

interface. Nessa prática, módulos recebem nomes pré-definidos ou são colocados em pastas pré-definidas, os quais direcionam o *framework* na chamada de serviços especializados. Por exemplo, usualmente existe um módulo PHP dentro do *plugin* com um nome padrão, responsável pela criação das tabelas que serão usadas por aquele *plugin* no banco de dados. A este tipo de funcionalidade damos o nome de interface, dado que ela se comporta como tal e, como apresentaremos adiante, pode ser substituída por uma interface declarada nos moldes tradicionais, ou seja, um módulo com nome pré-definido pode ser substituído por uma operação disponível na interface.

A prática de uso deste tipo de interface é comum em sistemas PHP principalmente antes da existência do PHP 5, onde a ausência dos conceitos de interface tornava necessário seu uso. Mesmo utilizando-se a versão 5 do PHP, alguns aplicativos que aceitam extensões na forma de componentes optam por usar uma estrutura física para descrever sua interface, com o objetivo de manter a retro-compatibilidade de versões da linguagem.

2.3 IDENTIFICAÇÃO DE COMPONENTES

Um mecanismo consistente de identificação de componentes é essencial, principalmente quando estes atuam de forma distribuída e podem ser versionados. Neste caso, são necessários mecanismos que garantam que a identificação de um componente seja única em ambientes distribuídos e em diferentes versões do componente.

Neste sentido, os modelos COM e XPCOM usam identificadores globais de 128 bits, o que garante sua exclusividade no mundo, ou seja, jamais se repetirá e garantirá a identificação de cada interface, classe ou objeto através do espaço e do tempo. Estes identificadores exclusivos globais são conhecidos como UUIDs (*universally unique IDs*).

Para estes modelos os nomes legíveis são atribuídos apenas por conveniência e utilizados apenas no escopo local. A utilização de UUIDs ajuda a garantir que os componentes COM e XPCOM, mesmo em redes gigantescas com milhões de outros componentes, não se liguem ao componente, interface ou método errado.

Outra estratégia de identificação universal única é aquela adotada por serviços Web. Em muitos aspectos os

2.4 COMPONENTES E *FRAMEWORKS*

Ainda em (BASS e outros, 2001) a tecnologia de componentes é definida a partir de um ambiente em que os componentes são executados; usualmente denominado de *framework* de componentes. A tecnologia inclui ferramentas para projetar, construir, combinar e entregar componentes ou sistemas construídos a partir de componentes. A ideia de *framework* de componentes remete à questão de que um modelo de componentes não diz respeito apenas à forma como cada unidade atua, mas a infraestrutura que dá suporte a execução do componente, conforme o modelo proposto. Modelos de componentes como EJB, COM e XPCOM são sempre acompanhados de um *framework* para o seu funcionamento.

3 OS DCCS

Esta pesquisa se realizou no contexto de um *framework* de gerenciamento de conteúdo em linguagem PHP, sobre o qual foi desenvolvida a infraestrutura de componentes aqui proposta. Tal infraestrutura exigiu um modelo de componentes que, ao mesmo tempo, fosse flexível o bastante para se adaptar às especificidades da linguagem PHP e estivesse preparado para o contexto de gerenciamento de conteúdo. Esta foi a razão para a escolha do *Digital Content Component* (DCC) ou Componente de Conteúdo Digital (SANTANCHÈ, 2006)(SANTANCHÈ; MEDEIROS, 2007) (SANTANCHÈ; MEDEIROS; PASTORELLO, 2007).

3.1 DCC

Um DCC é uma unidade auto descritiva capaz de encapsular tanto conteúdo em geral quanto código executável (SANTANCHÈ, 2006). Em ambos os casos, ele assume uma forma externa de componente, equivalente aos modelos de componentes de software.

Cada DCC é composto por quatro subdivisões (SANTANCHÈ; MEDEIROS, 2007):

1. o conteúdo que ele encapsula;
2. uma estrutura de gerenciamento – organiza as subpartes de um DCC e registra as relações entre elas;
3. interfaces do componente;
4. metadados descritivos.

Na subdivisão (1) está o conteúdo propriamente dito do componente, que pode ser, por exemplo, código executável ou artefatos multimídia (e.g., áudio, vídeo, texto) individuais ou combinados. A subdivisão (2) contém uma estrutura que organiza o que está disposto na subdivisão (1). Em muitos casos um DCC encapsula diversos artefatos, cuja organização e relação é descrita na seção (2).

Interfaces – subdivisão (3) – são um elemento fundamental dos DCCs e são elas que definem a forma como estes irão se relacionar. Além de seguir a clássica estrutura de interfaces

descrita na seção 2.2 , os DCCs utilizam padrões abertos da Web Semântica (RDF e OWL) para enriquecer semanticamente a descrição das mesmas.

Finalmente a subdivisão (4) inclui metadados descritivos do DCC como um todo. Tais metadados também envolvem padrões abertos da Web Semântica.

Os DCCs podem ser classificados em duas grandes categorias: DCCs Passivos e DCCs de Processo (SANTANCHÈ; MEDEIROS; PASTORELLO, 2007). Os DCCs passivos encapsulam qualquer tipo de conteúdo, mas não encapsulam software executável que implemente a funcionalidade declarada em sua interface; por esse motivo, para a sua operação sua infraestrutura os associa a um tipo de DCC chamado *Companion*, que implementa especificamente as operações declaradas em sua interface. A interface dos DCCs Passivos declara uma funcionalidade potencial, uma vez que descreve operações que não estão implementadas neste DCC, mas que são potencialmente executáveis e são realizadas pelo *Companion* DCC. Os DCCs de processo se comportam como componentes de software, encapsulando software executável que implementa suas operações.

Um DCC pode ser implementado de diferentes maneiras de acordo com o contexto da sua aplicação (SANTANCHÈ, 2006). Existem basicamente três formatos de implementação dos DCCs:

a) *Deployment* DCC – formato aberto de distribuição dos DCCs; utiliza padrões abertos da Web como XML e OWL para representação da estrutura e metadados;
b) *Storage* DCC – formato para armazenamento dos DCCs dentro de repositórios; o padrão não impõe nenhum formato específico para o *Storage* DCC, contanto que seja possível importar e exportar *Deployment* DCCs sem perda de conteúdo e semântica, por esse motivo, esse formato pode variar em diferentes soluções de armazenamento;
c) Runtime DCC – formato utilizado pelos DCCs em tempo de execução; varia de acordo com a solução de implementação do framework, em que um dos principais fatores é a linguagem de programação.

Para se aproveitar o pleno potencial dos DCCs é necessária a implantação dos três formatos. Por esse motivo, esta pesquisa faz parte de um projeto mais amplo no qual será adaptado o *framework* DCC completo para o contexto de PHP. Por se tratar de uma tarefa de grande complexidade, optamos neste projeto por implantar um subconjunto dos DCCs, sua infraestrutura de *Runtime*.

3.2 FLUID WEB

Conforme apresentado no Capítulo 2, todo modelo de componentes remete a um *framework* que dá suporte à sua execução e gerenciamento. Da mesma maneira os DCCs atuam sobre uma infraestrutura chamada *Fluid Web* (SANTANCHÈ, 2006). Tal infraestrutura atua simultaneamente como um *framework* de suporte aos *Runtime* DCCs e gerenciamento das tarefas de armazenamento e distribuição de DCCs.

3.3 ARQUITETURA GERAL DE *RUNTIME* DOS DCCS

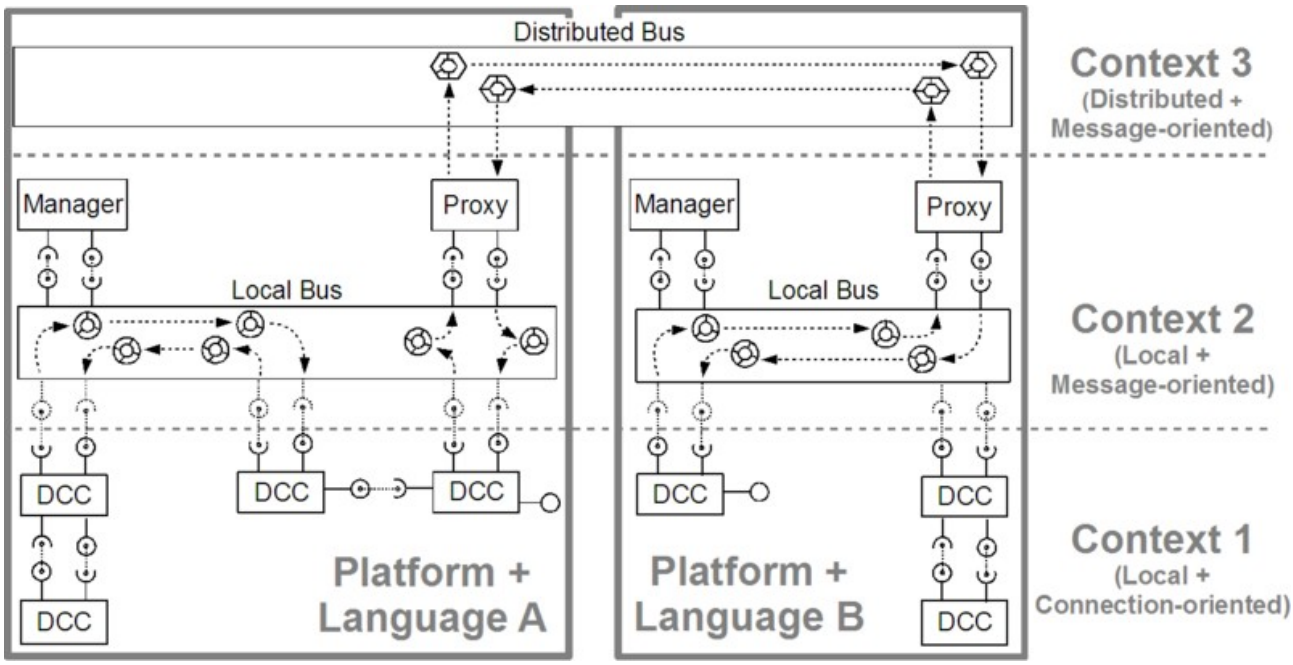

Figura 2 - Modelo dos contextos dos DCCs

A arquitetura de *runtime* dos DCCs é parte da infra-estrutura *Fluid Web* e tem como meta manter duas características essenciais: ser flexível e simples. A flexibilidade deve chegar ao ponto de poder aplicar o modelo em qualquer plataforma de desenvolvimento e linguagem, ver Figura 2. Esta primeira característica não deve, entretanto, comprometer a simplicidade no desenvolvimento de aplicações com DCCs.

Para sistematizar a forma como os DCCs se relacionam estes são organizados em contextos ou níveis. Os 3 níveis permitem uma variação entre flexibilidade × performance, flexibilidade × simplicidade, e uma maneira simples de trabalhar de forma distribuída. A Figura 2 representa uma descrição detalhada desta estrutura. Tendo a Figura 2 como guia, abaixo segue um detalhamento de seus três contextos.

Contexto 1 - O contexto 1 é a camada mais básica dos DCCs; este contexto é baseado em uma abordagem orientada à conexões. No diagrama é possível observar o funcionamento das composições de DCCs neste nível, que são conectados através suas interfaces usando conectores. Uma conexão é feita pela ligação de uma interface provida a uma requerida. Desta maneira os DCCs podem se compor para formar aplicações em que eles atuam de forma colaborativa. Este tipo de conexão exige que as interfaces sejam previamente especificadas e que as conexões sejam feitas em tempo de implementação, tornando a ligação entre os componentes bastante rígida.

Contexto 2 - Este contexto implementa uma abordagem orientada a mensagens, em que temos a entidade *bus* local, que é responsável pela transmissão das mensagens entre os DCCs. Os DCCs são ligados ao *bus* local através de uma interface especial padronizada (interface única). O componente *Manager* é responsável pelo gerenciamento das mensagens que trafegam pelo *bus*. Os objetos que trafegam no *bus* se apresentam na forma de objetos nativos da linguagem.

Contexto 3 - O contexto 3 entra em cena quando se torna necessária a comunicação de aplicações que estão sendo executadas em plataformas diferentes, seja *hardware*, SO ou linguagem de programação. O componente *Proxy*, que aparece conectado ao *bus* local na Figura 2, é responsável por comunicar o *bus* local com o *bus* distribuído do Contexto 3. A função básica do *Proxy* é transformar os objetos mensagens em um formato aberto e padronizado, apto para a distribuição na rede. Durante este processo de transmissão de mensagens, o *Proxy* serializa as mensagens dos DCCs que estão no Contexto 2 para o Contexto 3, assim como o *Proxy* também é responsável por desserializar a mensagem no caminho contrário.

A implementação do *framework* de *runtime* possui atualmente uma implementação de referência em Java. A Figura 3 ilustra a organização dos pacotes da versão Java do *framework*.

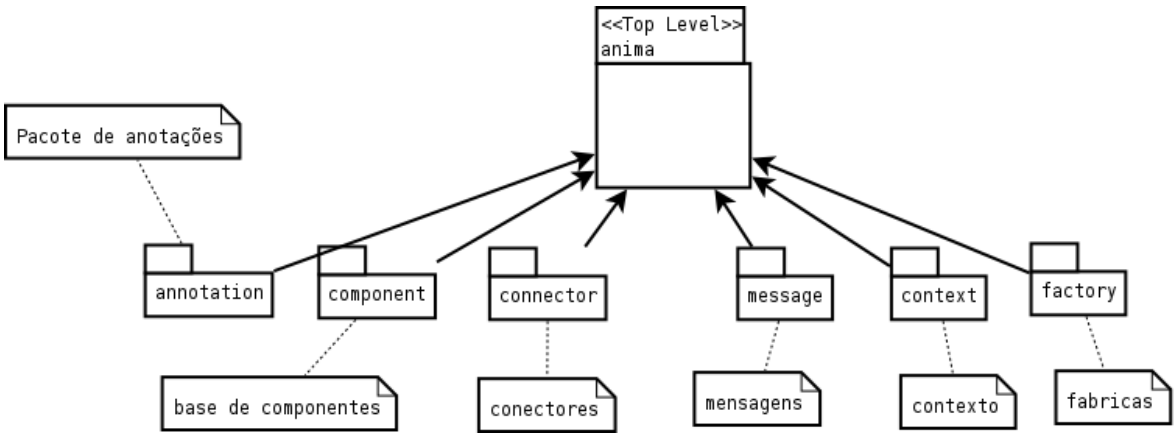

Figura 3 - Estrutura de pacotes do Fluid Web java

A seguir detalharemos a função da hierarquia superior dos pacotes:

Context – Reúne as ferramentas para caracterização do contexto, que será fundamental para a configuração da operação dos componentes.

Component – Fábricas, interfaces e implementações básicas dos componentes.

Connector – Fábricas, interfaces e implementações básicas dos conectores que ligam componentes.

Message – Representação das mensagens que circulam nos contextos 2 e 3.

Factory – Responsável pelo gerenciamento das fábricas que criam e entregam componentes, conectores e mensagens.

Annotation – Pacote onde estão concentrados os objetos de anotação, que são associados ao código através da sintaxe *annotation* do Java.

3.4 MODELO *RUNTIME* DOS DCCS

Dentro do modelo de *runtime* da infra-estrutura, cada DCC também deve seguir um modelo de *runtime*, que remete a um modelo de componentes de *software*. O modelo DCC busca combinar as vantagens dos principais modelos de componente de *software* em um modelo simples.

A seguir apresentaremos o modelo *runtime* dos DCCs a partir de um estudo de caso no qual será construído um DCC elementar chamado FishComponent. Este exemplo está disponível como programa introdutório na implementação de referência em Java (SANTANCHÈ, 2009).

3.4.1 Estudo de Caso

O componente Fish é um exemplo básico da aplicação das interfaces requeridas e providas em um DCC, assim como a criação e uso de sua identificação. O programa consiste em desenhar na tela do console um peixe baseado em caracteres, idêntico ao da Figura 4.

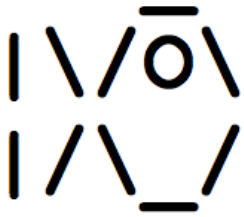

Figura 4 - Resultado final do exemplo componente Fish

Nesta pesquisa o exemplo Fish foi traduzido para o PHP como forma de realizar uma validação inicial do funcionamento do modelo DCC implementado em PHP. O exemplo apresentado a seguir está em sua versão original em Java e será usado como base comparativa para a versão em PHP, apresentada na Seção 6 .

Maior detalhamento técnico do exemplo em Java, seu desenvolvimento e onde baixar as ferramentas necessárias pode ser encontrado no site (FLUIDWEB...2009).

3.4.1.1 Definição do Componente

A Figura 5 representa de forma gráfica a configuração do componente que fará parte do exemplo. Em destaque, pode-se observar a interface provida (*provided Interface*) do componente *Fish*.

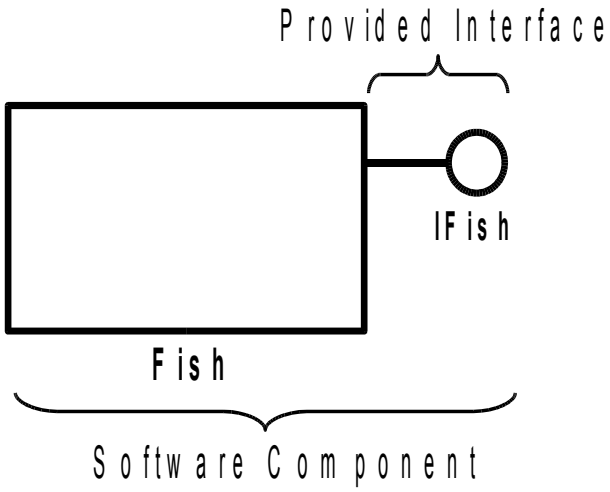

Figura 5 - O componente *Fish* e sua interface

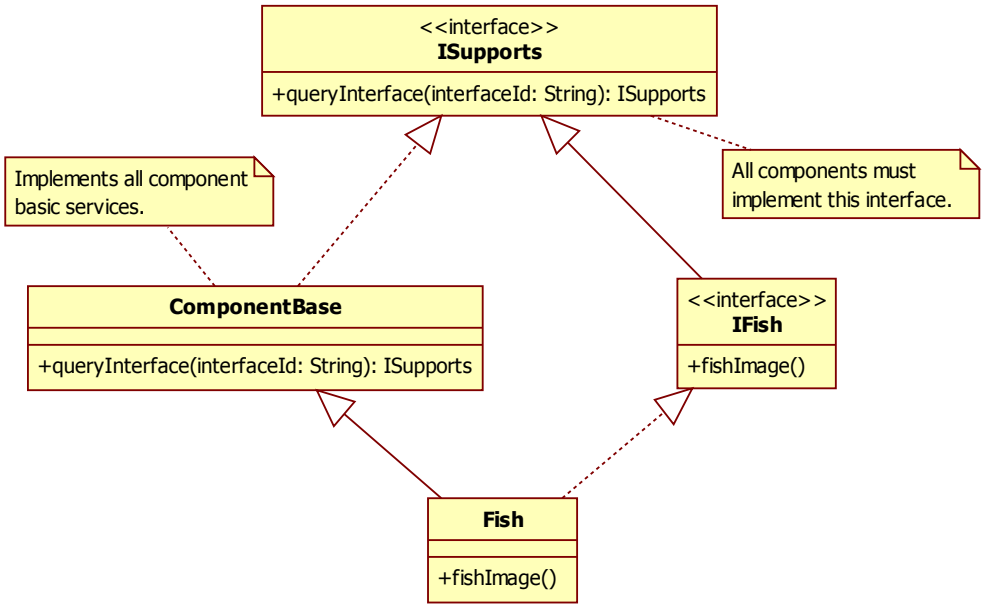

Figura 6 - Modelagem UML da implementação do FishComponent

Os métodos do DCC *Fish* são exclusivamente acessados através de sua interface provida. No caso deste exemplo é a interface IFish que especifica os serviços providos pelo componente.

A identificação do componente Fish deve ser feito através de uma URI. Mediante essa técnica obtém-se um identificador do componente Fish na forma de URI (vide mais detalhes no Capítulo 3.5 – Identificação dos DCCs):

```
http://purl.org/NET/dcc/examples.fish.s01.Fish
```

DCCs podem ser identificados por meio de qualquer URI válida, entretanto, para o contexto de Java há uma recomendação para a construção da URIs composta de 2 partes: O prefixo da URI (`http://purl.org/NET/dcc/`) e o caminho da classe *Class Path* (`examples.fish.s01.Fish`), resultando na URI apresentada acima.

A implementação do componente Fish se inicia pela implementação da interface provida IFish que deve estender a interface ISupports (Figura 6), em Java temos:

```
public interface IFish extends ISupports
```

A interface `ISupports` especifica as funcionalidades básicas que deve implementar cada componente como, por exemplo, métodos para *query* de interfaces e gerenciamento do ciclo de vida. Ela é derivada da interface de mesmo nome definida pelo XPCOM (MICROSOFT, 2009b).

Na versão Java dos DCCs, a identificação da interface é feita através da funcionalidade *annotation,* que insere metadados que "anotam" dados – o código.

Na versão 5 do Java foi adicionada uma API chamada de *Annotations* para o registro e manipulação de anotações. *Annotations* são metadados inseridos diretamente no código Java e que podem ser manipulados pela linguagem de programação. Tais metadados são relacionados com uma parte específica do código, podendo ser uma classe, um atributo, um método ou qualquer outro trecho de código. A escolha por tal mecanismo para identificação deve-se ao fato de que as *annotations* não interferem diretamente na operação do programa, mas agregam metadados que são acessíveis por ferramentas e bibliotecas que tratam o código. Em Java a identificação da interface do exemplo IFish usando *annotation* fica:

```
@ComponentInterface("http://purl.org/NET/dcc/examples.fish.s01.IFis
h")
```

A interface IFish por sua vez descreve o método fishImage mais especificamente:

```
public String fishImage();
```

3.4.1.2 Instanciação do Componente

A instanciação de um componente se dá através de uma fábrica abstrata (*Abstract Factory*). Os DCCs adotaram este padrão de projeto para instanciar todos os seus componentes. Detalhes deste padrão de projeto podem ser encontrados em (GAMMA e outros, 1995).

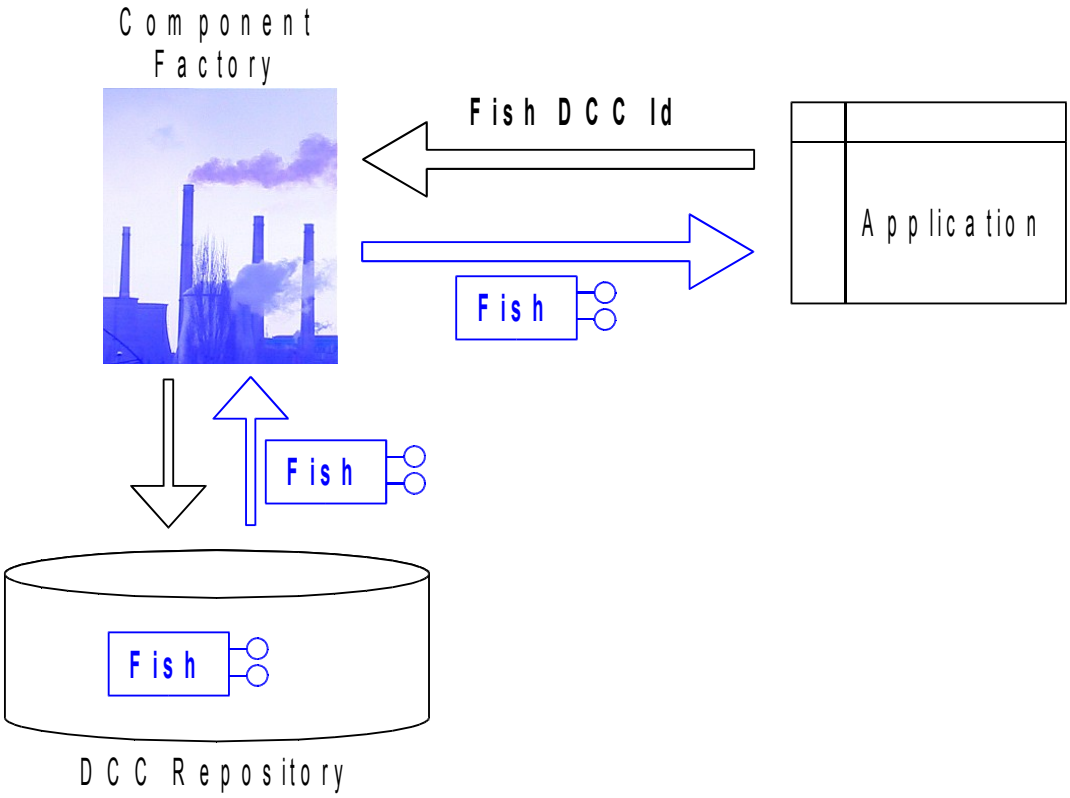

Figura 7 - Fábrica de componentes atendendo a solicitação de instanciação

Para ilustrar o modo como componentes são instanciados e usados existe uma aplicação que chama-se `FishTank`. Seu acesso ao componente se dá exclusivamente através da interface `IFish`. A aplicação usa a fábrica para solicitar ao gerenciador dos DCCs um componente que implementa a interface `IFish`. Tomando a Figura 7 como referência, a aplicação `FishTank` solicita à fábrica de componentes o componente *Fish* e a interface provida através de seus identificadores (URI), a fábrica por sua vez, solicitará ao repositório o componente que será acessado pela interface solicitada (`IFish`).

O repositório de DCCs deve ser capaz de gerenciar componentes permanentes, ou seja, aqueles que já estão homologados pelo sistema, e componentes temporários para os componentes que estão em fase de desenvolvimento ou testes. Esses componentes temporários são registrados

através do método `registerPrototype()` (Figura 8). Os componentes temporários ficam disponíveis na base apenas enquanto a aplicação que o registrou está em funcionamento. Isto confere ao componente o caráter provisório típico de um software em desenvolvimento.

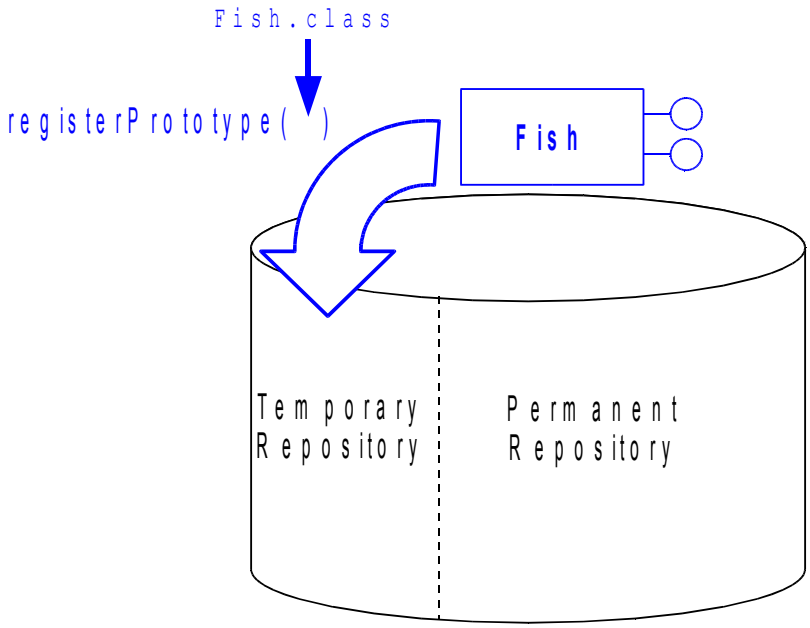

Figura 8 - Funcionamento do repositório de DCCs

A seqüência para instanciação e uso de um componente no modelo DCC é composta de quatro passos:

- Criar uma fábrica:

```
IGlobalFactory factory =
ComponentContextFactory.createGlobalFactory();
```

- Registrar o componente temporário no repositório (esta etapa só é necessária para os componentes temporários, os componentes permanentes não precisam de registro através de código, dado que já estão registrados na base permanente):

```
factory.registerPrototype(Fish.class);
```

- Instanciar o componente e sua interface provida baseando-se em suas URIs:

```
IFish component = (Ifish)factory.createInstance(
  "http://purl.org/NET/dcc/examples.fish.s01.Fish",
  "http://purl.org/NET/dcc/examples.fish.s01.IFish");
```

- Finalmente, usar o componente chamando o método desejado:

```
System.out.println(component.fishImage());
```

O resultado desta implementação deve ser equivalente à Figura 4 já apresentada anteriormente.

3.5 IDENTIFICAÇÃO DOS DCCS

Como citado no Capítulo 2.3, um modelo de componente precisa garantir a identificação única de cada componente em seu universo, mesmo quando se trata de ambientes distribuídos. Cada DCC deve ser identificado por uma URI válida, única no mundo. Esta identificação deve ser tão estável quanto possível. No melhor cenário, a identificação de um DCC nunca vai mudar. URIs persistentes são recomendadas para a identificação dos DCCs. Este tipo de URI utiliza um mecanismo de redirecionamento em que uma URI, registrada em um servidor de URIs persistentes, é associada a outra URI que indica seu lugar físico na rede (usualmente uma URL). Deste modo, modificações devido a alterações no *host* do servidor Web ou na sua organização interna, que podem causar modificações no endereço físico, não afetam a URI persistente, que só precisa ser redirecionada para novas configurações. Um site onde pode-se encontrar mais informações sobre URI persistentes, assim como criá-las é o (PURL, 2009).

A identificação por meio de URIs se diferencia daquela por meio de UUIDs adotada pelo COM e XPCOM, mencionada anteriormente. Por um lado o uso de URIs permite inserir a descrição de componentes na Web Semântica, por outro, as UUIDs dispensam qualquer serviço de nomes ou mecanismo de validação na rede.

3.6 VIEW DCC

Durante a elaboração desta pesquisa iniciou-se um processo de especificação de uma categoria especial de DCCs chamada *View* DCC ou VDCC. Esta especificação é uma das bases para uma das principais contribuições deste projeto, que consiste em usar o modelo VDCC na concepção de uma interface para sistemas CMS baseada na composição de unidades menores, capazes de

serem destacadas e utilizadas individualmente.

A especificação dos VDCCs trata de um conjunto específico de DCCs que têm a funcionalidade de apresentação. VDCC é um tipo especializado de DCC de Processo.

VDCCs podem ter as seguintes funções:

Root **-** Unidade central de apresentação, que implementa uma solução completa para se apresentar para o usuário, assim como apresentar os VDCCs que estão contidos nele, ou seja, componentes *Root* têm autonomia de se apresentar na tela sem estar inseridos em outros componentes visuais que os gerenciam. Os *Roots* devem implementar a interface `IViewRoot` (Figura 9).

Composite **-** Compõe outros VDCCs dentro sua área visual. Um *Composite* define as regras para a apresentação dos componentes que estão contidos nele. *Composites* devem implementar a interface `IViewComposite` (Figura 9).

Member **-** Membro de um *Composite* e parte da formação de soluções visuais *Composed*. Os *Members* devem implementar a interface `IViewMember` (Figura 9), que por sua vez requer a interface `IViewComposite`.

Figura 9 - Diagrama de classe UML das interfaces básicas do VDCC

As interfaces acima podem ser combinadas. Deste modo, um componente *Root* também pode ser um *Composite* e um componente *Composite* também pode ser um *Member*.

A Figura 10 ilustra um cenário típico com VDCCs, cuja relação é detalhada na Figura 11 , em que as interfaces `IViewComposite` e `IViewMember` estão representados. Conforme ilustrado na Figura 10, geralmente cada VDCC se encaixa em uma das três categorias: *Top, Middle* e *Atomic*.

Atomic VDCCs são as mais básicas unidades de apresentação. Normalmente eles não são autônomos e são combinados dentro de *Composite* VDCCs para produzir resultados mais complexos. Por ser atômicos eles não agregam outros VDCCs e, portanto, não implementam a interface `IViewComposite`.

Top VDCCs são autônomos o suficiente para se auto-apresentar e apresentar o seu conteúdo para o usuário. Portanto, eles não são contidos por qualquer outro VDCC e não precisam implementar a interface `IViewMember`.

Middle VDCCs desempenham um papel intermediário, combinando VDCCs em grupos para ser incluídos em um VDCC de nível superior. Eles não são autônomos.

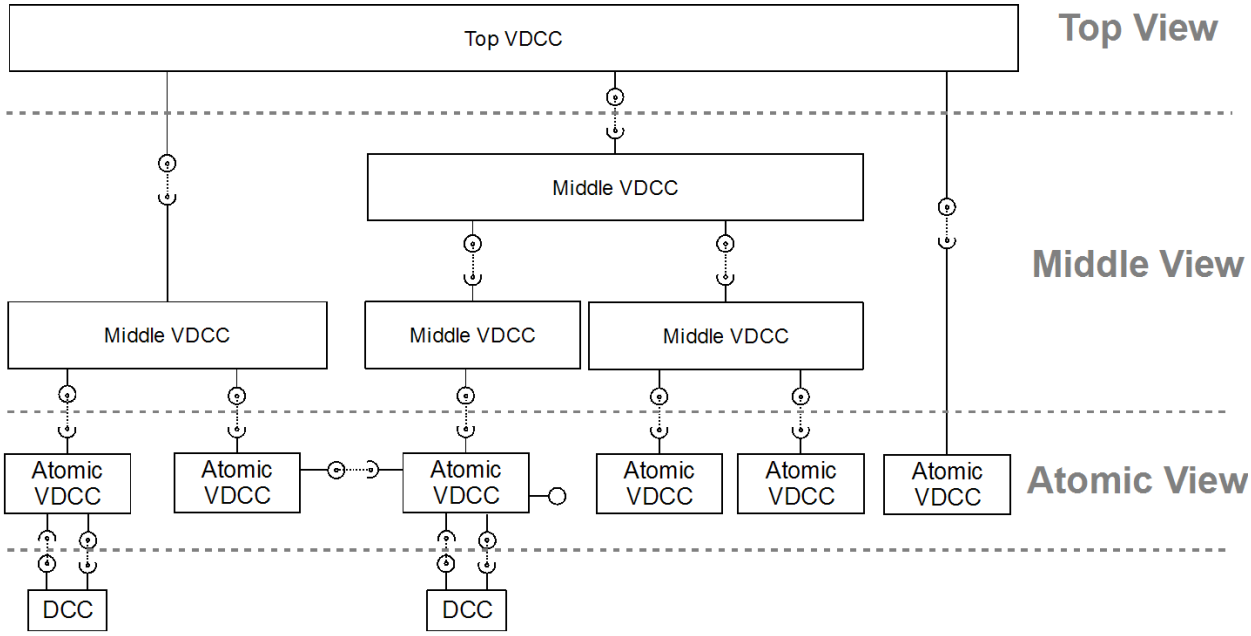

Figura 10 - Diagrama ilustrando um tipico cenário de VDCCs

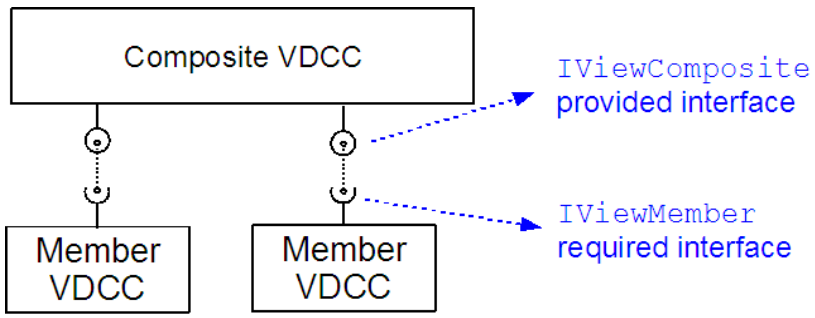

Figura 11 - Relacionamento Container/Component na Figura 10

3.7 *WIDGET* DCC (WDCC)

WDCC é um VDCC destinado a utilizar a *Graphic User Interface*. Interfaces do WDCC (`IWRoot`, `IWComposite` e `IWMember`) especializam as respectivas interfaces ViewDCC com a

finalidade de lidar com as especificidades de cada plataforma. Uma vez que cada linguagem e *toolkit GUI* tem a sua própria abordagem para representar objetos da *GUI*, as interfaces `IWRoot` e `IWComposite` podem ser adaptadas para se adequar a cada contexto. Conforme ilustrado na Figura 12, a interface `IWComposite` define uma operação *add* que recebe um `<contextualized_Widget_object>` como parâmetro. Na mesma linha de raciocínio, `IWRoot` retorna uma `<contextualized_Widget_object>`. Estes são objetos GUI típicos da linguagem ou do *framework* gráfico ao qual os WDCCs serão adaptados. A fim de esclarecer seu funcionamento, na seção seguinte serão apresentados os WDCCs no contexto Java.

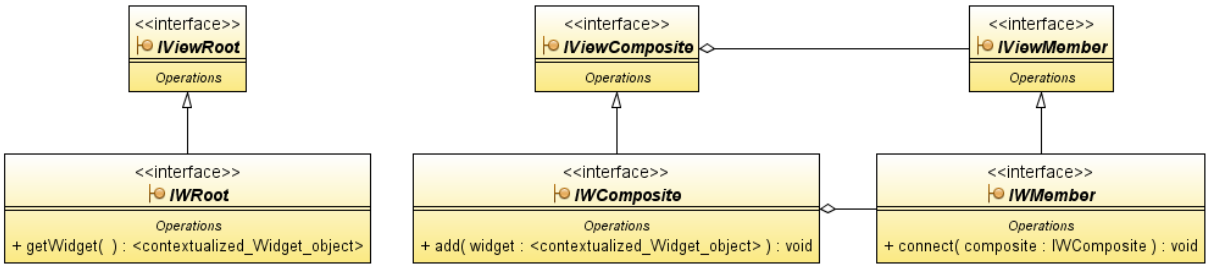

Figura 12 - Diagrama de Classe UML da interface básica do GUIDCC

A questão fundamental é que cada linguagem ou *framework* tem sua própria hierarquia de objetos gráficos que tratam sua interface. Os componentes fazem naturalmente uso destes objetos nativos, evitando a recodificação. Por outro lado, a hierarquia de classes dos objetos gráficos é independente da hierarquia de classes que implementam os componentes. Para resolver esta questão, os WDCCs são associados a objetos gráficos nativos da linguagem ou *framework* – objetos classificados como *Widgets* – e a composição de WDCCs reflete em uma composição paralela de Widgets.

Para realizar esta composição paralela, quando um *Member* WDCC (que implementa a interface `IWMember`) é inserido em um *Composite* WDCC (que implementa a interface `IWComposite`) o *Widget* associado ao *Member* também é inserido no *Widget* associado ao *Composite*, seguindo três etapas:

1. O *Composite* WDCC é ligado ao *Member* WDCC pelo método *connect* do componente.
2. O *Member* WDCC chama o método *add* do *Composite* WDCC para inserir seu objeto *Widget* no *Composite*. Normalmente, o *Member* WDCC insere apenas um *Widget* no recipiente, mas também é possível inserir vários objetos.
3. A *Composite* WDCC interage com o objeto *Widget*. Por exemplo, pode chamar o

método *repaint*.

3.7.1 WDCC no Contexto Java

A Figura 13 ilustra típicas interfaces `IWRoot` e `IWComposite` adaptados ao contexto Java *Swing*. A operação *add* do `IWComposite` recebe um objeto da classe `Component` do Java AWT. Esta classe é a base dos componentes visuais do Java, tanto de seu *framework* mais antigo (AWT) quanto daquele mais novo (*Swing*). O mesmo tipo é retornado pelo método `getWidget` do `IWRoot`.

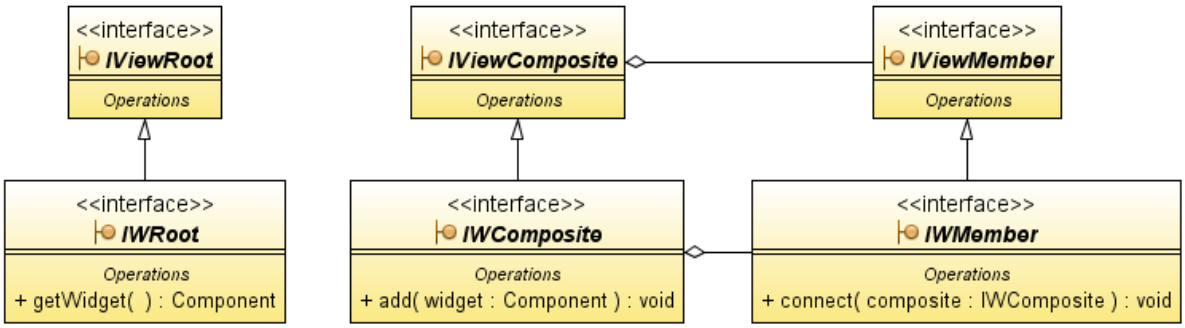

Figura 13 - Diagrama de classe UML – interfaces WDCC implementadas no contexto Java *Swing*

3.7.2 WDCC no contexto das Linguagens Web

Uma vez que as linguagens como JavaScript, PHP, ASP, etc. não têm um *framework* GUI padrão, os objetos *Widget* têm que ser definidos diretamente dentro do framework de DCCs e o acesso, aos mesmos, é feito através da interface `Iwidget` como está ilustrado na Figura 14.

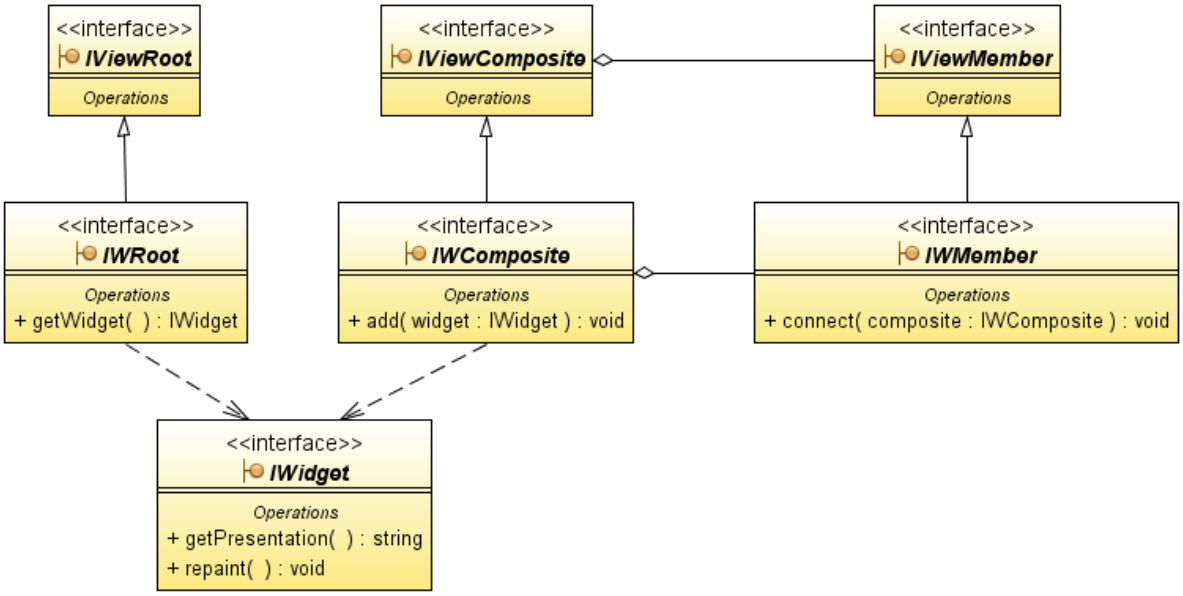

Figura 14 - Diagrama de classe UML, interfaces WDCC implementando o contexto das linguagens Web

Como será apresentado no Capítulo 8, esta estrutura visual dos DCCs será explorada para permitir que componentes visuais PHP possam ser embutidos em outras aplicações.

4 MODELOS DE COMPONENTES EM PHP

4.1 OBJETIVO DESTE CAPÍTULO.

No final desta pesquisa pretende-se alcançar um modelo de componentes genérico apto à linguagem de programação PHP e a sua aplicação em sistemas CMS. Neste Capítulo serão tratadas as principais características da estrutura de modelos de componentes e *plugins* PHP, bem como trabalhos relacionados ao contexto de PHP e CMS.

4.2 A LINGUAGEM PHP

Segundo o TIOBE *Programming Community Index* (TIOBE, 2009), um índice dedicado a medir a popularidade das linguagens de programação, o PHP é a quinta linguagem mais utilizada no planeta, vide a Figura 15 que traz um recorte extraído do site TIOBE .

Position Feb 2009	Position Feb 2008	Delta in Position	Programming Language	Ratings Feb 2009	Delta Feb 2008	Status
1	1	=	Java	19.401%	-2.08%	A
2	2	=	C	15.837%	+0.98%	A
3	5	⇈	C++	9.633%	+0.36%	A
4	3	↓	(Visual) Basic	8.843%	-2.76%	A
5	4	↓	PHP	8.779%	-1.11%	A

Figura 15 - Recorte de uma tabela comparativa do Site TIOBE

Nota: Figura extraída em 19 de Fevereiro de 2009

O PHP nasceu há cerca de 15 anos, em 1994, e foi criado por Rasmus Lerdof como um

conjunto de ferramentas para monitorar o seu site (VIANNA, 2007). PHP chamou a atenção de muitas pessoas e que deixou de ser o *Personal Home Page Tools*, para se chamar PHP: *Hypertext processor*. O PHP é *open-source* e pode ser baixado por meio de seu site oficial: www.php.net.

"É uma linguagem de programação utilizada para desenvolvimento para a web sendo a mesma de grande utilização, podendo ser mesclada dentro do código HTML." (NIEDERAUER, 2004).

Vantagens do PHP (SIQUEIRA, 2002):

a) É uma linguagem de fácil aprendizado;
b) Tem performance e estabilidade excelentes;
c) Seu código é aberto e não é preciso pagar por sua utilização;
d) Tem suporte nos principais servidores Web do mercado;
e) Suporta conexão com os bancos de dados mais utilizados do mercado, como: MySQL, PostgreSQL, Oracle, DB2;
f) Tem suporte aos sistemas operacionais mais utilizados no mercado.

Figura 16 - Interação entre o PHP, o servidor WEB e o browser do usuário

A Figura 16 mostra o fluxo de uma requisição http para o servidor Web. Analisando a figura observa-se que uma das características do PHP é que o código é executado no servidor. O que é enviado ao cliente é apenas HTML resultante do código PHP executado. Sendo assim, é possível se interagir com um banco de dados e outras aplicações existentes no servidor, sem que o cliente precise ter qualquer software extra além do navegador Web.

Durante os anos de existência do PHP, muitos *frameworks* foram criados para facilitar o desenvolvimento de sistemas Web e o gerenciamento de conteúdo. Abaixo segue um estudo de importantes *frameworks* que se popularizaram no domínio PHP. Neste Capítulo serão analisadas suas estruturas de codificação e modelos de componentes, além de um breve comparativo entre as

soluções.

Tais *frameworks* definem modelos de extensão baseados em módulos de *software*, derivados do modelo de componentes de *software*. Tais modelos compartilham de algumas características que são fundamentais para a nossa análise e para o modelo proposto nesta pesquisa. Por esse motivo, nas subseções a seguir detalharemos tais sistemas, bem como o modelo de componentes subjacente.

As ferramentas pesquisadas foram: Joomla! e o PEAR:

4.3 O JOOMLA!

O Joomla! é um *framework* que pode ser utilizado tanto como *Content Management System* (CMS) quanto como base de desenvolvimento de sistemas. Sua arquitetura motivou a criação de uma grande quantidade de *plugins* e componentes.

O Joomla! 1.5 é um sistema de três níveis, veja a Figura 17:

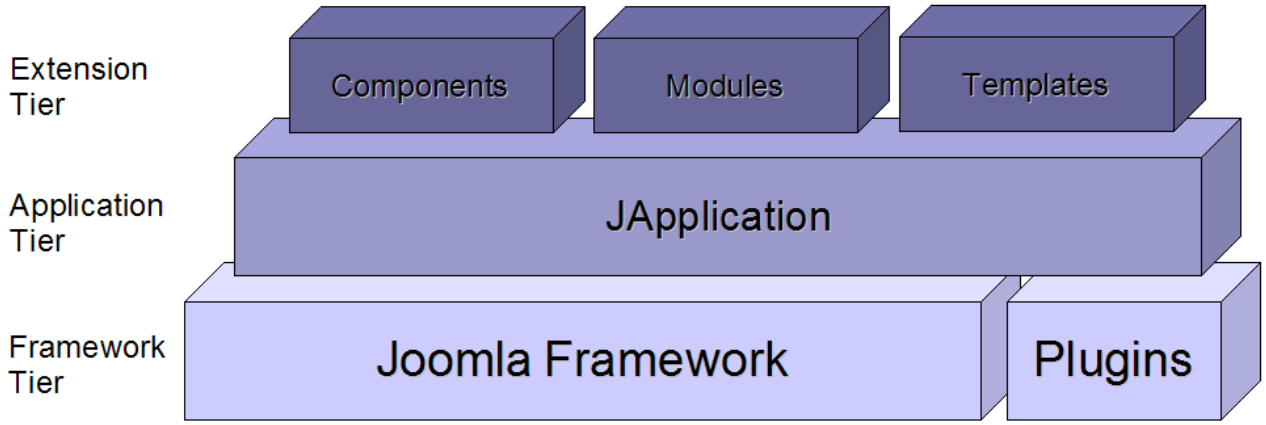

Figura 17 - Camadas do Joomla!

Fonte: (FRAMEWORK...2009)

O nível mais baixo é a camada do *Framework.* Esta camada é composta das bibliotecas e dos *plugins*. O segundo nível é a camada de aplicação, onde estão situadas as classes JApplication, que é conhecida como a classe básica para o CMS Joomla!. A terceira camada é o nível das extensões para o Joomla! É neste nível que todos os componentes, módulos, *templates* e lógicas são executados e renderizados.

Tal modelo de camadas permite a customização do CMS ou a inclusão de novas

funcionalidades ao sistema de forma sistemática, dispensando "*hacks*" ou modificações no código fonte do *kernel*. Se analisado aplicando conceitos de reuso de código, portabilidade e produtividade, este tipo de estrutura possibilita atingir parcialmente o objetivo da componentização.

Existem muitas vantagens em se escrever uma extensão ao invés de se alterar o núcleo de uma aplicação. Sempre que o *framework* for modificado, ou até mesmo, quando houver o lançamento de uma nova versão, é provável que o núcleo do código deste *framework* seja modificado, no entanto, se forem garantidas as interfaces com as extensões já desenvolvidas, não haverá necessidade de re-escrevê-las. Por outro lado, ainda que as interfaces mudem, o impacto de modificação será claramente definido pelas novas interfaces que facilitarão a localização das mudanças e construção de novas versões de extensões compatíveis com o novo núcleo.

Todas as extensões do Joomla! podem ser consideradas como os componentes, no entanto, os desenvolvedores do Joomla!, classificaram suas extensões em 3 tipos: Componentes, Módulos e *Plugins*.

Componentes - Analisando o sistema como um todo, dentre os três tipos, este é o principal. O Componente representa a base do que é visto nas páginas do sistema. Por este motivo, Joomla! foi desenvolvido para carregar somente um componente por vez em cada página gerada, e este será o responsável por coordenar a ação do que é apresentado e gerenciar o processo de *back end*, que envolve desde a consulta e atualização da base de dados como qualquer outra ação no servidor. Consequentemente o gerenciador de conteúdo do Joomla! é considerado um componente. Os componentes, em comparação com os *plugins*, trabalham em camadas superiores, sem interferência direta no *framework*.

Módulos - Em contraste com os componentes, os módulos têm tarefas complementares e atuam sobre os componentes. Por este motivo, qualquer quantidade de módulos pode aparecer em uma página do Joomla!. Os módulos são usualmente aplicados em tarefas complementares, como a composição de uma barra lateral, do menu da página, etc. Os módulos são colocados como painéis em um Joomla! *template*.

***Plugins* -** Quando existe um pedaço de código que será necessário em todo o *website*, este código é melhor implementado como um *plugin,* ou seja, de acordo com o gráfico na Figura 17 observa-se que *plugins* ficam em paralelo ao *framework.* Isto é formalmente chamado pelo Joomla! de "Mambot" ou "Mambo robô". Esse é um *apelido para os plugins* do Mambo, que é a aplicação percursora do Joomla!. *Plugins* são geralmente usados para formatar a saída de um componente ou módulo sempre que uma página é construída. *Plugins* são extensões para o *framework* do Joomla! que operam em um nível inferior aos componentes. Para ilustrar esta relação imagine que o usuário precise de um editor de texto HTML que funcione no navegador e se utilize dos recursos

WYSIWYG. O desenvolvedor pode optar por ferramentas em Javascript conhecidas na Web e adaptadas para serem *plugins* do Joomla! como, por exemplo, TinyMCE (MOXIECODE SYSTEMS, 2009) ou HTMLArea3 (A DIRECTORY...2009) ou FCKeditor (FCKEDITOR...2009) ou XSTANDARD (XSTANDARD...2009) sem que esta opção interfira no núcleo do *framework*.

4.4 O PEAR (PHP *EXTENSION AND APPLICATION REPOSITORY*)

O PEAR é um repositório e uma plataforma para a codificação de componentes em PHP. O projeto foi fundado em 1999 por Stig S. Bakken para promover a reutilização de código (MINETTO, 2007).

O projeto visa fornecer:

a) Uma biblioteca estruturada de código aberto para programadores PHP;

b) Um sistema de distribuição de código e gerência de pacotes;

c) Um padrão para a escrita de códigos em PHP;

d) Uma biblioteca de extensões para o PHP (PECL);

e) Um site, uma lista de *e-mails* e servidores para download de códigos.

Um pacote PEAR pode ser composto pelo código fonte e/ou os respectivos binários. O PEAR declara explicitamente todas as dependências em cada Pacote.

O PEAR é dirigido por uma comunidade de desenvolvedores de código aberto. Assim qualquer um pode contribuir com um código em forma de um novo pacote ou com melhoria ou um pacote existente. Durante anos o PEAR desenvolveu um conjunto de convenções, algumas delas estão evoluindo e outras se solidificaram, por isso para se contribuir com o PEAR existem sete exigências listadas a seguir:

1. O código de cada pacote deve ser apropriadamente licenciado sob uma licença *Open Source*.

2. O código deve estar disponível em um repositório de código público.

3. Código extensível é "*forward-compatible*". É importante sempre ter em mente que um código do PEAR deve ser extensível e fácil de adicionar novas características. Pois compatibilidade não é apenas sobre o apoio à mesma sintaxe e semântica de versões anteriores, é também sobre planejamento futuro ao escrever o código.

4. O código deve ter documentação apropriada em um dos formatos, Docbook XML [http://www.docbook.org] ou Texto ASCII

5. Devem ser implementados testes da regressão nos *frameworks* de teste: phpt [http://phpt.info/] ou formato de PHPUnit [http://www.phpunit.de/]

6. O desenvolvedor deve estar disposto a fornecer a sustentação para o pacote e deve também liberar as versões futuras com menos *bugs*.

7. O pacote deve estar de acordo com os padrões do estilo de código do PEAR. Abaixo segue uma explicação mais detalhada sobre este padrão.

4.4.1 O PEAR Coding Standards

O PEAR *Coding Standards* (frequentemente abreviado como CS) tem como principal objetivo manter o código consistente para ser facilmente legível e sustentável por mais desenvolvedores PEAR.

Algumas destas regras podem facilmente ser aplicadas em outros modelos de componentes independentemente da linguagem. Dentre estas regras definidas pelos desenvolvedores do PEAR podemos destacar: identação e o comprimento da linha para formatação dos códigos; especificação da forma como devem ser escritas as estruturas de controle; detalhamento na padronização nas chamadas de funções; formatação específica nas definições das classes e das funções, assim como as suas convenções de nomes; definição de padronização nos comentários para maior entendimento dos objetivos dos códigos escritos, dentre outras coisas. Para um melhor detalhamento destas convenções ver o Apêndice A.

4.4.2 Descrição e declaração de dependências.

O arquivo package.xml é um XML *well-formed* (EXTENSIBLE...2009) que contém todas as informações sobre um pacote do PEAR, inclusive a descrição detalhada das dependências deste pacote; o que é uma das características dos componentes de sucesso.

A manipulação das dependências é um dos pontos fortes do PEAR. Para que haja sucesso na sua implementação é crucial compreender como especificar uma dependência, e as maneiras diferentes de assegurar-se de que um pacote seja usado somente nos sistemas que o suportam.

A declaração de pacote do PEAR utiliza a *tag* <*dependencies*> que organiza as dependências em grupos, registra os atributos nas *tags* e não abrevia as palavras para maior clareza e legibilidade humana.

O controle de dependências do PEAR é feito nos seguintes itens (The PEAR Documentation Group, 2007):

Função	Tag	Descrição
Instalador	pearinstaller	Define as versões do PEAR que se pode instalar o pacote. Como todas as *tags* de dependência que suportam "versionamento", estas quatro *tags* são utilizadas para determinar e explicitar as versões suportadas.
Subpacote	subpackage	As dependências dos sub-pacotes compartilham do mesmo formato XML que as dependências do pacote. A dependência de sub-pacotes deve ser usada somente se um pacote for dividido em mais de um sub-pacote.
Sistema Operacional	os	A dependência do OS é usada para restringir um pacote a uma classe particular de Sistemas Operacionais (como Unix) e a um OS específico (como o Darwin, ou o Freebsd).
Arquitetura	*arch*	A dependência de arquitetura é usada para restringir um pacote a um OS e a uma arquitetura específica do processador.

Quadro 1 - Parâmetros para controle de dependências do PEAR

Para cada um dos itens acima podem ser especificados os seguintes elementos de controle:

Função	*Tag*	Descrição
Versão Minima	*min*	Versão mínima suportada pelo pacote. Esta *tag* é requerida em todos os arquivos package.xml.
Versão Máxima	*max*	Versão máxima do instalador suportada. Esta *tag* impedirá que o pacote seja instalado por qualquer um com uma versão mais nova do PEAR!
Versão Recomendada	*recommended*	A versão recomendada do instalador do PEAR
Versão Incompatível	*exclude*	Versões incompatíveis do instalador do PEAR.

Quadro 2 - Valores para os itens do Quadro 1

4.4.3 Considerações Finais sobre este Capítulo

A análise destes dois exemplos de *software* demonstra diferentes implementações de estruturas de componentes. O contraste entre as diferentes estratégias permite perceber os pontos fortes de cada uma, que podem ser reaproveitados em novas pesquisas e implementações.

O Joomla! apresenta uma divisão do sistema em camadas, em que componentes assumem papéis diferentes de acordo com a camada. A clara distinção entre o *core* e as extensões permite que atualizações tanto dos componentes como do *framework* não interfiram uns nos outros. Entretanto, o modelo de interfaces do Joomla! é baseado em arquivos e diretórios pré-definidos, não explicitando a exata interface entre módulo e *framework*.

O PEAR apresenta como destaque a necessidade de padronização do código fonte, já que este se trata de uma aplicação de código aberto e existe uma comunidade trabalhando num projeto em comum. Por isso a clareza e a uniformidade de estilo é necessária para facilitar na colaboração no projeto. Além disso, o PEAR demonstra sua maturidade na representação de dependências de sistemas, pacotes, arquiteturas, etc. para que um componente funcione. Isso permite que o desenvolvedor possa facilmente substituir um componente por outro com o mínimo de esforço e, possivelmente, sem a necessidade de qualquer alteração no código fonte.

Ambos os sistemas foram tomados como base no desenvolvimento do EVA, descrito no Capítulo 7. No ano de criação do EVA (2004), o Joomla! ainda não existia, porém seu percursor, o Mambo, teve uma forte contribuição na concepção do EVA. O Mambo e outros sistemas similares foram desmontados e agregados em forma de módulos do EVA, formando assim o que chamamos de EVAcms. Já o PEAR tem sua principal contribuição no EVA *framework*, onde são utilizados alguns de seus componentes como por exemplo o PEAR_MAIL, que é responsável por gerenciar o envio de e-mails dos módulos do EVA.

5 GERENCIAMENTO DE CONTEÚDO

Neste Capítulo será tratado o tópico gerenciamento de conteúdo, que é um tema chave deste projeto, dado que o modelo de componentes PHP aqui proposto foi desenvolvido sobre o EVA, que consiste em um *framework* com seu núcleo básico especializado no gerenciamento de conteúdo. Além do tópico específico sobre Sistemas de Gerenciamento de Conteúdo, trataremos também sobre Sistemas de Gerenciamento Educacional, que estão indiretamente relacionados com o tema principal deste projeto, mas que serão essenciais para ilustrar o diferencial da nossa proposta de combinação do EVAcms com DCCs.

5.1 CMS

Já é notório o crescimento constante da participação ativa dos usuários na geração e publicação de conteúdo na Web. O volume e a importância das informações digitais têm aumentado muito nos últimos anos, como consequência da facilidade de acesso e necessidade de agilidade na produção de informações no ambiente Web.

Com o objetivo de facilitar a criação e gerenciamento de conteúdo surgem os Sistemas de Gerenciamento de Conteúdo (SGC). A ideia surgiu em 1995 (REVISTA W, 2007) por duas empresas, a CNET e a Vignette, que disponibilizaram publicamente a tecnologia por eles utilizada e a denominaram de *Content Management System* (CMS) ou Sistema Gerenciador de Conteúdo.

O termo “gestão de conteúdo” relaciona-se a um amplo conjunto de áreas e tecnologias, incluindo gestão do conhecimento, gestão do arquivamento e ciclo de vida de documentos, gestão de conteúdo Web, gestão de mensagens e correio eletrônico, gestão de fluxo de trabalho e processos de negócio, portais, pesquisa, equipes e colaboração (BAX; PEREIRA, 2002).

CMS é principalmente um sistema na plataforma Web desenvolvido para auxiliar os usuários não técnicos a inserir, alterar e gerenciar conteúdo em um Web site de forma rápida e simples (ADIERS, 2007)(ROBERTSON, 2002). Assim os usuários encarregados pelo desenvolvimento de conteúdo de um portal podem gerenciar o conteúdo de forma colaborativa e

distribuída. Sistemas CMS alcançaram grande aceitação na comunidade, o que resultou em uma grande diversidade de soluções disponíveis (ADIERS, 2007) . Segundo Paul Browning e Mike Lowndes (BROWNING; LOWNDES, 2001), quando a criação e publicação de conteúdo é bem gerida a organização tem uma melhor relação custo-eficácia, tornando-se susceptível a conduzir melhor sua tomada de decisão.

Adiers (2007) cita algumas das principais características e funcionalidades de um CMS, como disponibilizar uma maneira interativa para a apresentação de conteúdo na forma de páginas Web, gerenciar o processo no qual usuários adicionam e editam conteúdo, a possibilidade de criar e gerenciar conteúdo descentralizadamente, oferecer suporte à indexação e busca de informação, etc. Estas aplicações voltadas para a Web variam desde páginas simples, passando por páginas complexas com separação conteúdo-*layout*, até a construção de páginas dinâmicas a partir de um banco de dados (ROBERTSON, 2002). Dentre os domínios de aplicação do CMS estão: materiais de formação, documentação (manuais, procedimentos etc.), documentos comerciais etc. (ROBERTSON, 2002).

5.1.1 CMS e Reuso de *Software*

Se por um lado CMSs são uma ferramenta para reuso de conteúdo, por outro, eles também podem ser vistos sob a perspectiva de reuso de *software*. Neste subcapítulo analisaremos esta faceta dos sistemas CMS sob duas perspectivas: extensão na forma de componentes de *software* e *framework* para o desenvolvimento de sistemas. Ambas as facetas são muito relevantes no contexto deste estudo, dado que a nossa proposta tem enfoque justamente no reuso de *software*.

Enquanto componentes de *software* são uma técnica para reuso de *software* na forma de uma solução pronta, na qual usualmente o cliente não conhece detalhes internos da implementação do mesmo, *frameworks* seguem uma direção diversa, dado que funcionam como um “esqueleto” para um novo sistema e, portanto, o objetivo é que o cliente tenha conhecimento da estrutura interna da implementação para reusar não apenas o código, como também o design (JOHNSON, 1997a).

5.1.2 Estendendo um CMS através de Componentes de *Software*

Como observado com o Joomla! e outros Sistemas Gerenciadores de Conteúdo, usualmente sua extensão é feita a partir da instalação de *plugins*. Para efeito da análise a ser realizada neste trabalho, tais *plugins* são considerados como um subconjunto de componentes de *software*. *Plugins* no universo dos CMSs são considerados componentes não essenciais e não fazem parte do núcleo base de um *software.* Em outras palavras, são componentes que servem para estender o aplicativo de forma simples e usualmente podem ser inseridos ou retirados por usuários, sem que esses tenham conhecimentos técnicos e sem a necessidade da interferência de um desenvolvedor.

5.1.3 CMS como *Framework*

Se por um lado o CMS pode ser visto como um sistema completo para gerenciamento de conteúdo, por outro, grande parte dos CMSs são projetados para serem utilizados como *framework* de suporte ao desenvolvimento de sistemas especializados. Nesse caso, o CMS se comporta como um "esqueleto" de *website*/portal, com recursos básicos e de manutenção e administração já prontamente disponíveis. Comparado a um simples "modelo de *website*", um CMS permite a criação, armazenamento e administração de conteúdo Web de forma dinâmica, através de uma interface de usuário via Web, ao invés de se limitar a conjunto de páginas HTML estáticas. A aparência de um *website* criado com um CMS é customizável através da utilização de *templates* ("modelos visuais" de *website*), que podem ser facilmente substituídos, detalhes da implementação dos *templates* no EVA, ver Apêndice B.

Especificamente em orientação a objeto, *framework* é um conjunto de classes com objetivo de reutilização simultânea de um design e de código (JOHNSON, 1997b), provendo um guia para uma solução de arquitetura em um domínio específico de *software*. *Framework* se diferencia de uma simples biblioteca (*toolkit*), pois esta se concentra apenas em oferecer implementação de funcionalidades, sem definir a reutilização de uma solução de arquitetura *(design)*.

Como já comprovado em Fábricas de *software* e grandes empresas de desenvolvimento, ao utilizar-se estes tipos de soluções os ganhos em produtividade e custo são enormes para uma grande parte dos casos. A maioria dos CMS populares estão disponíveis na forma de *software* livre o que praticamente elimina os custos com licença de uso e garantem o acesso ao código-fonte. A

presença de comunidades online crescendo em volta desses sistemas faz com que a mão de obra disponível torne-se cada vez mais abundante.

O Quadro 3 apresenta uma comparação entre as principais soluções CMS disponíveis no mercado. Esta tabela foi gerada a partir do Web site CMS Matrix, que é especialista em comparar características detalhadas sobre diversas soluções CMS. Dentre as características que podem ser comparadas têm-se segurança, suporte, requisitos de sistema, facilidade de uso, desempenho, interoperabilidade, flexibilidade, entre outros. Neste estudo foi selecionado um subconjunto destas características considerado por nós mais relevante para efeito da análise apresentada neste texto. Além das soluções analisadas no site, foi adicionado o EVAcms que é objeto de estudo deste documento.

A seguir segue uma breve descrição sobre cada um dos aplicativos analisados, sendo que o Joomla! já foi analisado no Capítulo 4.3 - O Joomla! e o EVAcms serão melhor detalhados posteriormente:

Joomla! - Segundo o site oficial (http://www.joomla.org) o nome Joomla vem do equivalente fonético da palavra Swahili "Jumla", que significa "todos juntos" ou "como um todo". Joomla! (pronuncia-se djumla) é um CMS (*Content Management System*) desenvolvido a partir do Mambo. É escrito em PHP e roda no servidor web Apache ou IIS e banco de dados MySQL. É um projeto de código aberto (licença GNU/GPL).

Mambo – É um CMS criado pela empresa *Miro International* no início de 2000 (versão 1.0). Seu site oficial é o (http://mambo-foundation.org/). Com um corpo técnico para o desenvolvimento de aplicações baseadas no conjunto AMP (Apache, MySQL e PHP). Depois de algum tempo de desenvolvimento do Mambo, um grupo de desenvolvedores se separou do grupo principal e criaram uma nova ferramenta muito semelhante ao Mambo: o Joomla! que foi citado anteriormente.

Xoops - XOOPS é um Sistema de Gerenciamento de Conteúdo. Pronuncia-se foneticamente como "czups" ou "zoo'ps" e é um acrônimo de "*eXtensible Object Oriented Portal System*". Segundo o site oficial o XOOPS (http://xoops.org/) está sob os termos da GNU - *General Public License* - GPL (Licença Pública Geral).

EVAcms - Um projeto de CMS nacional, 100% Software Livre, o EVAcms encontra-se hospedado na URL (http://www.evacms.com.br/) e é uma plataforma de desenvolvimento de sistemas escrita em PHP. O EVAcms é um programa de código aberto distribuído sob os termos da GNU *General Public License.*.

Product	Joomla! 1.5.9	Mambo 4.6.1	Xoops 2.0.18	EVAcms
Último Update		**01/11/09 12/16/2006**	**08/05/08**	
Requerimentos do sistema	Joomla!	Mambo	Xoops	EVAcms
Custo Aproximado	Gratis	Gratis	Gratis	Free
Banco de Dados	MySQL	MySQL	MySQL 4.23.xx ou later	Any Supported by ADODB
Sistema Operacional	Qualquer	Qualquer	Qualquer	Any
Linguagem de Programação	PHP	PHP	PHP 4.1.0 or later	PHP
Servidor WEB	Apache	Apache, IIS, com qualquer PHP habilitado, mas Apache Recomendado	Apache, IIS	Apache, IIS, any PHP enabled web server
Segurança	Joomla!	Mambo	Xoops	EVAcms
LDAP	Sim	Sim	Sim	Sim
Histórico de Login	Sim	Add On Gratuito	Add On Gratuito	Sim
Suporte	Joomla!	Mambo	Xoops	EVAcms
Manuais Comerciais	Sim	Sim	Sim	Não
Suporte Comercial	Sim	Sim	Sim	Não
Treinamento Comercial	Sim	Sim	Não	Sim
Comunidade de desenvolvimento	Sim	Sim	Sim	Sim
Ajuda on-line	Sim	Sim	Limitado	Não
API Plug and Play	Sim	Sim	Sim	Sim
Facilidade de Uso	Joomla!	Mambo	Xoops	EVAcms
Conteúdo Drag-N-Drop	Não	Não	Add On Gratuito	Sim
URL Amigável	Sim	Sim	Add On Gratuito	Add On Gratuito
Redimensionamento de Imagens	Sim	Não	Add On Gratuito	Sim
Assistente de Instalação	Não		Não	Sim
Linguagem de Templates	Sim	Sim	Sim	Sim
Editor WYSIWYG	Sim	Sim	Add On Gratuito	Sim
Performance	Joomla!	Mambo	Xoops	EVAcms
Replicação de banco de dados	Não	Não	Não	Não
Load Balancing	Sim	Não	Sim	Não
Cache	Sim	Sim	Sim	Sim
Exportar conteúdo estático	Não	Não	Não	Não
Gerenciamento	Joomla!	Mambo	Xoops	EVAcms
Gerenciamento de Anúncios	Sim	Sim	Sim	Add On Gratuito
Agendamento de conteúdo	Sim	Sim	Sim	Sim
Administração on-line	Sim	Sim	Sim	Sim
Sub-sites	Sim	Não	Add On Gratuito	Sim
Temas	Sim	Sim	Sim	Sim
Lixeira	Sim	Sim	Não	Não
Web Statistics	Sim	Sim	Add On Gratuito	Não
Gerenciamento Web de Templates	Sim	Sim	Sim	Sim
Gerenciamento de fluxo	Não	Não	Não	Não

Quadro 3 - Quadro comparativo de soluções CMS

5.2 LMS

A difusão da Educação à Distância (EAD) e das "universidades" corporativas, que utilizam ferramentas de EAD para propiciar educação continuada para seus funcionários, popularizou sistemas especializados no gerenciamento das informações e do processo de aprendizagem por computador, conhecidos com *Learning Management Systems* (LMS).

Enquanto o CMS é centrado no conteúdo, um LMS é centrado em pessoas (*people-oriented* (GRACE; BUTLER, 2005)). Desse modo, um LMS mantém o registro de alunos e professores, os organiza por turmas e disciplinas, gerencia o desenvolvimento do curso, aplicação de avaliação, possui ferramentas de interação como fóruns e mensagens, etc.

Um LMS muito popular e que será tratado em um estudo de caso neste texto é o Moodle – *Modular Object Oriented Distance Learning* (http://moodle.org). O Moodle é um LMS gratuito e *open-source* escrito sobre a linguagem PHP. Seguindo a mesma linha dos sistemas CMS apresentados, o Moodle pode ser estendido pelo uso de componentes. Esta capacidade deu origem a uma vasta comunidade internacional que desenvolve extensões para o Moodle (http://moodle.org/mod/data/view.php?id=6009).

A Figura 18 ilustra uma tela típica de um curso no Moodle. Na lateral esquerda, o Moodle reúne um conjunto de ferramentas que estão à disposição do aluno. Na tela central, que no recorte da Figura 18 aparece à direita, é organizado um índice de recursos que o professor/autor disponibiliza para o aluno.

Figura 18 - Tela típica do sistema Moodle

5.3 LCMS

Grande parte dos recursos colocados no LMS consiste em conteúdo produzido pelo professor, ou selecionado por ele. O gerenciamento de conteúdo é uma especialidade dos CMSs e, na medida em que a necessidade de se produzir e gerenciar conteúdo dentro de sistema aumenta, surge necessidade de uma convergência entre o LMS e o CMS (BERGSTEDT e outros, 2003). Alguns autores chegam a falar em LCMS (BRENNAN; FUNKE; ANDERSON, 2001), uma combinação de LMS com CMS.

Uma das barreiras para esta combinação consiste no fato da falta de componentização no projeto central do sistema, principalmente no que diz respeito à parte visual, ou seja, cada sistema assume a área visual como exclusiva e fica difícil fazer uma combinação entre ambos, dado que cada um deles teria que respeitar uma certa política de "componentização visual".

Esta política é justamente uma das contribuições deste trabalho. Como é apresentado no Capítulo 7.6, Autonomia Visual – Caso Prático. A política visual dos View DCCs será combinada

com uma capacidade singular do EVAcms de tratar cada elemento do sistema como uma entidade visual autônoma. Como resultado, produziremos componentes visuais que podem ser embutidos (*embedded*) dentro de sistemas de terceiros. Mostraremos, como estudo de caso, um exemplo em que o EVAcms será embutido dentro do Moodle.

Conforme mencionado no Capítulo 2.2, *frameworks* PHP têm uma abordagem particular na definição de seus componentes. Usualmente tais *frameworks* utilizam módulos com nome pré-definido ou arquivos em pastas pré-definidas. Certas funcionalidades do componente, tal como a criação de tabelas no banco de dados, são ativadas através do acionamento de *scripts* em pastas específicas. De certa forma, estes aspectos compõem o que chamaremos de interface do módulo, dado que é através deste mecanismo que o *framework* principal interage com ele.

A fim de traçar um perfil do mecanismo de relacionamento entre módulos e *frameworks*, escolhemos um *framework*/sistema CMS e um *framework*/sistema LMS e os confrontamos com o EVAcms.

No Quadro 4 é apresentada uma síntese comparativa de estratégias utilizadas por diferentes sistemas/*frameworks* na definição de seus componentes.

	Joomla!	**Moodle**	**EVA**
Inicialização do módulo	Descrito no arquivo XML de instalação	Realizada pelo *script* pré-definido em Index.php	Realizada pelo *script* pré-definido em Index.php
Criação de tabelas utilizadas pelo módulo	Arquivo SQL descrito no arquivo XML de instalação	Localizado dentro da pasta db um arquivo com o mesmo nome do banco de dados a ser instalado. Por exemplo: mysql.sql	eva_tabelas.php com sintaxe ADODB de criação de tabelas
Outros metadados	O arquivo XML com detalhes é empacotado com um arquivo zip e as demais partes do código		

Quadro 4 - Padrões de interface entre os componentes e o framework

Em todos estes sistemas, tal como acontece na versão do EVAcms antes da incorporação dos DCCs, a relação dos componentes com o sistema através de arquivos e diretórios chave torna a real interface e dependências entre módulo x *framework* obscura. Mais adiante demonstraremos como o uso do modelo de DCCs permitirá explicitar as interfaces, tornando os componentes verdadeiros *black-box*, mais apropriados para o reuso.

6 *FLUID WEB* EM PHP

Todo o conteúdo digital pode ser replicado, adaptado, decomposto, fundido e transformado. Esta visão foi nomeada de Fluid Web (SANTANCHÈ, 2006), cuja perspectiva foi citada anteriormente no Capítulo 3.2. A implementação da infra-estrutura *Fluid Web* no contexto do PHP é derivada da especificação pré-existente em Java. Apesar da implementação original ser em Java, por se tratar de uma especificação do tipo "*lightweight*", ou seja, que define um padrão a ser seguido para a construção dos componentes sem estabelecer um formato binário rigoroso, é possível adaptar o modelo para qualquer outra linguagem de programação. A versão PHP sofreu adaptações para que se tornasse compatível com a linguagem e com a cultura de desenvolvimento PHP. A maioria dos esforços foram concentrados na diferença entre as sintaxes das linguagens. Dessa forma, poucas alterações em termos de lógica foram necessárias para a transformação, já que a estrutura de orientação a objetos do PHP, a partir da versão 5, foi inspirada nos padrões Java e, portanto, somente a partir desta versão os conceitos de interface e classe abstrata foram incorporados.

Os conceitos de interface e classe abstrata se mostraram fundamentais para garantir que a versão do modelo de DCC implementada em PHP siga alguns fundamentos do modelo DCC, por exemplo, manter a consistência das conexões em tempo de compilação e execução através de interfaces requiridas e interfaces providas.

Neste projeto optamos por portar um subconjunto da infra-estrutura *runtime* que serve aos propósitos desta pesquisa. Especificamente, portamos o Contexto 1 da infra-estrutura, conforme descrito no Capítulo 3.3 , Arquitetura Geral de Runtime dos DCCs, como também a estrutura de apresentação dos VDCCs, conforme descrito no Capítulo 3.6, View DCC, os demais Contextos 2 e 3 não serão tratados neste momento. Como será apresentado adiante, por ser uma especificação em produção, a plataforma EVAcms contribuiu significativamente em uma concepção de política visual compatível com linguagens Web e especialmente o PHP.

A transcrição de contextos, Java para PHP, tiveram algumas perdas relativas a estrutura do componente. Mas versão do modelo de DCC em PHP é bem similar a versão já existente em Java, com a exceção do conceito de *annotation*, que ainda não existe nativamente no PHP. Os demais conceitos e estruturas necessários para implementar o Contexto 1 do modelo proposto, conforme apresentado no Capítulo 3.3 , Arquitetura Geral de Runtime dos DCCs, deste documento, puderam

ser desenvolvidos utilizando apenas core nativo do PHP versão 5.

A Figura 19 mostra um diagrama ilustrando as diferenças entre a arquitetura de DCC inicial e a arquitetura atualmente adaptada para o PHP. Nesta proposta de adaptação inicial, alguns pacotes foram removidos de sua estrutura básica para simplificação da solução: os pacotes *annotation*, *connector* e *message* foram retirados e serão reincorporados em pesquisas futuras.

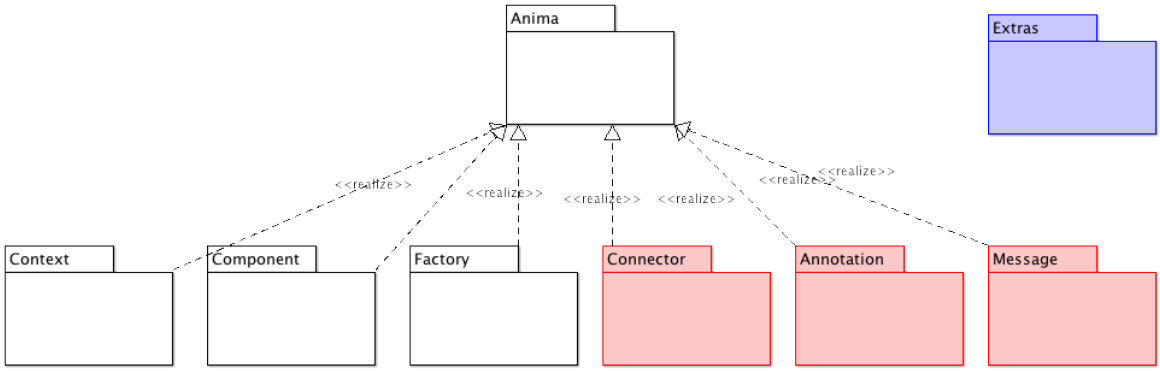

Figura 19 - Reorganização dos pacotes dos DCCs em PHP

Com o objetivo de manter a compatibilidade com o Java houve a necessidade de criar uma nova classe *Hashtable,* dado que não foi encontrada uma função nativa e equivalente em PHP 5. Porém já prevendo o aumento gradativo do objeto de estudo para a implementação dos demais contextos, inevitavelmente surgirão novas classes de compatibilização ou extensão, que serão incorporadas a este novo grupo de classes. Para manter a integridade e organização do projeto inicial, criou-se um pacote que reunirá estas novas classes não nativas. Este pacote tem o nome de "Extras".

A seguir detalharemos a versão DCC em PHP, os diferenciais e os resultados obtidos.

6.1 ESTRUTURA DO COMPONENTE

6.1.1 Interfaces Padrão

Basicamente a diferença das interfaces da versão Java para a versão PHP é a limitação do PHP de não suportar que seus métodos especifiquem valor de retorno, nem mesmo *null*, como é feito no Java. O PHP também limita-se em permitir apenas especificar tipos classe como parâmetro dos seus métodos, por exemplo: podemos especificar que um método espera como parâmetro um objeto de uma classe específica, mas não podemos especificar que esperamos uma *string* ou um inteiro como parâmetro, para exemplificar esta limitação um trecho de código PHP válido seria:

```
public function my_method(MyClass $var);
```

e um trecho inválido:

```
public function my_method(String $var);
```

No Quadro 5 podemos comparar a diferença da declaração final das versões Java e PHP da interface ISupports:

Java	PHP
`ISupports queryInterface(String interfaceId);`	`public function queryInterface($interfaceId);`
`public String getPrimaryKey();`	`public function getPrimaryKey();`

Quadro 5 - Tabela comparativa das versões Java e PHP da interface ISupports

6.1.2 Componente Base

Na classe `ComponentBase`, que implementa a interface `ISupports`, algumas modificações e cortes no código foram necessários. Abaixo segue um detalhamento de seus

métodos e diferenças:

1. Método construtor da classe onde o corpo deste método é vazio nas duas versões:

- **Java:** `public ComponentBase()`
- **PHP:** `public function ComponentBase()`

2. Método utilizado para retornar o valor da propriedade privada `primaryKey` declarada em Java como "`private String primaryKey`" e em PHP como "`private $primaryKey`"

- **Java:** `public String getPrimaryKey()`
- **PHP:** `public function getPrimaryKey()`

3. Este método atribui à propriedade `primaryKey` o valor do parâmetro com o mesmo nome, `primaryKey`. Caso o parâmetro seja nulo será gerado um identificador único através do método `generatePrimaryKey()`.

- **Java:** `public void setPrimaryKey(String primaryKey)`
- **PHP:** `public function setPrimaryKey($primaryKey)`

4. Neste método o Java usa a classe UUID para gerar este identificador único armazenado na propriedade privada `primaryKey`, ficando como resultado final de código: (`return UUID.randomUUID().toString();`). Já na versão do PHP é utilizado uma função nativa que tem a mesma finalidade: (`return uniqid();`).

- **Java:** `public static String generatePrimaryKey()`
- **PHP:** `public static function generatePrimaryKey()`

5. Este método retorna uma interface específica identificada pelo id.

- **Java:** `public ISupports queryInterface(String interfaceId)`
- **PHP**: `public function queryInterface($interfaceId)`

6.1.3 Identificação dos DCCs

A identificação dos componentes no modelo DCC, como já descrito anteriormente na versão

Java, utiliza a funcionalidade *annotation*. Como no PHP ainda não existe tal funcionalidade, alternativamente os valores necessários para a identificação do componente foram transferidos para propriedades da classe.

No exemplo FischComponent do Java a identificação se dava da seguinte forma:

```
@Component( id="http://purl.org/NET/dcc/examples.fish.s01.Fish",
            provides={"http://purl.org/NET/dcc/examples.fish.s01.IFish"})
```

Onde `id="http://purl.org/NET/dcc/examples.fish.s01.Fish"` é a identificação do componente e `provides={"http://purl.org/NET/dcc/examples.fish.s01.IFish"}` é a identificação das interfaces providas. Na versão PHP, após a adaptação dos identificadores, obtivemos o seguinte trecho de código:

```
const id = "http://purl.org/NET/dcc/examples.fish.s01.Fish";
public static $provides=array("http://purl.org/NET/dcc/examples.fish.s01.IFish");
```

Ambas as linhas são propriedades da classe Fish.

6.1.4 Fábrica de DCCs

A implementação da fábrica de DCCs é uma das classes onde foram necessárias algumas das mais importantes adaptações, para que a utilização das funcionalidades no PHP continuem similares com as já feitas em Java. Mais especificamente o método que mais se destacou foi o `createInstance()`. Este método é descrito na Interface `IglobalFactory` como um método que sofre 3 sobrecargas, exatamente como listado abaixo:

```
public ISupports createInstance(String classId, String primaryKey);
public ISupports createInstance(String classId);
public ISupports createInstance(String classId, String primaryKey,
                                String interfaceId);
```

Apesar da inspiração de suas características de Orientação a Objetos do PHP virem do Java, até a versão atual (PHP 5), este ainda não suporta sobrecarga da forma que foi implementada na

interface `IGlobalFactory`.

Para solucionar esta necessidade, foi criada uma simulação de sobrecarga utilizando a função `func_get_args` (Figura 20), que retorna um vetor contendo uma lista com os argumentos da função ou método. A partir daí, é possível saber quantos argumentos foram passados na chamada do método e direcioná-lo para respectivo trecho de código.

```
public function createInstance()
{
    $args = func_get_args();
    if(count($args) == 3){
        return $this->componentFactory->createInstance($args[0], $args[1], $args[2]);
    }else if(count($args) == 2){
        return $this->componentFactory->createInstance($args[0], $args[1]);
    }else{
        return $this->componentFactory->createInstance($args[0]);
    }
}
```

Figura 20 - Trecho de código da fábrica que simula sobrecarga

6.2 COMPATIBILIDADE *LIGHTWEIGHT*

Conforme abordado anteriormente, o modelo *runtime* dos DCCs pode ser considerado como um modelo *lightweight*, no sentido que ele não exige compatibilidade binária com outras linguagens, tal como faz o COM (MICROSOFT, 2009a) e o XPCOM (PARRISH, 2001). Desse modo, nos Contextos 1 e 2 o modelo DCC operam mais no sentido de recomendar um padrão de programação, enquanto no Contexto 3 é possível interligar componentes de diferentes linguagens. A fim de ilustrar como o padrão PHP se alinha com a referência em Java, a seguir retomamos o estudo de caso do componente Fish, apresentado no Capítulo 3.4.1, no qual faremos um paralelo entre seu uso em Java e PHP.

6.2.1 Usando um DCC

Nos sub-tópicos a seguir apresentaremos um programa simples que instancia dois DCCs (Fish e Aquarium) e os conecta para que funcionem em conjunto.

Este programa foi batizado de FishTank02, e até o momento existem versões Java e PHP deste programa.

6.2.2 Criando a Fábrica Global

A única diferença de como estas 2 linguagens criam suas fábricas é a maneira como os métodos estáticos são acessados.

- **Java:** `IGlobalFactory factory = ComponentContextFactory.createGlobalFactory();`
- **PHP:** `$factory = ComponentContextFactory::createGlobalFactory();`

6.2.3 Registrando Protótipos

Nesta fase do desenvolvimento é necessária mais uma adaptação no código para reduzir as diferenças entre a linguagem original e a nova versão. Por exemplo, para registrar os dois componentes em Java foram utilizadas as seguintes linhas de código.

- `factory.registerPrototype(Fish.class);`
- `factory.registerPrototype(Aquarium.class);`

Onde a possibilidade de se informar diretamente a classe (e.g, Fish.class) para se extrair as anotações é uma particularidade do Java. Em PHP, a adaptação ficou como abaixo.

- `$factory->registerPrototype('Fish');`
- `$factory->registerPrototype('Aquarium');`

Por não existir método equivalente em PHP, foram utilizados strings contendo o nome da classe a ser registrada para uso.

6.2.4 Criando Instâncias

As instâncias dos componentes são feitas através do método `createInstance` (Figura 21). Como parâmetros são fornecidos o identificador do componente e a interface requerida.

O exemplo da Figura 21 ilustra o FishTank02 implementado em PHP:

```
$componentFish = $factory->createInstance(
          "http://purl.org/NET/dcc/examples.fish.s01.Fish",
          "http://purl.org/NET/dcc/examples.fish.s01.IFish");

$componentAquarium = $factory->createInstance(
           "http://purl.org/NET/dcc/examples.fish.s01.Aquarium",
           "http://purl.org/NET/dcc/examples.fish.s02.IAquariumRequired");
```

Figura 21 - Instanciando componentes no exemplo FishTank02.

6.2.5 Conectando os Componentes

A conexão dos componentes é relativamente simples; ela é feita a partir de uma chamada ao método *connect* do componente que implementa a interface requerida, passando como parâmetro o componente que implementa a interface provida. A Figura 22 e a Figura 23 apresentam uma implementação para o método *connect* em PHP e Java respectivamente. Como pode ser observado, para manter a conexão, a classe que implementa o método *connect* guarda a referência para o objeto que define a interface provida. Neste exemplo especificamente, para fins de simplificação, após ter sido feita a conexão o componente inicia o processo de desenho.

```
public function connect(IFish $theFish)
{
    $this->theFish = $theFish;
    $this->drawAquarium();
}
```

Figura 22 - Método *Connect* versão PHP

```
public void connect(IFish theFish)
{
    this.theFish = theFish;
    drawAquarium();
}
```

Figura 23 - Método *Connect* versão Java

6.3 DCCS BASE PARA CMS

Neste Capítulo foi detalhada a adaptação do modelo DCC para o contexto PHP. Ela serviu como base para o trabalho que será apresentado nos próximos dois Capítulos que consiste em usar este modelo de componentes em um *framework* CMS, o EVAcms.

7 EVACMS

O nome EVA é a mistura das iniciais da palavra "*Earned Value*", que significa valor agregado, e o nome Eva que representa a mulher moldada pelo homem. Este nome tem o objetivo de unir o conceito de se poder construir qualquer "molde" de aplicação e agregar valor fazendo uma alusão ao reaproveitamento de código.

EVAcms (EVACMS, 2009) foi criado com o objetivo de atender uma demanda interna da EMBASA (Empresa Baiana de Águas e Saneamento) para desenvolvimento do portal corporativo dessa empresa. Posteriormente tornou-se *open-source*, tendo sido adotado por uma comunidade mais ampla de colaboradores como: Roberto Rander, Alex Campos, Getúlio Júnior, etc. A partir daí, as aplicações do EVAcms começaram a se expandir para outros nichos.

A plataforma EVA é escrita em PHP e seus módulos iniciais eram totalmente em português, no entanto, atualmente já existem traduções destes módulos para o inglês. Para a criação deste sistema diversos outros aplicativos similares foram estudados e analisados. Como exemplos de aplicativos utilizados como base para o desenvolvimento inicial do EVAcms podemos citar: Mambo (MAMBO, 2009), Postnuke (POSTNUKE, 2009), PHP-Nuke (PHP-NUKE, 2009), eGroupware (EGROUPWARE, 2009) e Xoops (XOOPS, 2009). Com a exceção do eGroupware os demais são *frameworks* CMS com finalidades semelhantes a do EVAcms. O eGroupware é um *software* de colaboração gratuito. Com ele é possível gerenciar contatos, compromissos, e outras tarefas corporativas. Tais sistemas foram exaustivamente instalados, desinstalados e utilizados. Destas ferramentas foram levantadas pontos positivos e negativos, assim como agregação de funcionalidades para atender a demanda da época.

Durante estes anos de existência diversos portais e *websites* foram desenvolvidos usando o EVAcms como *template* e como um CMS propriamente dito. Abaixo segue uma lista de alguns sites desenvolvidos usando o EVA, versão atualizada disponível em (EHVOLUA!, 2009):

1.	www.evacms.com.br	2.	www.cursoparaconcursos.com.br
3.	www.simuladosnaweb.com.br	4.	www.fabac.com.br
5.	www.portalesportivo.com.br	6.	www.joguelimpo.org.br
7.	www.coloniamundodalua.com.br	8.	www.escolagenesis.com
9.	www.erleiloes.com.br	10.	www.bandacheirodeamor.com.br
11.	www.infolex.com.br	12.	www.sindicerba.com.br
13.	www.angelinfo.com.br/os	14.	www.prosabersaude.com.br
15.	www.urivaladao.com	16.	www.redes.unifacs.br
17.	www.danielneto.com	18.	www.saude.ba.gov.br/SaudeBahia
19.	www.paypermind.com	20.	www.fezta.com.br
21.	www.samhop.com.br	22.	www.bolananet.com.br
23.	www.chroma.com.br	24.	www.zevende.com.br
25.	www.alugarcasaguarajuba.com.br		

Quadro 6 - Sites desenvolvidos usando o EVA

Veja o Apêndice C para detalhes de instalação desta ferramenta.

A editora Digerati publica mensalmente uma revista chamada de "*One Click*". Trata-se de uma revista de mercado especializada em desenvolvimento Web e Web design. Nas principais edições, acompanha a revista um CD com uma seleção de ferramentas julgadas interessantes pelos editores para o contexto. Na revista número 60 o CD distribuído veio com uma seção especial para CMS, no qual foram selecionadas quatro ferramentas/*frameworks* CMS: o Joomla!, o DragonFly, o Pure SEO CMS e o EVAcms. Esta distribuição voluntária comprova da aceitação deste sistema no mercado.

7.1 ESTUDO DE CASO

A Figura 24 apresenta uma tela típica de um site gerenciado pelo sistema EVAcms, desenvolvido por nós para o curso de redes da Unifacs. Utilizaremos este site como estudo de caso para apresentar as funcionalidades do EVAcms.

Figura 24 - Tela típica de um sistema EVAcms

É importante compreender que como um CMS, o EVA não se limita a armazenar e apresentar páginas estáticas ou dinâmicas. Ele constitui uma solução completa na qual o gerenciamento, configuração e extensão do site são feitos através da Web.

7.2 ARQUITETURA EVACMS SEM DCCS

O EVAcms é um sistema modular cujos módulos podem ser comparados com componentes, conforme explanação feita no Capítulo 2 Componentes.

Os módulos estendem as potencialidades do EVAcms dando-lhe novas funcionalidades. O próprio usuário final, não especialista em desenvolvimento de software, pode criar módulos em EVAcms usando o criador de módulos, que apresentam uma interface Web visual para este processo de criação, veja o Apêndice D para detalhes de criação de novos módulos.

A Figura 26 ilustra a arquitetura geral do EVA antes da sua combinação com os DCCs. As

características de CMS do EVA foram adquiridas através de um conjunto de módulos desenvolvidos no início, que atualmente constituem o seu núcleo principal. Como está ilustrado na Figura 26, os módulos que compõem o núcleo são:

Gerenciamento de usuários – Este módulo é responsável pelo gerenciamento dos usuários e grupos de usuários do EVA. O *login* deve ser feito por este módulo, fisicamente este módulo se encontra na pasta /modulos/eva_usuario/.

Gestão de conteúdo – Este módulo é o principal responsável pela característica de CMS do EVA. É através deste módulo que são criadas as seções de notícias e textos. Assim, esse módulo determina que grupo de usuários pode ver, editar, apagar ou inserir novos textos. Estas permissões podem ser mescladas em um único grupo de usuários através do módulo de gerenciamento de usuários, fisicamente este módulo se encontra na pasta /modulos/eva_conteudo/.

Gestão de blocos – Um bloco para o EVA é uma especialização de um módulo com a flexibilidade de poder ser apresentado em qualquer parte da interface, por exemplo: sabemos que o módulo de gestão de conteúdo se encarrega de alimentar a base de dados com notícias ou textos. Caso o site necessitasse exibir, em qualquer parte do site, as três ultimas notícias de uma determinada seção de conteúdo, criaríamos um bloco no módulo de conteúdo que atendesse esta demanda e o gerenciador de blocos posicionaria este novo bloco na interface do site. No bloco é possível controlar que grupo de usuários tem privilégios de acesso Figura 27. Fisicamente este módulo se encontra na pasta /modulos/eva_bloco/

Gestão de múltiplos sites – O EVA permite a criação de múltiplos sites com temas diferentes em cima de uma mesma instalação. Isto permite que sites diferentes compartilhem conteúdo, usuários e todos os subsistemas de um modo geral. Como exemplo da aplicação prática desta funcionalidade, podemos citar dois sites da UNIFACS que realizam este compartilhamento: o site do curso de redes de computadores (http://www.redes.unifacs.br) e o site de certificações também da UNIFACS (http://www.certificacoes.unifacs.br/). Fisicamente este módulo se encontra na pasta /modulos/eva_site/.

Configuração – Exibe informações da instalação do EVA e do PHP, assim como permite fazer uma reinstalação corretiva do sistema e permite consultar o *log* de *queries* executadas. Fisicamente este módulo se encontra na pasta /modulos/eva_config/

Gerenciamento de menu – Gerencia os menus e as permissões dos itens do menu através de uma interface intuitiva de criação de menus com níveis e subníveis, como visto na Figura 25. Fisicamente este módulo se encontra na pasta /modulos/eva_menu/

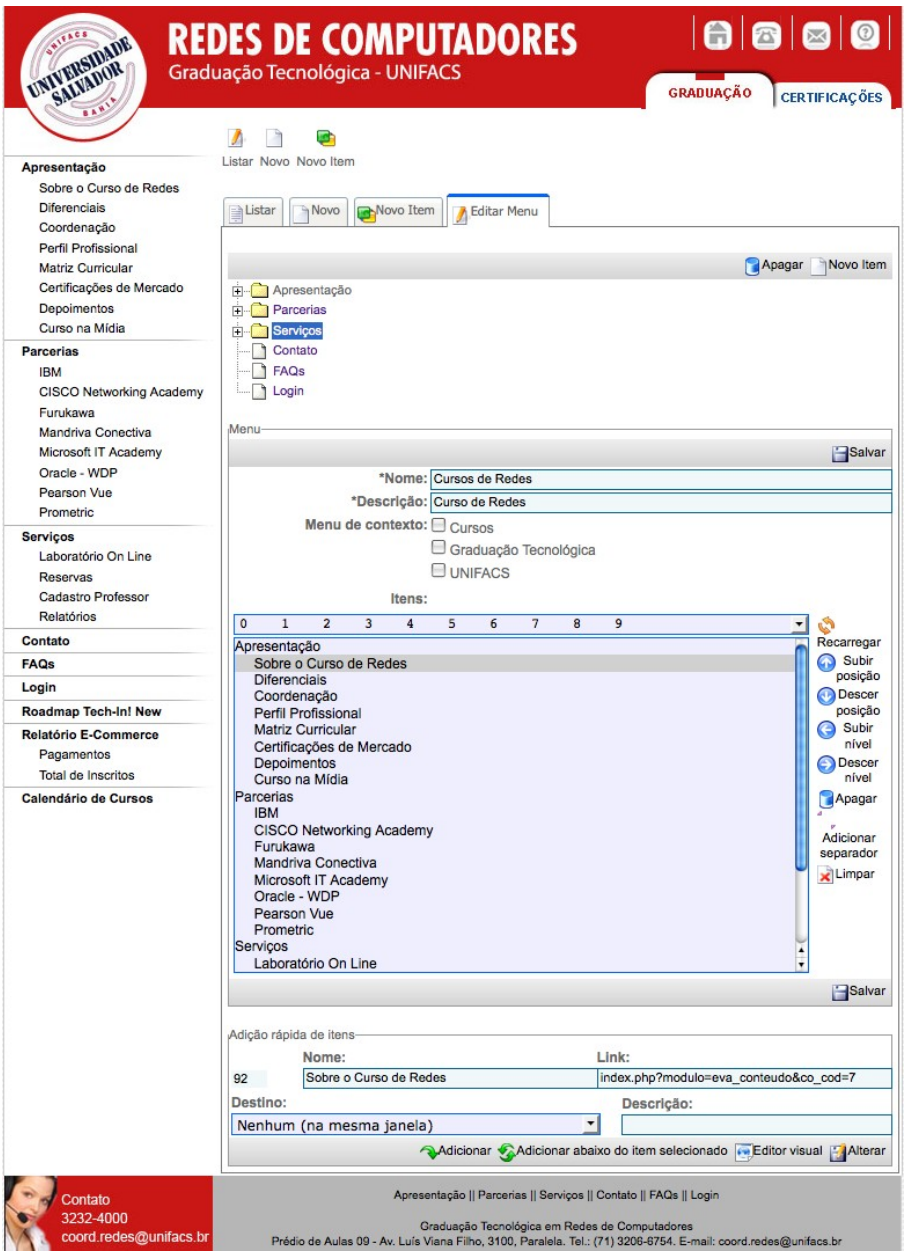

Figura 25 - Módulo de Menu

Todos estes módulos têm sofrido upgrades, mas até os dias de hoje continuam fazendo parte do núcleo básico do EVAcms. Como pode ser visto na Figura 26, estes módulos são desenvolvidos sobre um *framework* também chamado de EVA ou EVA *Framework*. A arquitetura EVA está organizada de tal maneira que ele possa ser utilizado de duas formas: como um CMS customizável e pronto para uso, ou como um *framework* de suporte ao desenvolvimento dos mais diversos sistemas e aplicações Web.

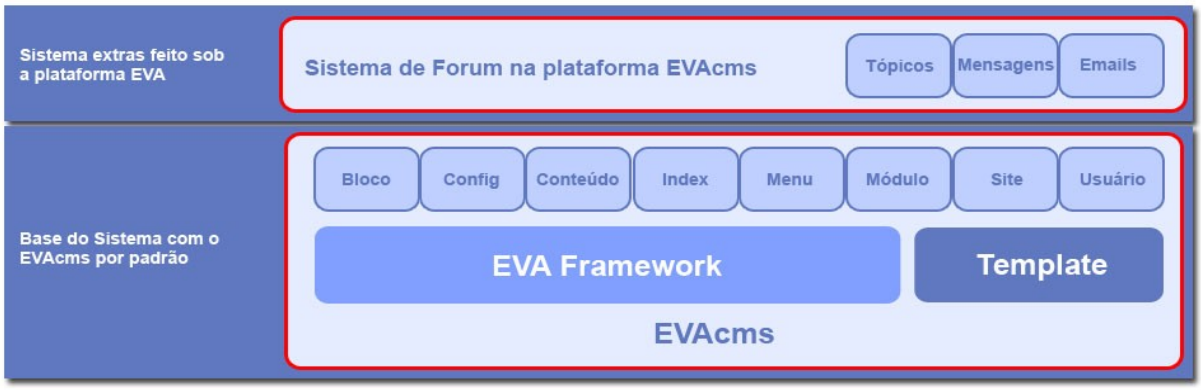

Figura 26 - Arquitetura do EVA

Conforme está ilustrado na Figura 26, a base do sistema EVA está subdividida em EVA *Framework*, o seu sistema de *Templates* e os módulos padrões acoplados a este *Framework*. Qualquer outro módulo para o EVA deve ser desenvolvido em cima desta base que chamaremos de Núcleo do EVAcms. O sistema de *Templates* consiste em imagens e arquivos HTML específicos para apresentação do EVA com seus módulos e blocos. Toda a aparência do EVA pode ser alterada através dos *templates*, também conhecidos como temas. Para se criar um novo tema para o EVA existem alguns critérios a serem seguidos como: nomes de arquivos e pastas; todos os novos temas devem ficar fisicamente dentro da pasta /arquivos/temas/; dentro desta pasta devem existir no mínimo os arquivos index.htm ,que é o arquivo principal do tema, um ou mais arquivos de folha de estilo em cascata na pasta ./css/ e a pasta imagens (onde as imagens do *layout* são salvas). O EVA tem seus próprios estilos que são necessários para que o sistema funcione corretamente e estes estilos devem ser incluídos na página principal do novo tema.

No arquivo mais importante do sistema de *templates*, o index.htm, é necessário que existam no mínimo 5 *tags* básicas: {topo}, {direita}, {centro}, {esquerda} e {rodapé}, todas embutidas no HTML. Um detalhe importante é que os módulos sempre aparecem na *tag* {centro} e os blocos aparecem na *tag* correspondente ao especificado pelo gerenciador de blocos (Figura 27).

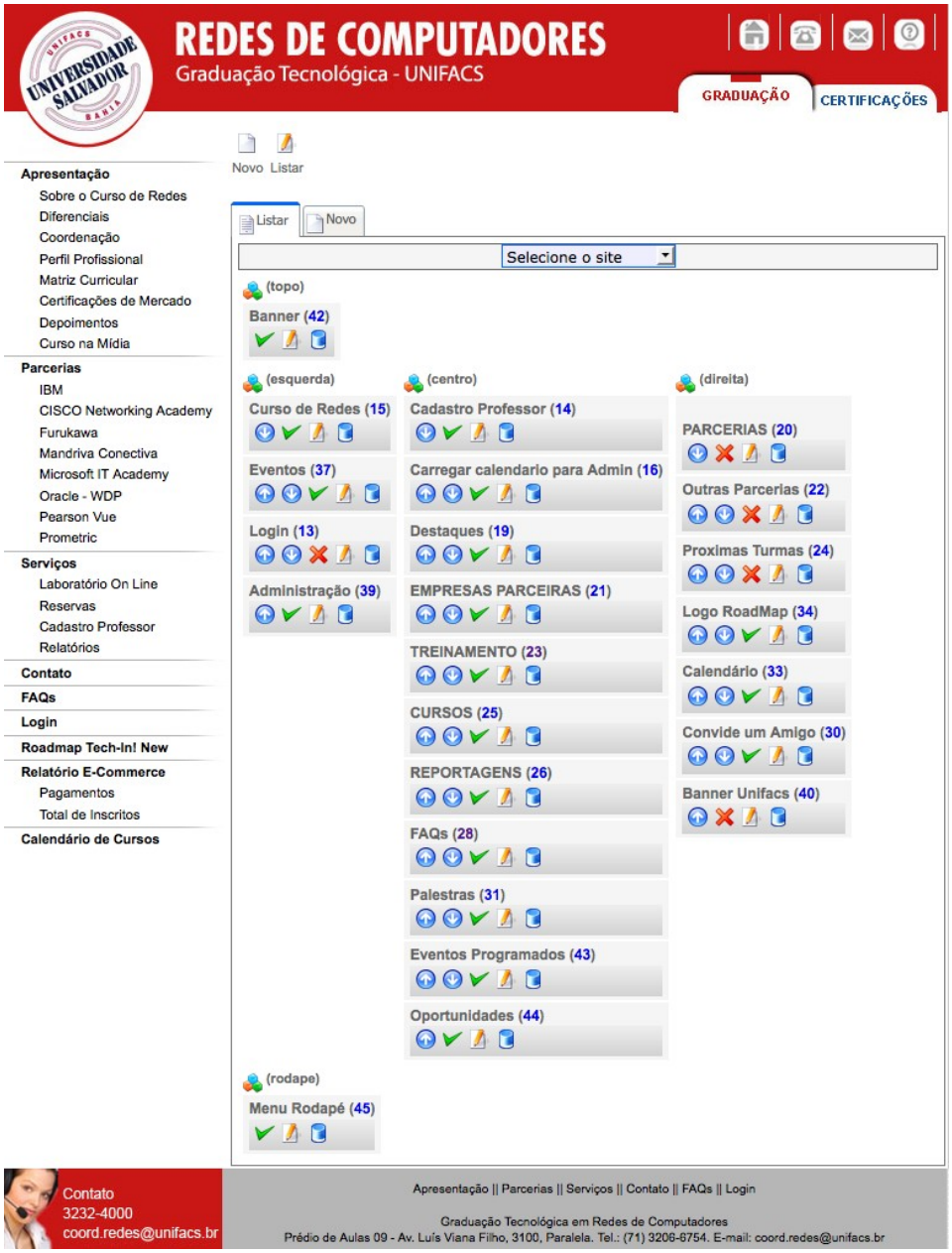

Figura 27 - Gerenciador de Blocos

Como observamos acima o EVAcms nada mais é do que o conjunto de módulos padrão, necessários para o funcionamento do sistema inicial.

7.3 ESTENDENDO O EVACMS ATRAVÉS DE MÓDULOS E BLOCOS

Como será demonstrado aqui, a arquitetura do EVA foi projetada para operar de forma completamente modular, não apenas do ponto de vista de construção e extensão do sistema, como

também nas atividades visuais. Isto torna o EVA um candidato para se combinar com o modelo de componentes DCC.

O EVA define duas principais entidades modulares de construção e extensão: Blocos e Módulos. Os **Blocos** são entidades mais genéricas, que podem ter qualquer funcionalidade e têm liberdade de apresentação em qualquer parte da tela. A Figura 28, por exemplo, delimita áreas que são operadas por blocos dentro do sistema maior. Nesta figura é possível observar blocos responsáveis pelo *banner*, menu, destaques, etc. Cada um é responsável por uma área visual delimitada que pode operar de forma autônoma. Esta autonomia é um diferencial do EVAcms que detalharemos a seguir.

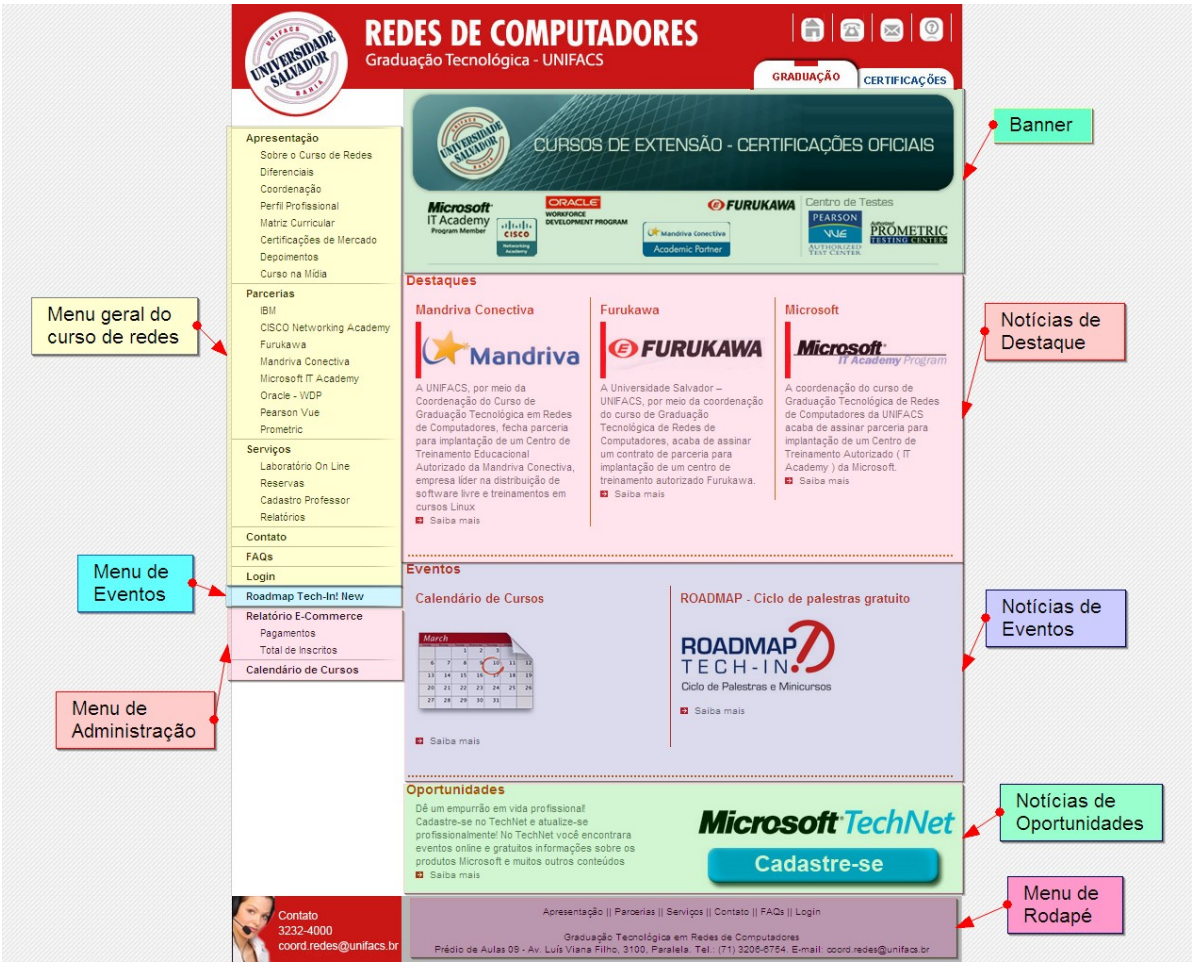

Figura 28 - Blocos em um site EVAcms

Os **Módulos** são entidades que possuem uma funcionalidade especializada. Eles usualmente são responsáveis por operações que envolvem inserção, atualização e remoção de dados no sistema. A Figura 29 ilustra um módulo em operação dentro do sistema EVA responsável pelo gerenciamento de parceiros do curso de redes de computadores da universidade.

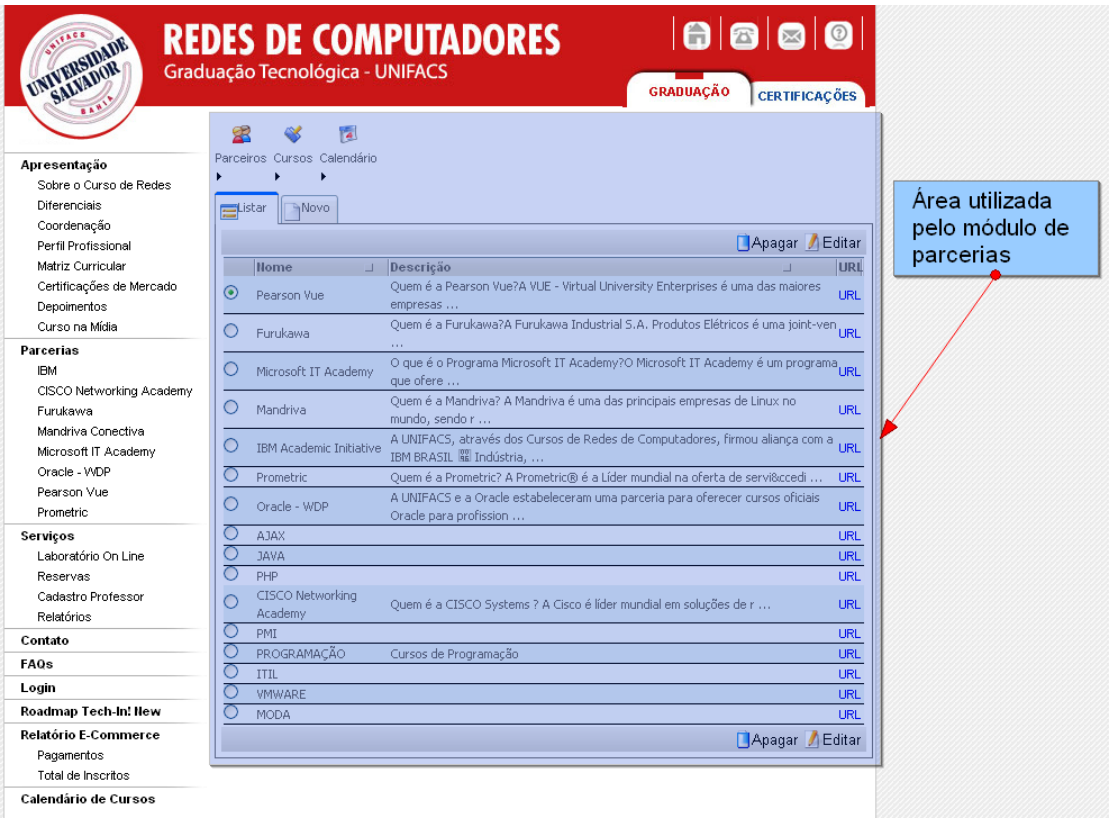

Figura 29 - Módulo no EVAcms

Ao contrário dos Blocos, os Módulos usualmente ficam restritos a uma área específica. O padrão é que eles sejam apresentados na área determinada pela *tag* {centro} no tema escolhido, tal como aparece na Figura 29.

Seguindo a tendência de *frameworks* PHP, a criação de Blocos e Módulos em EVA devem seguir um modelo, conforme está ilustrado na Figura 30. Neste exemplo, os arquivos que iniciam com ecommerce_cupom são referentes a uma tabela com o mesmo nome. Caso o módulo não interaja com nenhuma tabela, estes arquivos não existirão. Existem diretórios e arquivos que têm funcionalidades específicas determinadas pelo seu nome, conforme o detalhamento a seguir:

a) *controller* – pasta onde serão armazenadas todas as classes responsáveis pelo controle seguindo o contexto MVC.

b) idiomas – Pasta onde são armazenados os arquivos responsáveis pelas traduções do sistema para diversos idiomas.

c) *model* - pasta onde serão armazenados todas as classes responsáveis pelo modelo seguindo o contexto MVC.

d) *view* - pasta onde serão armazenadas todas as classes responsáveis pela interface seguindo o contexto MVC.

e) ecommerce_cupom.php – Gerenciar as funcionalidades do EVA na tabela ecommerce_cupom

f) ecommerce_cupom_cadastrar.php – Cadastrar dados na tabela ecommerce_cupom

g) ecommerce_cupom_cadastrar_editar.php – Editar dados da ecommerce_cupom

h) ecommerce_cupom_listar.php – Listar os dados da tabela ecommerce_cupom

i) eva_blocos.php – Arquivo que contém as funcionalidades dos módulos

j) eva_classe.php – Classe genérica do módulo que é sempre carregada quando o módulo é solicitado, mesmo que solicitado dentro de um outro módulo.

k) eva_comandos.php – Arquivo de comandos do EVA, geralmente usado para requisições AJAX e outras requisições que não precisem de interface com o tema do site, como download de arquivos, gerar objetos JSON, etc.

l) eva_config.php – Arquivo onde são armazenadas informações do módulo, como os blocos disponíveis neste módulo, versão, nome, etc.

m) eva_menu.php – Neste arquivo existe um vetor com o menu padrão do módulo, este menu é visualizado somente pelo administrador do site.

n) eva_tabelas.php – Aqui existe um vetor que posteriormente será transformado para a sintaxe ADODB de criação de tabelas.

o) index.php – Arquivo principal do módulo a partir da qual todas as chamadas para suas funções passam por ele.

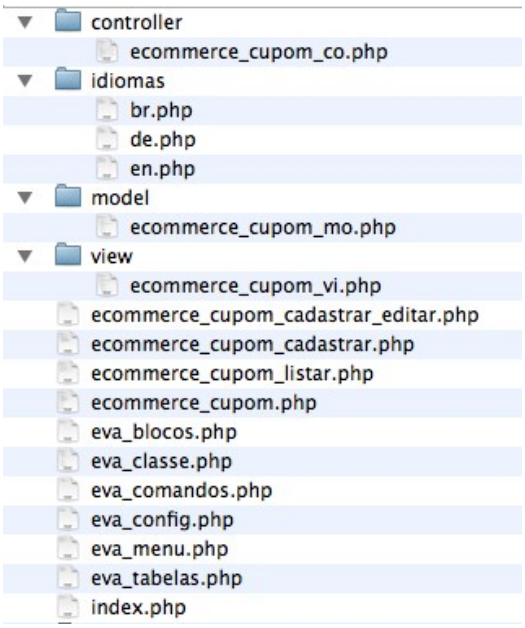

Figura 30 - Disposição das partes de um módulo MVC no EVA

7.4 MÓDULOS PADRÃO DO EVACMS.

Ao fazer o download do EVAcms o *Framework* já vem com alguns módulos configurados. Estes são chamados de módulos padrão, ou seja, módulos essenciais para o funcionamento do sistema e não devem ser desinstalados ou removidos, a menos que o usuário ou programador saiba exatamente o que está fazendo.

Estes módulos são:

a) Eva_bloco: Administração de blocos;
b) Eva_config: Instalação e configuração do EVAcms;
c) Eva_conteudo: Módulos para cadastro de Notícias e Conteúdo;
d) Eva_index: Módulo para configuração da Página de Inicial;
e) Eva_menu: Módulos para configuração e administração de Menus;
f) Eva_modulo: Sistema de Gerenciamento de Módulos do EVA;
g) Eva_site: Administração de Sites;
h) Eva_usuario: Gerenciamento de Usuários e Grupos de Acesso;

7.5 AUTONOMIA VISUAL DOS COMPONENTES

Conforme anunciado anteriormente, os componentes (Módulos e Blocos) do EVA possuem uma política de apresentação visual na Web, que é um diferencial em relação a outros *frameworks* CMS. A questão fundamental que *frameworks* CMS enfrentam é quando se faz necessário combiná-los com outros *frameworks*. Esta não é uma situação incomum. Por exemplo, conforme mencionado no Capítulo 5.3, sistemas CMS podem ser combinados com sistemas LMS. O problema é que sistemas CMS são usualmente projetados para tratar a tela do navegador como se fosse de exclusividade dele. Desse modo, torna-se difícil combinar sistemas.

O EVAcms, por outro lado, tem uma característica singular de autonomia dos componentes (Módulos e Blocos) na construção da sua interface. Como está ilustrado na Figura 28, na base do sistema existe o elemento fundamental: Bloco. Para o EVA todos as aparições visuais de seus componentes são feitas através de Blocos e são gerenciados pelo módulo eva_bloco (veja Figura 28). Até os Módulos, apresentados anteriormente, têm sua apresentação visual relacionada com um tipo especial de Bloco.

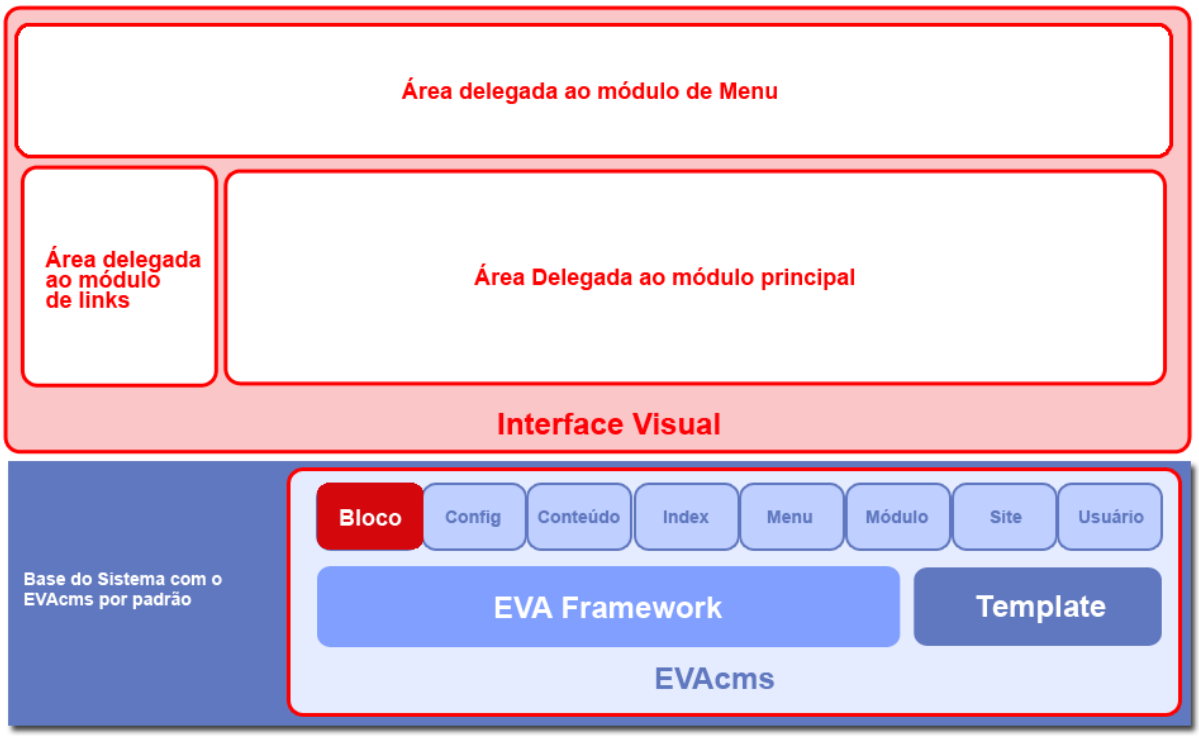

Figura 31 - Gerenciamento das áreas da interface de um Website feita pelo módulo Bloco

Retomando os Blocos apresentados na Figura 28, considere que se deseja utilizar individualmente o Bloco de "Menu Geral do Curso de Redes". O módulo pode ser acionado individualmente, diretamente do navegador, e gerará o resultado apresentado na Figura 32. Nesse caso, o módulo gera um resultado preparado para ser embutido em um ambiente terceiro. Como pode ser observado na figura, o módulo retorna inclusive sem estilos, de modo que possa se adaptar aos estilos do ambiente hospedeiro.

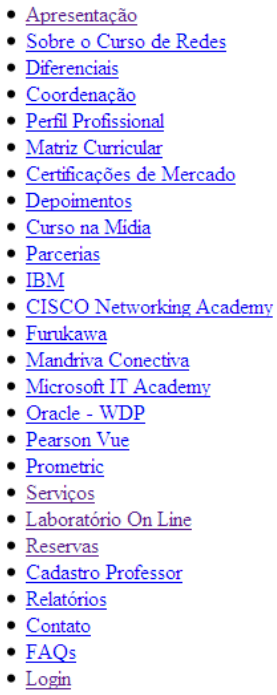

Figura 32 - Bloco de menu acionado individualmente

Os Módulos também atuam da mesma maneira. A Figura 33 ilustra o Módulo da Figura 29, desta vez acionado para se apresentar individualmente dentro de um ambiente terceiro.

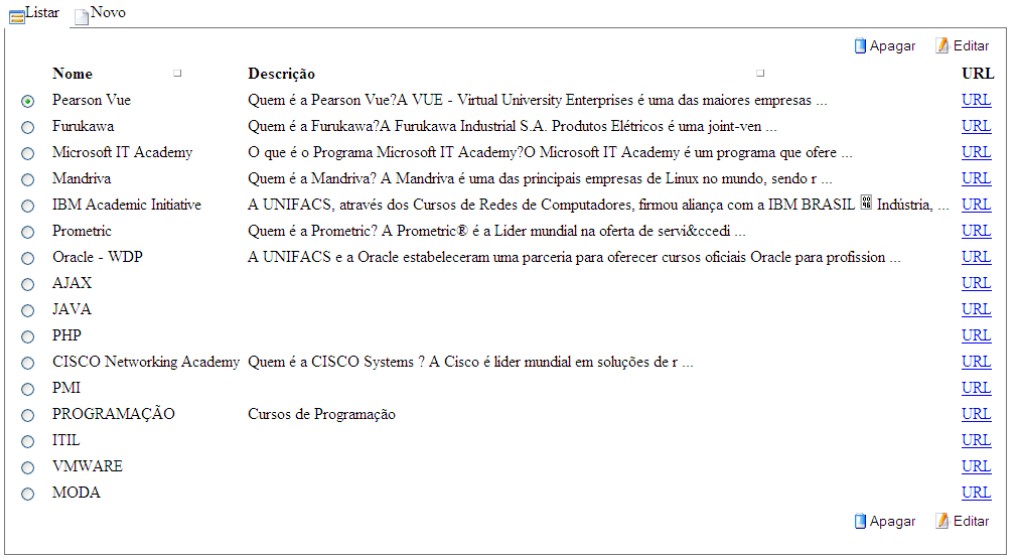

Figura 33 - Módulo de Gerência de Parceiros acionado individualmente

7.6 AUTONOMIA VISUAL – CASO PRÁTICO

A autonomia visual é também essencial do ponto de vista do reuso de Blocos e Módulos. Considere-se o seguinte caso prático: em novembro de 2008 aconteceu na UNIFACS o ESELAW 2008 (*5th Experimental Software Engineering)* traduzido para o português como Quinto Fórum de Engenharia de Software Experimental. Este evento é reconhecido mundialmente e apoiado por renomadas entidades como: *Institute of Electrical and Electronic Engineers* (IEEE), Sociedade Brasileira de Computação (SBC), Fundação de Amparo de Amparo à Pesquisa do Estado da Bahia (FAPESB), entre outros.

Na época da concepção do evento ESELAW 2008 foi detectada a necessidade de um sistema para o *site* do evento que suportasse o cadastro de palestras e cursos, administrasse e controlasse as vagas e pagamentos neste evento. Durante o levantamento destas necessidades, detectou-se que um sistema pré-existente – o *website* dos cursos de certificação da UNIFACS (vide Figura 34) – já possuía tal sistema com estas características, e este sistema foi desenvolvido utilizando a plataforma do EVA *Framework.* O Módulo a ser reusado – responsável pelo *e-commerce* dos cursos de certificação – era capaz de gerenciar sua própria interface visual independentemente dos demais blocos. Com isso foi imediatamente identificada a solução para este problema através do reuso no site do ESELAW 2008 do mesmo Módulo de *e-commerce* dos cursos de certificação da UNIFACS.

Figura 34 - Sistema de *e-commerce* do site de certificações da UNIFACS
Nota: (UNIFACS, 2009)

8 ARQUITETURA EVACMS COM DCCS

Como citado anteriormente no Capítulo 1 a arquitetura do EVAcms possuía um modelo de componentes próprio e limitado às extensões do sistema, além disso não implementava o conjunto completo características de um componente de *software*, que garantiria uma maior reusabilidade do código.

A partir da estrutura pré-existente apresentada, iniciou-se um processo de adaptação do EVA de forma que possa operar como DCCs em PHP. Esse processo foi dividido em duas etapas: (i) implantação do modelo DCC em extensões do EVA; (ii) inclusão do DCC como modelo de componentes do Núcleo EVA. Para o escopo deste projeto, foi implantada a primeira etapa e, especificamente, a transformação dos Módulos EVA para o modelo DCC.

O modelo de DCCs se encaixou no contexto pré-existente do EVAcms. Para isso foi usada uma estrutura que permite agregar funcionalidades ao EVAcms sem ter que alterar seu "core", mantendo com isso compatibilidade com as funcionalidades já desenvolvidas. No EVAcms a estrutura que permite tais características foi apresentada anteriormente como o nome de Módulos.

A relação entre Módulo e *Framework* no EVA antes da implantação dos DCCs pode ser sintetizada na Figura 35, ou seja, o *Framework* executa chamadas a funções do Módulo e vice-versa, sem que seja explicitada qual a exata interface entre ambos. Desse modo, a relação e as dependências entre Módulo e *Framework* ficam obscuras. Adicionalmente, o cliente do módulo precisa conhecer os detalhes internos do mesmo para chamar suas funções.

Figura 35 - Relação Módulo x Framework na versão anterior do EVA

A Figura 36 ilustra detalhes internos de implementação. Por adotar o modelo Model-View-

Controller (MVC) para a sua implementação, um módulo internamente é organizado em três classes abstratas: eva_model (*model*), eva_view (*view*) e eva_controller (*controller*). Cada uma destas classes implementa um conjunto de funcionalidades básicas do MVC e interfaces padronizadas de acesso. Conforme mostra a figura, classes concretas implementam estas classes abstratas. Nesse caso, haverá uma classe concreta de cada entidade MVC para cada tabela mantida pelo módulo.

Um problema fundamental apresentado na Figura 36 é que o Módulo antes da implementação dos DCCs não possui uma interface única de acesso. Desse modo, o *Framework* faz chamadas diretas em diversos pontos de acesso tanto para o *View* quanto para o *Controller*.

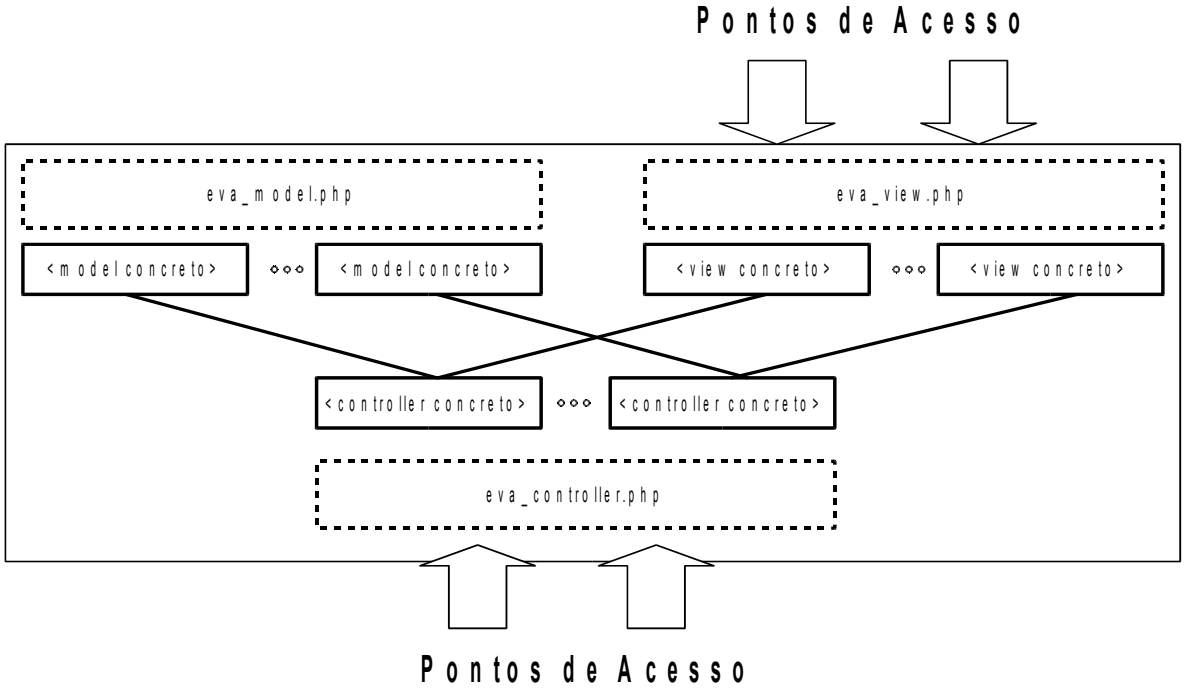

Figura 36 - Detalhe interno da organização de um Módulo

Como mostra a Figura 37, a transformação dos Módulos para o modelo DCC possibilitou a definição e explicitação de interfaces providas e requeridas na relação Módulo x *framework*. O EVA *framework* publica uma interface requerida na qual os módulos se conectarão através de suas interfaces providas.

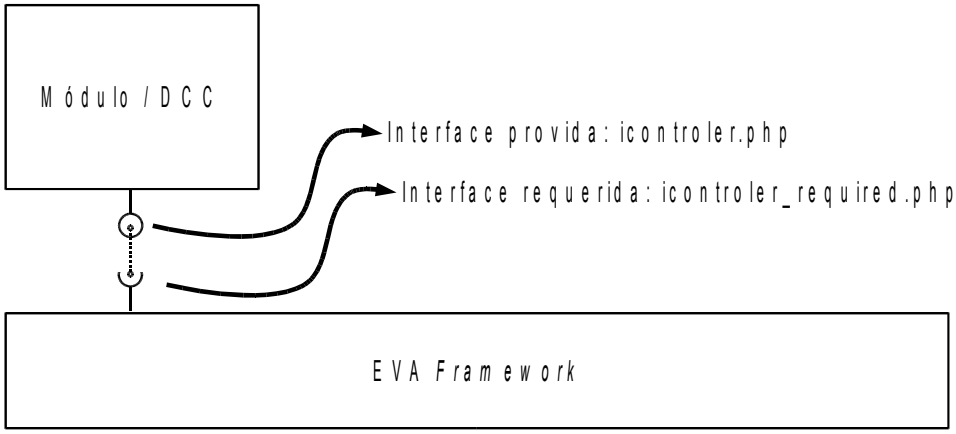

Figura 37 - Relação Módulo x *Framework* utilizando modelo DCC

Para lidar com o aspecto visual dos módulos no que diz respeito ao acesso ao módulo *View*, sem quebrar o encapsulamento e o acesso exclusivo pela interface, foi utilizado o padrão dos VDCCs apresentado anteriormente. Mais especificamente, a classe abstrata eva_view desempenha o papel de um widget, conforme descrito no Capítulo 3.6 View DCC, e por isso implementa a interface iwidget. A classe abstrata eva_contoller desempenha o papel de *root* e portanto implementa a interface iwroot.

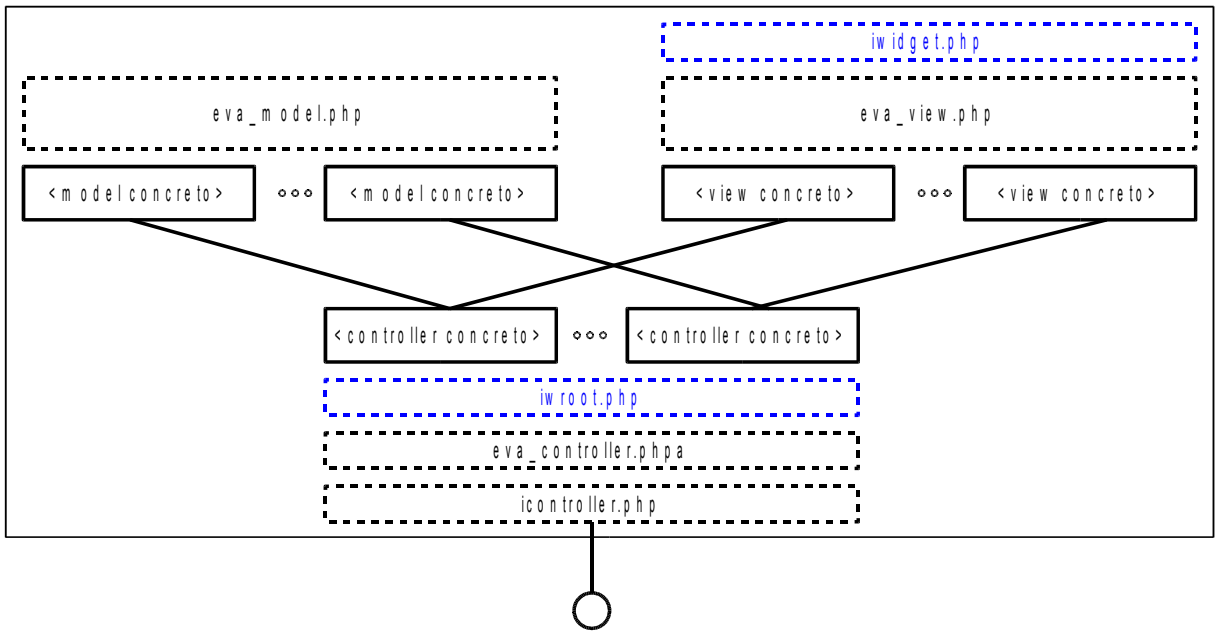

Figura 38 - Detalhe interno de um módulo usando o padrão VDCC

A Figura 39 ilustra a dinâmica de recuperação de um objeto gráfico sem quebrar o encapsulamento do Módulo. Ao invés do *Framework* fazer um acesso direto ao *View*, como

acontecia na versão anterior, agora ele terá que requisitar a referência para o *View* (widget) através de um método (getWidget) declarado na interface padrão.

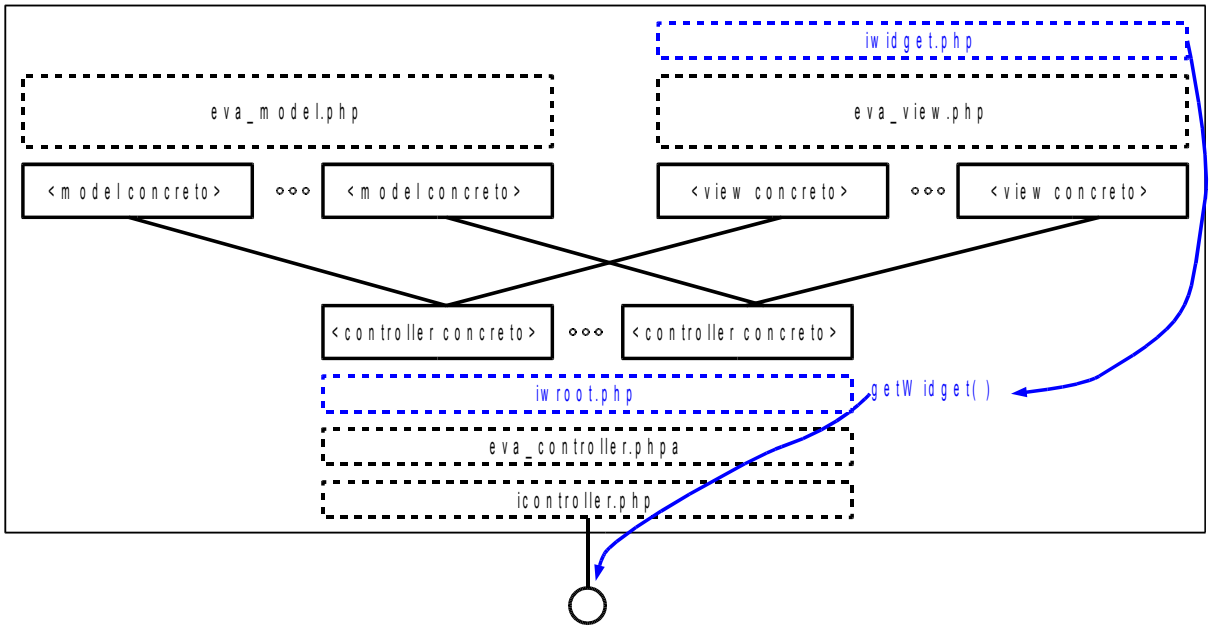

Figura 39 - Recuperação de widget através da interface padrão

A classe *View* dispõe de diferentes maneiras de apresentação a depender da funcionalidade desejada. Para cada variante a antiga versão de Módulos do EVA implementava um método, tais como os métodos form( ) e grid( ) apresentados na Figura 40. O padrão VDCC entretanto prevê que o acesso será feito exclusivamente pelo método getPresentation( ). Para realizar tal adaptação, a interface iwidget foi estendida de modo que suportasse um método extra denominado setPresentationType( ), responsável por configurar o widget para atuar de uma maneira específica. Baseado nessa configuração, o método getPresentation desvia para o método específico, conforme ilustra a Figura 40.

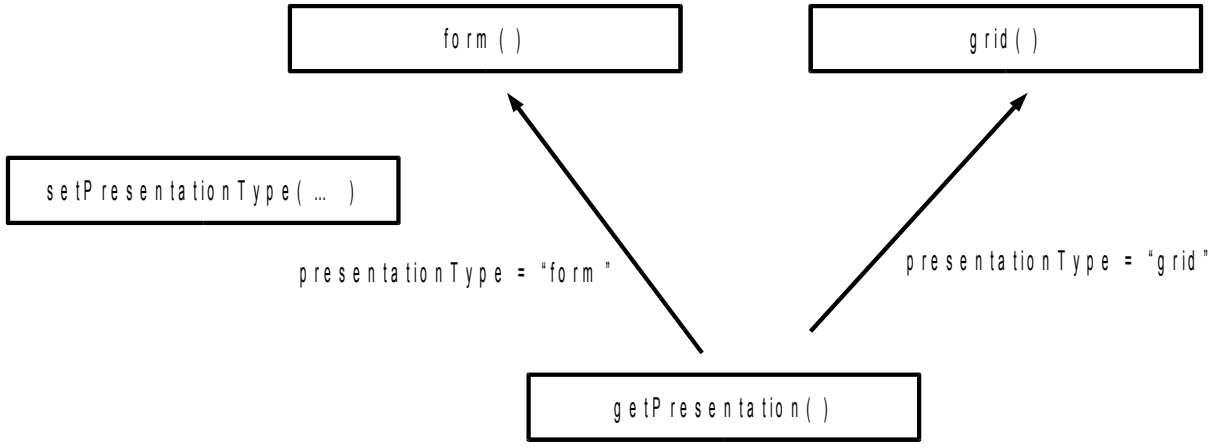

Figura 40 - Unificação da chamada do método de apresentação

8.1 GERADOR DE DCCS

Criar um novo Módulo no EVAcms é um processo simples. O EVA disponibiliza um sistema que cria automaticamente boa parte do código inicial, como as classes de acesso ao banco de dados, controle e visualização.

Atualmente existem duas abordagens diferentes para a geração de módulos em funcionamento: uma estrutura mais simples e mais antiga – usando orientação a objetos, compatível com o PHP 4, que apesar de nesta versão o PHP já suportar a programação OO ainda existem muitas limitações de conceitos e suporte a padrões de projetos – a outra versão mais atual que já incorpora o poder do PHP5 e usa um modelo explicitamente MVC, interfaces e classes abstratas. Porém para atender a necessidade de integração com o padrão DCC, demonstrada anteriormente, foi desenvolvido no EVA mais um gerador de módulo. Esse gerador permite que o EVAcms gere módulos compatíveis com os DCCs, além de assumir a tarefa de gerenciamento e comunicação dos mesmos. O EVA *framework* é bastante flexível e projetado para facilmente suportar módulos desenvolvidos por terceiros, assim como permite sua expansão sem a necessidade de modificação do código *core*.

A Figura 41 ilustra o gerador de módulos (que por sua vez também é um módulo) para a criação de Módulos/DCC, o que podemos chamar de: Criador de Módulos DCC. Este gerador foi batizado como eva_gerador_dcc e ficará encarregado das funções primárias de gerenciamento dos

DCCs e interligação entre DCCs e módulos já existentes no EVA.

Após incorporar este novo módulo gerador, tornou-se possível adicionar componentes (Módulos) no padrão DCC e, acima deste Módulo, criar outros componentes, que juntos formarão sistemas compatíveis com os DCCs (vide Figura 41).

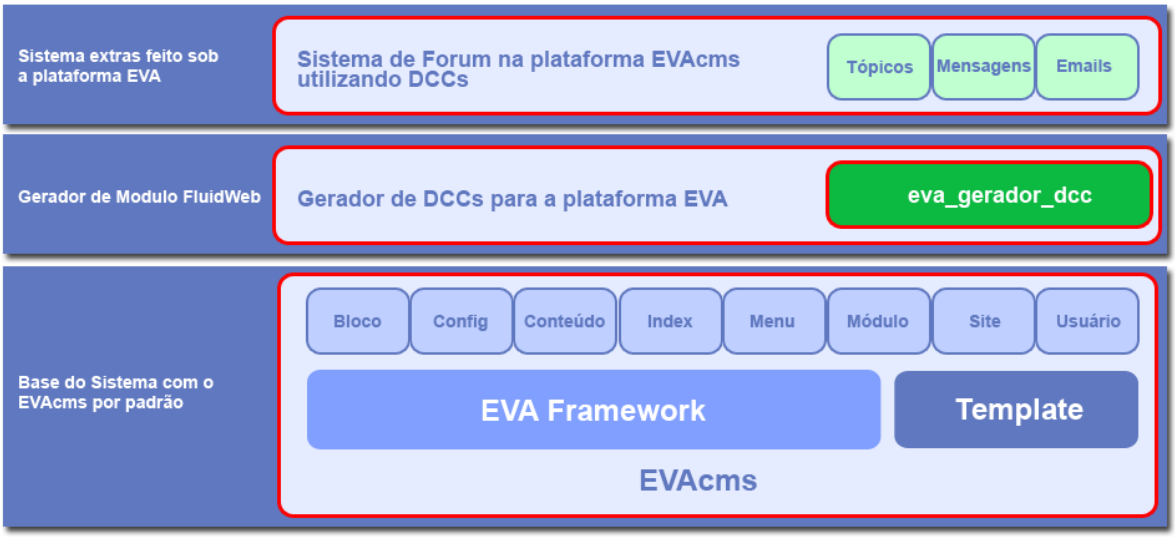

Figura 41 - Eva com o Gerador de DCCs incorporado

8.2 CASO PRÁTICO – GERADOR DE DCCS

Com o objetivo de validar o Gerador de DCCs mais um caso prático foi desenvolvido, logo este estudo de caso descreve os passos para utilizar o Gerador desenvolvido.

Para gerar um módulo utilizando o gerador automático baseado nos DCCs é necessário inicialmente estar autenticado no sistema, como demonstra a Figura 42.

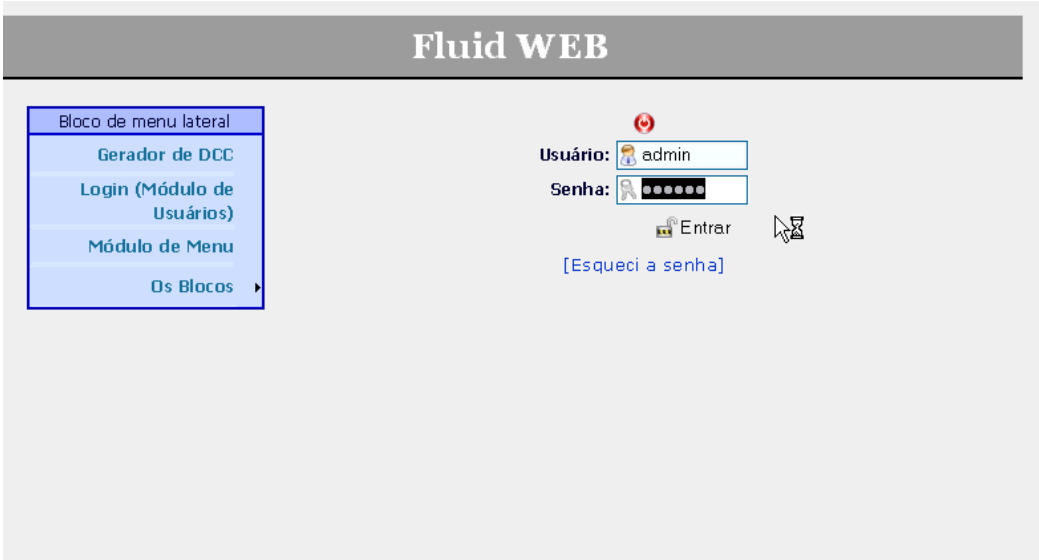

Figura 42 - Página de autenticação do EVA

Após autenticar o usuário terá acesso ao menu de contexto do sistema ver Figura 42, para visualizar este menu de contexto basta pressionar a tecla ESC no teclado do computador. O menu de contexto é considerado o centro de comando do EVA e nele é apresentado ao usuário todos os módulos em que o usuário tem acesso de administrador.

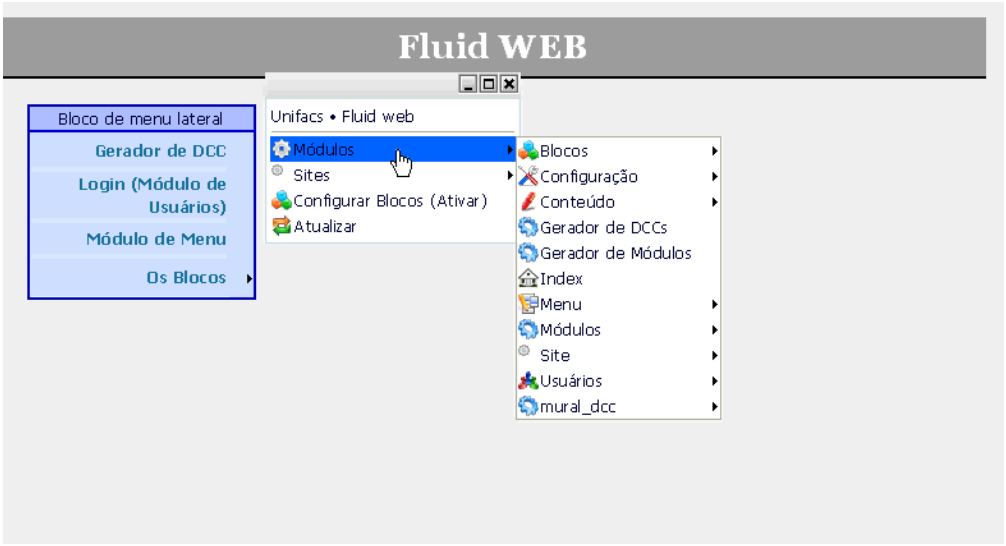

Figura 43 - Menu de contexto do EVA

Em algum momento pode ser necessário para o administrador do sistema verificar, atribuir ou excluir Módulos no EVA, para gerenciar e saber quais os módulos que o usuário possui acesso e qual tipo de acesso, deve-se acessar o módulo de gerenciamento de usuários, ver Figura 44. As permissões administrativas do(s) grupo(s) do seu usuário estão detalhadas na aba de edição de

grupos ver Figura 45

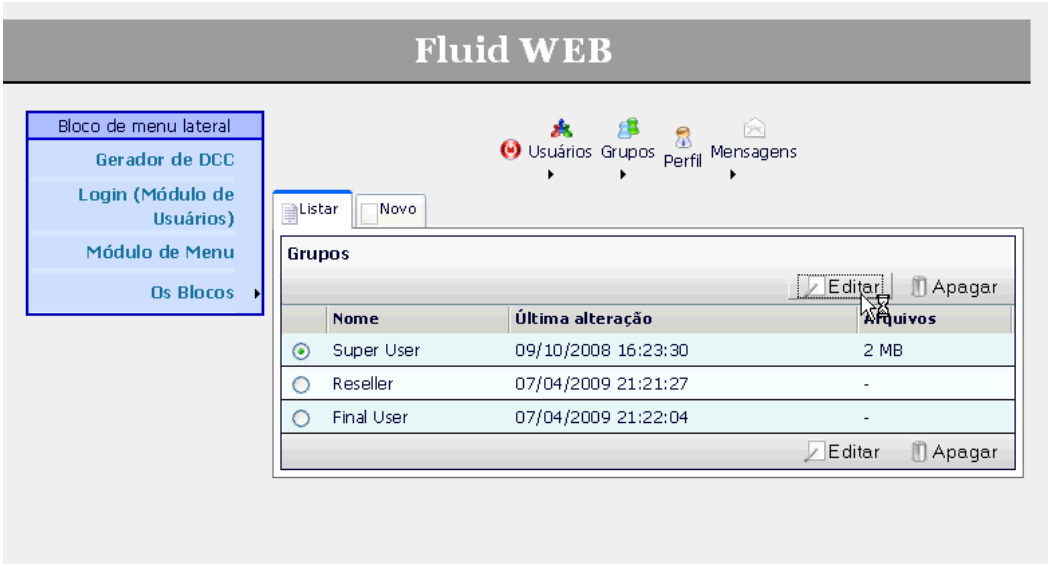

Figura 44 - Módulo de Gerenciamento de Usuários

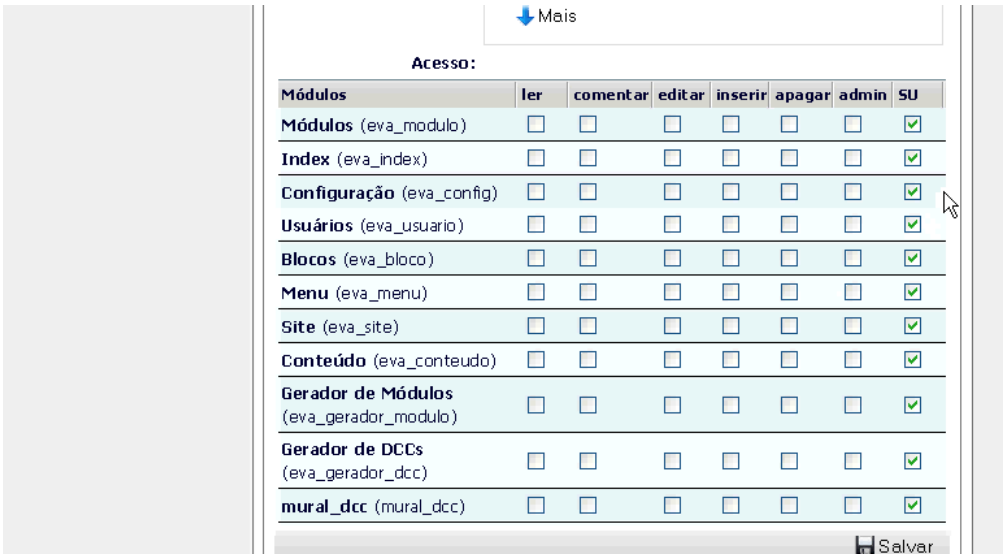

Figura 45 - Detalhes de Permissões do Grupo de Super Usuário

Mais um detalhe do EVA é a possibilidade de acessando o módulo de gerenciamento dos módulos verificar quais os módulos estão instalados, ver Figura 43 e quais estão disponíveis para instalação, ver Figura 46.

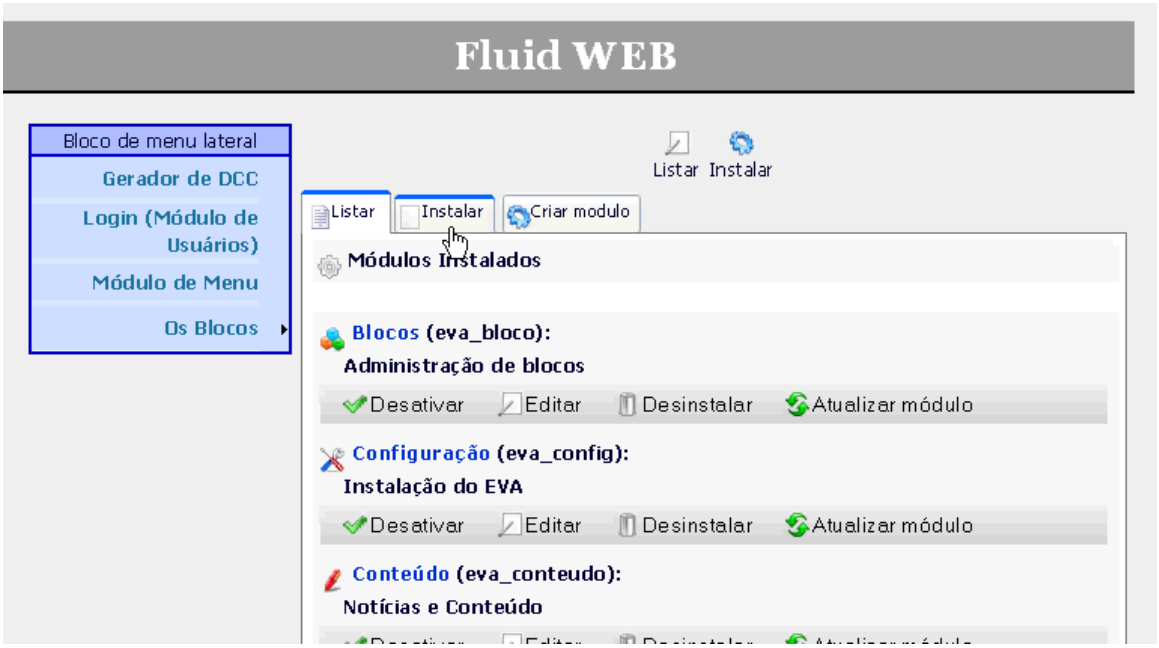

Figura 46 - Listagem dos Módulos Instalados

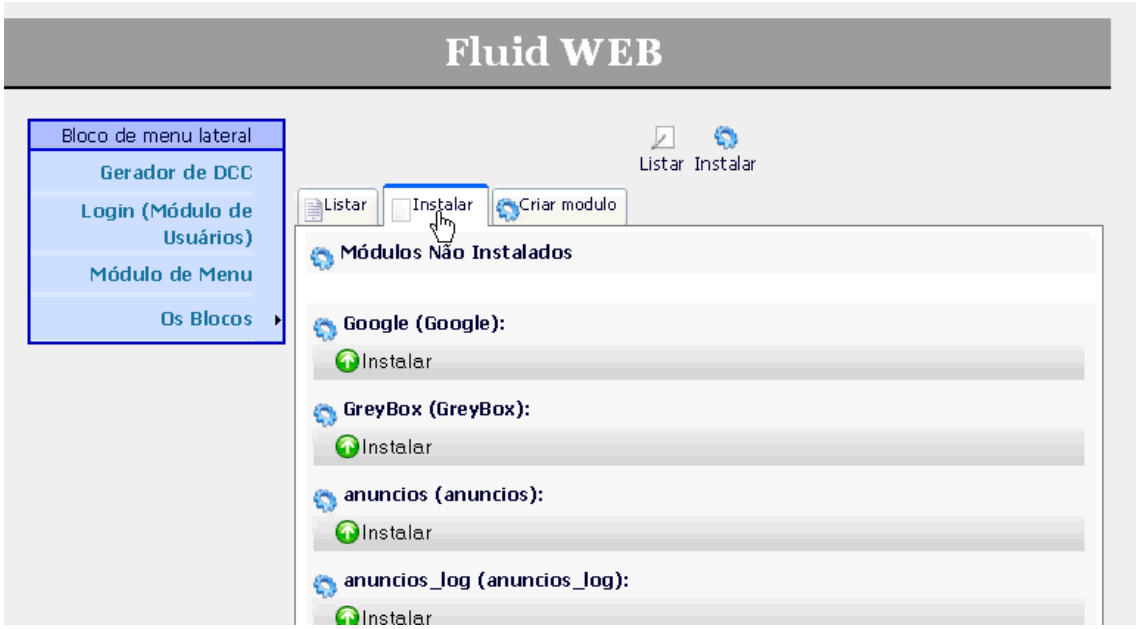

Figura 47 - Listagem dos Módulos Disponíveis para Instalação

Para validar o capítulo 7.5 Autonomia Visual dos Componentes, foi explorado neste caso prático a autonomia visual do EVA, como já citado anteriormente, trata-se da capacidade do EVA permitir que cada bloco ou módulo apresente-se isoladamente independente dos demais blocos instalados no sistema. Esta capacidade é de muito importante pois abre a possibilidade do EVA

combinar-se com outros sistemas, mesmo de plataformas diferente, como por exemplo mesclar sistemas CMS com sistemas LMS, facilitando o desenvolvimento de Modelos de Componentes para Sistemas CMS e LMS formando assim um LCMS, capítulo 5.3

Nos próximos parágrafos estão descritos os procedimentos necessários para usufruir desta flexibilidade visual.

Para avisar ao sistema que deseja exibir separadamente um bloco, utiliza-se variáveis GET para transportar a chave primaria do bloco a ser exibido para sistema EVA. No estudo de caso foi exibido um bloco de menu cuja chave primaria é o numero 3, A chave primaria sempre aparece ao lado do nome do bloco no módulo de gerenciamento de blocos ver Figura 48.

Ainda na figura Figura 48.observa-se uma área chamada pelo Gerenciador de Blocos de centro que em regra é reservada para a apresentação do módulo, mas com o uso da seguinte URL **http://seu.dominio/index.php?bloco=3** conduzirá para o sistema a chave primaria do bloco identificado pela variável **bloco** e valor igual a , estas informações avisam ao EVA o que deve ser apresentado no centro do sistema. No entanto é possível isolar o bloco mais ainda, retirando todo o tema que envolve o site basta incluir mais uma variável GET na URL do sistema **http://seu.dominio/index.php?bloco=3&popup=true** a variável GET popup destaca o bloco referente a chave primaria 3 e o isola do tema ver Figura 49.

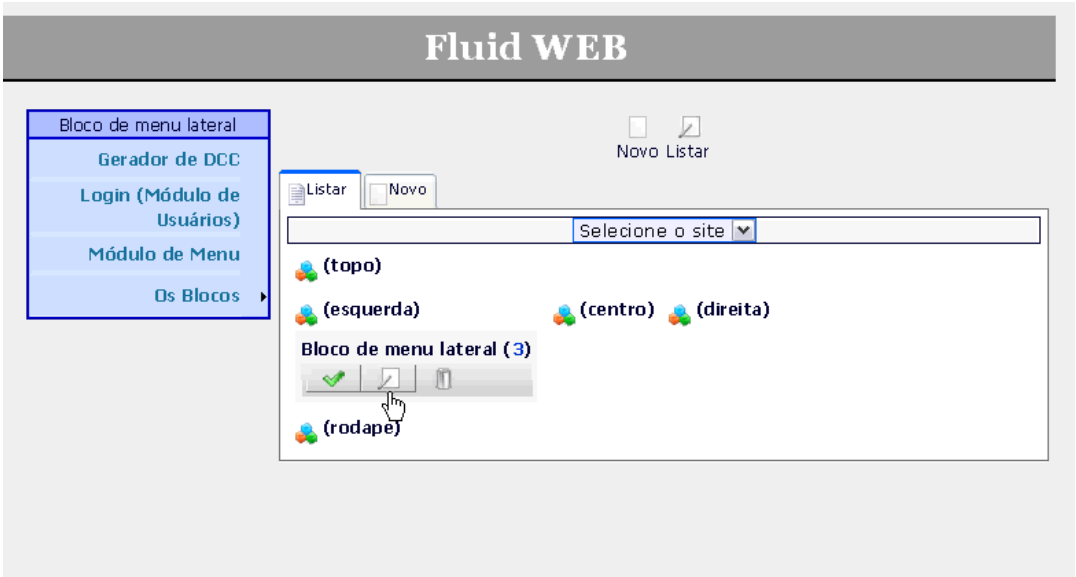

Figura 48 - Módulo de Gerenciamento de Blocos

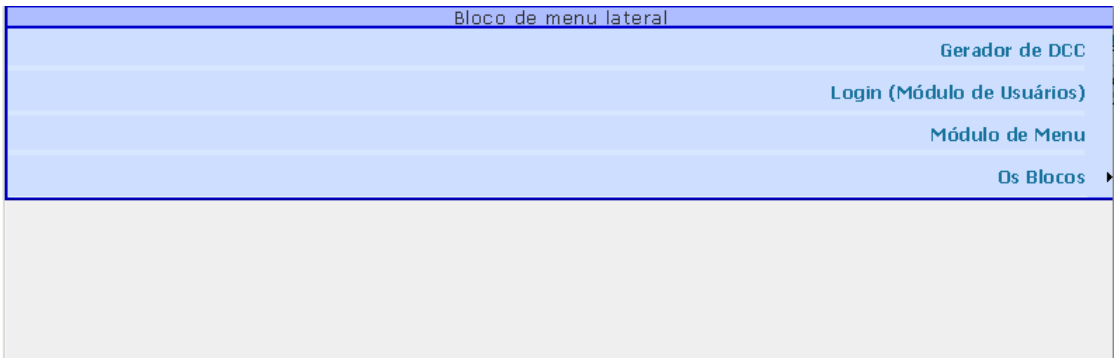

Figura 49 - Bloco destacado e isolado do tema

Nota: URL utilizada na Figura 49, http://seu.dominio/index.php?bloco=3&popup=true

E finalmente para simplificar os passos necessários para criar um novo módulo no caso prático foi utilizado uma tabela já pré existente. A tabela contém apenas 3 campos: mu_cod, a chave primaria, mu_nome, para armazenar um nome e mu_texto, para armazenar o texto do mural, ver Figura 50.

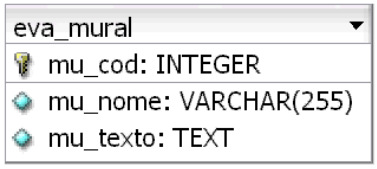

Figura 50 - Modelo de dados do estudo de caso

Para a criação do novo módulo deve-se acessar o Gerador de módulos DCCs, preencher os campos, nome e Descrição e selecionar na listagem a tabela mencionada anteriormente, ver Figura 51.

Após clicar no botão Criar módulo e receber a mensagem de confirmação, deve-se verificar no gerenciador de módulos, Figura 47, o módulo recém criado. Então, clica-se no botão de instalar e finalmente o módulo já esta disponível no sistema.

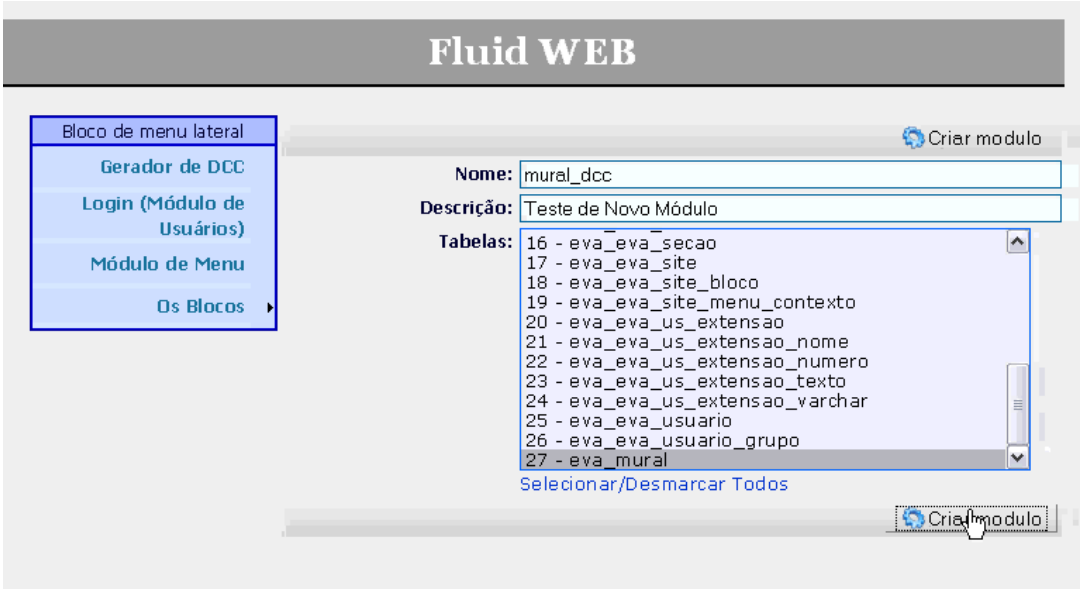

Figura 51 - Gerador de Módulos DCCs

O novo módulo por padrão é gerado inicialmente com as funções de listar, editar e novo. Após alguns ajustes no arquivo de idioma, um incremento no tipo de editor padrão para o campo mo_texto, teremos uma tela como a Figura 52.

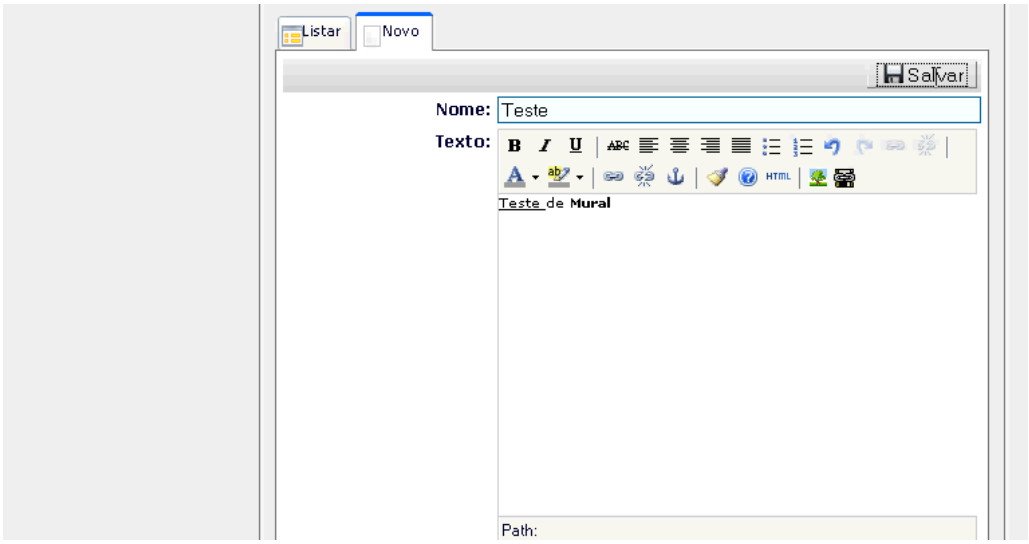

Figura 52 - Módulo Gerado pelo Gerador de Módulos DCCs

8.3 CONCLUSÃO

Ao final da adaptação concluiu-se o EVA *framework* e seus componentes tiveram uma clara evolução no âmbito estrutural, pois a relação anteriormente obscura entre componentes (Módulos) e *framework* foi explicitada através de uso de interfaces para descrever os serviços providos e requeridos entre seus módulos e módulos, e entre módulos e *framework*. Também observando-se a perspectiva da estrutura o padrão VDCC teve uma contribuição muito importante, pois através dela foi possível organizar a dinâmica de apresentação, centralizando as chamadas por objetos gráficos, ou seja, o acesso direto à camada View não existe mais, com a aplicação do modelo VDCC todo acesso foi centralizado e atualmente é feito através do método (getWidget).

9 CONSIDERAÇÕES FINAIS

Ao se pensar em componentes deve-se considerar duas esferas de aplicação. A primeira é o uso de componentes com a finalidade de reuso e distribuição, a outra muito importante, que não podemos deixar de lado, é a esfera da componentização visual, que é a capacidade do *framework* de associar aos componentes uma ou mais áreas da interface visual de um aplicativo. Sendo assim, o componente irá gerir toda forma de apresentação daquela parte específica da interface do *software.* Essa capacidade resulta em autonomia dos componentes na construção da interface visual. Nesta pesquisa demonstramos que tal independência visual dos componentes permite utilizar um componente de *software* de um aplicativo dentro de outro aplicativo distinto (embutido ou *embedded*) de forma transparente para o usuário. Foi demonstrado que o EVA *framework* possui um conjunto de características propícias para a componentização visual.

As três maiores contribuições desta pesquisa neste contexto foram:

a) A definição de um modelo de componentes de software baseado no modelo DCC para a linguagem PHP.
b) A implementação contextualização deste modelo como base para a expansão de *frameworks* CMS em PHP com aplicação prática no EVAcms.
c) A construção de uma política de composição visual para sistemas Web, que servirá como base para a construção de sistemas embutidos.

Comparado com o modelo de componentes do PEAR, os DCCs em PHP são mais completos, incorporando características importantes como múltiplas interfaces e interfaces providas e requeridas.

9.1 TRABALHOS EM ANDAMENTO

Conforme mencionado no Capítulo 5.3, que trata do tema *Learning Content Management System*, uma das grandes aplicações da componentização visual do EVAcms será a possibilidade de embuti-lo em ambientes terceiros, o que viabilizaria combiná-lo com LMS para a construção de um LCMS.

Este é um dos projetos que atualmente está em andamento, no qual se pretende embutir o EVAcms como CMS dentro do Moodle. A Figura 53 mostra um resultado preliminar deste projeto, no qual o EVAcms foi embutido para operar dentro do Moodle. Note que neste experimento o bloco de gerenciamento de parceiros é ativado diretamente dentro do ambiente Moodle.

Figura 53 - EVAcms funcionando embutido no Moodle

9.2 TRABALHOS FUTUROS

Em todo sistema desenvolvido sobre o EVA é sempre possível e desejável manter uma constante evolução alinhada com a próxima evolução do EVA. Para otimizar este processo de evolução, o padrão DCC passará a fazer parte diretamente do núcleo do EVA *framework* de modo que todos os componentes do EVA *framework* deverão seguir o padrão DCC e seu modelo de

comunicação, trabalhando diretamente em conjunto com o sistema de *templates* do EVA (vide Figura 54). Esta alteração permitirá uma maior flexibilidade na escolha do componente a ser utilizado para exercer uma determinada função, pois os componentes do EVA no dia de hoje não explicitam as interfaces requeridas e interfaces providas.

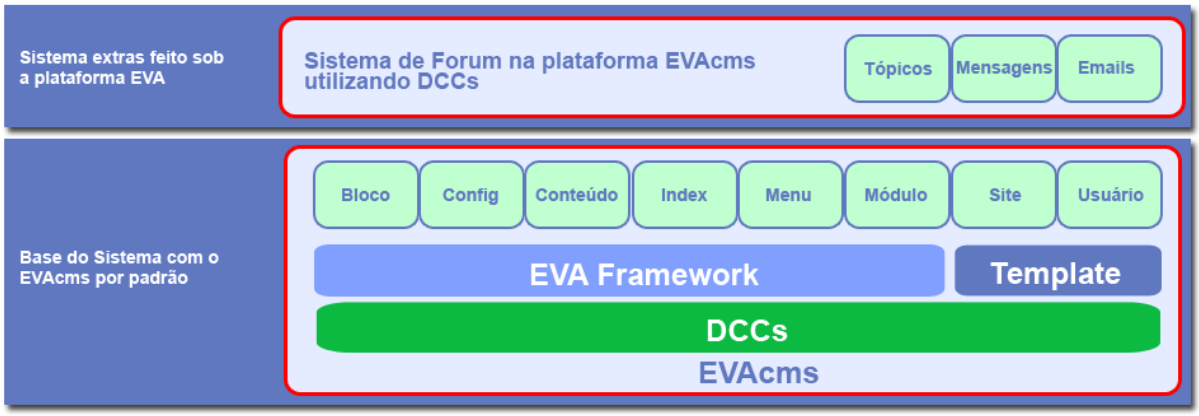

Figura 54 - Os DCCs trabalhando em conjunto e sua relação futura com o EVA *framework* e com o sistema de *templates* do EVA

REFERÊNCIAS

A DIRECTORY of browser-based WYSIWYG editor components. Disponível em: <http://www.htmlarea.com/> Acesso em: 1 abr. 2009.

ADIERS, Diego Ribas. **Gerenciamento de conteúdo na web usando Plone**: aplicação ao portal da informática da UFSM. 2007. 54 f. Monografia (Graduação) - Universidade Federal de Santa Maria, Santa Maria, 2007.

ALONSO, Gustavo et al. **Web services**: concepts, architectures and applications. Palo Alto: Springer, 2004. 354 p.

BASS, Len et al. **Market assessment of component-based *software* engineering** (CMU/SEI-2001-TN-007). Pittsburgh: Software Engineering Institute, 2001, 41 p. v.1.

BAX, Marcello P.; PEREIRA, Júlio. C. L. Introdução à gestão de conteúdos. In: WORKSHOP BRASILEIRO DE INTELIGÊNCIA COMPETITIVA E GESTÃO DO CONHECIMENTO, 3., 2002, São Paulo. **Anais ...** São Paulo: KM, 2002. p. 1 – 15.

BERGSTEDT, Stefan et al. Content Management Systems and e-Learning-Systems — A Symbiosis? In: IEEE INTERNATIONAL CONFERENCE ON ADVANCED LEARNING TECHNOLOGIES, 3., 2003, Athens - Greece. **Proceedings...** . Athens - Greece: Third Ieee International Conference On Advanced Learning Technologies, 2003. p. 155 - 159.

BRENNAN, Michael; FUNKE, Susan; ANDERSON, Cushing. **The learning content management system:** a new *e learning* market segment emerges. Framingham, MA, Estados Unidos: IDC, 2001.

BROWNING, Paul; LOWNDES, Mike. **JISC Techwatch report:** content management systems. Londres: [s.n.], 2001.

BROWNSWORD, Lisa; CLEMENTS, Paul. **A case study in successful product line development**. Pittsburg: Software Engineering Institute, 1996, 84 p.

BROY, Manfred et al. **What characterizes a (*software*) component?** 1998. Berlin, Heidelberg – Alemanha: Springer,1998.

CAPRETZ, Luiz Fernando; CAPRETZ, Miriam A. M.; LI, Dahai. Component-based software development. In: ANNUAL CONFERENCE OF THE IEEE INDUSTRIAL ELECTRONICS SOCIETY (IECON), 27., 2001, Denver, CO. **Anais...** Denver, CO : IEEE Computer Society, 2001, p. 1834-1837.

CMS chegou para ficar!. **REVISTA W**. São Paulo: Europa, n. 77, 20 jan. 2007.

EGROUPWARE. **EGroupware**: Home. Disponível em: http://www.egroupware.org/. Acesso em: 1 mar. 2009.

EHVOLUA!. Disponível em: <http://ehvolua.blogspot.com/> Acesso em: Acesso em: 1 mar. 2009.

EVA cms. **Eva cms - gerenciador de conteúdo 100% nacional**. Disponível em: <http://www.evacms.com.br/> Acesso em: 1 mar. 2009

EXTENSIBLE markup language (XML) 1.0. 5. ed. Disponível em: <http://www.w3.org/TR/REC-xml/> Acesso em: 1 abr. 2009.

EZRAN, Michel ; MORISIO, Maurizio; TULLY, Colin. **Practical software reuse**. London: Springer-Verlag, 2002, 222 p.

FCKEDITOR the text editor for internet. Disponível em: <http://www.fckeditor.net/> Acesso em: 1 abr. 2009.

FLUIDWEB. Disponível em: <http://sourceforge.net/apps/mediawiki/fluidweb/> Acesso em: 1 abr. 2009.

FRAMEWORK - Joomla! Documentation. Disponível em: <http://docs.joomla.org/Framework> Acesso em: 1 abr. 2009.

GAMMA, Erich et al. **Design patterns:** elements of reusable object-oriented software. Estados Unidos: Addison-Wesley, 1995.

GRACE, Audrey; BUTLER, Tom. Learning management systems: a new beginning in the management of learning and knowledge. **Int. journal of knowledge and learning,** v.1, n.1-2, 2005, p. 12-24.

HOPKINS, Jon. **Component primer.** New York: ACM, 000.

IBM. **CORBA Component Model (CCM).** Disponível em: <http://www.ibm.com/developerworks/library/co-cjct6/> Acesso em: 1 abr. 2009.

JOHNSON, Ralph. E. Components, Frameworks, patterns. In: PROCEEDINGS OF THE 1997 SYMPOSIUM ON SOFTWARE REUSABILITY, 1., 1997, Boston, Massachusetts, United States. **Proceedings...** Massachusetts: ACM, 1997. p. 10 - 17.

JOHNSON, Ralph E. **Frameworks = (components + patterns)**. New York: ACM, 1997.

MAMBO. **Mambo Support**. Disponível em: <http://mambo-support.org/> Acesso em: 1 mar. 2009.

MCILROY, Malcolm Douglas. Mass produced software components. In: SOFTWARE ENGINEERING: REPORT OF A CONFERENCE SPONSORED BY THE NATO SCIENCE COMMITTEE, 1., 1968, New York. **Proceedings...** New York: Scientific Affairs Division, 1968. p. 79 - 87.

MICROSOFT. **COM**: component object model technologies. Disponível em: <http://www.microsoft.com/com/default.mspx> Acesso em: 1 abr. 2009.

MICROSOFT. **The component object model:** a technical overview. Disponível em: <http://msdn.microsoft.com/en-us/library/ms809980.aspx> Acesso em: maio 2009.

MINETTO, Elton Luís. **Frameworks para desenvolvimento em PHP.** São Paulo: Novatec, 2007.

MOXIECODE SYSTEMS. **TinyMCE - Javascript WYSIWYG editor.** Disponível em: <http://tinymce.moxiecode.com/> Acesso em:1 abr. 2009.

MOZILLA. **Creating XPCOM Components – MDC.** Disponível em: <https://developer.mozilla.org/en/Creating_XPCOM_Components> Acesso em: 1 abr. 2009.

NIEDERAUER, Juliano. **PHP para quem conhece PHP**. São Paulo: Novatec, 2004.

OSGI. **OSGi Aliance | About / OSGi Technology.** Disponível em: <http://www.osgi.org/About/Technology> Acesso em: 1 abr. 2009.

PARRISH, Rick. **XPCOM Part 1: a**n introduction to XPCOM. Disponível em: <http://www.ibm.com/developerworks/webservices/library/co-xpcom.html> Acesso em: 1 abr. 2009.

PHP-NUKE. **PHP-Nuke**. Disponível em: <http://www.phpnuke.org/> Acesso em: 1 mar. 2009.

POSTNUKE. **Flexible content management system**. Disponível em: <http://www.postnuke.com/> Acesso em: 1 mar. 2009.

PURL. **PURL home page.** Disponível em: <http://purl.org> Acesso em: 1 abr. 2009.

ROBERTSON, James. **How to evaluate a content management system.** Disponível em: <http://www.steptwo.com.au/papers/kmc_evaluate/index.html>. Acesso em: jan 2002.

SANTANCHÈ, André. **Fluid web e componentes de conteúdo digital:** da visão centrada em documentos para a visão centrada em conteúdo. 2006. 400 f. Tese (Doutorado) - Unicamp, Campinas, 2006.

SANTANCHÈ, André. **Construindo um componente peixe (building a fish component - portuguese version)**. Disponível em: <http://www.scivee.tv/node/9610>. Acesso em: 1 abr. 2009.

SANTANCHÈ, André; MEDEIROS, Claudia Bauzer. A component model and infrastructure for a fluid web**. IEEE transactions on knowledge and data engineering.** Campinas, 2007.

SANTANCHÈ, André; MEDEIROS, Claudia Bauzer; PASTORELLO, Gilberto Zonta. **User-author centered multimedia building blocks.** Campinas: Multimedia Systems, 2007.

SIQUEIRA, Bruno Rodrigues. **Apostila PHP**. São Paulo: [s.n.], 2002.

SUN. **JGURU. Enterprise JavaBeans fundamentals.** Disponível em: <http://java.sun.com/developer/onlineTraining/EJBIntro/EJBIntro.html> Acesso em: 1 abr. 2009.

SZYPERSKI, Clemens. **Component software**: beyond object-oriented programming. Estados Unidos: Addison-wesley, 2002.

THE PEAR DOCUMENTATION GROUP. **PEAR manual.** Disponível em: <http://pear.php.net/manual/en/> Acesso em: fev. 2009.

TIOBE. **TIOBE programming community index.** Disponível em: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html> Acesso em: fev de 2009

UNIFACS. **Unifacs - cursos**. Disponível em: <http://www.certificacoes.unifacs.br> Acesso em: fev de 2009

VIANNA, Matheus Barreto Meireles. **Autenticação de usuários utilizando MySQL Aliado à um processo de unificação de senhas.** Minas Gerais, 2007.

W3C **W3C. Extensible markup language (XML)**. Disponível em: <http://www.w3.org/XML/>. Acesso em: 1 Mar. 2009.

XSTANDARD. **The XHTML WYSIWYG editor for desktop & web application**. Disponível em: <http://xstandard.com/> Acesso em: 1 abr. 2009.

XOOPS. **Powered by You! :** XOOPS CMS (content management system). Disponível em: <http://www.xoops.org/> Acesso em: 1 mar. 2009.