DEFINIÇÃO DE COMPONENTES ARQUITETURAIS REUTILIZÁVEIS: UM ESTUDO DE CASO NA UTILIZAÇÃO DE PADRÕES DE PROJETOS ARQUITETURAIS NO DESENVOLVIMENTO DE SISTEMAS PARA AMBIENTES COOPORATIVOS

L.M. Silva
Aluno do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas – CEFET-RN
Av. Salgado Filho, 1159 Morro Branco CEP 59.000-000 Natal-RN
E-mail: leogillies@hotmail.com

B.P. Pontes
Aluno do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas – CEFET-RN
Av. Salgado Filho, 1159 Morro Branco CEP 59.000-000 Natal-RN
E-mail: bppontes@hotmail.com

P. A. Souza Neto
Professor do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas – CEFET-RN
Av. Salgado Filho, 1159 Morro Branco CEP 59.000-000 Natal-RN
E-mail: placidoneto@cefetrn.br

RESUMO

A arquitetura de *software* é considerada uma ponte entre a análise de requisitos e a implementação no desenvolvimento de projetos de *software*. Uma arquitetura descreve como os componentes, que compõem o *software*, interagem entre si, onde ocorrem essas interações e quais são suas principais propriedades. A definição de uma arquitetura facilita o entendimento, reuso, construção, evolução, análise e administração do processo de desenvolvimento de *software.* Semelhante ao conceito de arquitetura, o padrão de projeto descreve a solução a partir de um problema comum, de forma que se possa reusar esta solução quantas vezes for necessário. As implementações dos padrões não se repetem. A partir dessas definições, o objetivo deste trabalho é definir e desenvolver componentes de *software* reutilizáveis a partir de padrões de projeto arquiteturais. Será apresentado um estudo de caso no desenvolvimento de um sistema corporativo, onde após uma definição arquitetural e aplicação de padrões, componentes podem ser produzidos e reutilizados em projetos de *softwares* semelhantes. Uma arquitetura poderá ser desenvolvida a partir do reuso desses componentes. Essa arquitetura poderá ser aplicada para atender aos requisitos de qualidade de *software*, os quais são classificados como requisitos básicos e requisitos extras. A finalidade da reutilização de componentes é aumentar a produtividade no processo de desenvolvimento de *software*.

PALAVRAS-CHAVE: arquitetura de *software*, padrões de projeto, sistemas corporativos, requisitos de qualidade, componentes reutilizáveis.

1 INTRODUÇÃO

A informação é o insumo básico para todas as atividades humanas. Seu objetivo é proporcionar alterações no comportamento das pessoas, reduzindo a incerteza. Sob uma perspectiva empresarial, (Cendón, 2002) afirma que no processo de tomada de decisões, a informação é usada na redução de incertezas, monitoração da concorrência, identificação de ameaças e oportunidades e melhoria da competitividade. O conjunto de informações usadas pelos administradores para redução de incertezas tem sido chamado de "informações para negócios". Tais informações englobam informações mercadológicas, financeiras, estatísticas, jurídicas e informações sobre empresas e produtos. Embora a necessidade dessas informações sempre estivesse presente, com a globalização da economia sua importância tornou-se mais significativa.

As novas tecnologias permitem à empresa maior flexibilidade na obtenção dos dados atualizados, como também oferecem aos usuários maior flexibilidade na busca e na manipulação dos dados.

Em ambientes corporativos, os sistemas são heterogêneos e consistem em uma combinação de numerosos sistemas operacionais, espalhados por múltiplos componentes de *hardware* e *software*. A construção de uma infra-estrutura de *software* capaz de permitir a comunicação entre esses diversos componentes se constitui num novo paradigma da computação.

Segundo (Morisseeau-Leroy et al., 2001) a computação de objetos distribuídos se refere à programas que realizam chamadas remotas a outros programas que residem em diferentes computadores, e até mesmo em diferentes redes, oferecendo suporte ao intercâmbio e à reutilização de objetos distribuídos e permitindo aos desenvolvedores construir sistemas a partir da montagem de componentes de diferentes fabricantes.

Este trabalho exibe uma solução para definição de componentes arquiteturais reutilizáveis através de um estudo de caso utilizando padrões de projetos arquiteturais para o desenvolvimento de sistemas coorporativos, onde, nesse desenvolvimento, seja mostrada a utilização de um componente pré-existente e a criação de um novo componente.

Este artigo está organizado da seguinte forma: na seção 2 serão mostrados alguns conceitos básicos de arquitetura de *software*, na seção 3, será exibida uma explanação sobre padrões de projeto, componentes serão abordados na seção 4, a UML na seção 5 e o estudo de caso na seção 6. Por fim, os resultados serão apresentados na seção 7 e a conclusão, na seção 8.

2 ARQUITETURA DE SOFTWARE

Há, aproximadamente, quatro décadas, *software* constituía uma pequena, senão ínfima, parcela dos sistemas computacionais quando comparado ao *hardware*. Hoje, porém, *software* é responsável por significativa porção dos sistemas computacionais (Mendes).

Todavia, ao mesmo tempo em que os sistemas computacionais têm se tornado quase ubíquos em inúmeras aplicações, o tamanho e a complexidade do *software* têm aumentado tanto que as técnicas de abstração utilizadas nos anos 80 já não são mais suficientes para lidar com esses novos problemas. Isso refere-se ao nível de organização de um sistema ou, mais propriamente, o nível de arquitetura de *software*.

Para lidar com a crescente complexidade e o tamanho de sistemas, engenheiros de *software* têm feito uso de princípios de projeto como, por exemplo, a ocultação de informações. Entretanto, à medida que os sistemas tornam-se cada vez maiores, o uso de uma disciplina deve ser enfatizado de modo a obter resultados de baixo custo e de maior qualidade. Dentro desse contexto, a arquitetura de software tem sido utilizada para lidar com sistemas grandes e complexos (Mendes).

O processo de desenvolvimento de *software* é composto de muitas etapas que vão da análise de requisitos até a sua implementação. Um dos pontos críticos nesse caminho é a definição e construção da arquitetura da aplicação. A arquitetura representa o projeto global, solução computacional ao problema identificado na fase de análise de requisitos.

Uma arquitetura de *software* é a principal parte do projeto de um sistema, mostrando como as partes que o compõem interagem entre si, onde ocorrem essas interações e quais são as principais propriedades dessas partes, dando assim uma descrição que serve para a análise e avaliação do sistema (Shaw, 1996). Segundo (Garlan, 2000), a arquitetura de *software* é considerada uma ponte entre a análise de requisitos e a implementação.

A arquitetura tem, no mínimo, seis importantes papéis dentro do processo de desenvolvimento de *software*, são eles:

- Entendimento: facilita e simplifica a compreensão do sistema, pois exibe uma abstração de alto nível do mesmo;
- Reuso: pode-se fazer reuso em vários níveis, nos quais os componentes estão integrados;
- Construção: provê o esqueleto inicial do sistema, indicando a maioria dos componentes e as dependências entre eles;
- Evolução: a arquitetura pode expor de forma clara a dimensão do sistema, separando os conceitos de funcionalidade dos meios pelos quais os componentes estão conectados, distinguindo explicitamente componentes e mecanismos. Essa separação concebe uma evolução fácil, pois permite mudanças em mecanismos de conexão sem perda de performance, interoperabilidade, prototipagem e reuso;
- Análise: permite checagem de consistência, conformidade das restrições impostas pelo estilo arquitetural, conformidade da qualidade dos atributos, análise de dependência e análise de domínio;
- Administração: a crítica evolução de uma arquitetura tipicamente leva a uma clareza maior no entendimento dos requisitos, estratégias de implementação e riscos potenciais.

Arquiteturas de *software* podem utilizar os mesmos componentes e conectores, podendo assim, ser classificadas. Tal classificação é importante pois possibilita o reuso da arquitetura em algum sistema semelhante.

Garlan e Shaw (Garlan et al., 1993) afirmam que um estilo arquitetural define uma família de sistemas em termos de um padrão de estrutura organizacional. Mais especificamente, um estilo arquitetural determina o vocabulário dos componentes e conectores que podem ser usados em instâncias daquele estilo, juntamente com um conjunto de restrições de como eles podem ser combinados.

Um sistema bem organizado, com todos os seus componentes devidamente relacionados, exibe uma estrutura lógica e modular bem como provê suporte à ocultação de informações. Tais princípios caracterizam um projeto de *software*. O objetivo de organizar os sistemas é uma necessidade natural para os projetistas e engenheiros de *software*. Uma forma de codificar o conhecimento sobre o projeto é dispor de um vocabulário do conjunto de conceitos, estruturas e padrões de uso existentes.

Mendes (Mendes, 2002) conclui que um estilo arquitetural permite que um profissional (engenheiro, projetista ou arquiteto de *software*) determine a classe a qual pertence à organização de um sistema.

O estilo em Camadas estrutura um sistema em um conjunto de camadas, onde cada uma delas agrupa um conjunto de tarefas em um determinado nível de abstração. Geralmente, em uma arquitetura em camadas, uma camada no nível N oferece um conjunto de serviços à camada no nível superior (isto é, N + 1). Para tanto, a camada N utiliza suas funções bem como faz uso dos serviços disponíveis na camada inferior (N - 1). O melhor exemplo desse estilo arquitetural é o modelo de referência OSI da ISO. Outros estilos podem ser vistos em (Gamma et al., 1995).

3 PADRÕES DE PROJETO

Segundo (Alexander et al., 1977) cada padrão descreve um problema que pode mais de uma vez no ambiente de desenvolvimento, e então descreve o âmago da solução deste problema, de forma que se possa usar esta solução um milhão de vezes, sem fazer o mesmo duas vezes.

Uma outra definição, segundo (Gamma et al., 1995), um padrão de projeto nomeia, abstrai e identifica os aspectos chave de uma estrutura de projeto comum para torná-la útil para a criação de um projeto orientado a objetos reutilizável.

Por fim, um padrão de projeto descreve uma solução para um problema que acontece frequentemente no desenvolvimento de *software*. Em geral, os padrões têm cinco elementos essenciais: o nome do padrão, o problema, o contexto, a solução e as conseqüências. Existem 23 padrões de projeto divididos em três categorias: padrões estruturais, padrões de criação e padrões comportamentais, que podem ser encontrados com detalhes em (Gamma).

Os padrões estruturais se preocupam com a forma como classes e objetos são compostos para formar estruturas maiores.

- *Facade*: o objetivo deste padrão é fornecer uma interface unificada para um conjunto de interfaces de um sistema. *Facade* define um nível mais alto de abstração que torna o subsistema mais fácil de ser usado. O uso do *Facade* se faz necessário quando se quer minimizar a dependência e a comunicação entre subsistemas. O uso deste padrão traz os seguintes benefícios: reduz o número de objetos com os quais o cliente tem de lidar; promove um fraco acoplamento entre o subsistema e seus clientes; e oferece a facilidade do reuso das classes do subsistema.

Os padrões de criação abstraem o processo de criação de um objeto. Eles ajudam um sistema a tornar-se independente do método de criação dos seus objetos, bem como eles são compostos e representados.

- *Singleton*: esse padrão garante que uma classe tenha somente uma instância e fornece um ponto global de acesso para a mesma. Alguns benefícios desse padrão são: permite acesso controlado à instância única, reduz o espaço de nomes e permite um número variável de instâncias

Os padrões comportamentais se preocupam com os algoritmos e a atribuição de responsabilidades entre os objetos.

- *Observer*: o padrão Observer tem como objetivo estabelecer uma relação um-para-muitos entre objetos, de maneira que quando um objeto muda de estado todos os seus dependentes são notificados e atualizados automaticamente. Alguns dos benefícios desse padrão são os seguintes: permite variar *subjects* e observadores de forma independente; o programador pode reutilizar *subjects* sem reutilizar observadores, e vice-versa; e permite acrescentar observadores sem modificar o *subject* ou outros observadores.

4 COMPONENTES

Componentes de *software* são artefatos autocontidos, facilmente identificáveis que descrevem e/ou executam funções específicas e têm interfaces claras, documentação apropriada e uma condição de reuso definida (Sametinger, 1997).

Brown (Brown, 2000) descreve um componente como sendo "um pedaço independentemente utilizável de funcionalidade que provê acesso a seus serviços através de interfaces". Diz ainda que: "um componente é muito mais que uma sub-rotina em um programa modular, que um objeto ou uma classe na orientação a objetos ou um pacote em um modelo de sistema".

Um melhor entendimento da definição apresentada por (Sametinger, 1997) pode ser alcançado através de uma discussão mais aprofundada dos termos indicados por sua definição:

- Autocontido: característica dos componentes de poderem ser reusáveis sem a necessidade de incluir/depender de outros componentes. Caso exista alguma dependência, então todo o conjunto deve ser visto como o componente reutilizável;
- Identificação: componentes devem ser facilmente identificados, ou seja, devem estar contidos em um único lugar ao invés de espalhados e misturados com outros artefatos de software ou documentação. Os componentes são tratados como artefatos por poderem assumir uma variedade de diferentes formas como, por exemplo, código fonte, documentação e código executável;
- Funcionalidade: componentes têm uma funcionalidade clara e específica que realizam e/ou descrevem. Componentes podem realizar funções ou podem ser simplesmente descrições de funcionalidades. Com isso se deseja também caracterizar como componentes toda a documentação do ciclo de vida do *software*, embora esta não inclua codificação propriamente dita de funcionalidades;
- Interface: componentes devem ter uma interface clara, que indica como podem ser reusados e conectados a outros componentes, e devem ocultar os detalhes que não são necessários para o reuso;
- Documentação: a existência de documentação é indispensável para o reuso. O tipo de componente e sua complexidade irão indicar a conveniência do tipo de documentação;
- Condição de Reuso: componentes devem ser mantidos de modo a preservar o reuso sistemático e a condição de reuso compreende diferentes informações como, por exemplo, quem é o proprietário, quem deve ser contatado em caso de problema, qual é a situação de qualidade, entre outras.

Os componentes são, na maioria das vezes, objetos comerciais que têm comportamentos predefinidos. Para se utilizar um componente precisa-se conhecer a sua interface com o mundo exterior, mas não precisamos conhecer os detalhes da sua implementação interna, que ficam ocultos nas interfaces, encapsulados em conjuntos de funcionalidades. As interfaces são o meio pelo qual os componentes se conectam e são constituídas por conjuntos de operações nomeadas, que são ativadas pelos clientes.

A motivação para a utilização de componentes no desenvolvimento de aplicações é que eles elevam o nível de abstração da resolução do problema de tal forma que se pode utilizá-los sem precisar ser um programador experiente. A utilização de componentes permite que os fornecedores construam ambientes de desenvolvimento visuais nos quais o conceito de conectar estes componentes de *software* forme a base de qualquer novo desenvolvimento.

Para os desenvolvedores e usuários de sistemas, o Desenvolvimento Baseado em Componentes é um caminho para reduzir os custos de desenvolvimento, aumentar a produtividade e permitir a atualização controlada dos sistemas face à rápida evolução tecnológica.

5 UML

UML (*Unified Modeling Language*) (UML) é uma família de notações gráficas, apoiada por um metamodelo único, que ajuda na descrição e no projeto de sistemas de *software*, principalmente daqueles construídos utilizando o estilo orientado a objetos (Fowler, 2004).

Fowler (Fowler, 2004) define que as pessoas podem utilizar a UML de três formas: esboço, projeto e linguagem de programação. No esboço, os desenvolvedores utilizam a UML para ajudar a transmitir alguns aspectos do sistema. Já no projeto, a UML tem foco na completeza, em um projeto detalhado. E a UML como linguagem de programação é utilizada quando os desenvolvedores desenham diagramas UML, que são compilados diretamente para o código executável e a UML se torna o código fonte. Abaixo, alguns dos principais diagramas da UML.

- Diagrama de classes: o diagrama de classes da UML é usado para descrever a visão estática de uma aplicação (Rumbaugh et al., 1999). Os principais constituintes são as classes e os seus relacionamentos. Uma classe é uma descrição de um conceito, e deve ter atributos e operações associadas.
- Diagrama de caso de uso: os diagramas de caso de uso servem para descrever o projeto na visão do cliente, além de auxiliar no entendimento dos requisitos funcionais do sistema.
- Diagrama de seqüência: esses diagramas descrevem a interação entre grupos de objetos em determinado cenário.

6 ESTUDO DE CASO: DESENVOLVENDO COMPONENTES REUTILIZÁVEIS

A importância dos componentes em aplicações corporativas é dada, visto que os componentes proporcionam um alto nível de abstração entre o cliente e a sua implementação.

Nesse contexto, foi desenvolvido um sistema corporativo, com o objetivo de mostrar a integração de componentes já existentes, bem como, defini-los, e apresentar a sua utilização em um contexto específico.

A arquitetura proposta é baseada no estilo arquitetural em Camadas e na arquitetura de referência J2EE (J2EE). Essa arquitetura é mostrada na Figura 1:

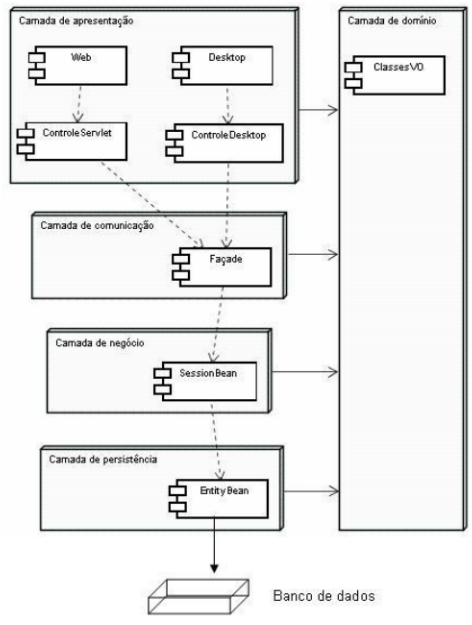

Figura 1: Arquitetura do Sistema

O funcionamento da arquitetura do sistema dar-se-á como segue.

A camada mais superior é a camada de apresentação que contém quatro componentes: *Desktop*, *Web*, ControleSerlet e

ControleDesktop. O componente *Desktop* representa a GUI (*Graphics User Interface*) *Swing* do programa. Esse componente terá um controlador próprio, o componente ControleDesktop. Outro componente, o *Web*, representa a GUI *Web* do sistema. Nele estão contidas as páginas HTML e JSP. Como o cliente *Desktop*, o cliente *Web* também terá um controlador próprio, o componente ControleServlet. Ambos os controladores, ao receberem uma requisição, repassam essa requisição para o método de negócio apropriado, situados na camada de negócio. Porém, para acessar esta camada, faz-se uso de uma camada intermediária, a camada de comunicação. Essa camada contém apenas um componente: *Facade*. Esse componente é uma interface que recebe as requisições da camada de apresentação e as repassa para a camada de negócio. Observe que o padrão de projeto *Facade* é aplicado. O *Facade* entrega a requisição dos controladores ao *bean* de sessão adequado para aquela ação. Dependendo da requisição, o *bean* pode requerer uma informação contida no banco de dados. Para tal, faz uso de um *bean* de entidade, situado na camada de persistência. O passo seguinte é o caminho inverso, até a camada de apresentação, onde os dados serão apresentados para o cliente. Vale salientar que existe ainda uma outra camada, camada de domínio, que está visível a todas as outras camadas e que contém as Classes VO (*Value Object*) do sistema.

A Figura 2 ilustra o relacionamento entre as classes de domínio do sistema, a visão estática do sistema. Trata-se de um diagrama de classes em sua visão conceitual.

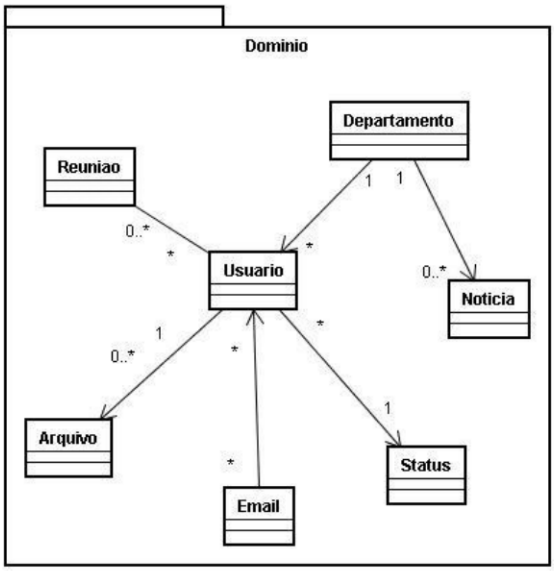

Figura 2: Diagrama de Classes

Como pode-se observar no diagrama, a entidade referência no portal é a classe Usuário. É nele que ficarão focadas as reuniões, o departamento, as notícias e o status.

A arquitetura do sistema foi definida também com padrões de projeto. Os padrões utilizados foram: *Facade*, *Observer* e *Singleton*.

Pode ser visto na Figura 3 outro diagrama UML, o diagrama de caso de uso, que retrata as interações típicas entre os usuários e o sistema.

O usuário (colaborador) do sistema pode realizar a maioria dos casos de uso do diagrama, são eles: cadastrar reunião, cadastrar notícia, ler notícias, visualizar reuniões, enviar e-mail, alterar *status*, fazer *upload*/*download* de arquivos e efetuar *login*.

Os casos de uso serão realizados a partir dos módulos do sistema. No módulo de reuniões, um usuário poderá agendar uma reunião e relacionar os usuários envolvidos, que serão informados através de e-mail. Além disso, o usuário poderá visualizar as reuniões da qual ele faz parte. O módulo de notícias permitirá o aumento desejado de transferência de informações da empresa. Nesse módulo, um usuário poderá cadastrar novas notícias relacionadas ao seu departamento, bem como visualizar notícias já cadastradas. O módulo de arquivos possibilitará ao usuário do sistema centralizar seus diversos arquivos em um único espaço destinado no Portal. Desta forma, poderá inserir (realizar *upload*), visualizar e baixar (realizar *download*) arquivos independentemente da máquina que estiver. No módulo de *status*, o usuário poderá indicar o seu *status* de forma semelhante ao software de comunicação Microsoft *Messenger*, como por exemplo: “Em

Reunião", "Ausente" e "*Online*". O usuário poderá especificar, também, uma data prevista para o término do seu *status*. Este módulo, porém, ainda não está implementado no sistema.

Existem ainda dois casos de uso que não se situam em módulo algum: enviar e-mail e efetuar *login*. No caso de uso enviar e-mail, o usuário poderá enviar um e-mail para um usuário específico do sistema. Já no caso de uso efetuar *login*, como o nome sugere, o usuário poderá se logar no sistema.

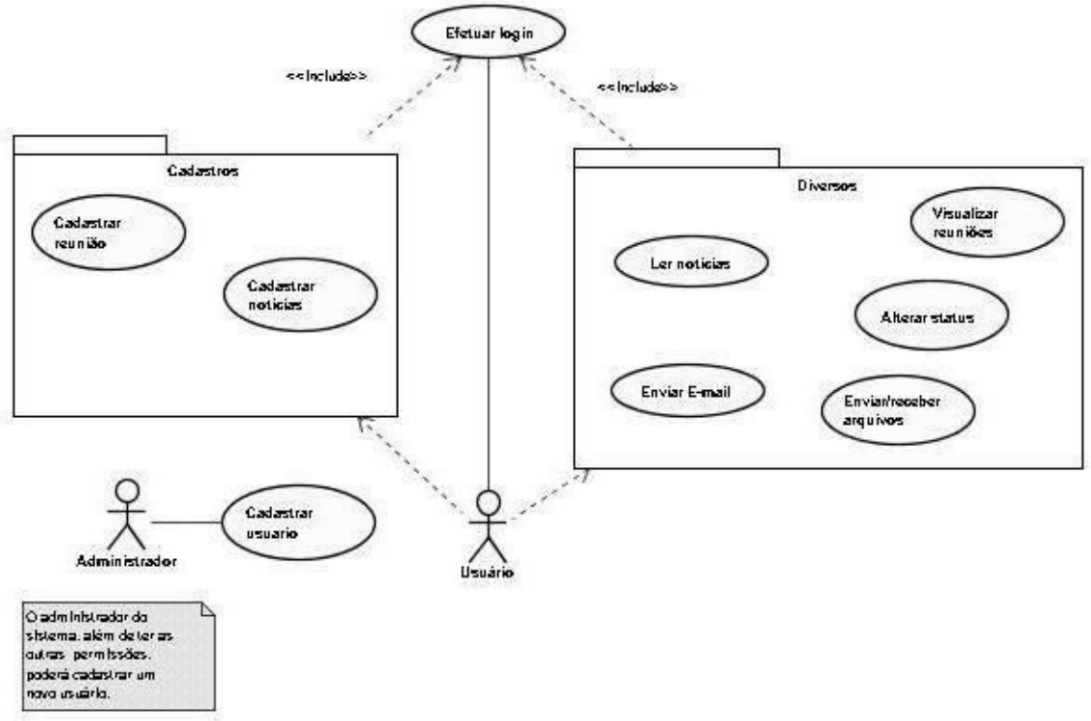

Figura 3: Diagrama de Casos de Uso

Para ilustrar um componente utilizado no sistema, será apresentado o caso de uso Enviar E-mail. Esse caso de uso permite que o sistema, após o agendamento de uma reunião, encaminhe um e-mail para cada participante dessa reunião. Na descrição, faremos uso de outro diagrama UML, o diagrama de seqüência (Figura 4).

Embora o diagrama mostre o procedimento apenas para o cliente *Web*, pode ser aplicado analogamente ao cliente *Desktop*, substituindo apenas o componente controlador. Porém, vale salientar que, para o cliente *Desktop*, o componente controlador será substituído por um componente de negócio. Observe que a utilização do padrão de projeto *Facade* propiciou a utilização desse componente, visto que foram utilizados métodos dispostos na interface. O componente utilizado para o envio de e-mail foi o JavaMail (JavaMail API).

O padrão *Observer* foi aplicado no contexto de *Login,* entre duas das camada de apresentação: Tela Principal e Módulo *Login*.

Outro padrão utilizado foi o *Singleton*. Esse padrão garante que só exista uma instância de uma determinada classe, além de oferecer um ponto global de acesso para a mesma. Estes componentes previamente definidos, podem ser compartilhados em projetos subseqüentes, pelo fato de serem componentes comuns em um contexto coorporativo. Alem disso, pe importante perceber que o componente *Java Mail* foi reutilizado.

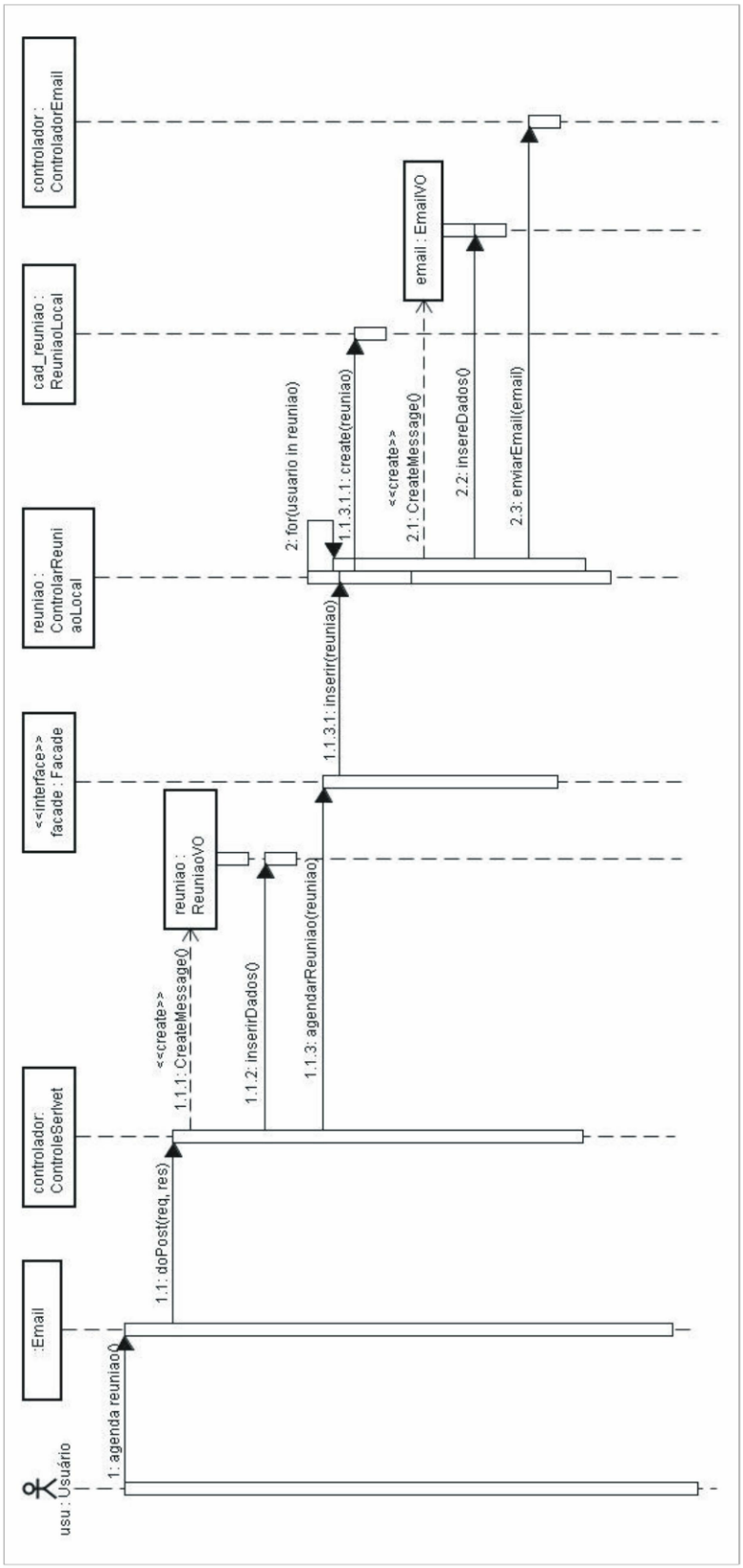

Figura 4: Diagrama de Sequência

No contexto do sistema, o padrão foi aplicado para garantir apenas uma instância da fachada. O método *getReferencia()*, aplicado ao módulo de *Login*, é instanciado pelo controlador, pois, de acordo com a arquitetura do sistema, apenas o controlador terá acesso à fachada. O código do componentes é apresentado na Figura 5.

```
public static Facade getReferencia() {
    if (facade == null) {
        try {
            facade = Cliente.conectar();
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(null,
                    "Erro ao conectar-se com o servidor. " +
                    "Verifique as suas " +
                    "configurações ou tente novamente" +
                    " mais tarde.", "Aviso!", 1);
        }
    }
    return facade;
}
```

Figura 5: Código do método *getReferência()*

O uso do padrão *Facade* fez-se necessário, pois minimizou a dependência entre as camadas de controle e de negócio, promovendo a facilidade do reuso desses componentes em outros sistemas, a disponibilização de poucos objetos ao cliente, além do fraco acoplamento entre os subsistemas. Com isso, pode-se facilmente modificar a camada de negócio sem que seja necessário modificar a camada de controle.

É exibido no diagrama de componentes (Figura 6), os componentes utilizados e definidos no sistema. Observe que eles estão estereotipados de acordo com o sua função.

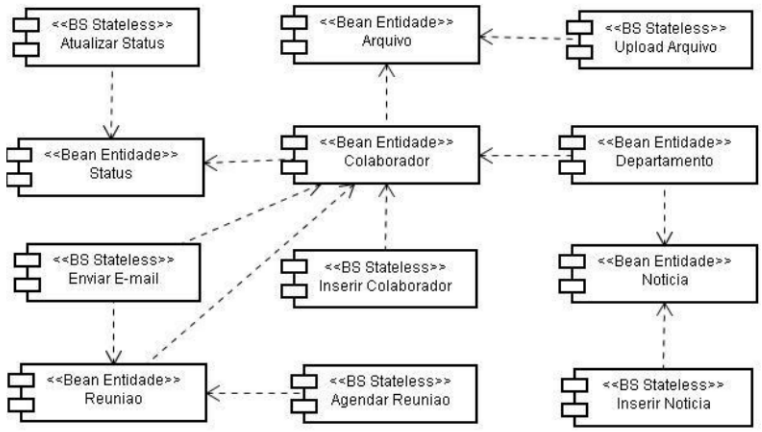

Figura 6: Diagrama de componentes

Um dos componentes desenvolvidos para a implementação do sistema foi o componente "Notícia". O funcionamento deste componente dar-se-á como segue.

Um colaborador da empresa, estando previamente *logado*, poderá realizar várias tarefas, dentre elas inserir uma notícia. Ao cadastrar uma notícia, a mesma ficará disposta para toda a empresa. O usuário poderá cadastrar notícias apenas do seu departamento. A notícia terá informações como resumo, período inicial e final de vigência e a descrição. O código fonte da fachada é mostrado na figura que segue.

```
public String inserir(br.cefetrn.portal.dominio.NoticiaVO noticia)
throws CreateException {

    getHome().create(noticia);
    return null;

}
```

Figura 7: Método inserir (*NoticiaVO noticia*)

Essa é a parte do componente que o cliente interage, podendo utilizar os seus métodos, neste caso, o método inserir(NoticiaVO noticia). Observe que o objeto NoticiaVO é passado como parâmetro pelo cliente.

A Figura 8 apresenta o código do componente que cria uma notícia *ejbCreate(NoticiaVO noticia)*, que é a implementação do método *inserir(NoticiaVO noticia)* na fachada.

```
public NoticiaPK ejbCreate(br.cefetrn.portal.dominio.NoticiaVO noticia)
    throws javax.ejb.CreateException {
    // TODO Auto-generated method stub
    setTitulo(noticia.getTitulo());
    setResumo(noticia.getResumo());
    setDescricao(noticia.getDescricao());
    setData_cad(noticia.getData_cad());
    setData_inicio(noticia.getData_inicio());
    setData_fim(noticia.getData_fim());
    return new NoticiaPK(getId());
}
```

Figura 8: Método ejbCreate(NoticiaVO noticia)

O padrão de projeto *Facade* é de fundamental importância para a definição deste componente, pois uma interface de comunicação é definida, tendo a sua implementação em uma classe à parte. O cliente, ao utilizar o componente, interagirá somente com a interface, sem conhecer detalhes da implementação do método utilizado.

7 RESULTADOS

A partir do estudo de componentes e aspectos relacionados, destaca-se a possibilidade de reuso dos mesmos. Essa possibilidade faz com que o desenvolvimento de software baseado em componentes seja bastante utilizado nos ambientes de desenvolvimento, pois propicia um aumento na produtividade e maior qualidade do produto final, por usar componentes testados. O desenvolvimento baseado em componentes facilita também a manutenção dos sistemas, pois possibilita que eles sejam atualizados pela integração de novos componentes e/ou substituição dos componentes existentes.

Como resultado, percebe-se o aumento de produtividade com a utilização de componentes pré-existentes. O desenvolvimento do sistema ocorreu em um tempo melhor, devido ao fato de reutilizar alguns componentes e implementar componentes não ambíguos e reusáveis. Componentes como envio de e-mail, calendários, componentes *Swing* para interface gráfica e *upload* de arquivos são alguns exemplos dos componentes utilizados no desenvolvimento do portal.

8 CONCLUSÃO

Foi apresentado neste trabalho um estudo de caso sobre um sistema coorporativo no tocante à utilização e definição de componentes

Uma arquitetura em camadas foi definida e aplicada ao desenvolvimento, tendo como arquitetura de referência a arquitetura J2EE, uma plataforma para o desenvolvimento de aplicações multicamadas e distribuídas. A arquitetura foi definida baseada na reutilização e definição dos componentes (gráficos, de controle, de negócio, de persistência, etc.).

Foram definidos componentes reutilizáveis a partir de padrões de projeto arquiteturais, como *Facade*. Para tal, exibiu-se um estudo de caso no desenvolvimento de um sistema corporativo, onde após uma definição arquitetural e aplicação de

padrões, componentes foram produzidos e poderão ser reutilizados em projetos de *softwares* semelhantes. Com a definição desses componentes e a utilização de outros, o aumento da produtividade no processo de desenvolvimento de *software* foi evidenciado.

9 REFERÊNCIAS

Java Platform, Enterprise Edition (Java EE). http://www.sun.com/.

The Unified Modeling Language. http://www.uml.org/.

C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. **A Pattern Language: Towns, Buildings, Construction**. 1977.

B. V. Cendón. **Bases de dados de informação para negócios**. 31, 2002.

A. M. S. Filho. **Sobre a importância da arquitetura de software no desenvolvimento de sistemas de software**.

A. M. S. Filho. **Arquitetura de Software**. 2002.

M. Fowler. **UML Distilled: A Brief Guide to the Standard Object Modelling Language**. 2004.
E. Gamma, R. Helm, R. Johnson, and J. Vlissides. **Design Patterns: Elements of Reusable Object-Oriented Software**. 1995.

D. Garlan. **Software architecture: a roadmap**. The Future of Software Engineering, 2000.

D. Garlan and M. Shaw. **An introduction to software architecture**. World Scientific Publishing Company, 2:1–39, 1993.

J. Rumbaugh, I. Jacobson, and Booch. **The Unified Modeling Language Reference Manual**. 1999.

M. Shaw. **Software Architecture: Perspectives on an Emerging Discipline**. 1996.

MOURISSEAU-LEROY, Nirva; SOLOMON, Martin K.; BASU, Julie. **Oracle 8i Programação de Componentes Java com EJB, CORBA e JSP**. Rio de Janeiro: Campus, 2001.

SAMETINGER, Johannes. **Software Engineering with Reusable Components**. New York: Springer, 1997. 271p.

BROWN, Alan W. **Large-Scale, Component-Based Development**. USA: Prentice Hall PTR, 2000.

JavaMail API, http://java.sun.com/products/javamail/.