Arquitetura PRODEPA: Uma solução para criação de sistemas distribuídos

Amanda Monteiro Sizo, Adriana Nunes Teles Xisto, José Augusto da Silva Fernandes, Leila Riodades Daher Santos, Cláudio Roberto de Lima Martins

PRODEPA – Processamento de dados do Pará
Rodovia Augusto Montenegro km 10, s/n – 66.820-000 – Belém – PA – Brasil

{amanda.sizo,adriana.teles, augusto.fernandes,leila.daher, claudio.martins}@prodepa.pa.gov.br

Abstract. *This paper describes the components of the development architecture applied at the PRODEPA company (Processamento de Dados do Estado do Pará), how it integrates with different frameworks to provide an increase in productivity in the company's software factory, and demonstrate how it allows to create distributed applications from the use of EJB technology. A case study demonstrates its applicability and the benefits achieved from the beginning of its implementation within a public organizational settings demanding software development.*

Resumo. *Este artigo descreve os componentes da arquitetura de desenvolvimento utilizado na Empresa de Processamento de Dados do Estado do Pará – PRODEPA, como ela se integra a diferentes frameworks visando prover aumento na produtividade na fábrica de software da empresa, além de demonstrar como ela cria aplicações distribuídas a partir da utilização da tecnologia EJB. Um estudo de caso demonstra sua aplicabilidade e benefícios alcançados desde sua implantação dentro de um cenário organizacional público, com forte demanda em construção de software.*

1. Introdução

Uma arquitetura de software de um programa ou sistema computacional é a estrutura que abrange os componentes de *software*, propriedades externamente visíveis desses componentes e as relações entre estes componentes [Pressman 2001]. Pode ser entendida também como a organização fundamental de um sistema, incluindo seus componentes, o relacionamento entre eles e com o ambiente computacional, além dos princípios que definem o desenho e a evolução dos componentes [IEEE 2000].

Definir um padrão arquitetural é uma maneira de estimular o reuso de componentes e padronizar a estrutura através do uso rotineiro de soluções existentes. A implantação de uma arquitetura para criação de sistemas distribuídos norteou o ciclo de desenvolvimento de *software*, tornando-o mais ágil.

Diversas arquiteturas de *software* são propostas para o desenvolvimento de *softwares* e sistemas, para as mais diversas plataformas. Apesar de uma grande quantidade permanecer privada e pertencer a grandes organizações, diversas arquiteturas estão publicamente disponíveis [Carvalho 2009]. Como é o caso da plataforma de desenvolvimento Pinhão [Pinhao 2009], e a plataforma de desenvolvimento Demoiselle

[Demoiselle 2009] que dão suporte às aplicações no mesmo ambiente computacional, isto é, no mesmo servidor de aplicação.

A crescente demanda para desenvolvimento de novos sistemas de forma ágil e eficiente nas mais variadas plataformas, estimulou a necessidade da criação de uma arquitetura padrão para o desenvolvimento de sistemas distribuídos na PRODEPA – Empresa de Processamento de Dados do Pará. Esta arquitetura, denominada "Arquitetura PRODEPA", integrada ao *framework* definido para uso interno (chamado Muiraquitã), veio suprir a necessidade de prover um meio de reutilização de componentes e arcabouços no ciclo de construção de sistemas na fábrica de *software* da empresa.

Este trabalho visa apresentar a Arquitetura PRODEPA, que segue o padrão arquitetural em camadas com algumas adaptações, mas fundamentada em alguns princípios de GoF [Gamma et al 1995] e padrões estabelecidos em Alur *et al* (2003). Foram integrados nessa arquitetura os *frameworks* Jboss-Seam, EJB3 e Hibernate, auxiliando o trabalho do desenvolvedor de software da empresa, além de viabilizar o processo proposto neste trabalho.

Diversas melhorias na qualidade do produto final podem ser observadas desde sua implantação, como: maior robustez, flexibilidade, extensibilidade, reusabilidade e alta disponibilidade. Tais características são difíceis de serem alcançadas em uma empresa pública, onde as tecnologias são freqüentemente substituídas.

O restante deste artigo está organizado como descrito a seguir: a seção 2 define o cenário organizacional, a seção 3 examina os *frameworks* utilizados, na seção 4 é apresentada a Arquitetura PRODEPA, na seção 5 é mostrada a otimização da arquitetura, na seção 6 um estudo de caso é implementado e finalmente a seção 7 apresenta as conclusões e os trabalhos futuros relativos a este artigo.

2. Contexto Organizacional

Atualmente a PRODEPA conta com aproximadamente 330 colaboradores e sua atual missão é "*Prover serviços baseados no uso de tecnologia da informação e comunicação, aos poderes públicos e à sociedade do Estado do Pará, em benefício do cidadão*" [Prodepa, 2009].

Culturalmente, a empresa apresenta fortes aspectos políticos e, por conseguinte, alta rotatividade de tecnologias. Nas gestões anteriores, os sistemas eram desenvolvidos em plataformas variadas, como Natural/Adabas, Delphi/Oracle, Lótus Notes entre outras. Porém, na atual gestão da PRODEPA foi observada a necessidade de uma mudança completa de paradigma que abrangesse não só o processo, mas também a engenharia de software para viabilizar o desenvolvimento mais ágil de sistemas, visando atender às necessidades de TI – Tecnologia da informação dos demais órgãos do Estado, promovendo assim a inserção do Estado no "*e-government*".

A criação da fábrica de software, a adoção de metodologias e padrões, a implantação do Nível F do MPS-br, além de uma reciclagem completa dos técnicos, foram vitais para conseguir avanços no ciclo de desenvolvimento de software. Dentro desse contexto, a criação da arquitetura PRODEPA e do *framework* Muiraquitã são exemplos do progresso obtido na fase de projeto e desenvolvimento, viabilizando e

uniformizando os procedimentos, diminuindo prazos e aumentando a produtividade, proporcionando a obtenção de melhores resultados com menores custos.

3. Frameworks Utilizados

Frameworks são estruturas de software cujo princípio é ser uma solução reutilizável, estável e bem documentada, que têm como objetivo auxiliar o trabalho do desenvolvedor [Minetto 2007]. Pode-se dizer ainda que *frameworks* são um conjunto de classes que colaboram entre si gerando um projeto reutilizável para um tipo específico de *software*. Cagnin (2005) conclui que nesta abordagem de reutilização, está implícito não apenas o aproveitamento do código, mas também os esforços realizados em todas as fases do desenvolvimento de *software*.

3.1. *Framework* Muiraquitã

Integrado à arquitetura PRODEPA está o *framework* Muiraquitã, que fornece vários recursos, agregados em três componentes, como ilustra a Figura 1 e detalhados a seguir.

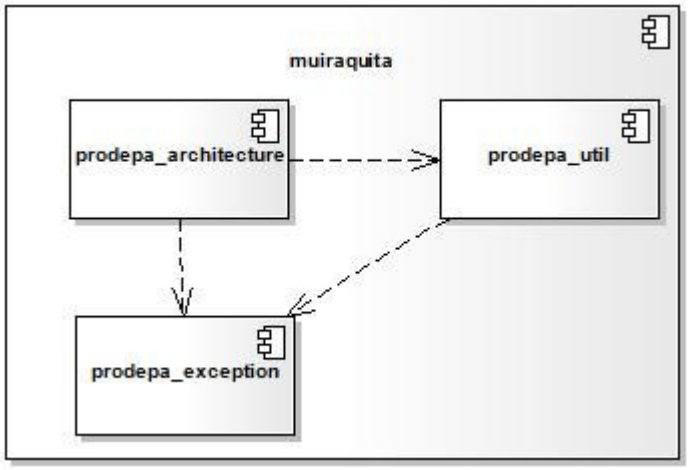

Figura 1. Framework Muiraquitã

a) ***prodepa-architecture***: fornece uma biblioteca de classes reutilizáveis comum a todas as aplicações desenvolvidas sob a Arquitetura PRODEPA, para prover recursos básicos que possibilitam que o projeto mantenha o foco no negócio, em vez da tecnologia. Alguns exemplos desses recursos são: criação de relatório, criação de DAOs genéricos, criação de DTOs, entre outros;

b) ***prodepa-utils***: fornece uma biblioteca de classes que oferecem alguns recursos adicionais já definidos pela linguagem Java, porém adaptados à Arquitetura PRODEPA como, por exemplo, manipulação de arquivos, datas, coleções, entre outros;

c) ***prodepa-exception***: responsável por padronizar as exceções decorrentes do tratamento de erros e validações exigidos pela arquitetura PRODEPA.

Alguns *frameworks* abertos foram adotados como padrões na Arquitetura PRODEPA, como o Jboss-Seam, EJB3 e Hibernate, proporcionando maior

produtividade a partir da reutilização de seus componentes. A seguir, são explicadas essas tecnologias.

a) EJB3: é um modelo de componente padrão do lado do servidor para aplicativos de negócio distribuído. Ele permite que os objetos chamados de *enterprise beans* sejam expostos como serviços *web,* de modo que seus métodos possam ser invocados por outro aplicativo J2EE, e também por aplicativos escritos por outras linguagens de programação em diferentes plataformas. Nesta versão, tem-se o recurso de persistência na própria especificação, chamada de Java *Persistence* API (JPA), que define uma maneira de mapear objetos Java simples (POJO) para um banco de dados relacional [Burke 2007]. É o principal *framework* responsável por prover o acesso distribuído aos serviços disponibilizados pelos sistemas desenvolvidos na arquitetura;

b) Jboss-Seam: unifica o modelo de componentes do JSF e do EJB3, eliminando o código de integração, deixando o desenvolvedor dispensar mais atenção às regras de negócio [Seam Reference 2007]. Segundo Yuan e Heute (2007, p. 5), o Seam foi projetado para aplicações Web com estado (*Statefull*). O *framework* garante, por exemplo, que os componentes de uma conversação sejam excluídos da sessão do usuário caso ela seja finalizada ou expirada;

c) Hibernate: é responsável por mapear objetos Java em tabelas de um banco de dados relacional, estreitando o *gap* conceitual que existe entre os dois, deixando o desenvolvedor livre para se concentrar no negócio da aplicação [Bauer 2007]. Na arquitetura é utilizado por fornecer alguns recursos adicionais no que tange o mapeamento objeto-relacional.

Na seção seguinte é apresentado como elas se integram na arquitetura PRODEPA.

4. Arquitetura PRODEPA

A arquitetura aqui definida utiliza o padrão arquitetural Model-View-Controller (MVC) que, por definição, desacopla as camadas provendo maior flexibilidade. O padrão MVC tem como objetivo básico separar as funções da aplicação (*model*) das funções de apresentação (*view*), permitindo a comunicação entre elas através de um *controller* [Araujo 2009].

A principal meta desta arquitetura é modularizar o *software* de tal forma que possa oferecer facilidades de visualização, reuso e na construção dos componentes, ao mesmo tempo em que busca um alto grau de satisfação nos atributos de confiabilidade, funcionalidade, desempenho e segurança.

Em termos gerais, a arquitetura PRODEPA busca atender alta coesão e baixo acoplamento entre os componentes, pois cada camada possui uma interface que define os serviços que devem ser implementados. A arquitetura possui fundamentos no padrão *facade* [Gamma et al 1995] que define um único ponto de acesso para os recursos do sistema. O domínio da aplicação é tratado em uma camada específica, onde são trabalhadas todas as funcionalidades que deverão ser atendidas pelo sistema, ficando a camada *business* responsável por validar as regras de negócio, e a camada DAO é

responsável por persistir os dados; o transporte de dados entre as camadas se dá por meio de DTO´s. Uma visão geral da arquitetura PRODEPA é vista na Figura 2. Os principais elementos da arquitetura são descritos adiante.

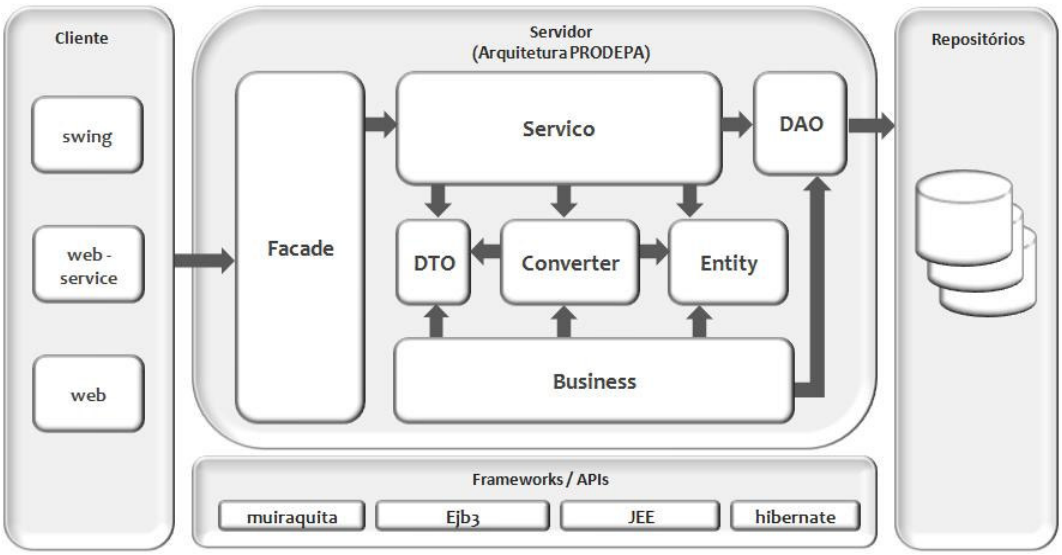

Figura 2. Integração da Arquitetura PRODEPA com demais componentes

4.1. Facade

O *facade* [Gamma et al 1995] é o padrão que simplifica o uso de serviços oferecidos pela aplicação corporativa aos clientes (*web services*, páginas *web*, etc). É a porta de entrada para efetuar uma transação. Funciona como ouvinte da camada de apresentação, ou seja, toda ação efetuada pelo usuário do sistema será "ouvida" pelo *facade* e repassada para o serviço executar.

Uma vez que o *facade* seja visto como uma porta de entrada, é responsabilidade dele receber pedidos de diferente formas, seja por meio de RMI-IIOP [RIIOP, 2009] ou *web-service*s.

4.2. Serviço

A camada de serviço segue o padrão "Roteiro de Transação" [Fowler 2006], que organiza a lógica de negócio em procedimentos, onde cada procedimento lida com uma única solicitação da apresentação. As classes encontradas neste pacote são responsáveis por executar as transações que a aplicação cliente necessita; as classes são compostas por anotações EJB3 e instancia os componentes de persistência.

4.3. DAO

O *Data Access Object* (DAO) é um padrão conhecido na literatura como responsável por intermediar o acesso aos dados armazenados em um banco de dados [Fowler 2006].

Cada DAO deve possuir uma interface, que especifica os métodos de manipulação de dados. Os serviços acessam apenas as interfaces dos DAOs,

desconhecendo a implementação utilizada. Na arquitetura há duas opções para trabalhar com DAO´s, através de instâncias nos serviços ou estendendo a classe *AbstractServico*. Esta é uma classe do *framework* Muiraquitã que já traz encapsulada a lógica de persistência utilizando os *frameworks* de persistência, além de agregar a funcionalidade de geração automática de consultas parametrizadas (usando EJBQL).

4.4. Business

A camada *business* é semelhante ao padrão *Business Delegate* [Alur et al, 2003], isto é, o código cliente limita-se a enxergar invocações locais, e toda a complexidade das chamadas remotas ficam ocultas dentro dele. Caso haja a necessidade de migrar de EJB3 para qualquer outra tecnologia de componentes de negócios, a aplicação cliente permaneceria intacta, ficando as conseqüências limitadas apenas nos *Business Delegates.*

A camada *business* não é obrigatória, é aplicável somente quando existem objetos de interação complexa com outros objetos persistidos ou não.

4.5. Entity

O *entity* [Alur 2003] é o modelo de classes de domínio da aplicação, também conhecido por POJO (*Plain Old Java Objects*), contendo apenas os atributos do domínio do negócio. Objetos *entity* podem ser utilizados tanto pela camada *business*, como à camada DAO, que os utiliza para persistir através de um mapeamento objeto-relacional.

4.6. DTO

São objetos de transporte entre as camadas cliente e servidora (*facade* e serviço). O objetivo é expor somente os contratos estabelecidos nos serviços, fornecendo informações suficientes para realizar cada funcionalidade da aplicação, mantendo alta coesão com baixo acoplamento.

4.7. Converter

O *Converter* é uma camada que tem a função de traduzir o DTO em entidades e vice-versa. Ao expor DTO´s no *facade*, isola-se os modelos de negócio do mundo externo, podendo publicar um serviço *web* sem expor os modelos de negócio a terceiros.

O diagrama de classes apresentado na Figura 3 mostra a integração entre as camadas citadas neste tópico.

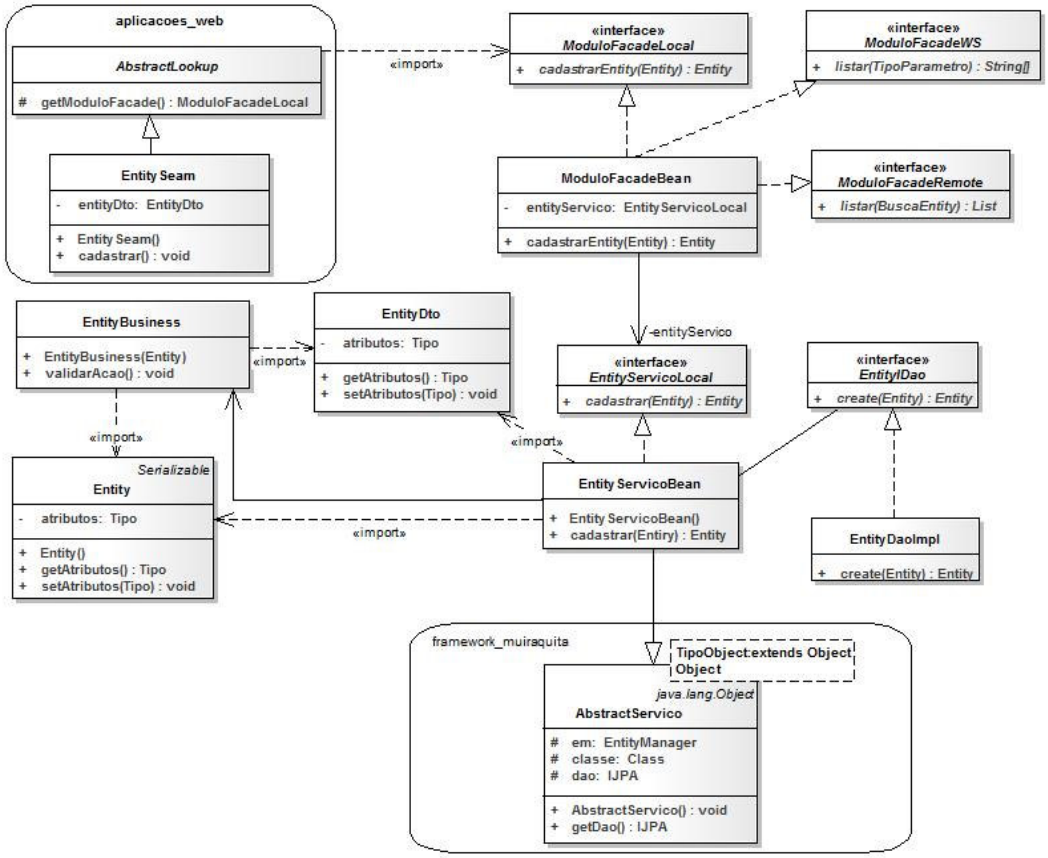

Figura 3. Diagrama de classes - Integração entre as camadas da arquitetura

5. Customização da Arquitetura

De acordo com Hohmann (2003), uma das características principais de uma arquitetura de software é sua flexibilidade e adaptabilidade. A arquitetura deve ser capaz de ser estendida e modificada a partir da contextualização da aplicação, do qual se pode formar um plano para a implementação ou modificação do sistema.

Em termos arquiteturais, a Arquitetura PRODEPA tem como característica a adaptabilidade, possibilitando a instanciação da mesma de forma personalizada. A arquitetura pode ser instanciada e simplificada para atender sistemas de baixa complexidade, que não têm seus serviços distribuídos, e onde esses serviços se restringem basicamente a métodos CRUD (*Create, Read, Update e Delete*) e métodos que não precisam de uma estrutura mais sofisticada.

A instância da arquitetura PRODEPA resultante desta otimização inclui a camada *facade*, serviço (sem injeção de EJBs) e camada *business,* se houver regras de negócio para serem validadas. As camadas de DTO e *converter* são retiradas, e na camada DAO não há necessidade de criação de classes e interfaces para esta camada, pois as classes de serviço herdam os serviços genéricos de persistência já disponibilizados pela classe *AbstractServico*.

6. Estudo de Caso

Frameworks de integração de *middleware* são usados para integrar aplicações e componentes distribuídos. Estes *frameworks* escondem o baixo nível da comunicação entre componentes distribuídos, possibilitando que os desenvolvedores trabalhem em um ambiente distribuído de forma semelhante a que trabalham em um ambiente não distribuído [Junior 2006].

Para demonstrar a utilização da arquitetura no desenvolvimento de sistemas distribuídos foi implementado um sistema de controle de acessos que fornece serviços de autenticação e autorização para diversas aplicações clientes, independente da linguagem utilizada. Na Figura 4, vê-se o diagrama de implantação que apresenta os componentes do sistema de controle de acesso (aplicação servidora) e as interfaces disponibilizadas para as aplicações clientes, dependendo da linguagem utilizada por tais aplicações. A *aplicacao_cliente2,* desenvolvida na linguagem de programação Java, acessa os serviços disponibilizados pela aplicação servidora através da interface remota (*FacadeRemote*); a *aplicação_cliente1*, desenvolvida em PHP, obtém os serviços da aplicação servidora através da interface *web service*. A camada cliente da própria aplicação servidora disponibiliza suas funcionalidades acessando a interface local (*FacadeLocal*).

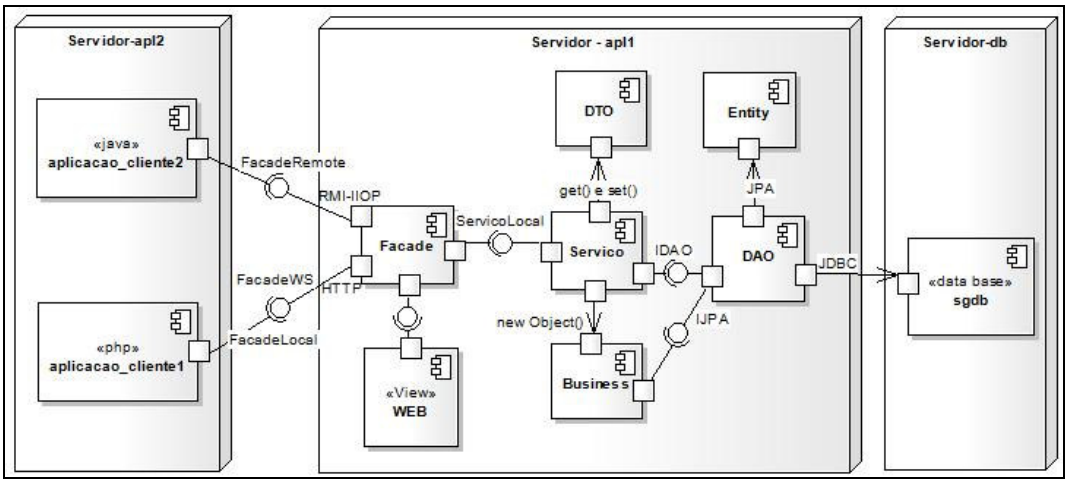

Figura 4. Diagrama de implantação - Integração entre as aplicações distribuídas

A Figura 5 fornece uma visão estrutural de como as classes utilizadas pela aplicação servidora interagem para disponibilizar seus serviços para uma aplicação cliente Java. A classe ModuloFacadeBean e a interface ModuloFacadeRemote fazem parte da camada *facade* da arquitetura PRODEPA e, para disponibilizar seus recursos remotamente, a classe ModuloFacadeBean utiliza anotações disponibilizadas pela API JEE.

A *aplicacao_cliente2*, que também foi desenvolvida na arquitetura PRODEPA, acessa a fachada da aplicação servidora utilizando o método *getProxy*(String), disponibilizado pela classe *BusinessFactory* que faz parte do componente *prodepa_util*, do *framework* Muiraquitã.

Para manipular os dados e acessar os serviços disponibilizados pela aplicação servidora, a aplicação cliente depende do componente cliente (*apl_servidora_cliente*), onde se encontra classes e interfaces disponibilizadas pela aplicação servidora.

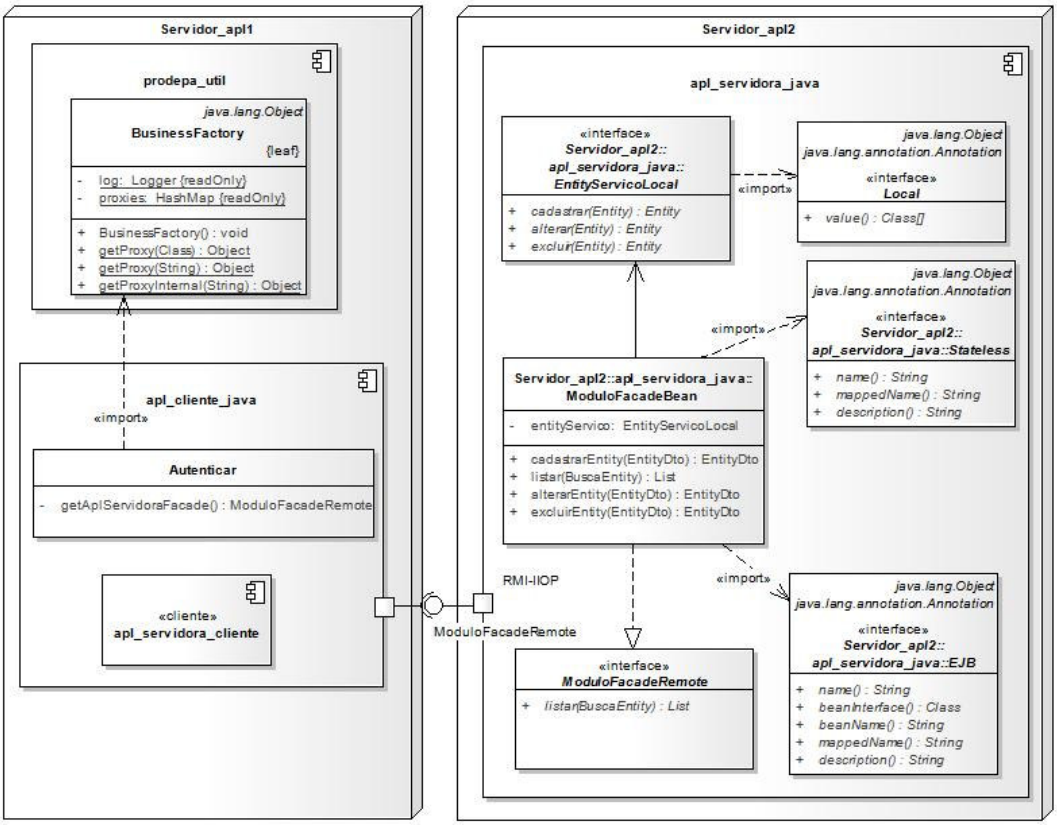

Figura 5. Diagrama de classes - Integração entre as camadas da arquitetura

Na Figura 6, é utilizado um diagrama de seqüência para representar uma visão dinâmica de como ocorre a comunicação entre as duas aplicações distribuídas e dos recursos que são utilizados.

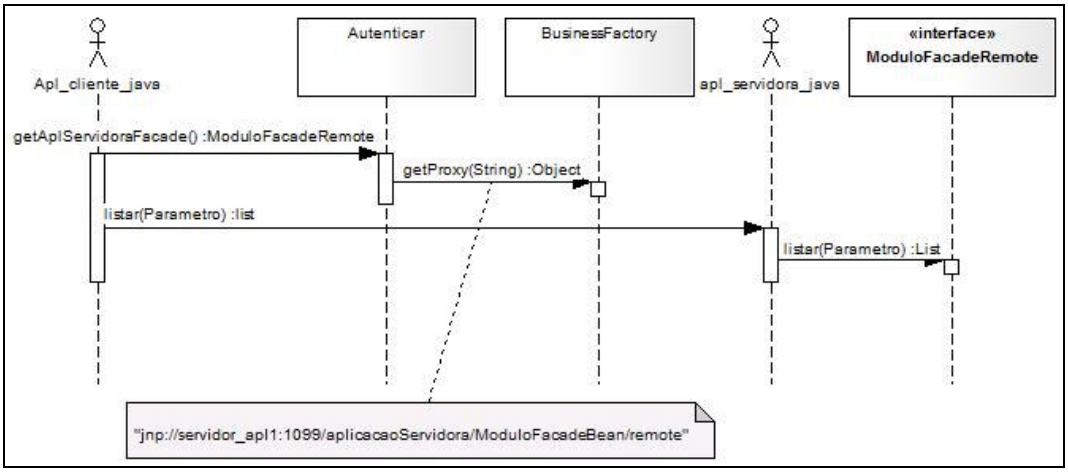

Figura 6. Diagrama de seqüência - Integração entre aplicações distribuídas

Diversas aplicações utilizam serviços disponibilizados pelo sistema de controle de acesso. A maioria dessas aplicações foram desenvolvidas seguindo a arquitetura PRODEPA, como por exemplo, o Sistema de Protocolo e o Sistema de Patrimônio do Estado do Pará.

O uso da arquitetura PRODEPA proporciona maior agilidade no desenvolvimento das aplicações, visto que permite a reusabilidade das estruturas pré-existentes, diminui o esforço dispensado para resolver detalhes de acesso a dados e padroniza o desenvolvimento de sistemas. Tal padronização é fundamental para diminuir a curva de aprendizagem e facilitar a comunicação entre os membros da fábrica de software, pois estabelece um vocabulário comum e um fluxo padrão de tarefas. Desta forma, é possível definir quais componentes serão empregados, que tipo de integração ocorrerá entre eles, e quais *frameworks* auxiliares devem ser utilizados para complementar a solução. Assim, o desenvolvedor é direcionado para resolver problemas relacionados com as regras de negócios que surgem no domínio da aplicação, e não com a construção e adaptação de componentes de infraestrutura.

7. Conclusão e Trabalhos Futuros

O presente trabalho descreveu a arquitetura de software utilizada na Empresa de Processamento de Dados do Pará – PRODEPA, e como ela se integra ao *framework* denominado Muiraquitã, objetivando reusabilidade, maior produtividade e qualidade nos produtos gerados em sua área de desenvolvimento de sistemas.

A arquitetura apresentada tem como diferencial integrar *frameworks* consagrados na plataforma Java, para desenvolvimento de aplicações distribuídas.

O emprego da arquitetura conjuntamente com o framework Muiraquitã trouxeram os seguintes benefícios:

Para a PRODEPA:

a) unificação, padronização e reuso de soluções implementadas por diferentes equipes facilitando a manutenção e evolução de sistemas;

b) maior rapidez no desenvolvimento de novos sistemas;

c) capacitação dos técnicos envolvidos a partir do conhecimento adquirido com os padrões de mercado, o que também elevou a auto-estima da equipe.

Para o Estado:

a) desenvolvimento ágil de soluções com menor custo, permitindo dispensar mais recursos para projetos da área fim de cada órgão;

b) maior integração das soluções adotadas pelos mais diversos órgãos do Estado, o que facilita o combate às fraudes e corrupção.

Para o cidadão:

a) atendimento mais rápido devido à automação dos processos;

b) mais serviços de auto-atendimento são disponibilizados, permitindo que o cidadão não perca tempo em filas de espera, por exemplo.

Como trabalho futuro, pretende-se adicionar o padrão *Factory* [Gamma et al 1995] para criação de objetos de persistência, acrescentar recursos como paginação por demanda e transformar em *frameworks* os módulos de auditoria e controle de acesso, a fim de utilizá-los de forma não intrusiva, facilitando a reutilização destes módulos.

Referencias

Alur, Deepak; Crupi, John; Malks;Dan. Core J2EE Patterns: Best Practices and Design Strategies, Second Edition. Pearson. 2003.

Araujo, D.F. Machado,R.P. Um Framework para Desenvolvimento de Aplicações em Ajax, Disponível em: http://prestesmachado.com.br/artigos/DanielaFeitosaAraujo.pdf , Acesso: 16 mar 2009.

Bauer, C., King G., Java Persitence com Hibernate. Ciência Moderna, 2007

Burke,B., Monson-Haefel,R., Enterprise javaBeans 3.0. 5th. Peason, 2007

Cagnin, M. I. P., uma contribuição para a reengenharia de software baseada em linguagens de padrões e frameworks. São Paulo: USP, Tese de doutorado. Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2005. Disponível em: http://www.teses.usp.br/teses/disponiveis/55/55134/tde-08092005-152316/. Acesso: 12 mar 2009.

Carvalho,A. G. Carvalho Uma Proposta de Arquitetura de Software Genérica para Mobile Games Utilizando J2ME. Disponível em: http://www.unibratec.com.br/jornadacientifica/diretorio/DEFININDO.pdf, Acesso: 12 mar 2009

Demoiselle, Sitio Oficial do Demoiselle Framework. Disponível em: http://www.frameworkdemoiselle.gov.br/menu/framework/download-1/das, Acesso: 30 mar. 2009.

Fowler, M. Padrões de Arquitetura de Aplicações Corporativas. Bookman, 2006.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. Design patterns: Elements of reusable object oriented software. Reading, MA. 1995.

Hohmann, L., Beyond Software Architecture: Creating and Sustaining Winning Solutions. Addison-Wesley Professional, 2003.

Junior, C.G.B., Agregando Frameworks de Infra-Estrutura em uma Arquitetura Baseada em Componentes: Um Estudo de Caso no Ambiente AulaNet. Disponível em: http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0410823_06_cap_02.pdf, Acesso: 21 mar 2009

IEEE 2000. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, IEEE Std 1471-2000. New York, NY: Institute of Electrical and Electronics Engineers.

Minetto, Elton Luís; MASETO, Jhony Maiki. Análise avaliativa entre frameworks de PHP. PHP Magazine, n.2, p.9-15, mar. 2007.

Pinhao Sitio Oficial da Plataforma Pinhão Paraná. Disponível em: http://www.frameworkpinhao.pr.gov.br/modules/conteudo/conteudo.php?conteudo=5#pfc_CelRUP_arquitetura.pdf, Acesso: 18 mar 2009

Pressman, R. S. Engenharia de Software. 5th ed. McGraw Hill. 2001.

Prodepa, Sitio da empresa de processamento de dados do Pará. Disponível em: http://www.prodepa.gov.br/index.php?q=node/117. Acesso: 12 mar 2009

RMI-IIOP (2009): Documentação da tecnologia RMI-IIOP. Disponível em <http://java.sun.com/products/rmi-iiop/>. Acesso: 18 mar 2009.

Seam, Sitio oficial do framework seam Disponível em: http://seamframework.org/Documentation/SeamDocumentation#H-SeamReferenceDocumentation, Acesso: 16 mar 2009

Yuan, M. J.; Heute, T. JBoss Seam Simplicity and Power Beyond Java EE, Primeira Edição, Upper Saddle River: Prentice Hall, 2007.