

Suíte de Produtos TCE-GO
DAS - Documento de Arquitetura de Software

Versão 0.1

Histórico de Revisões

Data	Versão	Descrição	Autor
08/03/2016	0.1	Criação da estrutura do documento, editando o documento de arquitetura de software da Suíte d produtos do TCE-GO.	Igor Vinicius

Introdução

Propósito

Escopo

Referências e Acrônimos

Diretrizes

Focos da Arquitetura

Infraestrutura

Camadas

Interface com Usuário (“UI”)

Interfaces e Fábricas (“InterfacesFabricas” - Contratos)

Serviço (“Servico”)

Serviços (“Servicos”)

Repositório (“Repositorio”)

Negócio (“Negocio”)

Banco de Dados (SGBD)

Componentes

Padrões

Sistemática de Qualidade

Qualidade Interna

Qualidade Externa

Descrição do Modelo de Arquitetura

TCE.Compartilhado

TCE.Componentes

Camadas

Interface com Usuário (“UI”)

Interfaces Fábricas

Serviço

Repositório

Negócio

Ferramentas e Tecnologias

1. Introdução

1.1. Propósito

A finalidade do Documento de Arquitetura de Software - DAS da Suíte de Produtos do TCE-GO é definir o modelo arquitetural dos produtos de software. Esse modelo deverá ser utilizado no desenvolvimento de todos os produtos novos e produtos que foram migrados para essa nova arquitetura de software.

O modelo arquitetural é baseado em SOA com o objetivo de seguir princípios de qualidade como: Alta Coesão, Baixo Acoplamento, Reutilização, Integração, Verificação e Validação, Documentação de Código e aplicação de Design Patterns.

1.2. Escopo

Este documento contempla a definição do modelo arquitetural de software, os padrões de software utilizados, componentes padrão utilizados no desenvolvimento dos software - como frameworks, ferramentas de desenvolvimento, servidores de aplicação e de banco de dados.

São apresentados nesse documento, a sistemática de integração dos produtos de software, apresentando a forma de comunicação desses produtos na visão interna e externa do ambiente físico do TCE-GO.

Esse documento também será utilizado como manual de desenvolvimento dos novos produtos de software, de modo a orientar a equipe de desenvolvimento na criação e configuração dos componentes padrão utilizados pelos produtos.

1.3. Referências e Acrônimos

DAS - Documento de Arquitetura de Software;

TCE-GO - Tribunal de Contas do Estado de Goiás;

DTO - Objeto de Transferência de Dados (Driver Transfer Object);

SOA - Arquitetura Orientada a Serviço (Service Oriented Architecture);

GER-TI - Gerencia de Tecnologia da Informação;

UI - Interface com o Usuário (User Interface);

MVC - Modelo Visão Controlador (Model View Controller);

DLL- Biblioteca de Vinculo Dinâmico (Dynamic link library);

CRUD - Insere, Consulta, Atualiza e Deleta (Create, Read, Update, Delete);

SGBD - Sistema Gerenciador de Banco de Dados;

TAD - Tipo abstrato de Dado;

2. Diretrizes

1. A arquitetura das soluções de software do TCE-GO está voltada primeiramente para a integração dos produtos, de modo a tornar cada produto específico para o seu propósito, e ser reutilizado em outras soluções através das chamadas via serviço.

- II. Todas as UIs seguiram o padrão MVC, tanto os projetos Web quanto Desktop ou Console. Porém, os objetos do Modelo serão objetos do tipo DTO, os quais serão mantidos ou adaptados com base nas regras de acesso por usuário e interface, UI ou de Serviço.
- III. A camada UI dos projetos, não conheceram a camada de Serviços concretos, nem a de Repositório e nem a de Negócio. O acesso aos serviços serão mantidos e tratados pela camada de Interfaces e Fábricas (*InterfacesFabricas*), a qual será responsável por estabelecer a comunicação do serviço solicitado pela UI, com a instância correspondente, seja via DLL ou via *Web Service*.
- IV. Todas as operações do sistema será realizada via serviço, o qual ficará responsável por validar a entrada de dados, controlar o acesso, e dar continuidade às operações executadas chamando outro serviço ou acessando as operações do repositório.

3. Focos da Arquitetura

O modelo arquitetural estabelecido nesse documento, será possível a partir da utilização das especificações seguintes:

3.1. Infraestrutura

As soluções de software serão criadas com base no seguinte modelo arquitetural de projetos que segue:

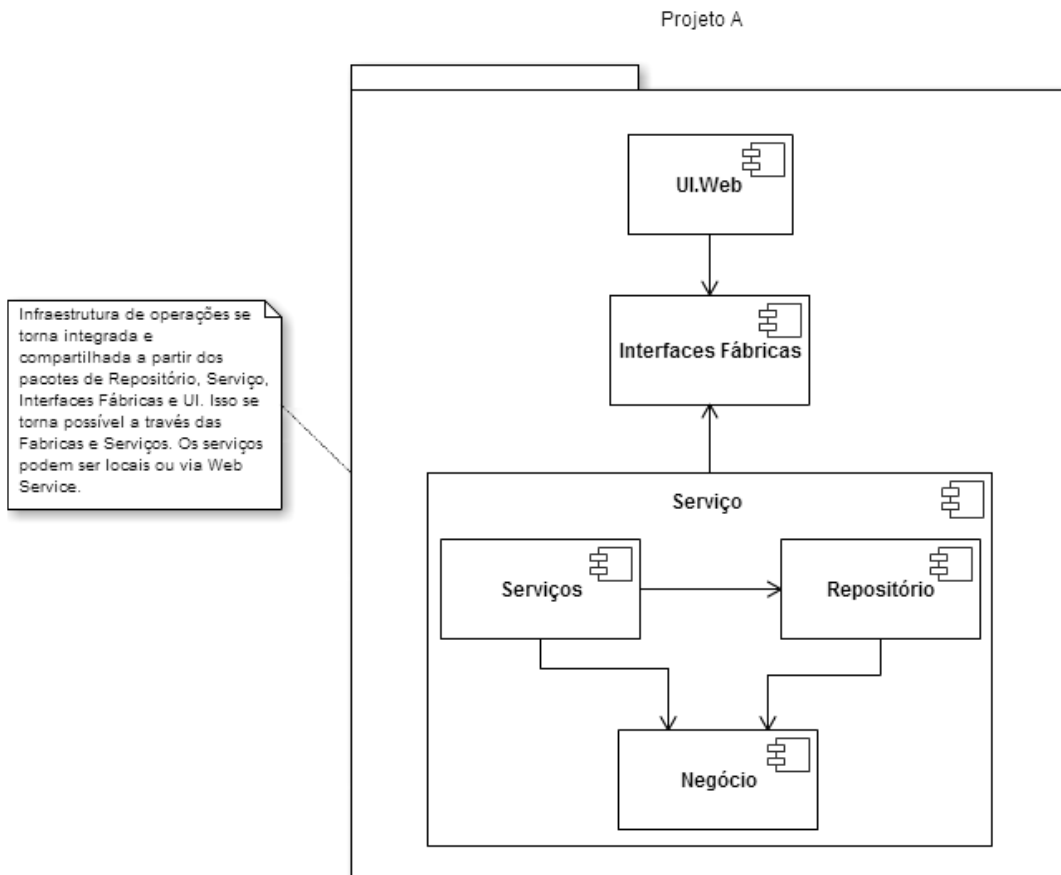
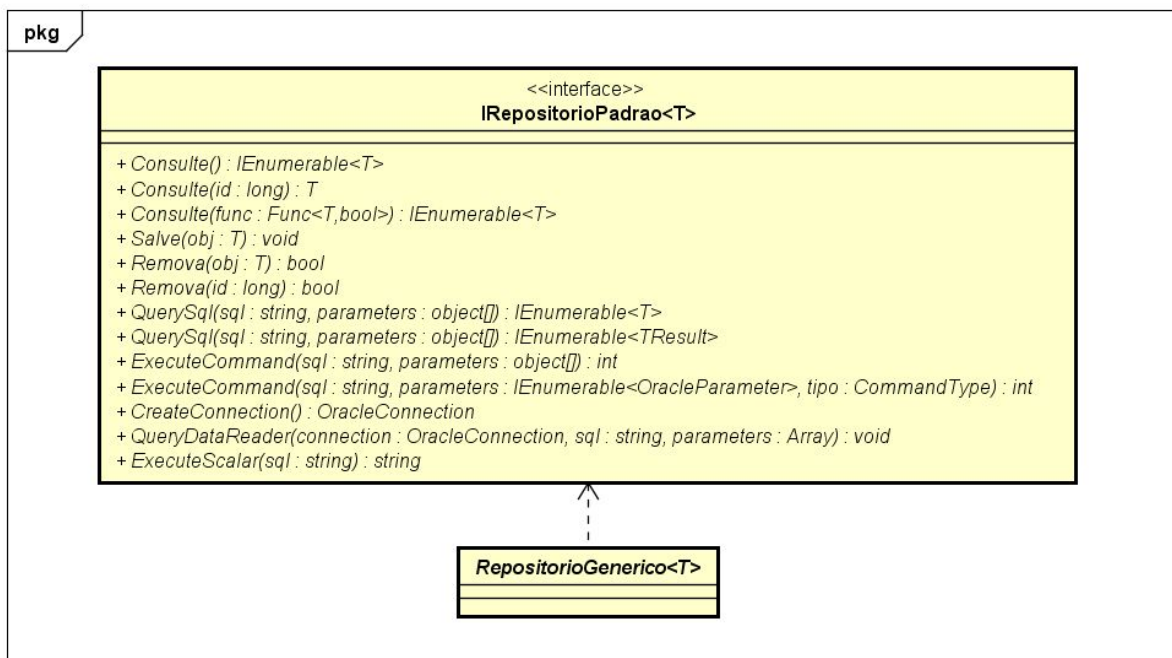


Figura 1 - Arquitetura de Pacotes

Essa infraestrutura de informação está consolidada na solução TCE.Compartilhado, a qual servirá como especificação de objetos de negócio a partir da herança da classe abstrata “ObjetoPersistido.”, também especifica o contrato padrão das interfaces de repositório e de serviço, bem como as classes de implementação padrão de repositório e de serviço, “RepositorioGenerico” e “ServicoPadrao”, respectivamente.

A implementação padrão do repositório realiza as principais operações de CRUD com o banco de dados, como apresentado no diagrama abaixo:

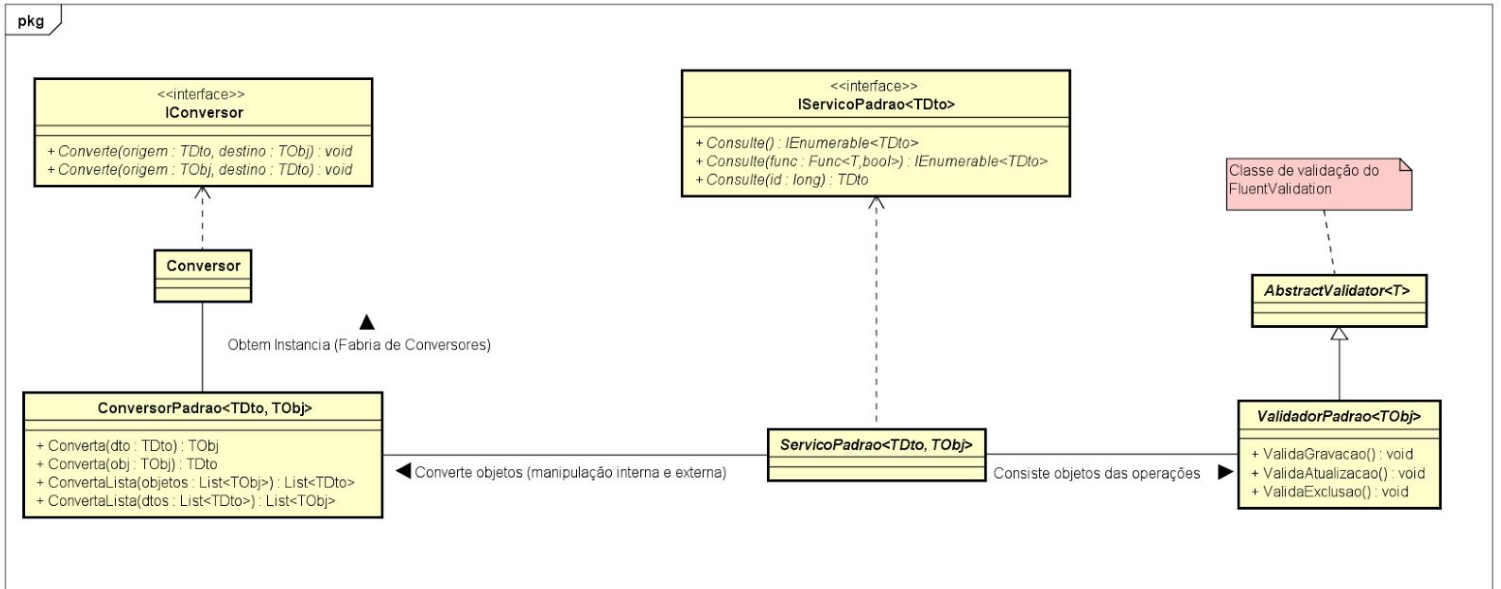


powered by Astah

Figura 2 - Diagrama de Classes - Repositório

Já a implementação padrão do serviço além de realizar as principais operações de CRUD, executa nas operações padrão de manipulação dos objetos (DTOs), as validações padrão, definidas na classe padrão de validação, “ValidadorPadrao”. Por esse motivo, todo serviço que herde da implementação padrão do serviço, deverá ter uma classe de validação, que herde da implementação padrão dos validadores, “ValidadorPadrao”.

A implementação padrão de validação de objetos, utiliza o framework *FluentValidation*, o qual possui várias funções de validação de objetos, customizadas nesta implementação padrão.



powered by Astah

Figura 3 - Diagrama de Classes - Serviço

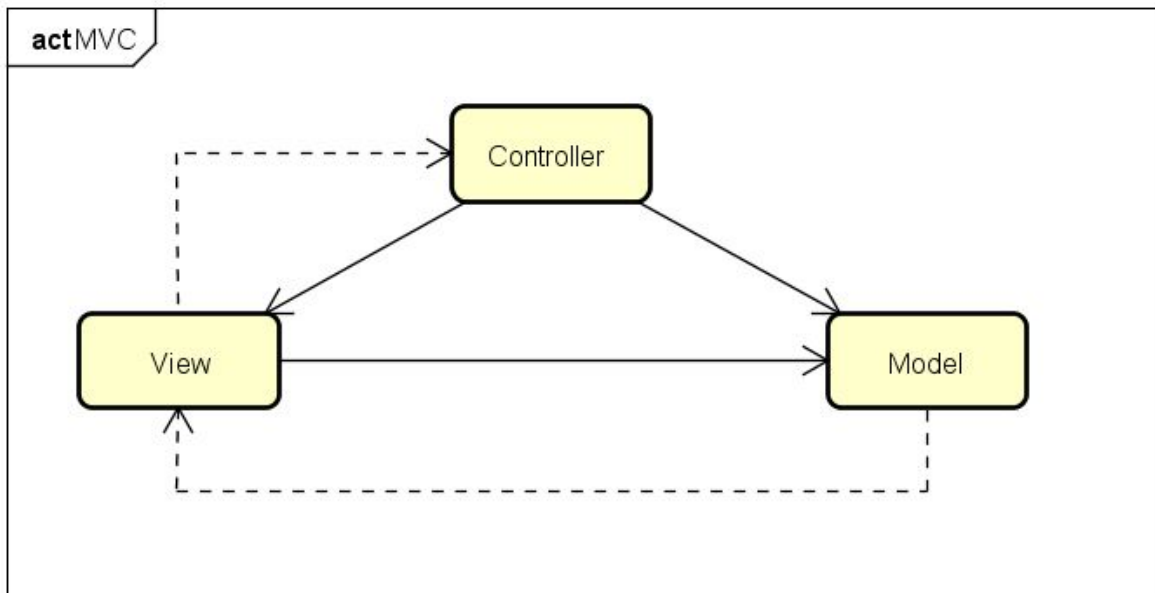
A UI realiza as operações com o sistemas utilizando DTOs. Cada DTO contém atributos e operações de acordo com as permissões de uso do usuário, seja usuário final ou outros sistemas. Todas as funcionalidades manipuladas pela UI, serão realizadas sempre pela camada de serviços, que por sua vez será obtida via contrato (Interfaces), instanciadas pelas fábricas, que poderão ser instanciadas em uma chamada local de DLL ou remota, via *Web Service*.

3.2. Camadas

A implementação em camadas é uma abordagem de decomposição granular da solução de software, a fim de permitir a manutenção e substituição das partes sem afetar o todo, e de forma a aumentar a coesão de cada camada, identificada na arquitetura como Macro Pacotes.

3.2.1. Interface com Usuário (“UI”)

Toda interface com o usuário será implementada utilizando o padrão MVC, porém o Model será implementada pela camada de Interfaces Fábricas (*InterfacesFabricas*), através da manipulação de objetos DTO.



powered by Astah

Figura 4 - Fluxo MVC

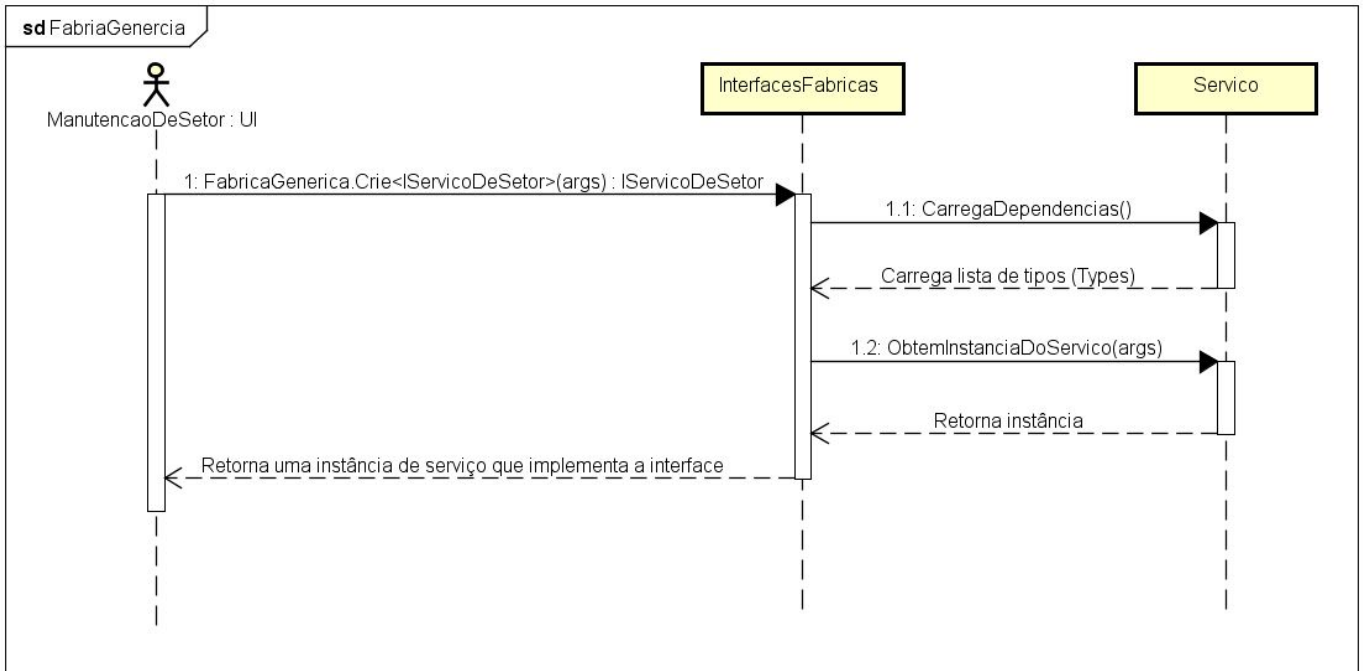
Os controladores executam as operações através dos serviços implementados, fazendo a chamada sempre via contrato (interface). Por esse motivo, as UIs não conhecem a implementação concreta dos serviços, as chamadas dos contratos serão realizadas via fábrica (fábrica genérica ou fábrica específica) ou via *Web Service*.

As *Views* da UI, não terão qualquer implementação de regra de negócio, devendo se limitar apenas no tratamento de apresentação dos dados ao usuário e converter as informações de entrada do usuário em objetos de transito , DTO, para que esse seja enviado na chamada da funcionalidade pelo serviço. Os serviços são responsáveis de validar as informações transitadas (DTOs), verificar as regras e controles de acesso, e executar as operações solicitadas pelo usuário através das UIs.

3.2.2. Interfaces e Fábricas (“*InterfacesFabricas*” - Contratos)

Essa camada é responsável por definir os contratos de operações dos serviços - interfaces de serviço, os objetos de transito (DTOs) e as fábricas de objetos e de operações.

O projeto TCE.Compartilhado, utilizado como base de implementação para os demais produtos de software do TCE-GO, disponibiliza a fábrica de instanciação de serviços em chamadas locais, a partir da fabrica genérica. Essa fábrica instancia de forma referencial, via reflexão, as implementações padrão dos serviços especificados pelos pacotes de Interfaces Fábricas dos projetos.



powered by Astah

Figura 5 - Diagrama de Sequência - Fábrica Genérica

3.2.3. Serviço (“*Servico*”)

A camada de serviço é composta por três pacotes principais, o pacote de Serviços, de Repositório e o de Negócio.

Essa camada é responsável por manter e controlar todas as regras de negócio, dês da validação das entradas e saídas das operações dos serviços até a persistência das informações no banco de dados.

Todas as chamadas de funcionalidades de manipulação de dados de outros produtos de software, serão realizadas por essa camada, através das chamadas de serviços locais ou via *Web Service*.

3.2.3.1. Serviços (“*Servicos*”)

O pacote de Serviços será responsável pela implementação dos serviços concretos das interfaces de serviços estabelecidos na camada de Interfaces Fábricas.

A entrada de dados via serviço será sempre através de tipos primitivos de dados ou via DTO. Já a manipulação das informações internamente ao serviço, poderá ser executada a partir de objetos de negócio, os quais essa camada conhece.

As saídas das operações do serviço, serão sempre via DTO, sempre em estruturas serializáveis, de modo a possibilitar a chamada do mesmo serviço em requisições remotas via *Web Services*.

Os serviços herdados da implementação padrão (*ServicoPadrao.cs*), sempre será necessário implementará um validador e um conversor do objeto de negócio para o objeto de

transferência (DTO). Já os serviços que não herdam da implementação padrão, podem especificar a forma de manipulação de dados, porém se necessário validar as entradas e saídas de informação, o validador deve herdar da implementação padrão de validador, assim como o conversor de DTO para objeto de negócio e vice e versa, caso o serviço manipule as informações de entrada a nível de negócio.

3.2.3.2. Repositório (“*Repositorio*”)

A camada de Repositório é responsável por realizar as operações com o banco de dados, de modo a encapsular a complexidade do mapeamento dos objetos de negócio com modelo relacional do banco de dados.

As operações padrão de CRUD e funções úteis de manipulação de SQL estão implementadas na classe abstrata do Repositório Genérico (*RepositorioGenerico.cs*), o qual obedece a interface padrão do Repositório Padrão (*IRepositorioPadrao.cs*).

Esse pacote será estruturada da seguinte forma:

- **Interfaces** - pacote onde serão especificadas as interfaces dos repositórios dos objetos de negócio, com as operações públicas.
- **Mapeadores** - nesse pacote serão registradas as classes de mapeamento dos objetos de negócio com suas tabelas correspondentes do banco de dados. Essas classe utilizam o mapeador *EntityTypeConfiguration* do framework Objeto-Relacional, *Entity Framework*, que trata a complexidade de configuração das propriedades dos objetos de negócio com as colunas da tabela do banco de dados correspondente.
- **Repositórios (*Repositorios*)** - pacote onde serão implementadas as classes concretas dos interfaces de repositório definidas no pacote Interfaces. No produto TCE.Compartilhado, utilizado como base de implementação dos produtos na arquitetura que é tratada nesse documento, traz a implementação padrão dos repositórios, a classe abstrata Repositório Genérico (*RepositorioGenerico.cs*), a qual deverá ser herdada na implementação dos repositórios que utilizaram o *Entity Framework* como *framework* de persistência de objetos.

3.2.3.3. Negócio (“*Negocio*”)

O pacote de Negócio contem as classes de objetos de negócio, implementadas com base na modelagem abstraída da análise de requisitos de usuário. Todas as classes persistidas no

banco de dados, devem herdar da classe abstrata *ObjetoPersistido*, para que o repositório genérico possa manipular em suas operações padrão.

3.2.4. Banco de Dados (SGBD)

O TCE-GO utiliza como banco de dados principal para a suíte de produtos, o banco Oracle 11g, porem existem projetos mantidos pela equipe da GER-TI, que utilizam os bancos de dados SQL Server 2012, MySQL e Postgres.

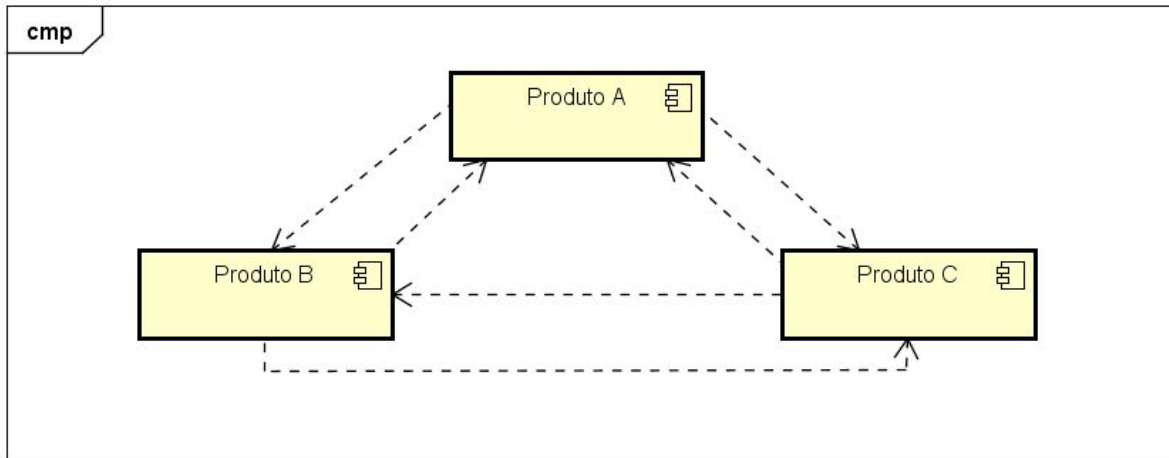
A modelagem do banco de dados principal, Oracle 11g, é realizada pela ferramenta Oracle Design.

3.3. Componentes

O modelo arquitetura definido nesse documento, modela os produtos de software em uma estrutura de componentes, ao separar cada produto em pacotes específicos. Essa abordagem facilita o desenvolvimento dos produtos de software, possibilitando não somente uma melhor manutenibilidade, mas a possibilidade de dividir esforços da equipe técnica na concepção de projeto de software dividindo a construção por componentes (pacotes) e permite a substituição de todo ou em partes dos componentes que formam o produto.

Partindo de uma visão *top-down*, cada produto de software concebido nesse modelo arquitetural, torna-se um componente de negócio que compõem a Suíte de Produtos do TCE-GO. Podemos afirmar isso, por que cada produto implantado localmente em uma solução ou disponibilizado de forma remota, está acessível a outro projeto, por meio da chamada dos serviços disponíveis no componente de negócio. Como os serviços e a infraestrutura de acesso a esses serviços, são baseadas no estilo arquitetural SOA, que estabelece a centralização das funcionalidades através do uso de Serviços de Software.

Com base nessa análise *top-down*, podemos visualizar a suíte de produtos do TCE-GO da forma representada no diagrama que segue:



powered by Astah

Figura 6 - Diagrama de Componentes - Integração dos Produtos

3.4. Padrões

Os projetos implementados baseados nessa arquitetura deverão seguir o padrão de nomenclatura dos projetos, o padrão estrutural de projetos (pacotes) da solução, bem como a organização estrutural dos pacotes interno dentro de cada projeto, como apresentado nas figuras abaixo:

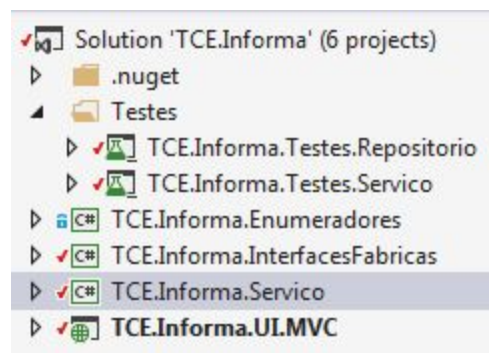


Figura 7 - Padrão Estrutural de Projetos

Deve ser observado o padrão de nomenclatura dos projetos de cada solução (produto de software) implementada baseada nesse modelo arquitetural. Todos os projetos deve ter como prefixo, o nome "TCE.". Esse prefixo é utilizado como identificador dos produtos que utilizam a arquitetura definida nesse documento. A partir desse identificador, e utilizando o padrão de projeto *Factory Method Pattern*, os serviços e repositórios disponíveis em cada produto da suíte ficarem acessíveis em qualquer produto, utilizando a função "Crie", da Fábrica Genérica ("*FabricaGenerica*"), a partir da chamada via contrato (Interface).

O padrão estrutural dos pacotes por projeto, seguirá o exemplo apresentado nos itens abaixo:

- Camada Interfaces Fábricas (*InterfacesFabricas*): a organização das subpastas de segundo e terceiro nível, organizadas com base no

propósito de cada pacote, deverá obedecer a organização dos pacotes da camada de Negócio, localizada por padrão no projeto de Serviço, no pacote “Negocio”.

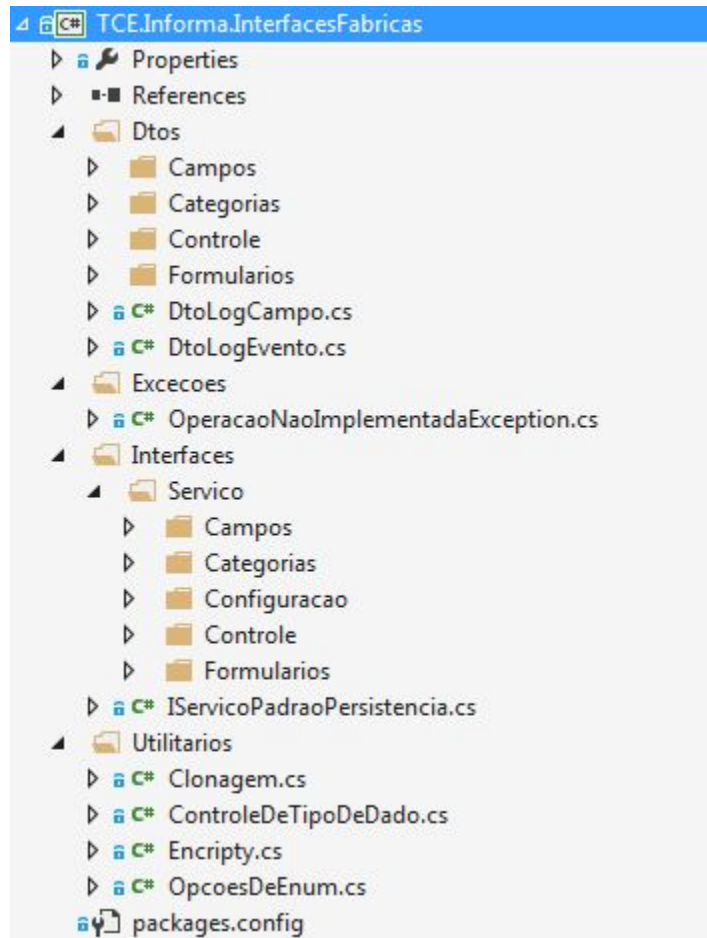


Figura 8 - Padrão do Projeto InterfacesFabricas

- Camada Serviços (*Servico*): a estrutura de pacotes utilizado no sub pacote de Negócio, também serão organizadas as classes contidas nos pacotes de “Conversores”, “Repositorio/Interfaces”, “Repositorio/Mapeadores”, “Repositorio/Repositorios”, “Servico” e “Validadores”, como apresentado na figura abaixo.

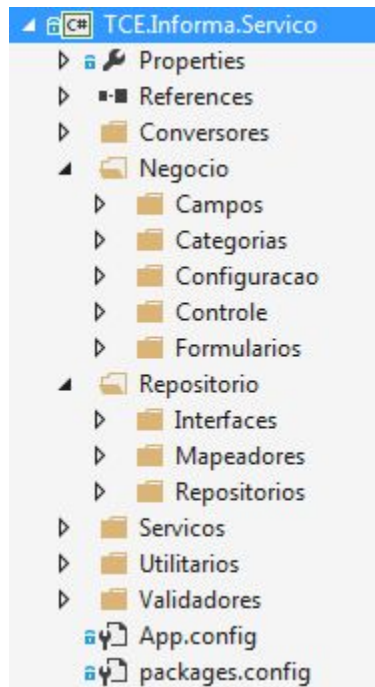


Figura 9 - Padrão do Projeto Servico

- Camada de Testes: o nome dos projetos de teste deve conter antes do nome da camada a qual será testada o prefixo “Testes”. Por padrão, os projetos de teste inicialmente serão dois, os de teste da camada de repositório e o da camada de serviço. Não obstante a isso, caso o produto de software realize integração com produtos de terceiros, nesse projeto terá outro projeto de testes cujo sufixo será “Integracao” (Integração), que será responsável por verificar as chamadas integradas dos serviços do projeto do TCE com os serviços de terceiros. Para projetos que vierem a ter teste de UI automatizado, o projeto de testes deverá respeitar o mesmo padrão de nomenclatura, por exemplo para o projeto TCE.Informa, o nome seria “TCE.Informa.Testes.UI.MVC”. Os projetos de teste de repositório deverá sempre testar a comunicação e operações reais com o banco de dados em ambiente de homologação. Já os projetos de testes de serviço, poderá mocar o repositório e outras estruturas de dados ou serviços que fogem da competência da equipe da GER-TI garantir a funcionalidade. A API de moque de objetos utilizadas pelo TCE-GO, será o RhinoMocks.

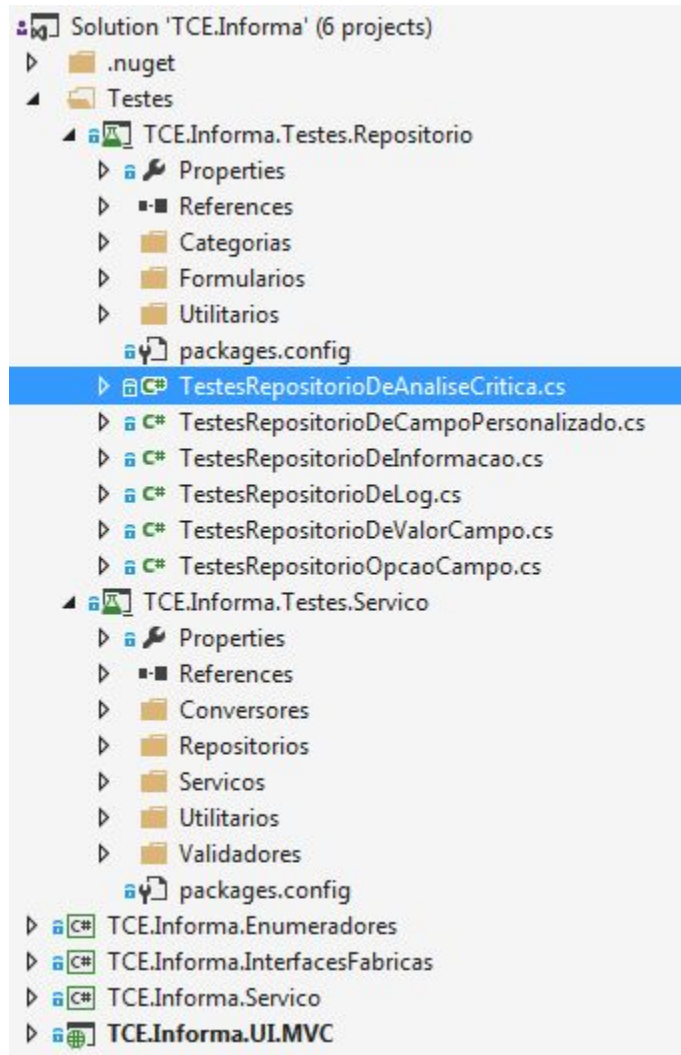


Figura 10 - Padrão do Projetos de Testes

- Camada UI: todos os projeto UI, independente do tipo Web, Desktop ou Console terão o identificado no nome do projeto que identifica que o projeto é UI e qual o padrão utilizado, por exemplo: se for um projeto UI Web MVC, utilizando o produto Informa, o nome do projeto será “TCE.Informa.UI.MVC”. Se o produto Informa tiver uma interface UI Desktop, o nome do projeto seria “TCE.Informa.UI.Desk”. E se tivesse uma interface *Console Application*, o nome do projeto seria “TCE.Informa.UI.Console”. Os projetos UI Web seguindo o modelo MVC, devem respeitar a organização estrutural apresentada na figura abaixo.

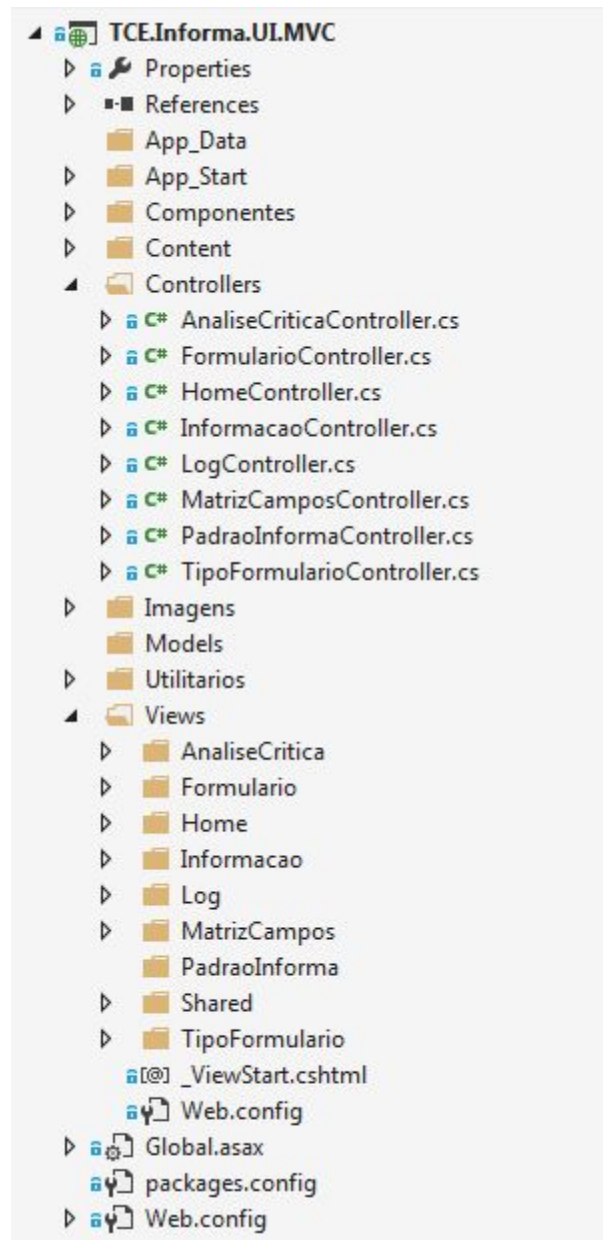


Figura 11 - Padrão do Projetos UI MVC

4. Sistemática de Qualidade

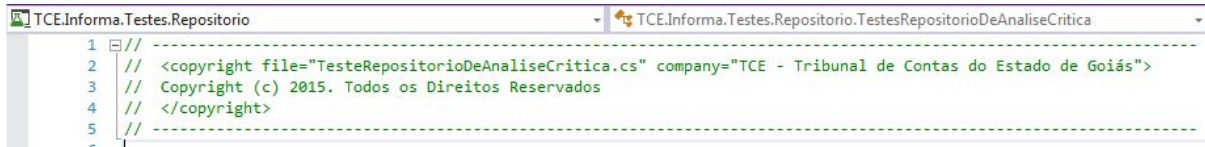
A qualidade dos produtos desenvolvidos utilizando essa modelo arquitetural não se limita apenas na verificação e validação das regras e estruturas de dados especificados nos documentos de requisitos. Para garantir a qualidade interna e externa dos produtos de software, foram estabelecidos alguns padrões de codificação, os quais seguem abaixo organizados por visão de qualidade:

4.1. Qualidade Interna

A qualidade interna do produto de software será determinada quando a estrutura lógica do código, quanto a organização dos métodos e variáveis, quando a capacidade de apreensibilidade do código lido, nomenclatura de métodos e variáveis e quanto ao tamanho das estruturas (quantidade de linhas de código por método e classe).

Para estabelecer um padrão de qualidade interna dos produtos de software aderentes à arquitetura de software que trata esse documento, foi definido os seguintes padrões de código:

- **Estilo de código** - o padrão de estilo de código utilizado nos projetos será o padrão determinado pela Microsoft, com alterações definidas pelas estruturas de Style Cop adotadas pela equipe de desenvolvimento da GER-TI. Todo arquivo de classe de projeto deve ter o cabeçalho contendo a propriedade intelectual do TCE-GO, como apresentado na imagem abaixo.



```
1 //
2 // <copyright file="TesteRepositorioDeAnaliseCritica.cs" company="TCE - Tribunal de Contas do Estado de Goiás">
3 // Copyright (c) 2015. Todos os Direitos Reservados
4 // </copyright>
5 //
```

Figura 12 - Cabeçalho de Classe

- **Estrutura do código** - a estrutura interna das classes serão organizadas inicialmente pelas variáveis, propriedades, métodos e por último as classes internas a classe, sendo essa última deve se restringir apenas ao escopo da classe que o contem, caso essa regra não seja atendida, a classe interna deverá ser retirada da classe mãe e ser incluída em um arquivo de classe separadamente. Partindo de uma visão macro das estruturas das classes, as variáveis, propriedades e métodos serão organizadas da seguinte forma: inicialmente as variáveis e estas se organizado com base no nível de acesso (*private*, *internal*, *protected* e *public*), posterior mente as propriedades e métodos, organizados com base no nível de acesso de cada estrutura, inicialmente pelo acesso, público estático, em seguida o acesso público não estático, seguido pelo acesso interno, protegido e por ultimo pelo privado. A organização entre as estruturas do mesmo tipo são apresentados abaixo:
 - **Variáveis** - as variáveis de escopo da classe serão organizadas na seguinte ordem: variáveis constantes serão as primeiras iniciadas a partir das públicas, seguida das internas, posteriormente as protegidas e por fim as privadas. Já as variáveis diferente de constantes serão ordenadas após as constantes em ordem de acesso iniciadas a partir das públicas, seguida das internas, posteriormente as protegidas e por fim as privadas.
 - **Propriedades da classe** - as propriedades da classe serão organizadas iniciando pelas propriedades públicas, em primeiro as estáticas, seguidas das públicas não estáticas, posteriormente pelas

internas, protegidas e por último as provadas, sendo os construtores de classe organizados em primeira posição.

- **Métodos** - os métodos serão organizados inicialmente por métodos públicos iniciados pelo tipo estáticos, seguidos de métodos internos, protegidos e posteriormente por métodos privados. Seguindo o manual de boas práticas de construção de código, *Clean Code*, os métodos das classes devem ser estruturados de acordo com a dependência entre si, isto é, os métodos utilizados por um método deve ser alocado logo abaixo desse método, de modo que o código seja auto explicativo, sempre sabendo que o detalhe da execução do método vem após a chamada.
- **Comentário de Bloco** - o comentário de bloco será permitido para os métodos de interfaces e classes de forma obrigatória para os métodos públicos e facultativo para os métodos privados. Para os métodos de classes que utilizam a especificação por interfaces, o comentário do método pode ser único pela interface, ficando facultativo replicar o comentário no método da(s) classe(s) concreta(s), devendo essas serem sempre atualizadas caso o comentário sofra alterações.
- **Comentário de Linha** - o comentário de linha de código não poderá ser extenso, devendo se limitar, de forma intuitiva, a explicação do uso da estrutura comentada e seu propósito.
- **Nomenclatura de Variáveis e Métodos** - não existe limitação quanto ao tamanho do nome para variáveis e métodos. O nome dessas estruturas deve sempre ser intuitivo de modo a facilitar o entendimento do propósito e para que serve a estrutura, dessa forma o nome das mesmas devem ser auto explicativas. O nome dessas estruturas que utilize mais de uma palavra, para as variáveis deve iniciar com letra minúscula e para métodos com letra maiúscula, e para ambas as palavras subsequentes a primeira deve sempre iniciar com letra maiúscula. As palavras que compõem o nome dessas estruturas deve conter ao menos duas letras, para não ferir o padrão de iniciar maiúsculo da segunda palavra em diante. O nome de métodos deve iniciar sempre com verbo, de forma a indicar a ação executada pela estrutura, como "Obtenha", "Consulte", "Salve", "ExisteLancamento" etc.
- **Nomenclatura de Classes** - toda classe preferencialmente deve ser um TAD, de modo a representar objetos do mundo real. Porém, há possibilidade de ser necessário criar classes úteis funcionais para determinadas operações típicas, como classes de extensão, classes de verificação de tipos conhecidos, como validação de CPF e CNPJ, entre outras. Para isso essas classes utilitárias, devem ser alocadas em um pacote intitulado como "Útil" ou "Utilitários", as quais devem agrupar as funções de mesmo propósito, isto é, a classe útil de verificação de CPF e CNPJ, deve concentrar esses métodos de verificação na mesma classe que poderá ser chamada de "ValidadorUtil" ou "FuncoesUteis". Já as classes de extensão podem ter o prefixo ou sufixo "Helper".
- **Tamanho da estrutura** - as classes não têm tamanho máximo fixado, porém, classes acima de 2 mil linhas de código devem ser refatoradas para

classes de controle específico, de modo a granular suas operações separando por escopo. Já os métodos, devem ser o mais curto possível, tendo como tamanho máximo 200 linhas de código. Ao analisar uma estrutura, deve contar como sendo uma linha de código estruturas do tipo *Linq* e do tipo *Lambda Expression*, e também para tratamento de exceções, estrutura *catch*, para as tratativas que não ultrapassem 10 linhas de código por tratamento de tipo de exceção. Para operações que requerem *loops* e condições que atingem uma alta complexidade, estas devem ser quebras em métodos ou alteradas para funções recursivas.

4.2. Qualidade Externa

A qualidade externa nos produtos de software desenvolvidos na arquitetura descrita nesse documento será medida tanto pelo uso de técnicas de Verificação e Validação, quando pela Qualidade de Uso.

Como a abordagem arquitetural está baseada na granularidade do projeto, separando as estruturas granulares em forma de componentes, em micro e macro escopo, todos os componentes são testados a nível de Verificação e Validação.

A verificação dos componentes se dá pela construção e execução de testes de unidade de negócio e repositório, a partir dos projetos de teste dos componentes de Unidade e de Repositório, o primeiro de classes e estruturas dos componentes de negócio e o segundo dos componentes de repositório (mapeadores e operações com o banco de dados).

A validação será realizada através dos projetos de testes de serviços, de integração e a nível funcional através de Testes Funcionais de Usuário. Como esse documento se limita ao escopo de arquitetura de software, será tratado aqui apenas no tocante a testes dos componentes de Serviço e de Integração.

Os testes de unidade, de repositório, de serviço e de integração dos projetos, serão projetos de Teste da plataforma .Net, do tipo Unit Test Project, o qual utiliza o framework de teste da plataforma .Net.

A elaboração dos testes devem se atentar ao escopo de cada estrutura a ser testada, o que significa que os testes de estruturas mais complexas como exemplo os serviços, não precisam testar a matéria de operação com banco de dados, visto que por padrão essa operação será verificada pelo teste de repositório da classe de repositório que o serviço faz uso. Da mesma forma para os testes de serviço que fazem integração com produtos de terceiros, a verificação da integração será escopo do teste de integração, por tanto, o teste de serviço deve garantir o escopo do serviço, as operações e estruturas que o serviço mantém, partindo do pré-suposto que a integração está funcionando corretamente. Partindo dessa concepção, a execução dessas estruturas anteriormente testadas, na execução dos testes deveram ser mocadas, para não tornar o processo de teste mais honroso ou impossível de ser realizado em um tempo hábil.

Os testes de repositório, como farão operações no banco de dados, o desenvolvedor deve manipular registros que seu teste venha a criar para que ao terminar os testes, esses registros de teste sejam apagados, para não deixar lixo no

banco de dados, salvo os registros utilizados nos testes que não foram criados pelo teste, como registros padrão do sistema, como descritores, setores e até usuários.

Os testes de cada estrutura - classe, funções e métodos, devem atentar para verificar o espaço amostral válido, inválido e de exceção, que verifique ao menos uma opção em cada extremo desse espaço amostral, e deve atentar também percorrer por todos os caminhos e todas as condições das estruturas implementadas. A fim de garantir que os produtos desenvolvidos estejam sempre igual ou acima da meta de qualidade do processo de desenvolvimento de software da GER-TI do TCE-GO.

5. Descrição do Modelo de Arquitetura

O departamento de Gerência de Tecnologia da Informação (GER-TI) do TCE-GO, desenvolve produtos de software na plataforma .Net, mas mantém outros produtos em outras linguagens e plataformas de desenvolvimento de software.

A arquitetura de software descrita nesse documento não tem restrição de linguagem de programação ou plataforma de desenvolvimento, entretanto, como a GER-TI, mantém e desenvolve, por padrão, produtos na plataforma Microsoft .Net, e como a equipe tem mais afinidade em desenvolver software na linguagem de programação C#, os projetos desenvolvidos nessa arquitetura, será sempre em linguagem C#.

Não obstante a essa restrição, caso a por motivos organizacionais o TCE-GO, venha a mudar o padrão de linguagem de programação adotada para seus produtos, a arquitetura de software proposto por esse documento será também utilizada no desenvolvimento dos produtos de software na plataforma e/ou linguagem definida pelo TCE-GO. Devendo esse documento ser atualizado apenas no que se refere a padrões estabelecidos que referenciam os padrões da plataforma .Net, como por exemplo o padrão de codificação e nomenclatura, uma vez que cada plataforma e linguagem estabelece o seu padrão.

Como apresentado anteriormente na seção *Focos da Arquitetura*, esse modelo arquitetura está focado na granularização das estruturas do projeto, analisando dessa forma a arquitetura de projeto a projeto, e a componentização dos produtos desenvolvidos, em uma visão macro, da suíte de produtos, a fim de possibilitar a integração dos produtos desenvolvidos baseados nessa arquitetura de software.

A GER-TI desenvolveu alguns componentes referencias que serão utilizados no desenvolvimento de produtos nessa arquitetura. Esses projetos servirão como padrões para os novos projetos, de modo a disponibilizar estruturas padrão para serem utilizadas, evoluídas ou sobrescritas nos projetos que fazem uso deles.

Os projetos de referência são:

5.1. TCE.Compartilhado

O projeto "TCE.Compartilhado", foi desenvolvido para ser a infraestrutura dos projetos dessa arquitetura. Nele está definido a estrutura padrão de objetos de negócio, mapeamento com o banco de dados, repositório padrão de operações com o banco de dados, serviço padrão, validadores e conversores. Todas essas

estruturas servem como referência e apoio para os novos projetos, de modo a padronizar as estruturas do projeto.

O TCE-GO possui projetos anteriores a arquitetura descrita nesse documento, chamados pela equipe da GER-TI de produtos legados. Como a demanda de migração desses projetos para essa arquitetura se dá de forma gradual, algumas estruturas de negócio do TCE-GO, foram migradas de forma pontual para a arquitetura descrita nesse documento, e foram incluídas nesse projeto, de forma temporária. Essas estruturas serão incluídas nos projetos que as mantêm, quando esses projetos forem migrados para a arquitetura descrita nesse documento.

Essa abordagem de desenvolvimento será utilizada sempre que situações de necessidade exigirem essa migração parcial, para beneficiar um ou vários projetos da suíte de produtos do TCE-GO.

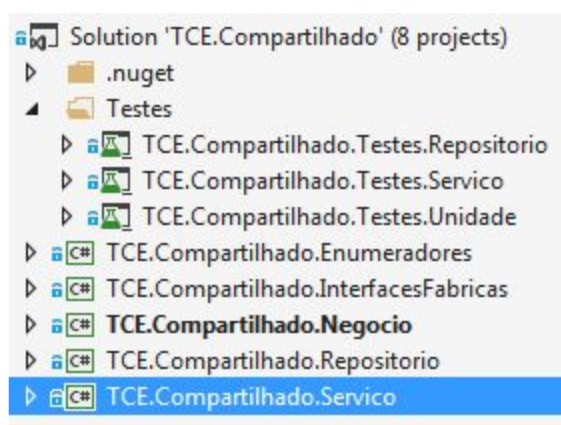


Figura 13 - Estrutura do projeto TCE.Compartilhado

O projeto TCE.Compartilhado foi desenvolvido na primeira versão da proposta de arquitetura dos softwares do TCE-GO. Essa versão diferente da proposta desse documento, os pacotes Negócio, Repositório e Serviço eram projetos separados. Na proposta desse documento, esses três projetos são organizados em um único projeto, e suas estruturas são organizadas em pacotes separados com os nomes do que antes eram projetos, pacotes de Negócio, Repositório e Serviços, como podemos visualizar na **Figura 7** e na **Figura 9**.

5.2. TCE.Componentes

O projeto TCE.Componentes tem a função de padronizar as estruturas de menor granularidade, que serão utilizadas de forma pontual, componente de um produto, seja no aspecto visual ou de estrutura de dados, independente de negócio.

Atualmente, esse projeto define o padrão do pacote visual UI MVC, para projetos Web que utilizam a tecnologia Asp.Net MVC na versão 4.5 e 5. Esse componente visual está presente em vários projetos, em que um dos componentes mais utilizados é o componente de *Login* e o componente de *Categoria Download*.

Estruturação da arquitetura proposta nesse documento, incluído os projetos de infraestrutura e componentização, está demonstrada na figura abaixo:

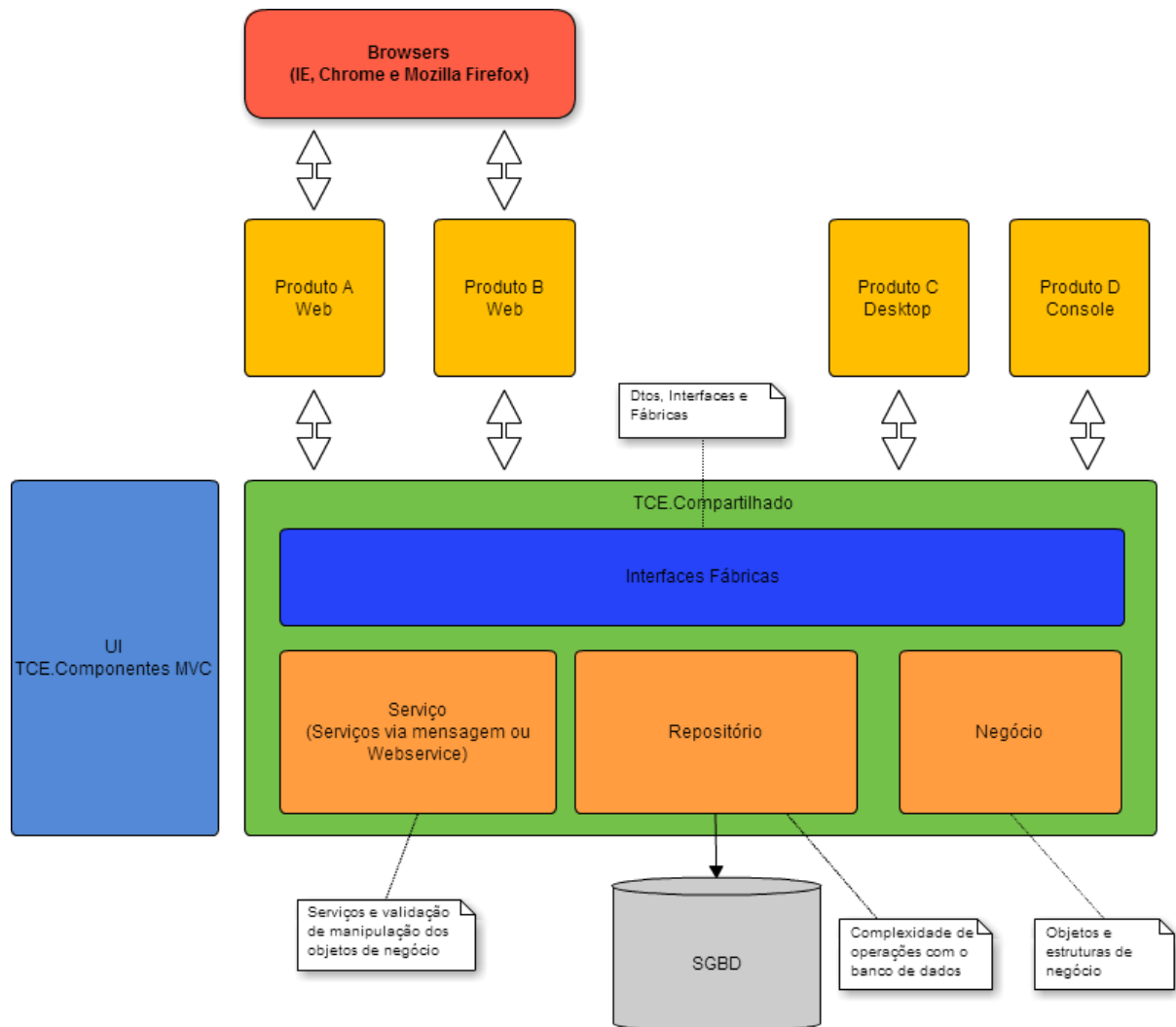


Figura 14 - Arquitetura da Informação - Suíte de Produtos

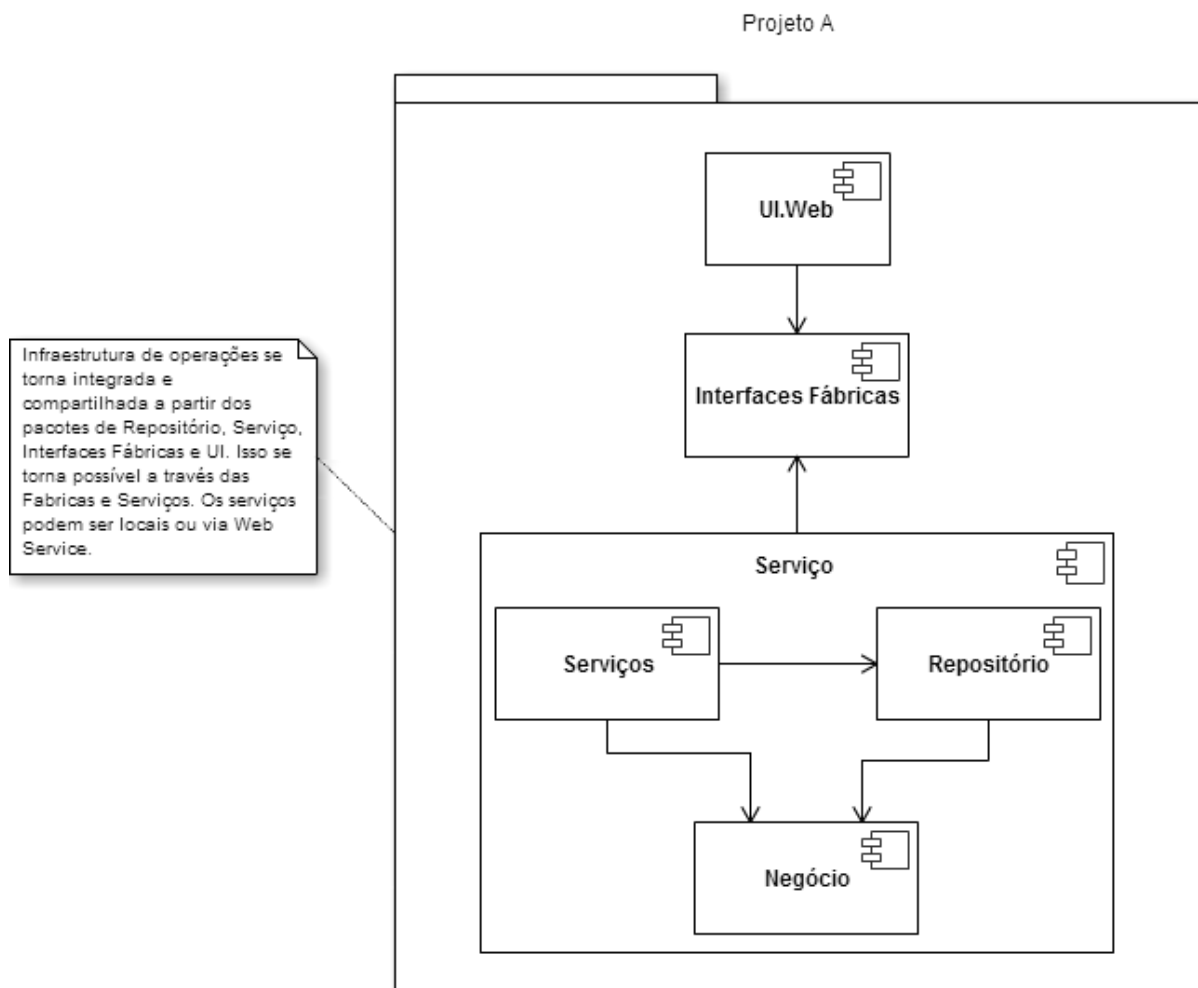


Figura 15 - Arquitetura de Pacotes

6. Camadas

A abordagem granular da arquitetura possibilita que cada camada possa ser substituída sem prejuízo ao funcionamento das demais camadas, desde que cada estrutura referenciada sejam mantidas, a fim de não ter quebra de contrato.

Essa abordagem possibilita a implantação da camada de serviços e de repositórios em um servidor central, de modo a possibilitar o balanceamento de carga de acesso a essas estruturas de forma remota e escalável.

A inclusão de novas interfaces ou a substituição das interfaces existentes, nessa arquitetura se torna possível, visto que a camada de Apresentação, a camada UI, não mantém nenhuma regra de negócio, apenas gerencia os aspectos visuais, deixando a cargo do serviço realizar as verificações e validações dos dados recebidos, transformados ou consultados pela interface. Possibilitando assim, incluir ou substituir as interfaces UI sem se preocupar com regras e restrições de negócio.

6.1. Interface com Usuário (“UI”)

A camada de apresentação implementada em qualquer tecnologia de UI, seja Web, Desktop ou até Mobile, será implementada seguindo a abordagem MVC.

No uso do padrão MVC na camada UI, a camada Model, será utilizado os DTOs da camada Interfaces Fabricas - seção 6.2. Os DTOs representam os objetos de apresentação para o usuário, seja usuário convencional ou sistema de terceiros. Porém, em casos de Model específico da interface UI, como coleção de uma combo, para apresentação em uma tela de cadastro, podem ser utilizados DTOs da camada UI, mas esses DTOs não representam objetos de negócio, por esse motivo não podem ser transitados entre UI e Interface Fábrica, tão pouco a ultima com a camada de Serviços.

Como as regras de negócio são mantidas pela camada de serviço, na chamada das operações de serviço pela interface UI, o serviço irá controlar o acesso a informação, até o nível de controle por usuário. Desse modo, a interface UI fica responsável apenas pela entrada e saída de dados.

Todo serviço ao validar o DTO enviado na chamada da operação, caso em qualquer situação da verificação ou validação do serviço identificar inconsistência, o serviço irá retornar uma exceção do tipo “*ValidationObjectException*”, a qual possui alguns métodos para obter as mensagens de verificação e validação, as quais contem a mensagem de inconsistência e o nome da propriedade ou unidade inconsistente. A interface UI deve tratar esse tipo de exceção, para apresentar da forma mais apropriada a mensagem de verificação ao usuário solicitante da operação.

Toda instanciação de serviço pela interface deve respeitar a interface do serviço utilizada pelo projeto, se a instanciação for local, esta deverá ser feita via Fabrica Genérica (fábrica padrão disponível na biblioteca do projeto TCE.Compartilhado), caso contrário, a chamada será realizada via Web Service.

Os projetos Web devem estar adaptados a rodar nos navegadores Internet Express 9 ou superior, Mozilla Firefox 3.6 ou superior e Google Chrome 49.0.2623.87 ou superior.

6.2. Interfaces Fábricas

A camada de Interfaces Fábricas, como o nome já sugere, contem as especificações das interfaces de serviço e as fábricas de objetos (Design Patterns - Factory Method Pattern). Cada projeto detém as especificações de seu contratos de serviços bem como as regras implementadas nos serviços concretos, que implementam tais contratos. As fábricas não são diferente, cada projeto se necessário for, irá implementar suas fábricas para tratar das estruturas e manipulações de dados que necessitarem.

Por padrão, a instanciação de serviços e repositórios quando referenciados de forma local, será realizada via classe *FabricaGenerica*, pelo método *Crie<T>()*, classe implementada e mantida pelo projeto TCE.Compartilhado.

Nessa camada no pacote Dto, serão especificadas as classes de Objetos de Transferência de Dados, os quais devem ser utilizados pela camada de apresentação UI, para apresentar e obter as informações dos usuários para as chamadas e operações dos serviços. Todos os DTOs devem herdar da classe abstrata definida pelo projeto TCE.Compartilhado, os quais serão serializáveis para realizarem operações remotas de serviços remotos.

Em projetos UI Web, implementados em Asp.Net MVC 4, estabelecemos um padrão organizacional dos componentes de cada estrutura como é sugerido por esse padrão. Nesses projetos o pacote Controllers contém todas as classes de controladores das Views, sendo que o nome de cada controlador, separado o sufixo "*Controller*", terá uma subpasta no pacote Views, na qual terá suas Views e suas PartialViews. As Views e PartialViews contêm estruturas de estilo e de script, as quais serão organizadas no pacote Componentes, na subpasta "*tela*", em que cada pasta de View terá uma pasta de Componente, que contém os arquivos de estilo (.css) e o arquivo de script (.js). Nesse projeto, as fontes utilizadas são organizadas no pacote Content na subpasta "*fonts*", e o estilo e script que manipula será implementado e definido pelo componente contido na pasta "*custom*", localizado no pacote Componentes. Todas as imagens manipuladas pelo projeto serão organizadas na pasta Imagens, como apresentado na imagem abaixo:

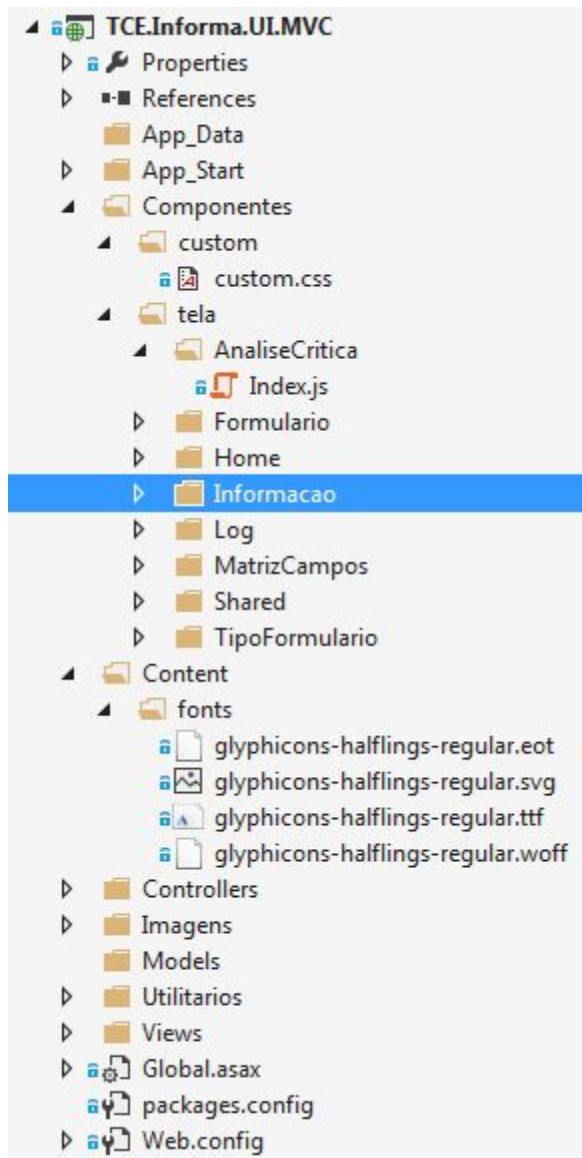


Figura 16 - Estrutura Projeto MVC

6.3. Serviço

A camada de Serviços dos projetos baseados na arquitetura descrita nesse documento, e o componente mais importante do projeto, visto que toda regra de negócio proposta pela solução é mantida e controlada por essa camada.

Como visto anteriormente, essa camada é responsável em manipular os objetos de negócio a partir de chamadas de serviço originadas pela camada UI ou por outros serviços, requisitadas a partir de um objeto de modelo UI - DTOs - ou por parâmetros. As operações que tratam de negócio, como operações de banco ou consistências de dados, essas operações requerem verificação e validação dos registros manipulados. Para isso, os serviços utilizam a estrutura de validação de objetos do *framework* FluentValidation. Esse *framework* permite que a série de validações realizadas em um objeto sejam agrupadas em uma lista de inconsistências, e que essa lista de inconsistências seja agrupadas por nome da propriedade e mensagem de validação, um dicionário de validações. Essa validação

após ser executada, caso tenha inconsistências, o serviço deverá retornar o dicionário de inconsistências utilizando uma exceção do tipo `ValidationObjectException`. Que por sua vez será tratado na UI para apresentar as validações campo a campo.

Os serviços que recebem DTOs de negócio deveram utilizar a estrutura de conversão padrão de DTO para objeto de negócio. Os conversores deveram implementar a interface `IConversor<TDto, TObj>`, e devem implementar a conversão de propriedades diferente dos tipos primitivos e do tipo coleção. Qualquer tratamento de conversão diferente do padrão, para cada chamada de conversão, seja de DTO para objeto de negócio ou de objeto de negócio para DTO, deve ser implementado nessa classe. O serviço fará uso dos conversores instanciados pela classe `ConversorPadrao<TDto, TObj>`, que irá obter a instância dos conversores com base nos tipos de objeto e com base nas propriedades convertidas.

Todo serviço que será disponibilizado para ser utilizado pela UI e por outros serviços, deverá implementar uma interface previamente estabelecida no pacote Interfaces Fábricas.

Os serviços que herdarem da implementação padrão `ServicoPadrao<TDto, TObj>`, devem ter implementado o validador que herda da classe abstrata `ValidadorPadrao<T>`, e deve ter implementado também o conversor que implemente a interface `IConversor<TDto, TObj>`.

6.4. Repositório

O pacote de Repositório contém toda estrutura de manipulação dos objetos de negócio no banco de dados.

O mapeamento das classes de negócio com as tabelas do banco de dados, através das classes de sufixo “*Map*”. Essas classes de mapeamento são carregadas em tempo de execução e agrupadas na coleção de mapeadores do projeto, através da classe de contexto “*EntitiesEF*” via reflexão.

A organização dos pacotes de repositório nesse modelo arquitetural fica no projeto de Serviço, na pasta Repositório, na qual contém as pastas de Interfaces (interfaces de repositório do projeto), de Mapeadores (classes de mapeadores dos objetos persistidos de negócio - classes que herdam da classe `EntityTypeConfiguration<T>`, do Entity Framework) e de Repositórios que contém a implementação concreta dos repositórios com base na interface que especifica.

No projeto TCE.Compartilhado, no pacote Repositorio, está disponível para herança, a implementação padrão do repositório, a classe `RepositorioGenerico<T>`. Essa classe já implementa as funcionalidades padrão de repositório que são especificadas pela interface `IRepositorioPadrao<T>`. Essa especificação deve ser utilizada quando o banco de dados do produto de software desenvolvido, for o banco de dados Oracle do TCE-GO.

6.5. Negócio

O pacote de negócio contém as classes que representam os objetos do negócio da solução de software. Esses objetos são manipulados pelas camadas de repositório (se forem objetos persistidos) e pela camada de serviço.

Esses objetos são completos, com todos os atributos de negócio, logo não tem a restrição quanto ao controle de acesso, visto que o controle de acesso é realizado pela camada de serviço, na consulta e operações da UI ou de outros serviços.

As propriedades que representam entidades de classes persistidas não devem ser criadas como sendo do tipo “virtual”, devido o Entity Framework por padrão realizar a carga no relacionamento quando não definido como *No Lazy Load*. Para que as consultas no banco de dados não carreguem informações desnecessárias, a consulta em que o objeto possui relacionamento com outras entidades, caso o desenvolvedor deseje que o repositório carregue a instância correspondente no relacionamento, este deverá fazer na função de consulta personalizada ou sobrescrever a consulta padrão que deseja carregar o objeto relacionado. Para isso a consulta deverá carregar o objeto dando um *Include*, na referencia ou fazer um *Load* após a consulta, sendo o primeiro uma função na referencia da entidade através da expressão lambda (*System.Data.Entity*), e o segundo uma função do Entity Framework, como apresentado abaixo:

```
public override Acordao Consulte(long id)
{
    using (var conexao = ObtenhaConexao())
    {
        var acordao = conexao.Set<Acordao>().FirstOrDefault(x => x.Id == id);

        if (acordao != null)
        {
            conexao.Entry(acordao).Reference(x => x.Autuacao).Load();
            conexao.Entry(acordao).Reference(x => x.Responsavel).Load();
            conexao.Entry(acordao).Reference(x => x.AlteradorDocumento).Load();
            conexao.Entry(acordao).Reference(x => x.CriadorDocumento).Load();
            conexao.Entry(acordao).Reference(x => x.SessaoPlenaria).Load();
        }

        return acordao;
    }
}
```

Figura 17 - Exemplo Load Entidade Relacionada

```
public override Usuario Consulte(long id)
{
    using (var conexao = ObtenhaConexao())
    {
        var resultado = (from usuario in conexao.Set<Usuario>().Include(x => x.Pessoa)
            where usuario.Id == id
            select usuario).FirstOrDefault();

        return resultado;
    }
}
```

Figura 18 - Exemplo Include Entidade Relacionada

7. Ferramentas e Tecnologias

As ferramentas utilizadas pela equipe da GER-TI, para o desenvolvimento de produtos de software baseados nessa arquitetura, são:

- Visual Studio 2013;
- Team Foundation Server 2010;
- PL/SQL Developer 7.0.3;
- Filezilla 3.10.2;
- Chrome 48.0.2564.116;
- Internet Explorer 9 ou superior;
- Oracle Design 10.1.2;
- Oracle 11g.

Os produtos de software desenvolvidos nessa arquitetura utilizam o Framework .Net na versão 4 ou superior, Asp.Net MVC, Entity Framework 5, ADO.Net, DevExpress versão 13.2, e WCF.