Procedimento Técnico SSE-DSV-99

*Objetivo do documento*

Histórico da Revisão

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versão** | **Descrição** | **Autor** |
| DD/MM/AAAA | 1.0 | Versão inicial | Autor |
|  | 1.1 |  |  |

Índice Analítico

1. Objetivo 4

2. Escopo 4

3. Acrônimos e Abreviações 4

4. Representação Arquitetural 4

5. Requisitos e Restrições arquiteturais 4

6. Visões de Casos de Uso – Principais para Arquitetura 5

7. Visão Lógica 5

8. Visão de Implementação 8

8.1 Core Domain – Visão Geral 8

8.2 Core Domain – Suporte eventos de domínio 9

8.3 Core Domain – Suporte regras de negócio 10

8.4 Core Infra – Visão Geral 13

8.5 Core Mensagens – Visão Geral 13

8.6 Camada Infra – Registro de atividades 14

8.7 Camada Infra – Persistência de dados 17

8.8 Camada Domínio – Entidades e Regras de Negócios 17

8.9 Camada Domínio – Criar entidades 18

8.10 Camada Domínio – Criar regras de negócio (Especificações e Validações) 19

8.11 Camada de Aplicação – Visão Geral 20

8.12 22

8.13 22

8.14 22

8.15 22

8.16 Paginação 22

8.17 Camada Apresentação 22

8.18 Estrutura de Projetos 22

8.19 Domain Model – Entidade 23

9. Visão de Implantação 24

10. Dimensionamento e Performance 24

11. Qualidade 24

12. Modelagem da Solução 24

13. Estrutura de Projetos 25

14. Módulos Core.Domain e Core.Infra 25

15. External Service 25

16. SpecificationValidation 25

17. Módulos Core Domain e Core Infra 25

18. Specs 26

19. External Services 26

20. XXXXXXXXXXXXXXXXX 26

21. XXXXXXXXXXXX 26

22. XXXXXXXXX 26

23. XXXXX 26

24. XXXXX 26

25. XXXX 26

26. Glossário 26

27. Criar procedimento 26

# Objetivo

Este documento fornece uma visão arquitetural abrangente do **Sistema Integrado Procon**, usando diversas visões de arquitetura para representar diferentes aspectos do sistema.

O objetivo deste documento é capturar e comunicar as decisões arquiteturais significativas que foram tomadas em relação ao sistema. Sugestões de complementação e atualização deste documento serão bem-vindas.

# Escopo

Este documento se aplica ao **Sistema Integrado Procon** que é composto por subsistemas com negócios específicos (Segurança, Fiscalização, Protocolo, Atendimento e Administrativo).

Dentre uma série de atividades que em esses subsistemas atuam podemos destacar:

* Comunicação com os fornecedores de bens e/ou serviços (pessoa jurídica e/ou pessoa física)
* Recebimento e processamento de reclamações administrativas, individuais e coletivas, contra fornecedores de bens ou serviços;
* Orientação aos consumidores e fornecedores acerca de seus direitos e obrigações nas relações de consumo;
* Fiscalização do mercado consumidor para fazer cumprir as determinações da legislação de defesa do consumidor;

O documento fornece informações básicas

# Acrônimos e Abreviações

# Representação Arquitetural

Este documento irá detalhar as visões baseado no modelo “4+1” [KRU41], utilizando como referência os modelos definidos na MDS. As visões utilizadas no documento serão:

# Requisitos e Restrições arquiteturais

Esta seção descreve os requisitos de software e restrições que tem um impacto significante na arquitetura.

|  |  |
| --- | --- |
| Requisito | Solução |
| Linguagem | Microsoft .NET Framework 4.5  C# ,ASP.NET MVC 5 , Jquery, Javascript, Bootstrap , HTML5 ,CSS |
| Plataforma | Windows Server 2012 IIS 8.5  Microsoft .NET Framework 4.5 |
| Segurança | O sistema deve prover de mecanismos de autenticação e autorização de forma segura para os funcionários do Procon e parceiros conveniados abrangendo a intranet e a internet.  Características básicas  - Prover acesso por Active Directory e  - Prover manutenção de recursos que serão acessados.  - Prover associação de menus aos recursos cadastrados  - Prover manutenção de usuários e respectivos perfis associados aos recursos cadastrados  - Prover acesso de forma segura via intranet e internet  - Auditoria de operações que modificaram informações no sistema |
| Persistência | Operações registradas devem ser persistidas fisicamente via bancos de dados relacionais. SQL SERVER 2014 |
|  | Execução da aplicação pelo menos nas últimas três versões dos principais browsers do mercado (Google Chrome, Firefox e Edge) |

# Visões de Casos de Uso – Principais para Arquitetura

# Visão Lógica

O Sistema Integrado PROCON é composto pelos seguintes subsistemas:

* Sistema Segurança
* Sistema Fiscalização
* Sistema Protocolo
* Sistema Atendimento
* Módulo Administrativo

**Protocolo**

É o sistema para gestão do protocolo do PROCON-SP, baseando-se no Protocolo e na tramitação a partir de uma unidade administrativa. As suas funcionalidades foram elaboradas na visão da política estadual de arquivos e gestão de documentos e baseadas nas normas e procedimento de protocolo para a administração pública do estado de São Paulo.

**Atendimento**

É o sistema para gestão do atendimento do PROCON-SP. As funcionalidades desta seção atendem basicamente a área da DAOC quanto às solicitações de orientação, de consultas e de atendimentos dos consumidores do estado de São Paulo. O módulo de atendimento deve ser interativo de forma a envolver os consumidores e os fornecedores, comunicando entre todos os envolvidos as necessidades de informação.

**Fiscalização**

É o sistema para atender à fiscalização do PROCON-SP. As funcionalidades desta seção atendem à necessidade do PROCON-SP, na fiscalização aos fornecedores, coletando e registrando todas as informações da fiscalização na sua diligência fiscal.

**Sistema Segurança**

É o sistema para coordenar a gestão de segurança, contemplando todos os módulos do Sistema Integrado.

**Módulo Administrativo**

É o módulo para administração das informações cadastrais do PROCON-SP. As funcionalidades desta seção atendem a necessidade do PROCON-SP, nas informações corporativas no contexto administrativo, para dar base a todas as unidades da fundação.

**Abordagem Sistêmica**

A solução adotada procurou na medida do possível aplicar uma abordagem de uma modelagem de software que segue um conjunto de práticas com objetivo de facilitar a implementação de subsistemas e suas respectivas regras/processos de negócios por contextos dentro de um modelo de **domínio (Domain Driven Design).** A partir dessa abordagem temos os seguintes componentes guias de implementação:

Core Domain

Contempla o Core básico para o suporte técnico de implementações para o Modelo de Domínio (Domain Model). Ex: Eventos de domínios, modelos para regras de negócios (especificações e validações), e funções de suporte a entidades, etc.

Core Infra

Contempla o core básico para o suporte técnico de implementações de serviços externos. Ex: serviço de e-mail, serviço de notificação, serviço de auditoria, etc.

Domain Model

Contempla as entidades de domínio, estados e comportamentos, lógica de negócio, interfaces para os serviços de domínio e de repositório e estruturas (objetos de valor) que agregam valor as entidades.

Domain Service

Serviço de Domínio que orquestra as operações sobre as entidades de domínio podendo trabalhar com diversas entidades, realizando persistência através de repositórios.

Application Service

Serviço de Aplicação que orquestra ações disparadas pela camada de apresentação e fornece DTOs para a comunicação, compondo adaptadores, interfaces para camadas superiores, suporte para implementação de controle de transações, registro de operações (logs). Também serve como fábrica de serviços para as camadas superiores.

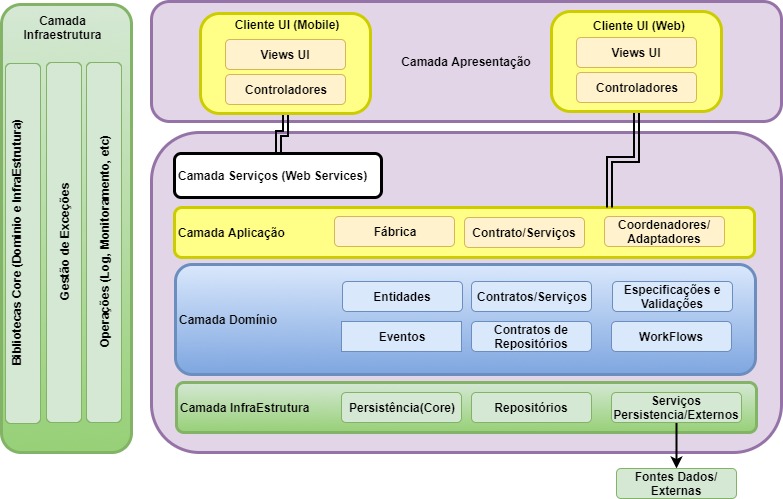
Repository

Realiza a persistência das entidades se comunicando diretamente com o meio de acessos aos dados.

External Service

Serviços externos que realizam a consulta/persistência por meios externos.

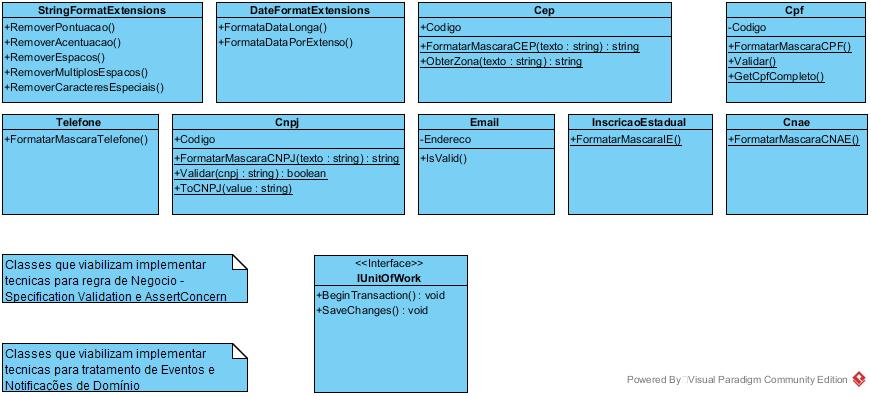
Seguem abaixo diagramas descrevendo conceitos informados acima:





# Visão de Implementação

## Core Domain – Visão Geral

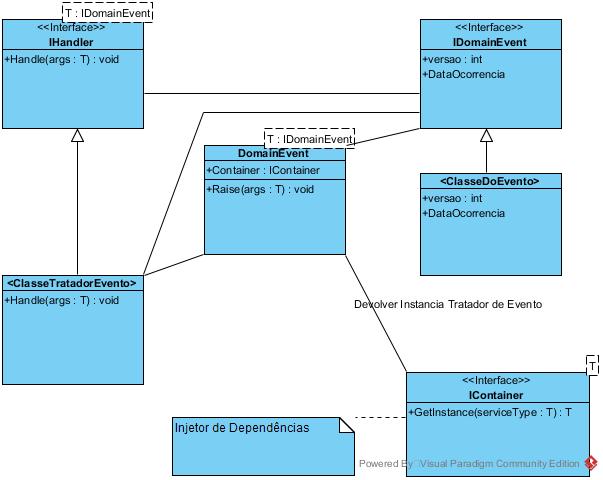


**Objetivo:**

Fornecer funcionalidades de uso comum para atender demandas de entidades de domínio:

* Formatações de strings e datas;
* Validações e formatações de entidades/atributos, como por exemplo: CEP, CPF, CNPJ,telefone, e-mail,inscrição estadual, CNAE, etc;
* Modelos para implementação de regras de negócio nos contextos (técnicas Specification/Validation);
* Classes que viabilizam implementar tratamento de eventos e notificações de domínio;
* A implementação dessa biblioteca deve estar livre na medida do possível de qualquer infraestrutura

## Core Domain – Suporte eventos de domínio

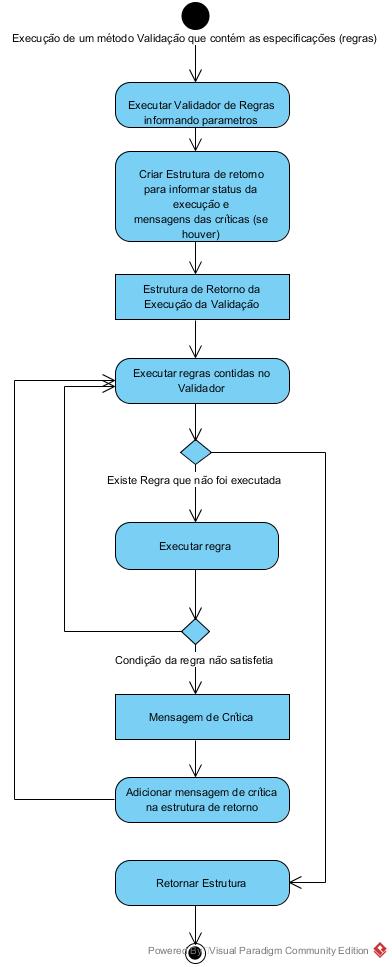


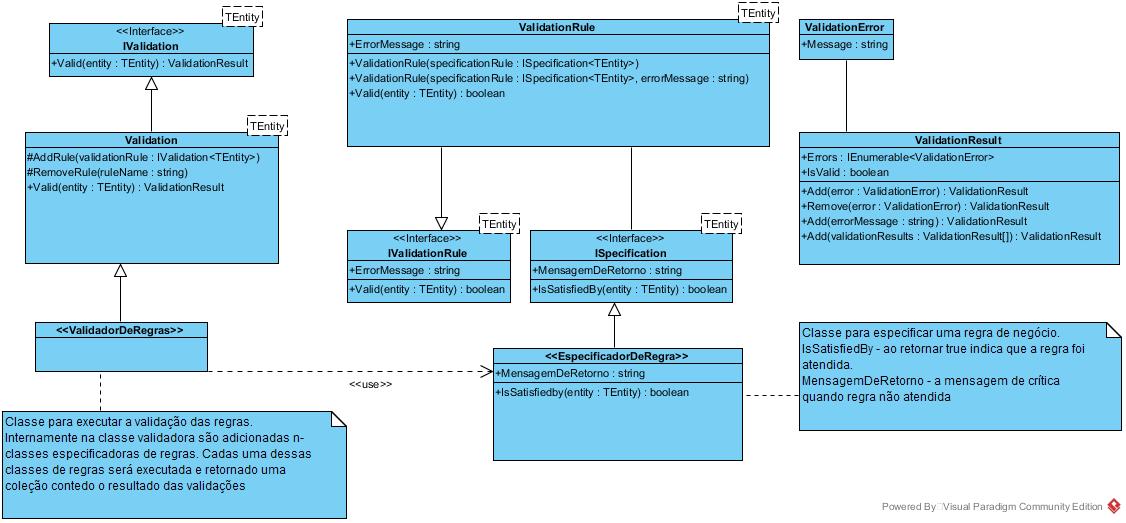
**Objetivo:**

Fornecer funcionalidades de uso comum para atender demandas de entidades de domínio:

* Formatações de strings e datas;
* Validações e formatações de entidades/atributos, como por exemplo: CEP, CPF, CNPJ,telefone, e-mail,inscrição estadual, CNAE, etc;
* Modelos para implementação de regras de negócio nos contextos (técnicas Specification/Validation);
* Classes que viabilizam implementar tratamento de eventos e notificações de domínio;
* A implementação dessa biblioteca deve estar livre na medida do possível de qualquer infraestrutura

## Core Domain – Suporte regras de negócio



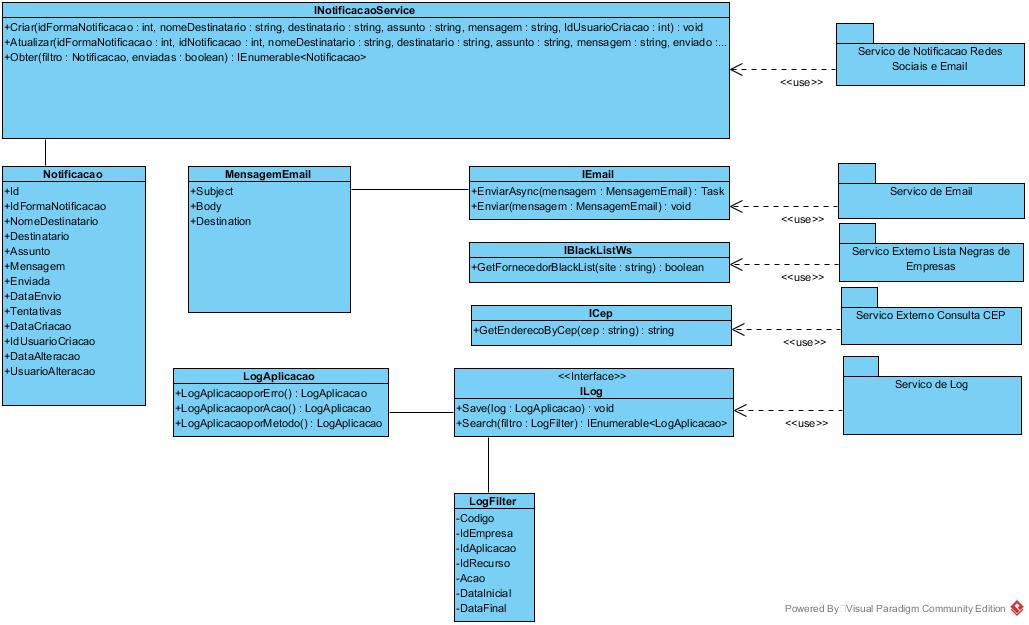


**Objetivo:**

Fornecer infraestrutura básica para implementação das regras nos serviços de negócio propiciando baixo acoplamento.

* Modelos para implementação de regras de negócio nos contextos (técnicas Specification/Validation);

## Core Infra – Visão Geral



**Objetivo**:

Fornecer funcionalidades de uso comum para atender demandas de infraestrutura:

* Disponibiliza interfaces e estruturas básicas para serviços possibilitando a abstração na criação dos mesmos e facilidade de manutenção em possíveis atualizações futuras como, por exemplo: serviço de e-mail, serviço de consulta lista negra de Empresas, serviço de notificações de redes sociais e de e-mail, serviço de consulta de CEP, serviço de log, etc.

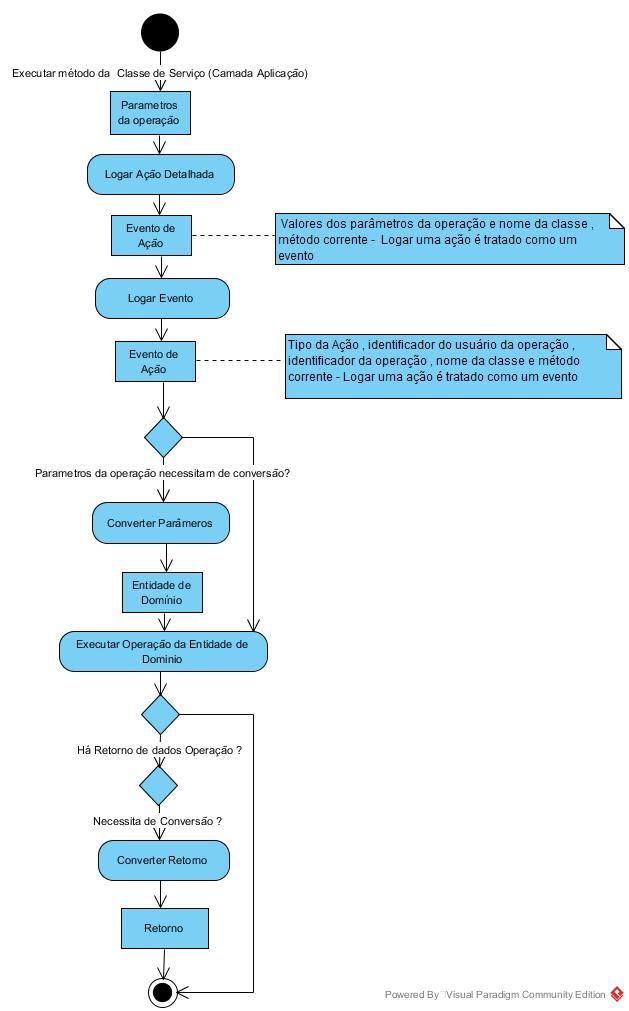
## Core Mensagens – Visão Geral

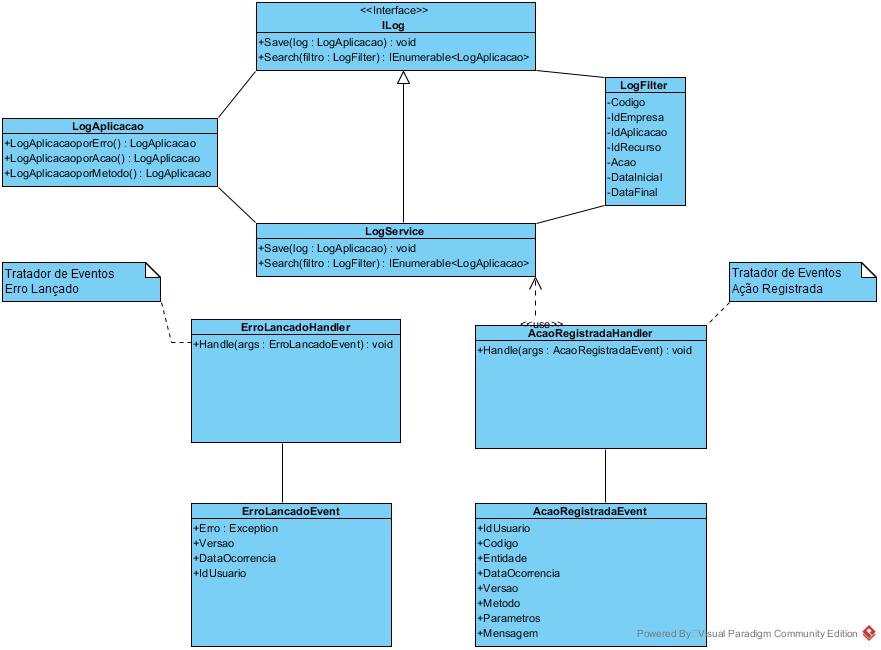


**Objetivo:**

Fornecer de forma centralizada um dicionário de mensagens utilizadas pela aplicação em situações como, por exemplo: retorno de críticas de validação de regras de negócios, status de operações e outras situações que por ventura sejam necessárias para emissão de mensagens.

## Camada Infra – Registro de atividades



****

**Objetivo:**

Fornecer um serviço para permitir o registro de ações ocorridas nos sistemas por questões de auditoria e/ou diagnóstico de problemas possam vir a ocorrer no sistema.

Registro da ação:

Por questões de auditoria é necessário manter os registros das seguintes operações de banco de dados:

* *Data da última alteração e identificador do usuário que realizou a operação de modificação do registro.*
* *Data de criação e identificador do usuário que realizou a operação de inclusão do registro*.

Esse tipo de informação deve constar em toda tabela implementada no banco de dados que for auditada através dos seguintes campos:

*dt\_inclusao, id\_usuario\_criacao, dt\_ultima\_alteracao e id\_usuario\_alteracao*.

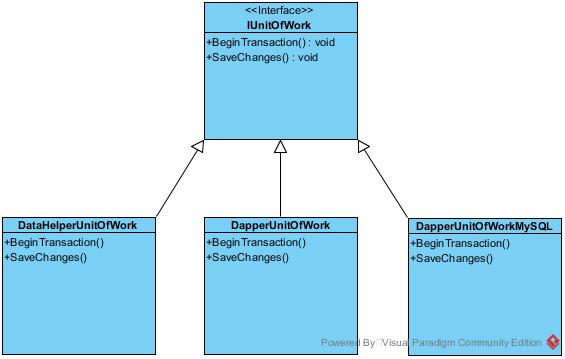
Para atender esse requisito dentro de cada operação disponibilizada no serviço de negócio deve ser implementado a persistência dessas informações;

Além de registrar na própria tabela que está sendo auditada, também é registrada em uma tabela específica de logs de operações (seguranca.tb\_log\_aplicacao) constando as informações da operação em questão, do identificador do usuário e data de execução. Para atender esse requisito foi implementado um tratador de eventos que se comunica com o serviço de log conforme descrito diagramas anteriores.

O registro da operação(log) é implementado como um tipo de evento de domínio (ação registrada).

Diagnóstico de Problema

## Camada Infra – Persistência de dados

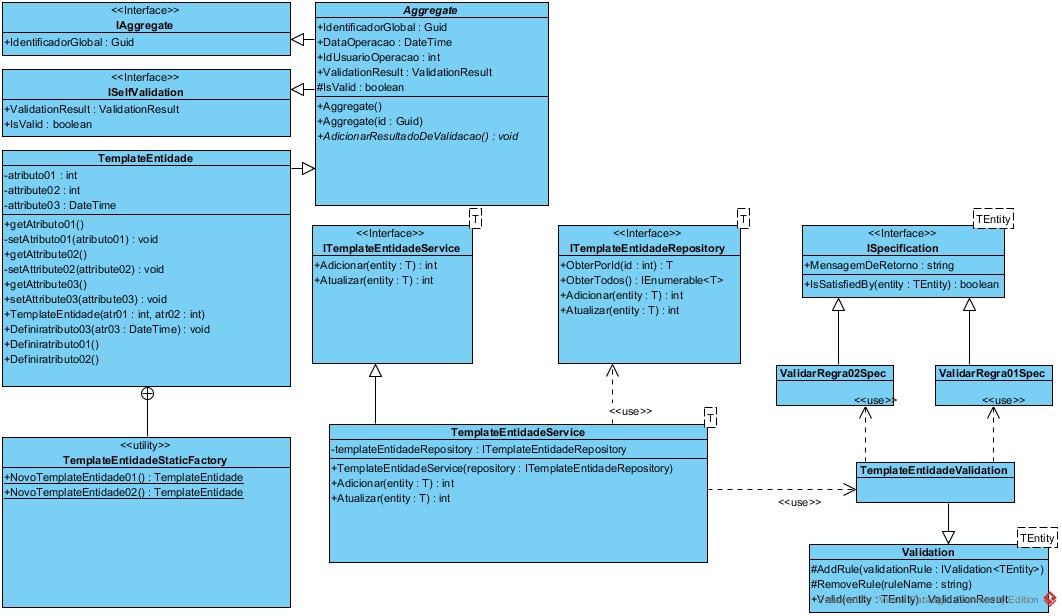


**Objetivo:**

Fornecer um serviço para possibilitar comunicação com as bases de dados:

* Implementação das abstrações em ADO.NET (SQL Server);
* Implementação das abstrações pelo componente Dapper (SQL Server e MySQL);
* Prover suporte transacional em operações de banco de dados;

## Camada Domínio – Entidades e Regras de Negócios



**Objetivo:**

Fornecer os serviços de negócio, os contratos para os repositórios e de negócio e entidades apresentando algumas características como:

* As entidades e lógica de negócio são implementadas através do padrão Domain Model;
* A implementação dessa camada deve estar livre na medida do possível de qualquer definição de infraestrutura de dados;
* Especificamente para essa camada não devem ser acrescentadas novas referências de assemblies considerando que as existentes já sejam suficientes para implementar a demanda dessa camada;
* Implementação das regras de negócio a partir de modelos usando técnicas de especificação e validação;

## Camada Domínio – Criar entidades

Temos as seguintes orientações básicas para implementar uma entidade (classe que representa a entidade de Negócio):

Herança

Ela deve herdar da classe *Aggregate* (presente na Core.Domain). Caso a entidade precise herdar de uma classe base, então a classe base é que herdará da classe *Aggregate.*

Propriedades

Os membros devem ser definidos como públicos para exposição e privados para recebimento de valores. (public <tipo> <nome do membro> { get; private set; }

Para que um cliente externo(camadas superiores) possa definir os valores nessas propriedades após o objeto ter sido criado deve ser feito por métodos.

Cria-se um método sem retorno recebendo como parâmetro o valor do cliente externo e repassando para a propriedade (técnica adhoc setters).

Construtores

A Entidade deve ter somente um construtor se possível.

Caso excepcionalmente necessite de outros construtores alternativos deve ser usado o recurso de fabricar a instância com classe aninhada.

Observação: É necessário criar um construtor adicional sem parâmetros para atender uma exigência da camada de dados (Restrição de dependência para poder criar o objeto a partir do banco de dados).

Dependências

A Entidade deverá basicamente terá dependências de:

Core.Domain (Core de Domínio)

System

System.Core

System.Xml

System.Xml.Linq

**ESSAS REFERÊNCIAS GERALMENTE SÃO O SUFICIENTE PARA RESOLVER OS PROBLEMAS** de regras de negócio. Se houver necessidade de incluir nova referência, reavalie ou converse com o Arquiteto. Provavelmente a solução do problema deve ser aplicada em outro lugar (camada).

Identificador Global

Toda entidade seguindo a risca o DDD deve possuir uma propriedade de identificação única.

(ex: identificadorGlobal tipo GUID). Por questões técnicas de performance do banco somente persistimos essa informação na tabela de log. Em outras tabelas ela não será persistida. Essa propriedade deverá constar também nas views models.

## Camada Domínio – Criar regras de negócio (Especificações e Validações)

Para implementar as regras de negócio foram utilizados duas técnicas (Specification e Validation). Através dessas técnicas as regras de negócios podem ser recombinadas encadeando-as usando a lógica booleana. O padrão é freqüentemente usado no contexto de DDD.

A implementação pode ser implementada da seguinte forma:

Criar a classe com o nome da regra a ser especificada.

A classe implementa um contrato que solicita a codificação do método da regra que deve retornar verdadeiro caso a regra seja atendida e uma mensagem de retorno caso a condição não seja satisfeita.

Exemplo de implementação:

*public class AutoValidarExistenciaItemSpec :ISpecification<Auto>*

*{*

*public bool IsSatisfiedBy(Auto entity)*

*{*

*return (entity.Itens.Count() > 0);*

*}*

*public string MensagemDeRetorno*

*{*

*get{ return "O auto deve conter um item.";*

*}*

*}*

Uma vez que a especificação já esteja implementada ela pode ser utilizada de diferentes formas de acordo com a necessidade.

O uso comum para validação de regras de negócio é criar outra classe agregadora das regras.

Essa classe deve herdar da classe Validation que valida as regras e gera uma coleção de resultados caso a regra não seja satisfeita.

Devem ser instanciados cada uma das regras utilizadas e serem adicionadas na coleção de regras da classe base.

Exemplo de implementação:

Criado a classe AutoValidation. Herdado da classe Base Validation.

Adicionado as regras na classe base através do método base.AddRule.

*public class AutoValidation : Validation<Entities.Auto>*

*{*

*publicAutoValidation(IAutoRepositoryautoRepository = null)*

*{*

*var existenciaItem = new AutoValidarExistenciaItemSpec();*

*var fiscalResponsavel = new AutoValidarFiscalResponsavelItemSpec();*

*base.AddRule(newValidationRule<Auto>(existenciaItem, existenciaItem.MensagemDeRetorno));*

*base.AddRule(new ValidationRule<Auto>(fiscalResponsavel, fiscalResponsavel.MensagemDeRetorno));*

*}*

*}*

Para o uso basta instanciar a classe agregadora de regras e executar o método Valid.

Caso alguma regra não seja atendida seja será gerado uma coleção de resultados (Classe ValidationResult) .

Exeemplo de implementação:

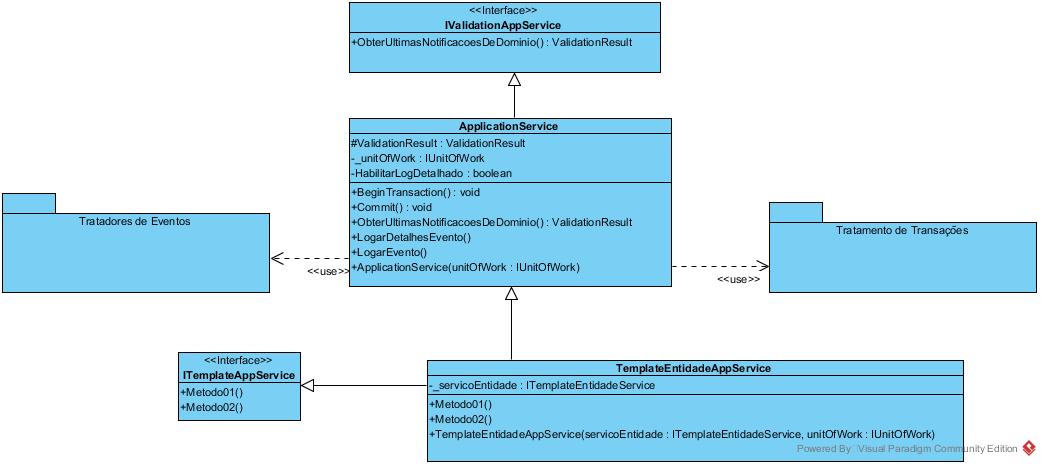
*var validationResult = new AutoValidation(\_autoRepository).Valid(entidade);*

*foreach (var validationResultError in validationResult.Errors)*

Opções comum de retorno

Essa lista de resultados pode ser enviada para a camada superior retornando por uma coleção de Resultados (classe ValidationResult), ou sendo adicionado a classe que representa a entidade. Toda entidade possui um membro do tipo ValidationResult. Pode retornar a classe entidade e verificar o membro ValidationResult.

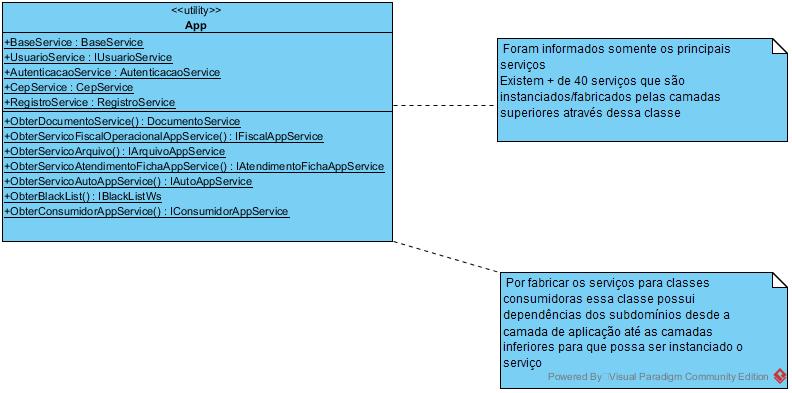
## Camada de Aplicação – Visão Geral



**Objetivo:**

Fornecer serviços para possibilitar comunicação entre as camadas de apresentação e camadas inferiores apresentando algumas características como:

* Coordenar chamadas entre os serviços de domínio e serviços externos;
* Expor operações realizadas pelos serviços domínio de forma que façam sentido ao domínio normalmente se utilizando da linguagem ubíqua;
* Suporte para implementação de recursos como: controle de transação, tratamento de evento, registro de ações;
* Prover adaptadores entre as camadas possibilitando expor N modelos de visão a partir das entidades de domínios;
* Expor os modelos de visão (views models) as camadas superiores;
* Fábrica de serviços expondo as abstrações para as camadas superiores;
* Resolução de dependências



**Objetivo:**

Fornecer serviços para possibilitar comunicação entre as camadas de apresentação e camadas inferiores apresentando algumas características como:

* Coordenar chamadas entre os serviços de domínio e serviços externos;
* Expor operações realizadas pelos serviços domínio de forma que façam sentido ao domínio normalmente se utilizando da linguagem ubíqua;
* Suporte para implementação de recursos como: controle de transação, tratamento de evento, registro de ações;
* Prover adaptadores entre as camadas possibilitando expor N modelos de visão a partir das entidades de domínios;
* Expor os modelos de visão (views models) as camadas superiores;
* Fábrica de serviços expondo as abstrações para as camadas superiores;
* Resolução de dependências

## 

## 

## 

## 

## Paginação

## Camada Apresentação

## Estrutura de Projetos

Para suportar a solução foram implementados um conjunto de projetos

[namespaceFaseInicial].Core

[namespaceFaseInicial].Interfaces

[namespaceFaseInicial].Model

[namespaceFaseInicial].Core.Services

[namespaceFaseInicial].Application

[namespaceFaseInicial].Infrastructure

[namespace].Core.Data

[namespace].Core.Domain

[namespace].Core.Domain.Mensagens

[namespace].Core.Infra

[namespace].[subsistema].Application

[namespace].[subsistema].Data

[namespace].[subsistema].Domain

[namespace].[subsistema].Infra

[namespaceFaseInicial] – Inicialmente o a solução foi desenvolvida com a visão de atender somente subsistema Fiscalização e módulo Administrativo. Por conta disso, o namespace utilizado foi o **br.procon.Fiscalizacao**.

A solução posteriormente passou a contemplar mais subsistemas/funcionalidades. A solução inicial teve que ser revisada para comportar os novos módulos visando facilitar a manutenção e escalabilidade da mesma. Dessa forma os novos projetos foram implementados de forma segmentada (por subsistemas) e utilizando um namespace mais abrangente o **br.procon.si**.

|  |  |
| --- | --- |
| Projetos – br.procon.Fiscalizacao | Descrição |
| Application | Fábrica de Serviços, Resolução de Dependências de Serviços, Tratador de Eventos |
| Core | Funcionalidades comuns as camadas superiores (Core.Services e UI) |
| Core.Services | Contempla o Módulo Administrativo e parte do Subsistema Fiscalização (Combustível) |
| Infrastructure | Implementações dos serviços externos e de persistência da fase inicial |
| Interfaces | Conjunto de Interfaces para atender Serviços de Negócio e Serviços Externos implementados na fase inicial |
| Model | Conjunto de ViewsModels |
| UI | Representa a camada de apresentação(web) do Sistema Integrado Procon |

|  |  |
| --- | --- |
| Projeto – br.procon.si | Descrição |
| Atendimento.Application | Componente representa a camada de aplicação do módulo de Atendimento |
| Atendimento.Data | Componente representa a camada dos repositórios do módulo de Atendimento |
| Atendimento.Domain | Componente representa a camada de domínio do módulo de Atendimento |
| Atendimento.Infra | Componente que implementa serviços externos como por exemplo emissão de relatórios para o módulo de Atendimento |
| Atendimento.Tests | Componente de Testes unitários e integração para o módulo de Atendimento |
| Core.Data | Componente core que implementa o acesso a dados (ado.net e dapper) |
| Core.Data.Tests | Componente de Testes para o projeto Core.Data |
| Core.Domain | Componente para suporte as entidades e implementação de regras de negócio |
| Core.Domain.Mensagens | Componente centralizador das mensagens utilizadas no sistema |
| Core.Infra | Componente para suporte para implementar serviços/funcionalidades externas |
| FiscalizacaoOperacional.Application | Componente representa a camada de aplicação do módulo de Fiscalização Operacional |
| FiscalizacaoOperacional.Data | Componente representa a camada de aplicação do módulo de Fiscalização Operacional |
| FiscalizacaoOperacional.Domain | Componente representa a camada de aplicação do módulo de Fiscalização Operacional |
| FiscalizacaoOperacional.Infra | Componente que implementa serviços externos como por exemplo emissão de relatórios para o módulo de Atendimento |
| FiscalizacaoOperacional.Tests | Componente de Testes unitários e integração para o módulo de Fiscalização Operacional |
| Fornecedor.Application | Componente representa a camada de aplicação do módulo de Fiscalização Operacional |
| Infra.Log | Componente que implementa o serviço externo de log |
| Infra.Email | Componente que implementa o serviço de email |
| Infra.Notificacao | Componente que implementa o serviço de notificação |
| Infra.Tests | Componente de Testes unitários e de integração para o Módulo Infra |
| Parada.Application | Componente representa a camada de aplicação do módulo de Fiscalização Operacional |
| Protocolo.Application | Componente representa a camada de aplicação do módulo de Fiscalização Operacional |
| Protocolo.Data | Componente representa a camada de aplicação do módulo de Fiscalização Operacional |
| Protocolo.Domain | Componente representa a camada de aplicação do módulo de Fiscalização Operacional |
| Protocolo.Infra | Componente que implementa serviços externos como por exemplo emissão de relatórios para o módulo de Protocolo |
| Protocolo.Tests | Componente de Testes unitários e integração para o módulo Protocolo |
| Seguranca.Application |  |
| UI.ConsumidorApp |  |
| Api.ConsumidorApp |  |
|  |  |
| DataTables.AspNet.Core |  |
| DataTables.AspNet.Mvc |  |

## Domain Model – Entidade

# Visão de Implantação

# Dimensionamento e Performance

# Qualidade

# Modelagem da Solução

A solução procurou na medida do possível aplicar uma abordagem de uma modelagem de software que segue um conjunto de práticas com objetivo de facilitar a implementação de complexas regras/ processos de negócios que tratamos como **domínio (Domain Driven Design).**A partir dessa abordagem foram definidos os principais componentes guias de implementação:

Domain Model

Contempla as entidades de domínio, estados e comportamentos, lógica de negócio (specs), interfaces para o serviços de domínio e de repositório e objetos(valueobjects) que agregam valor as entidades.

Domain Service

Serviço de Domínio que orquestra as operações sobre as entidades de domíniopodendo trabalhar com diversas entidades, realizando persistência através de repositórios.

Application Service

Serviço de Aplicação que orquestra ações disparadas pela camada de apresentação e fornece DTOs para a comunicação, compondo adaptadores, interfaces para camadas superiores, suporte para implementação de controle de transações, registro de operações (logs). Também serve como fábrica de serviços para as camadas superiores.

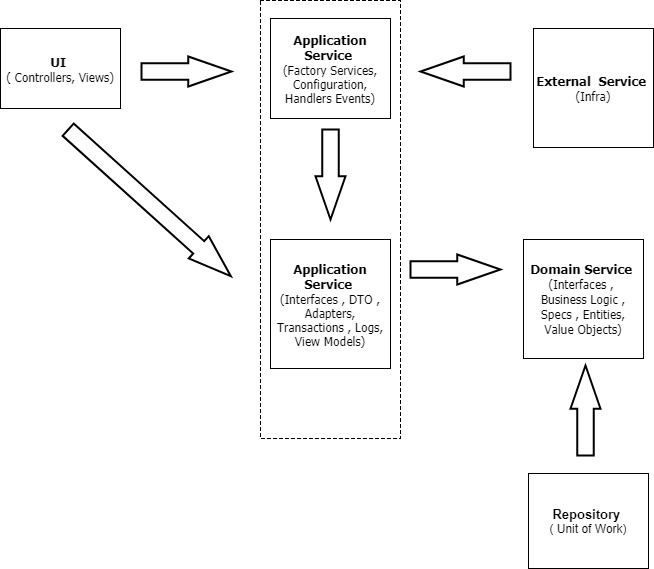
Repository

Realiza a persistência das entidades se comunicando diretamente com o meio de acessos aos dados.

External Service

Serviço externo (infra) que realiza a consulta/persistência por meios externos.

O diagrama exibe o relacionamento entres os componentes e dependências entre os mesmos:



# Estrutura de Projetos

# Módulos Core.Domain e Core.Infra

Os módulos Core.Domain e Core.Infra servem como base para todos os outros módulos.

O Core.Domain disponibiliza recursos de suporte operações de domínios como por exemplo:

* Domain Events (permite enviar eventos de domínios)
* Domain Notifications (técnica para notificação de domínios)
* SpecificationValidation (técnica facilitadora para implementação das regras de negócio).
* AssertConcern (técnica facilitadora para implementação das regras de negócio).
* ValueObjects – classes que agregam valor as entidades de domínio como por exemplo:

Telefone,Cep, Cnae, Cnpj, Cpf, DataFormatExtensions , Email, InscricaoEstadual

# External Service

Falar sobre o Log

Falar sobre

# SpecificationValidation

# Módulos Core Domain e Core Infra

Falar sobre

# Specs

# External Services

# XXXXXXXXXXXXXXXXX

# XXXXXXXXXXXX

# XXXXXXXXX

# XXXXX

# XXXXX

# XXXX

# Glossário

Domain Model

Uma entidade de domínio, possui estados e comportamentos, lógica de negócio,etc

# Criar procedimento

PPPPPPPPPPPPPPPPP