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 – Eventos de Domínio 9

8.3 Core Domain – Eventos de Domínio 9

8.4 Core Domain – Regras de Negócio 9

8.5 Core Infra – Visão Geral 11

8.6 Camada Infraestrutura – Registro de Atividades 12

8.7 Camada Infraestrutura Persistência - Core Data – Visão Geral 15

8.8 Camada Domínio – Entidades e Regras de Negócios 15

8.9 Camada de Aplicação – Visão Geral 16

8.10 Mensagens 18

8.11 18

8.12 18

8.13 18

8.14 18

8.15 Especificações e Validações 18

8.16 Entidade 19

8.17 Repositorio e Unit of Work 19

8.18 Paginação 19

8.19 Camada Apresentação 19

8.20 Estrutura de Projetos 19

8.21 Domain Model – Entidade 20

9. Visão de Implantação 20

10. Dimensionamento e Performance 20

11. Qualidade 20

12. Modelagem da Solução 20

13. Estrutura de Projetos 22

14. Módulos Core.Domain e Core.Infra 22

15. External Service 22

16. Specification Validation 22

17. Módulos Core Domain e Core Infra 22

18. Specs 23

19. External Services 23

20. XXXXXXXXXXXXXXXXX 23

21. XXXXXXXXXXXX 23

22. XXXXXXXXX 23

23. XXXXX 23

24. XXXXX 23

25. XXXX 23

26. Glossário 23

27. Criar procedimento 23

# 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 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.NETFramework 4.5  C# ,ASP.NET MVC 5 , Jquery, Javascript, Bootstrap |
| 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 |

# 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 à 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 principais 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 (valueobjects) 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.

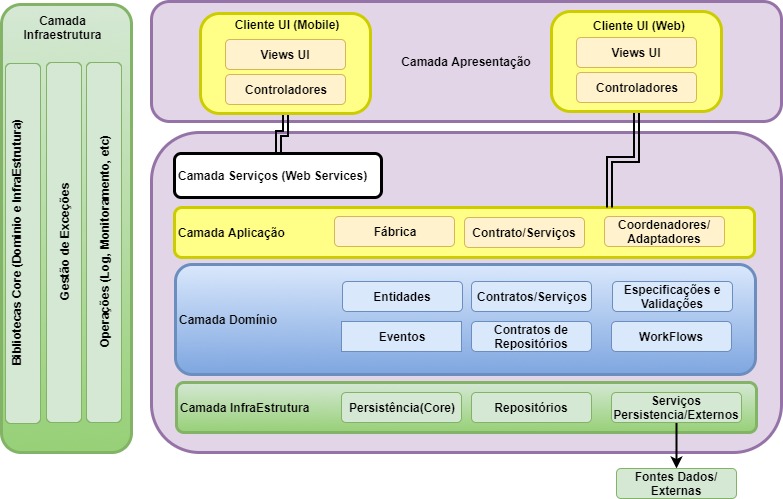
Repository

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.

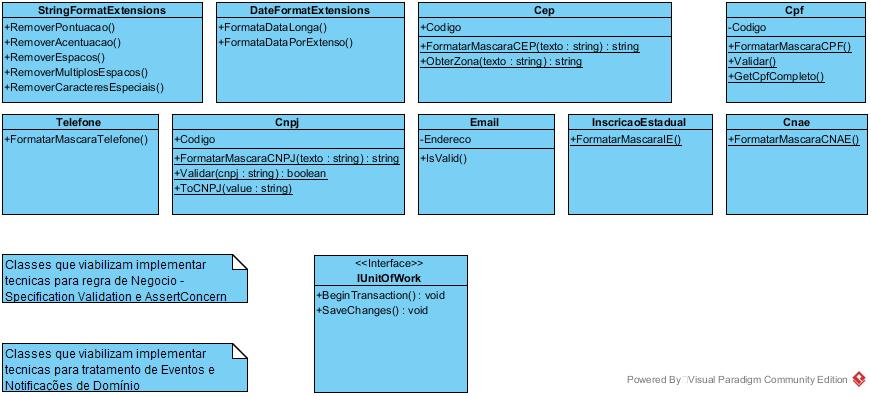
Seguem diagramas guiasdescrevendo os principais conceitos informados acima:





# Visão de Implementação

## Core Domain – Visão Geral



**Objetivo:**

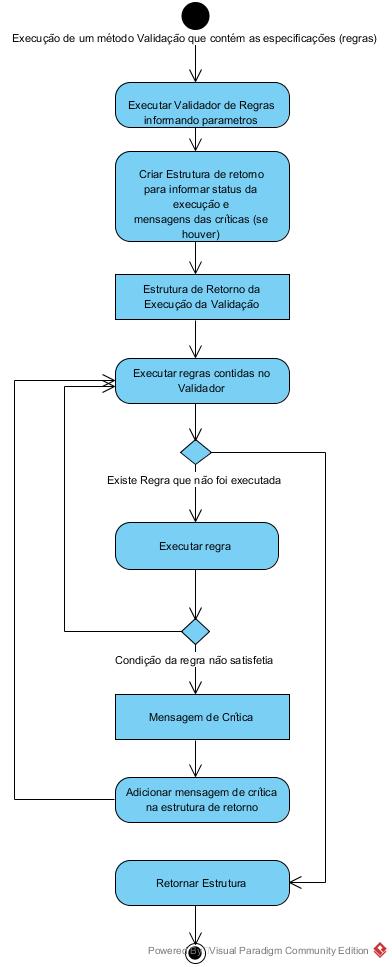
Fornecer funcionalidades de uso comum para atender demandas de entidades de domínio:

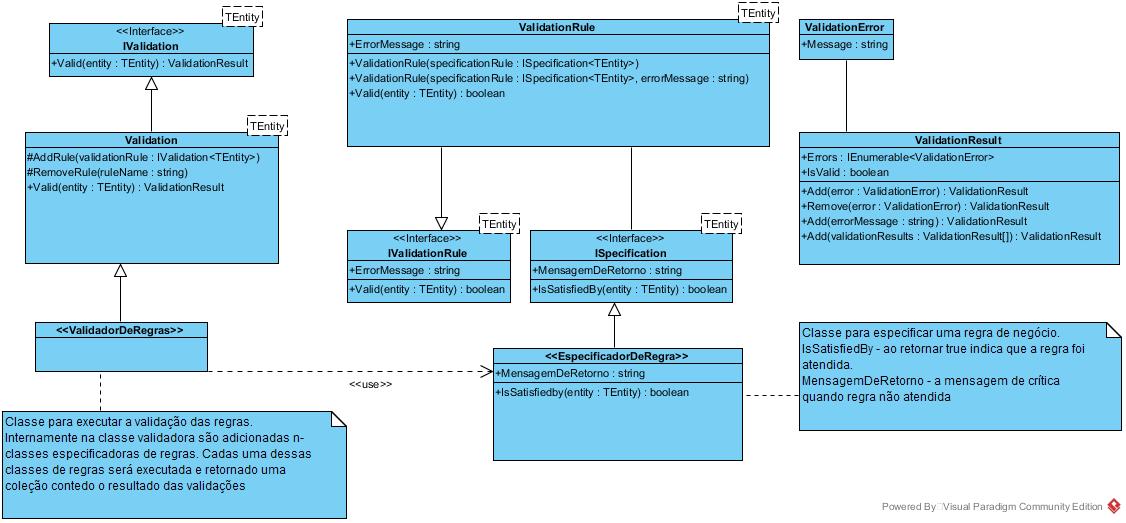
* Formatações de strings e datas;
* Validações e formatações de entidades/atributos, como por exemplo:CEP, CPF, CNPJ,telefone, e-mail,inscrição estadual, CNAE, etc;
* Modelospara implementação de regras de negócio nos contextos (técnicas Specification/Validation);
* Classes que viabilizam implementar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

## Core Domain – Eventosde Domínio

## Core Domain – Eventos de Domínio

## Core Domain – Regras de Negócio



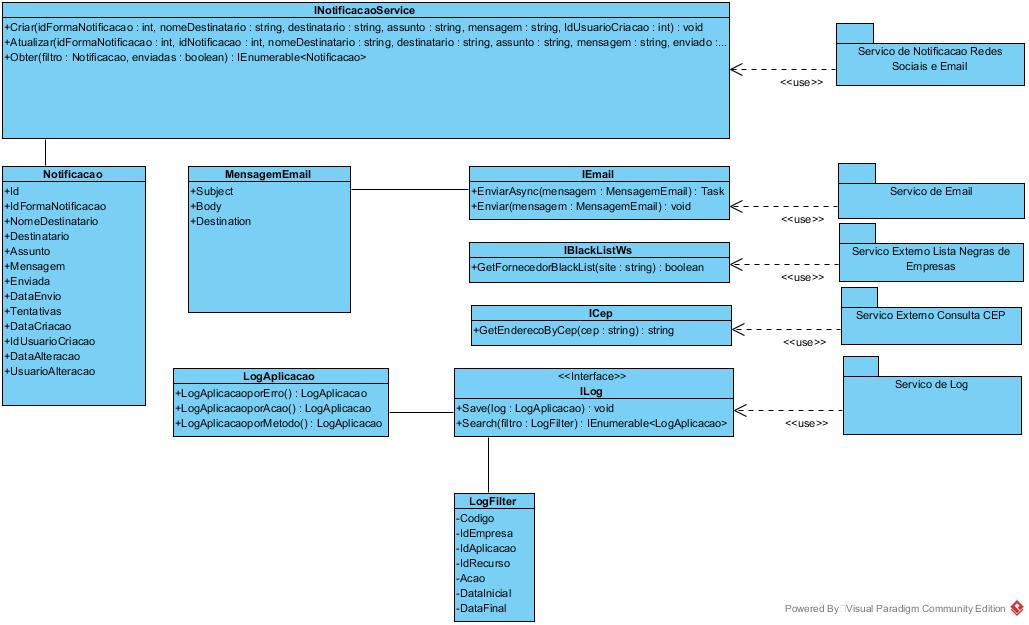


**Objetivo:**

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

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

## Core Infra – Visão Geral

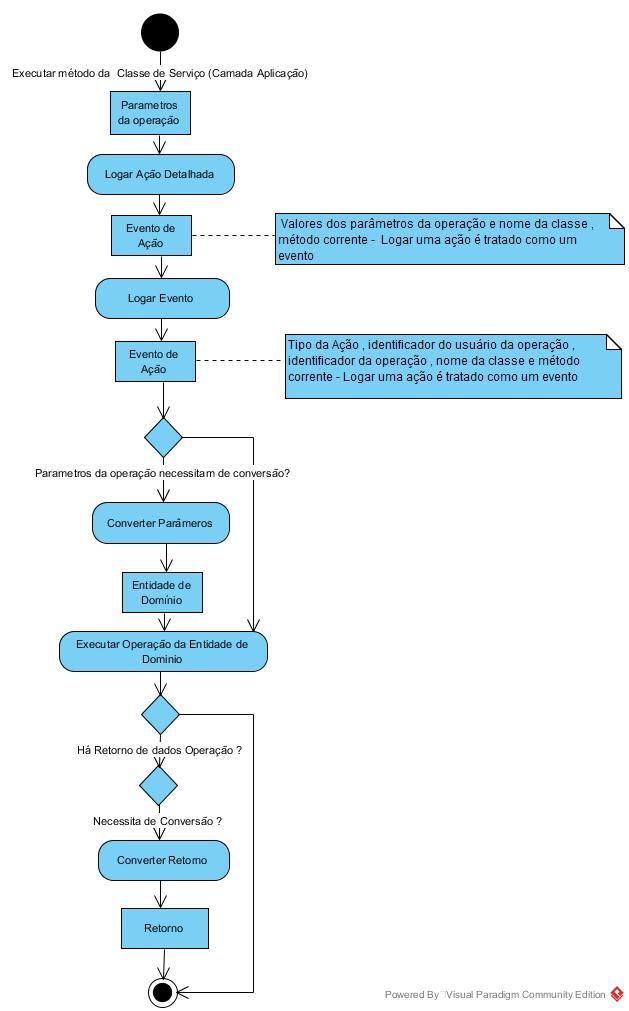


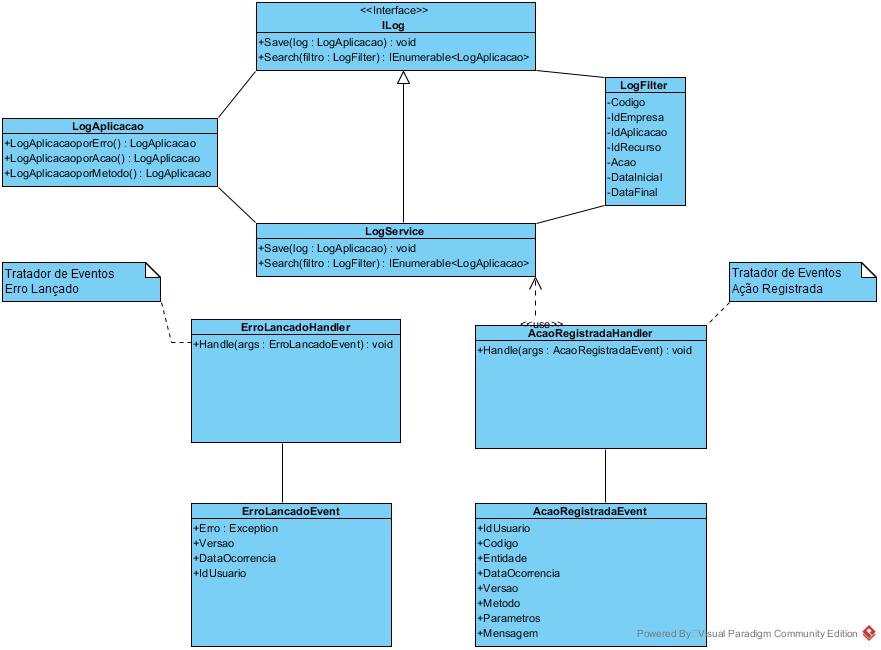
**Objetivo**:

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;

## Camada Infraestrutura – Registro de Atividades



****

**Objetivo:**

Fornecer um serviço para permitir o registro de ações ocorridas nos sistemas por questões de auditoriae/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\_criacaodt\_ultima\_alteracao, id\_usuario\_alteracao*.

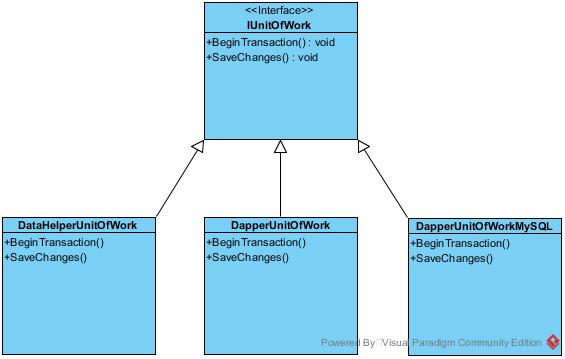
Para atender esse requisito dentro de cada operação disponibilizada no serviço de negócio deve ser implementado internamente a persistência dessas informações;

Além de registrar na própria tabelaque está sendo auditada, também deve ser criado um registro dessa operação em uma tabela específica de logs de operações constando as informações da operação, 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 diagramas anteriores.

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

Diagnóstico de Problema

## Camada Infraestrutura Persistência -Core Data – Visão Geral

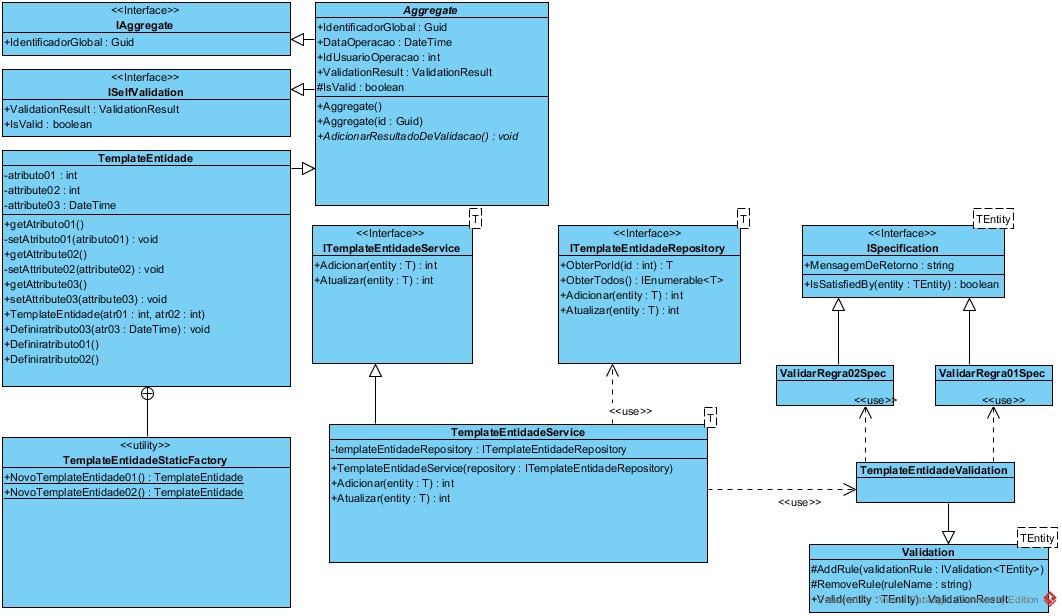


**Objetivo:**

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;
* Aimplementação dessa camadadeve estar livre na medida do possível de qualquer definição de infraestrutura de dados;

Conforme orientação acima temos as seguintes referências comuns implementadas nos projetos que representam essa camada:

* Especificamente para essa camada não devem ser acrescentadas novas referências considerando que as existentes sejam suficientes para implementar a demanda dessa camada;
* Implementação das regras de negócio a partir de modelos usando técnicas de especificação e validação;

## Camada de Domínio – Criação de Entidades

Temos as seguintes orientações básicas para montar 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 é feito por métodos.

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

Construtores

A Entidade deve ter somente dois construtores públicos.

Caso necessite de outros construtores alternativos deve ser usado o recurso de Factory com classe aninhada.

Nota: Um dos construtores deve ser 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 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. Provavelmente a solução do problema deve ser aplicada em outro lugar (camada).

Identificador Global

Toda entidade deve possuir uma propriedade de identificação única. (propriedade 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 de Domínio – Criação de regras de negócio (Especificações e Validações)

Objetivo:

Padrão de especificação é um padrão de design de software específico, pelo qual regras de negócios podem ser recombinadas encadeando as regras de negócios 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 de a 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:

A infraestrutura para implementação de recurso está disponibilizada br.procon.si.Core.Domain

No caso do contrato é o ISpecification.

publicclassAutoValidarExistenciaItemSpec :ISpecification<Auto>

{

publicboolIsSatisfiedBy(Auto entity)

{

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

}

publicstringMensagemDeRetorno

{

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 que tem a implementação para validar as regras e gerar 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:

A infraestrutura para implementação de recurso está disponibilizada no br.procon.si.Core.Domain

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 efetuar a execução para validar as regras e gerar a coleção de resultados

Instanciar a classe agregadora de regras e executar o método Valid.

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

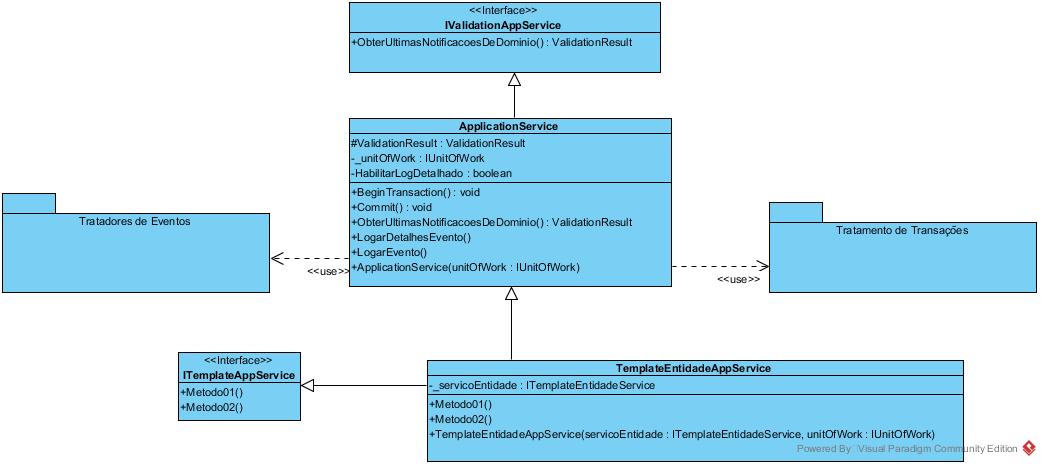
var validationResult = new AutoValidation(\_autoRepository).Valid(entidade);

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.

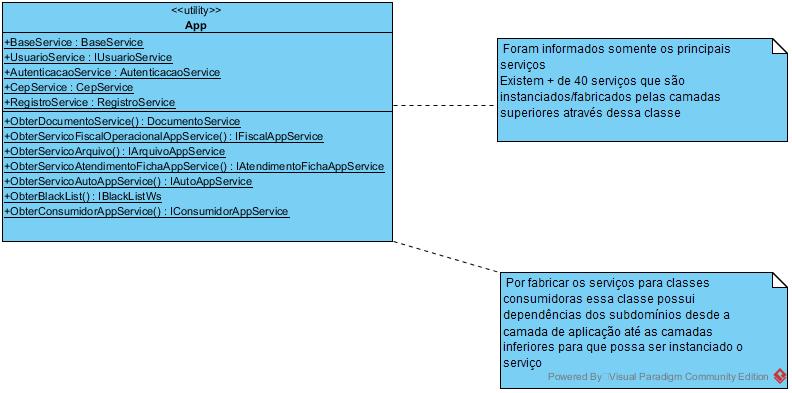
## Camada de Aplicação – Visão Geral



**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 (viewsmodels) 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 (viewsmodels) as camadas superiores;
* Fábrica de serviços expondo as abstrações para as camadas superiores;
* Resolução de dependências

## Mensagens

**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 se utilizar de retorno de algum tipo de mensagem.

## 

## 

## 

## 

## Especificações e Validações

Objetivo:

Padrão de especificação é um padrão de design de software específico, pelo qual regras de negócios podem ser recombinadas encadeando as regras de negócios usando a lógica booleana. O padrão é frequentemente usado no contexto de DDD.

A implementação pode ser implementada da seguinte forma:

Criar a classe com o nome de a 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:

A infraestrutura para implementação de recurso está disponibilizada br.procon.si.Core.Domain

No caso do contrato é o ISpecification.

publicclassAutoValidarExistenciaItemSpec :ISpecification<Auto>

{

publicboolIsSatisfiedBy(Auto entity)

{

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

}

publicstringMensagemDeRetorno

{

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 uma outra classe agregadora das regras.

Essa classe deve herdar da classe que tem a implementação para validar as regras e gerar 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:

A infraestrutura para implementação de recurso está disponibilizada no br.proconsi.Core.Domain

Criado a classAutoValidation.

Herdado da Classe Base Validation.

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

publicclassAutoValidation : 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 efetuar a execução para validar as regras e gerar a coleção de resultados

Instanciar a classe agregadora de regras e executar o método Valid.

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

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.

## Entidade

## Repositorio e Unit ofWork

## 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. Para facilitar a manutenção e posterior escalabilidade a solução inicial teve que ser revisada. 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 |  |
| Atendimento.Data |  |
| Atendimento.Domain |  |
| Atendimento.Infra |  |
| Atendimento.Tests |  |
| Core.Data |  |
| Core.Data.Tests |  |
| Core.Domain |  |
| Core.Domain.Mensagens |  |
| Core.Infra |  |
| FiscalizacaoOperacional.Application |  |
| FiscalizacaoOperacional.Data |  |
| FiscalizacaoOperacional.Domain |  |
| FiscalizacaoOperacional.Infra |  |
| FiscalizacaoOperacional.Tests |  |

## 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.

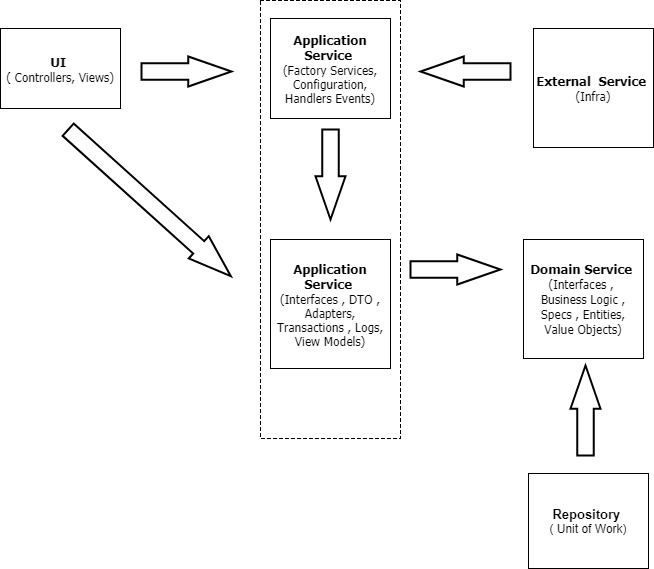
Repository

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