Consistência e Validação

Conceitos e Modelagem

2014-07-17

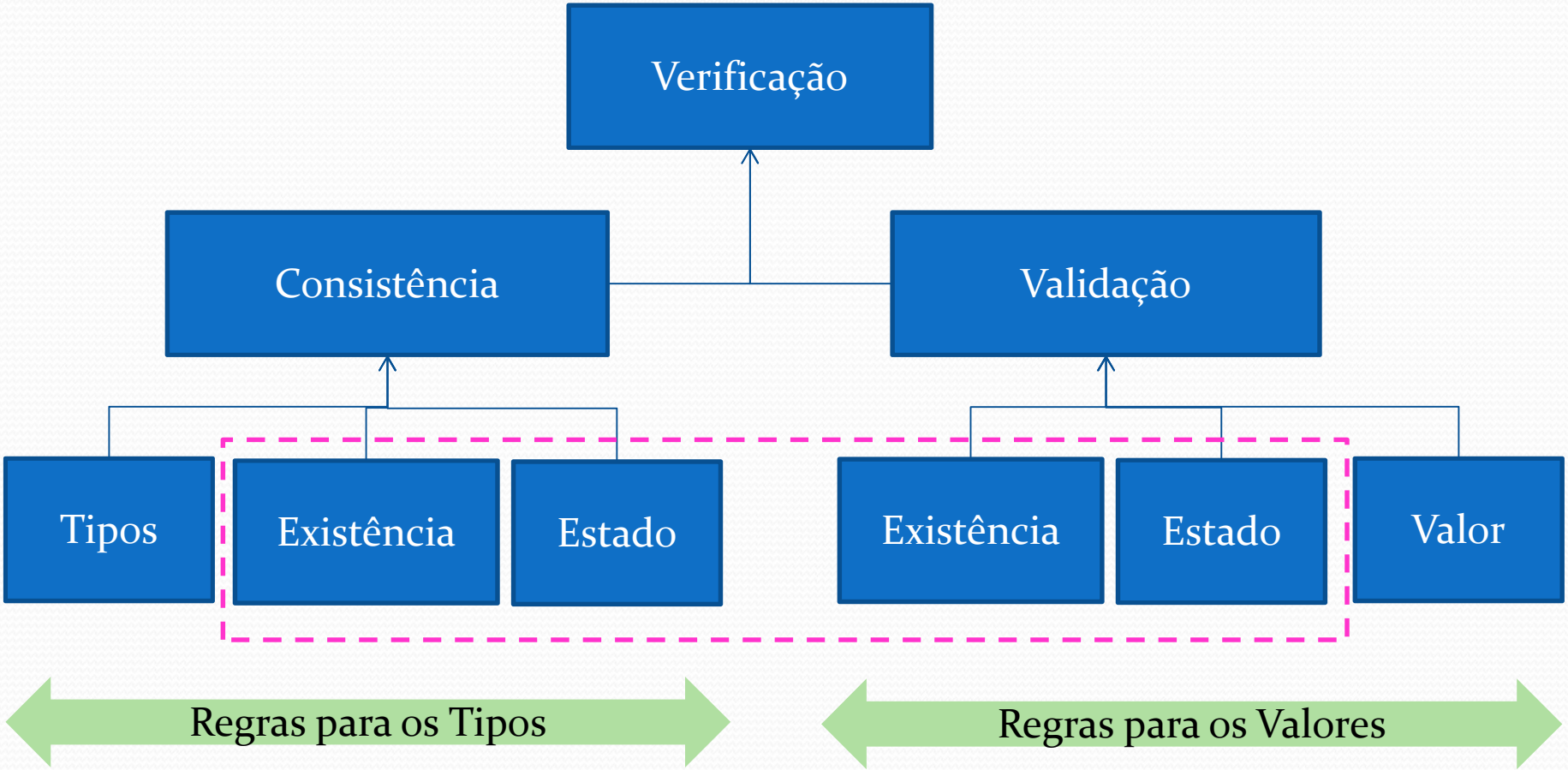
sergiotaborda@zbra.com.br

Conteúdo

- Conceitos
  - Verificação : Consistência vs Validação
  - Consistência, Compiladores e Linguagens de Programação
  - Validação e Restrições do Espaço de Estados
  - Conceito de Objeto vs PropertyBag Pattern
- API
  - Requisitos
  - Apresentação
  - Exemplo de Uso
  - Composição
  - Exemplo Completo
  - Anotações
  - Consistência por Validação (Integração com AOP)

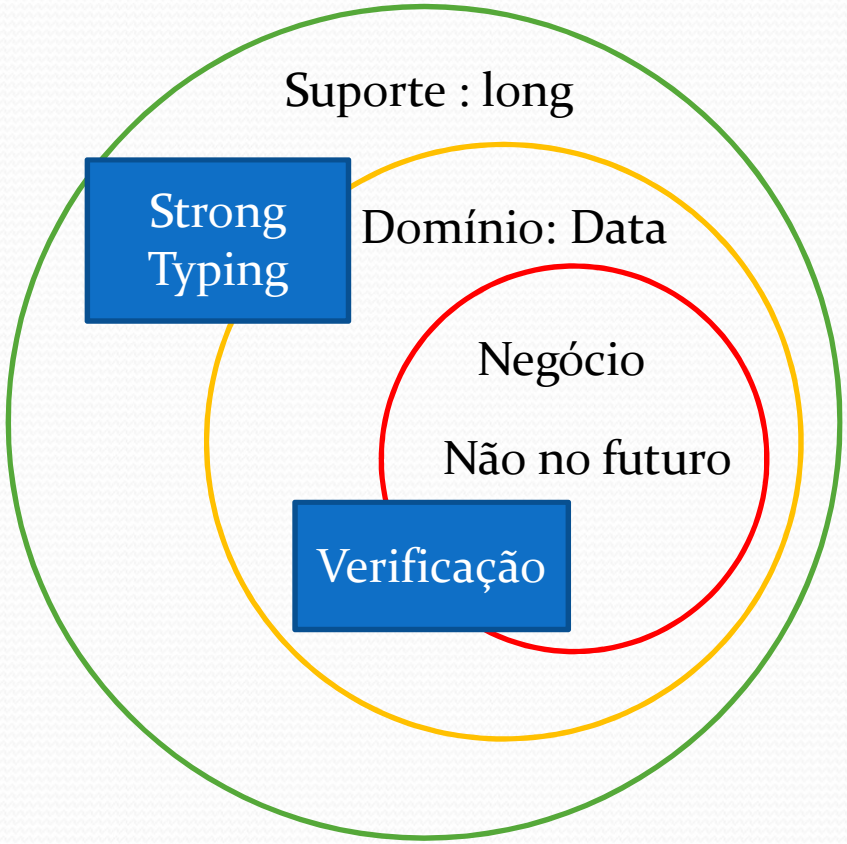
Conceitos

Verificação



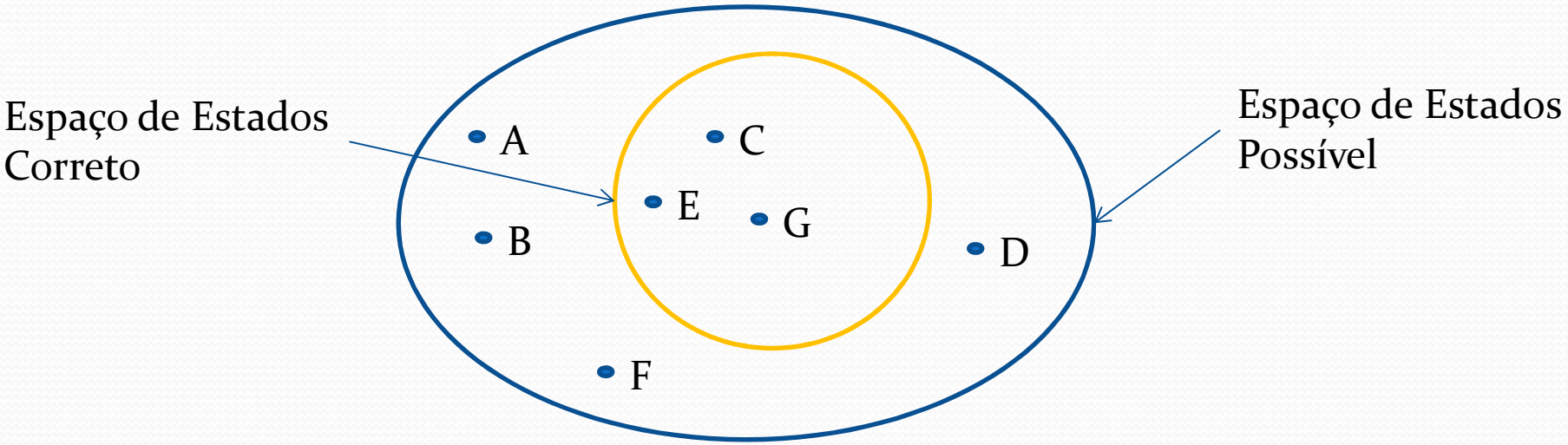
Suporte, Domínio e Negócio

- Suporte – O tipo "mais primitivo" que é utilizado como abstração para guardar o Espaço de Estados
  - *Tempo que passou desde a meia-noite de 01/01/1970 em milisegundos*
- Domínio – Regras relativas à abstração.
  - *Datas não se somam nem se multiplicam (longs sim)*
- Negócio – Constrangimentos suplementares às regras do Domínio que moldam as regras especificadas para o software
  - *Só pode emitir nota fiscal para factos no passado ou hoje*
  - *Um mesmo domínio pode ter regras diferentes dependendo do que ele representa.*



Espaço de Estados

- Estados possíveis para o objeto podem não ser corretos dentro de uma regra de negócio
  - int pode ser negativo ou positivo, mas a idade de uma pessoa é sempre positiva (uint)
  - NaN é um valor possível para double, mas inútil como argumento da função seno
  - Datas podem ser no futuro ou no passado, mas fatos acontecem apenas no passado e planejamentos apenas para o futuro



Consistência

- Automática
  - Pelo Compilador em tempo de compilação
    - var palavra = “oi”; return palavra * 2;
  - Pela VM em tempo de execução
    - *System.Reflection.RuntimeMethodInfo.**CheckConsistency**(Object target)*
      - *Object does not match target type.*
    - checked { return (int.MaxValue - 4) * 2}
- Manual
  - Pelo Programador
    - If (typeof(string).IsInstanceOfType (object))

Consistência e Linguagem

- Organização
  - Script – código solto num arquivo
  - Rotinas/ Funções – organizado por propósito
  - Objetos – organizado por abstração
  - Pacotes / Namespace – organizado por cooperação
  - Componentes / Módulos – organizado por uso
- Strong Typing
  - Tipos fortes ajudam o compilador e o programador a manter a consistência de tipos e portanto a consistência da abstração sendo usada aproximando o Domínio do Suporte.
    - If ('2' * 3 == 6) If ('2' * 3 == '6') If ('2' * 3 == '222')
- Boa abstração de Tipos Básicos
  - Menos categorias melhor
    - object, interface, mixin, enum, structs, primitives, annotations/attributes
  - Modelo
    - (C#) O que Int16, Int32, Int64 e SByte têm em comum ?
    - (Java) O que Short, Integer, Long e Byte têm em comum ?
    - (C#) Array.Length: int VS Array.LongLength:long
    - (C#) Enumerable.Count VS Enumerable.LongCount VS (Java) Stream.Count() : long VS Infinite Enumerable
    - (C#) enum: int VS enum:long VS var index = (int) enum;

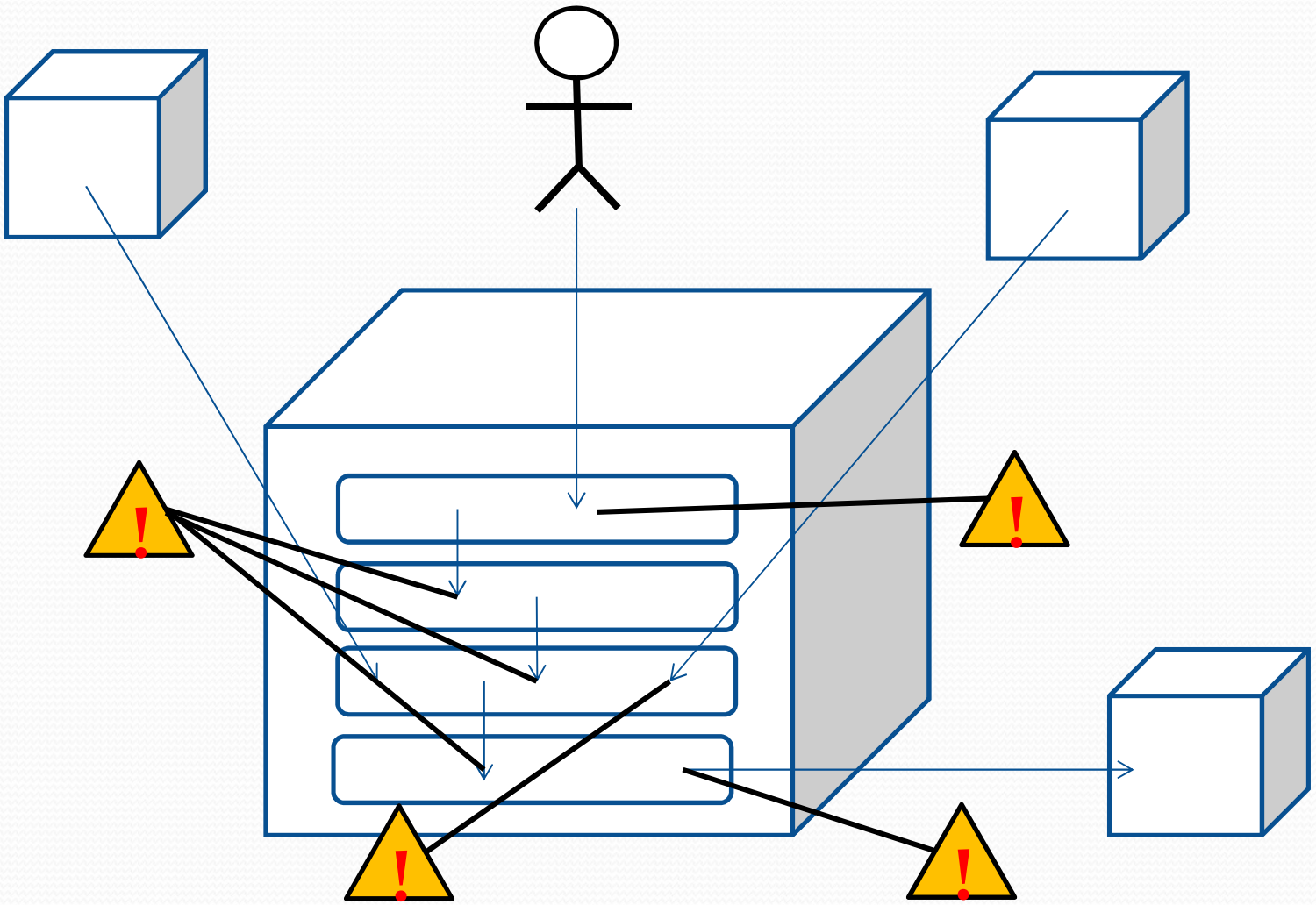
Validação

- Manual
  - Pelo Programador
    - Sempre que uma regra de domínio ou negócio precisa ser verificada
    - Sempre que um dado vem do exterior (do usuário, de outro sistema, de outra camada, ...)
- Automática
  - Por Frameworks de ajuda
    - Por exemplo: sempre que um dado estiver para ser gravado no banco de dados
    - Sempre que um dado vem do exterior (do usuário, de outro sistema, de outra camada, ...)

O quê verificar?

- Consistência
  - Tipos, casts, conversões
  - Existência de tipos, métodos, enums (reflection)
- Validação
  - Espaço de Estados
    - Se o estado resultante da ação pertence ao espaço de estados
    - Se o estado resultante pertence ao domínio (de negócio)
  - Mudança de Estado
    - Se a ação é possível para o estado corrente

Onde Validar ?



Onde Verificar?

- Sempre que, no fluxo de execução, há uma mudança de:
  - Nodo
    - *"Trust No Other"*
    - Todos os nodos devem verificar o que receberam de outros nodos (p.e.: webservices, sockets, ajax, importação de arquivos)
    - Neste sentido os usuários são considerados nodos, assim como outros sistemas ou aplicações.
  - Andar
    - *"Don't Trust yourself"*
    - Todos os andares devem verificar o que receberam de andares superiores.
  - Camada
    - *"Don't trust what others have trusted"*
    - Todas as camadas devem verificar o que receberam de outras camadas
      - Por exemplo, a API de envio de e-mail, valida o e-mail, mesmo quando o Service que a executa já validou antes, e o sistema já validou na entrada dos dados pelo usuário.
  - Objetos
    - *"You can't fool me" ou "You shall not pass!"*
      - Todos os métodos públicos devem realizar consistência
      - Todos os métodos públicos devem realizar validação
    - *"It's my code and I trust what I want to"*
      - Métodos privados não precisam de validação, mas podem precisar de consistência. Cada objeto sabe o que precisa ser feito para garantir a sua integridade interna

Como validar ?

- Sequência de Decisões
  - If-else
  - If-else-if
  - Switch
- Sequência condicional de Decisões (if de if)
- Repetição condicional de Decisões (while de if)
- Sequência condicional de Repetições condicionais de Decisões (if de while de if)
- Frameworks de Validação

Exemplo

```
public IList<EconomicSubActivity> FindEconomicSubActivities (
EconomicActivity activity)
{
     if ( activity == null){ // Consistência
               throw new ArgumentNullException("EconomicActivity is required")
     }
    return DoRealFind(activity)
}

public  int CountEconomicSubActivities ( EconomicActivity activity)
{
     var list =   FindEconomicSubActivities (activity); // Delegação de consistência !
     if ( list == null || list.Count  == 0){ // Validação
           throw new Exception("Expected list of sub-economic ativities is required");
     }
    return list.Count;
}
```

Consistência Manual

- Complexo
  - Lembrar de todas as regras
- Repetitivo
  - Não se aproveita o código e leva a programação por copy-paste (PCP) ou delegação
- Fácil de Esquecer
  - Escrever o mesmo código todas as vezes
- Chato
  - É simplesmente muito mecânico para ser divertido

Consistência Manual - Opções

- Conjunto de classes (Camada/Library) para encapsular as regras e comandos mais usados de uma forma conveniente de usar.
  - O Programador ainda tem que lembrar de usar estas classes e saber minimamente como usá-las
- Utilização de AOP junto com o conjunto de classes para forçar a verificação mesmo quando o programador se esquece.
  - Pode obrigar a que todas as classes passem por AOP
- Utilização de Pos-processadores. Um programa especifico analisa o código e/ou o compilado para descobrir problemas
  - Ainda é possível esquecer ou ignorar os resultados.
- Utilização de Pré-Processadores de compilação. O compilador faz o trabalho e trava a compilação se encontrar problemas.
  - É a melhor opção, mas precisa de suporte do compilador.

Consistência pelo Compilador

- É a tendência moderna
- Modificar o compilador para entender regras especiais
  - Através de anotações do código
  - Através de inspeção do código
  - Através de *plugins* (extensões do compilador)
- Compiladores
  - Oracle Java 8
    - Type Annotations
      - public @NotNull List<@NotNull T> Get()
      - public @NotNull Optional<T> Get(); // auto ?
    - The Check Framework
  - Scala Compiler Plugins
  - Microsoft Roslyn Compiler Platform

PropertyBag Pattern

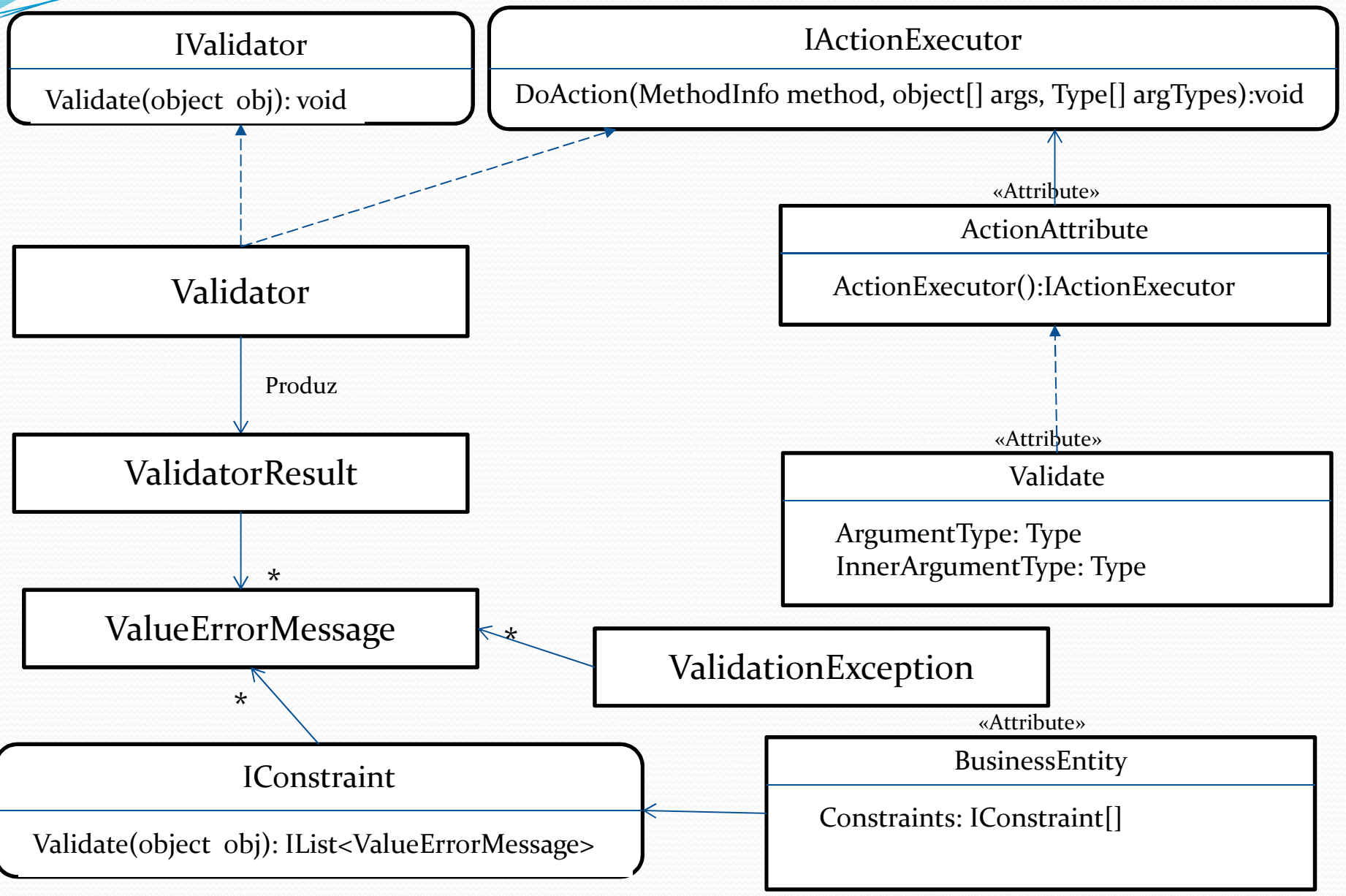
- Um objeto que contém apenas propriedades
  - As propriedades podem ser readonly – normalmente associadas a algum cálculo/algoritmo envolvendo as outras propriedades
  - As propriedades podem ser writeonly – normalmente em conjunção com propriedades readonly
- Padrões derivados
  - (D)TO – (Data) Transfer Object = PropertyBag serializável
- Um objeto que não contém regras sobre os valores das propriedades
  - Então precisa de validação por meios externos. A validação depende do contexto que recebe o objeto e não do objeto em si.

Validação

- De Valores de Propriedades
  - Dado Vazio/Obrigatório
  - Em um Intervalo (datas , quantidades)
  - Url, e-mail, CNPJ, CPF, CEP, ...
- De Agregados de Propriedades (PropertyBag pattern)
  - Valor em todas as Propriedades/Nenhuma
  - Compatibilidade entre valores de Propriedades diferentes (Estado » Pais)
  - Regra dependente do valor de outra propriedades (CEP » Pais)
- De Listas
  - Apenas um usuário pode ter a capacidade de realizar uma ação
  - Todos os itens do pedido devem ter uma quantidade
- Depende de Regras de Negócio
  - Obrigatório Se ...
  - Maior/Menor que outro campo , ou outro campo de outro objeto
  - Não mais que um (Único, Único Se)
  - Existente (em todo o Sistema)
- Depende do Domínio
  - Verificar que o CNPJ realmente foi atribuído a uma empresa, e a aquela empresa
  - Verificar que o e-mail realmente existe e pode receber mensagens
  - Verificar que uma data realmente existe no calendário (29/02/1600)

API de Validação

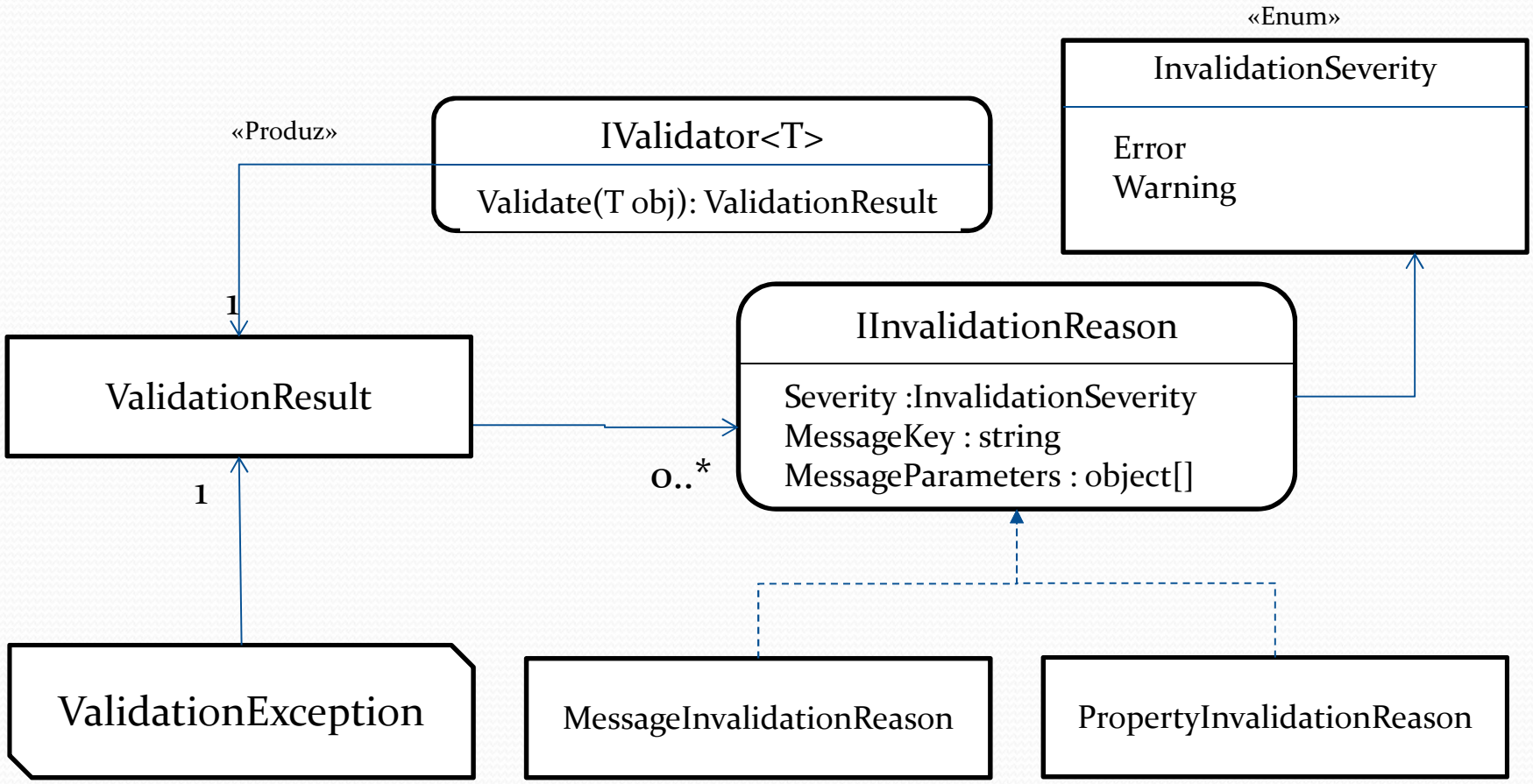
Fênix API


IValidator
Validate(object obj): void
IActionExecutor
DoAction(MethodInfo method, object[] args, Type[] argTypes):void
«Attribute»
ActionAttribute
ActionExecutor():IActionExecutor
Validator
Produz
ValidatorResult
«Attribute»
Validate
ArgumentType: Type
InnerArgumentType: Type
*
ValueErrorMessage
*
ValidationException
*
«Attribute»
BusinessEntity
Constraints: IConstraint[]
IConstraint
Validate(object obj): IList<ValueErrorMessage>

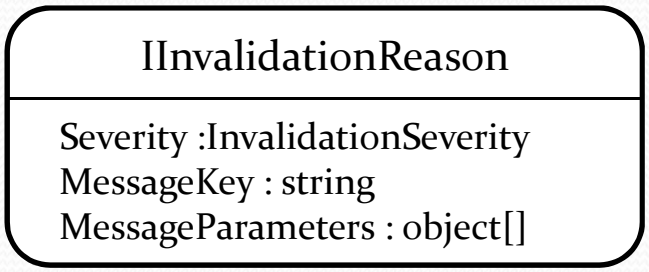
Requisitos

- Reuso
  - Algoritmos encapsulados (validar e-mail, cnpj, telefone, ...)
  - Composição (**Não é Vazio** *e* é um **E-mail válido**)
  - Configurável (O tamanho é entre *min* e *max*)
  - Meta-configurável (Uso de anotações, xml, etc...)
  - Internacionalizado (a API não depende do sistema)
  - Extensível (outras bibliotecas podem adicionar validadores)
- Contexto
  - Independência de Camada (View / Presenter / Service)
  - Capaz de acessar outros recursos (DB, rede, arquivos)
  - O algoritmo de validação não depende de para quê é usado.

API – Algoritmo Encapsulado



API - Internacionalização



- O primeiro parâmetro é o valor validado
- Se é uma propriedade, o segundo parâmetro é o nome da propriedade.
- Os outros parâmetros são os valores de validação
  - LengthValidator : min, max
- É possível criar um modelo mais fortemente tipado
  - Por exemplo: LenghtInvalidationReason
  - Útil se a API for usada como biblioteca.

API – Implementando IValidator

```
 public class NotNullValidator<T> : IValidator<T>
{
    public string InvalidMessageKey { get; set; }

    public NotNullValidator()
    {
          this.InvalidMessageKey = "Invalid.Is.Null";
    }

    public ValidationResult Validate(T candidate)
     {
       var result = new ValidationResult();
       if (candidate == null)
       {
          result.AddReason(MessageInvalidationReason.Error(InvalidMessageKey));

       }
       return result;
    }
}
```

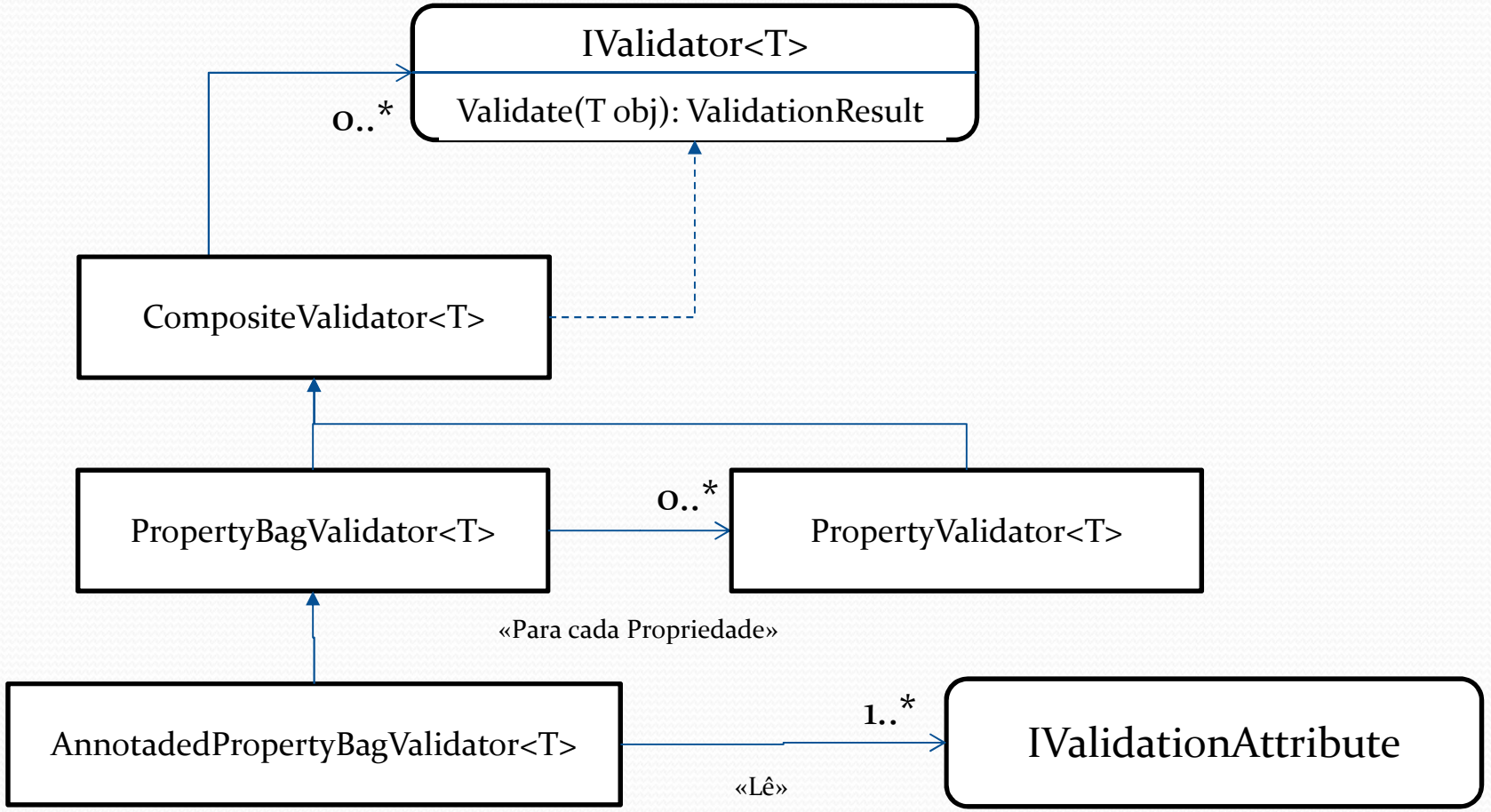
API – Usando

```
 public  void Save (Company  company)
 {
      var validator = new CompanyInsertValidator (dataContext); // acesso ao ambiente
      var result = validator.Validate(company);

      if (!result.IsValid())
      {
         throw new ValidationException (result);
      }
      . . .
}
```

API – Usando Diferente

```
public  void Save (Company  company)
{
          if (company.IsNew() )
          {
               company.ValidateWith (new CompanyInsertValidator (dataContext));
          }
          else
          {
               company.ValidateWith (new CompanyUpdateValidator (dataContext));
          }

          . . .
}
```

API - Composição



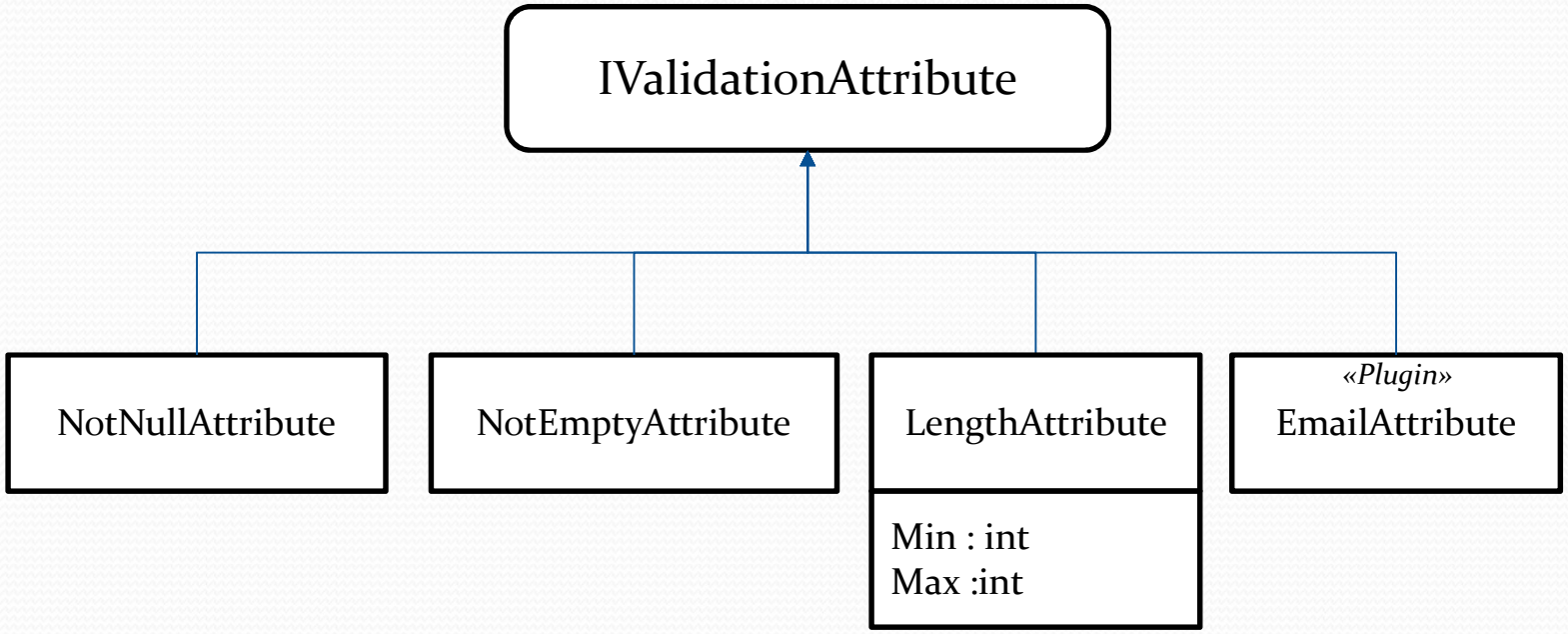
API – Validando Propriedades

```
public class AddressValidator : PropertyBagValidator<Address>
{
     public AddressValidator()
     {
        AddPropertyValidator( a => a.Country,    new CountryValidator()); // configuração , composição
     }
     public override ValidationResult Validate ( Address address)
     {
        if (address == null)
        {
            return new ValidationResult(); // não valida null. Isso é outro Valdiator que faz (SRP)
        }

        var result = base.Validate (address); //  composição

        if (result.IsValid())
        {
            var addressModel = AddressModelFactory.GetFactory().GetAddressModel(address.Country);

            return  addressModel.GetAddressValidator().Validate(address);
        }
        return result;
      }
 }
```

API – Tudo junto

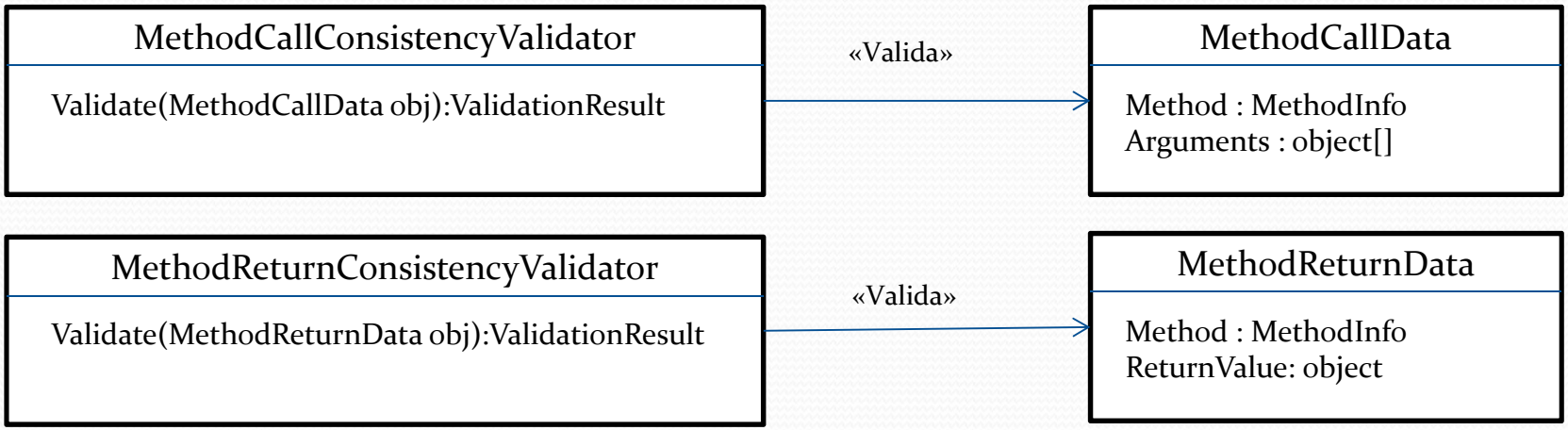
```
public class CompanyCreationValidator : AnnotadedPropertyBagValidator<Company>
{
    IReferenceDataService referenceDataService;
    IIndicadoresDataContext dataContext;

     public CompanyCreationValidator (
          IReferenceDataService referenceDataService, IIndicadoresDataContext dataContext)
    {
        this.referenceDataService = referenceDataService;
        this.dataContext = dataContext;
        Add(new CompanyUpdateValidator(referenceDataService));        // composição
    }
    public override ValidationResult Validate ( Company company)
    {
        var result = base.Validate(company);       // composição

         var attachedCompany = dataContext.Query<Company>().SingleOrDefault(c => c.Id == company.Id);

         if (attachedCompany == null && company.Country != null && company.FiscalNumber != null
           && Exists(company.Country, company.FiscalNumber))              {
            result.AddReason (
        MessageInvalidationReason.Error("Company.FiscalNumber.Duplicated"));
        }
        return result;
    }
}
```

API - Anotações



API – Consistência por Validação

- Usar mecanismo de Validação para realizar Consistência



API – Exemplo AOP

```
[return : NotEmpty]
public IList<EconomicSubActivity> FindEconomicSubActivities ( [NotNull] EconomicActivity activity)
{
        ...
}
```

API – Exemplo AOP

```
public class AnnotationConsistencyAdvice :  AopAlliance.Intercept .IMethodInterceptor
{
        private readonly MethodCallConsistencyValidator callValidator = new MethodCallConsistencyValidator();
        private readonly MethodReturnConsistencyValidator returnValidator = new MethodReturnConsistencyValidator();

        public object Invoke ( IMethodInvocation invocation)
        {
              ValidationResult result;
              if (invocation.Arguments != null && invocation.Arguments.Length > 0)
              {
                    result = callValidator.Validate ( new MethodCallData ( invocation.Method, invocation.Arguments));
                    if (!result.IsValid())
                    {
                        throw new ReceivedParametersConsistencyException (result);
                    }
              }
              object returnValue = invocation.Proceed();
              result = returnValidator.Validate ( new MethodReturnData( invocation.Method, returnValue ));

              if (!result.IsValid())
             {
                 throw new ReturnParameterConsistencyException (result);
              }
              return returnValue;
        }
}
```

References

- JGoodies Validation
  http://www.jgoodies.com/freeware/libraries/validation/
- Spring.NET Validation
  http://www.springframework.net/doc-latest/reference/html/validation.html
- Roslyn Compiler Platform
  http://msdn.microsoft.com/en-us/vstudio/roslyn.aspx
  http://channel9.msdn.com/Events/Build/2014/2-577
- Composite Pattern
  http://en.wikipedia.org/wiki/Composite_pattern
  http://www.javabuilding.com/academy/patterns/composite-object.html
- PropertyBag Pattern
  http://www.javabuilding.com/academy/academy/patterns/property-bag.html