FUTURE DECODED

6-7 OTT '16 / MILANO

IN PARTNERSHIP WITH: CommunityDays.it

www.futuredecoded.it

#FutureDecoded

Extending C# with Roslyn and Code-Aware Libraries

Carlo Pescio

www.futuredecoded.it
#FutureDecoded

Goals

- Perspective on a new technology
  - Stimulate new ideas
- Some practice
  - How to start using it
  - A simple (but not trivial) example
- Back on a larger perspective
  - Overcoming an OOP issue / C# limitation

Extend vs. Constrain

Libraries de facto **extend** the language

"Libraries cannot provide new **inabilities**"

Crista Lopes / Mark Miller

(http://tagide.com/blog/research/constraints)

Adding "inabilities" to the language through a library??

WHY ???

Example 1

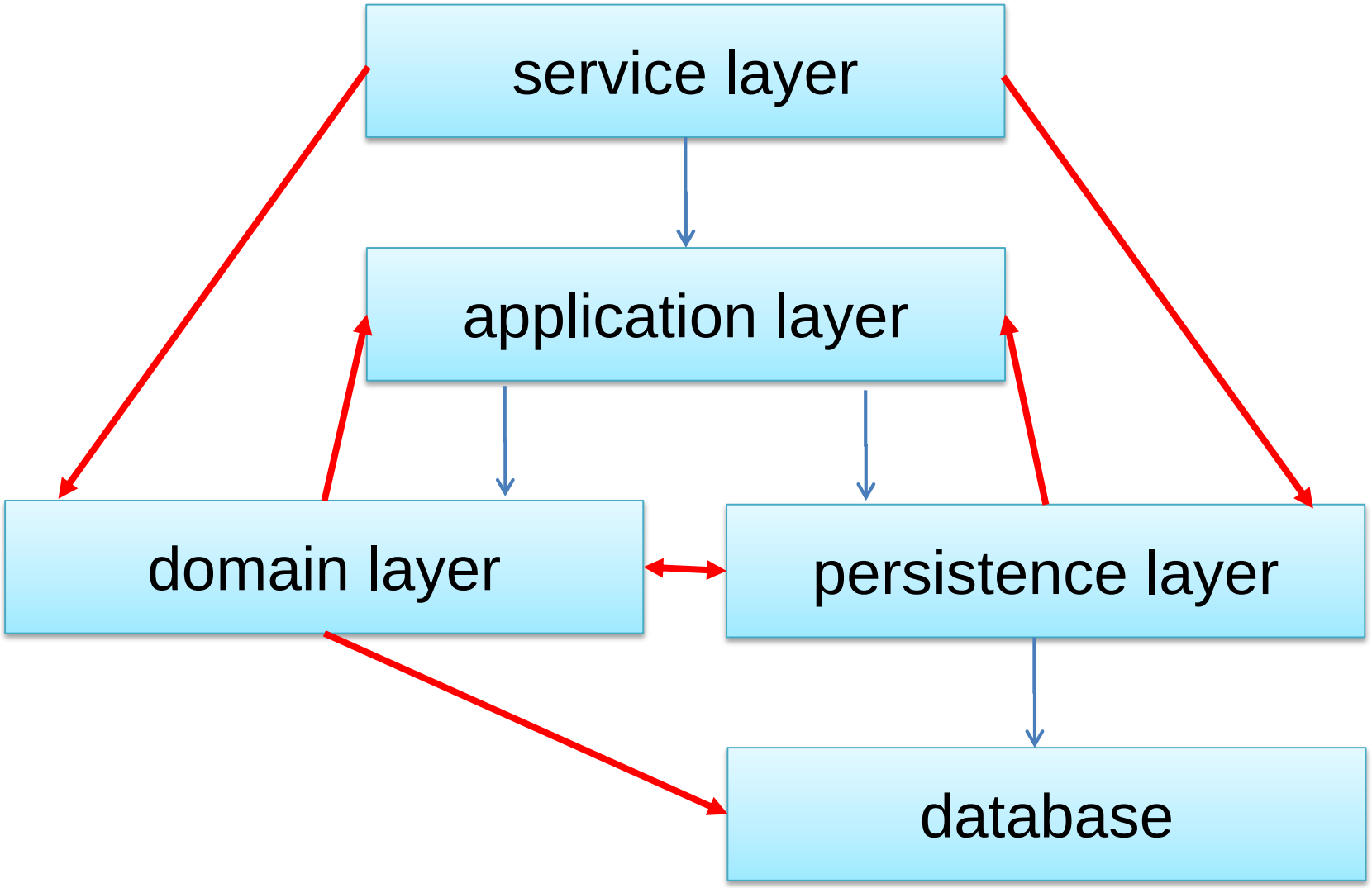
where I2 : interface

```
public static class To<I2> where I2 : class
{
  public static I2 Safely<I1>(I1 from)
    where I1 : class
  {
    return from as I2;
  }
}
```

where I1 : interface

```
Ifc2 q1 = To<Ifc2>.Safely(o1);
```

Example 2


service layer
application layer
domain layer
persistence layer
database

Who’s gonna check?

Not the compiler

Code review?

Some external tool?

The library / module / code itself!

Code-Aware Library

Leveraging the Roslyn compiler model

A library can include “analyzers”

- Analyzers run at compile time
- will see code **using** the library

Library + Analyzer = Code Aware Library

Can provide warnings or errors (and fixes)

Cannot extend syntax (no new keywords)

First steps

Install the .NET Compiler Platform ("Roslyn") SDK from aka.ms/roslynsdktemplates

Create a new project:
File | New Project | C# | Extensibility ->
Analyzer with Code Fix (NuGet + VSIX)

-> sample project will warn if a type name contains lowercase letters

Inside the Solution

- Analyzer project
- “Debugging project” (VSIX)
- (Test project)

Running the debugging project will open another instance of visual studio with your analyzer active.

Create / Open a solution there using your library

Use that to develop and debug your analyzer

Analizers

Not a full tutorial – see “resources” for details

Attribute to be found by roslyn

```
[DiagnosticAnalyzer(LanguageNames.CSharp)]
public class InterfaceCheckAnalyzer : DiagnosticAnalyzer
```

Implementation inheritance to provide common behavior & polymorphic hooks.

Minimal set-up (1)

Diagnostics Descriptors are templates for warning / error messages

```
public override ImmutableArray<DiagnosticDescriptor>
  SupportedDiagnostics { get { … } }
```

```
private static readonly LocalizableString Rule1Title = "'To' parameter must be an interface";
private static readonly LocalizableString Rule1MessageFormat =
   "Safe cast error: 'To' type parameter must be an interface but is {0}";
private static readonly LocalizableString Rule1Description =
   "Safe casts must be from <Interface1> to <Interface2>";
private static readonly string Rule1Category = "Typing";
private static DiagnosticDescriptor Rule1 = new
 DiagnosticDescriptor(DiagnosticId, Rule1Title, Rule1MessageFormat,
                      Rule1Category, DiagnosticSeverity.Error,
                      isEnabledByDefault: true, description: Rule1Description);
```

Minimal set-up (2)

Initialize: register callbacks on syntax tree

```
public override void Initialize(AnalysisContext context)
{
  context.RegisterSyntaxNodeAction(
   AnalyzeSafelyCall, SyntaxKind.InvocationExpression);
}
```

Several SyntaxKind, like type declaration etc.

Key notion: Syntax tree

A syntactic tree of the (parsed) source code.

Rich structure, “learn by exploration”

```
class Program
{
    static void Main(string[] args)
    {
        I1 o1 = new C1();
        I2 q1 = To<I2>.Safely(o1);
        object v = To<string>.Safely("aa").
    }
}
```

Syntax Tree

- InvocationExpression [310..333)
  - SimpleMemberAccessExpression [310..327)
    - GenericName [310..320)
      - IdentifierToken [310..312)
      - TypeArgumentList [312..320)

Key notion: Semantic model

Answers questions like “in which class “Safely” was declared?”

```
class Program
{
    static void Main(string[] args)
    {
        I1 o1 = new C1();
        I2 q1 = To<I2>.Safely(o1);
        object v = To<string>.Safely("aa").
    }
}
```

```
var safelySymbol =
context.SemanticModel.GetSymbolInfo(memberAccessExpr).Symbol as
IMethodSymbol
```

Example: Interface checking

```
public override void Initialize(AnalysisContext context)
{
  context.RegisterSyntaxNodeAction(AnalyzeSafelyCall,
                                   SyntaxKind.InvocationExpression);
}

public void AnalyzeSafelyCall(SyntaxNodeAnalysisContext context)
{
  var invocationExpr = (InvocationExpressionSyntax)context.Node;
  var memberAccessExpr = invocationExpr.Expression as
                         MemberAccessExpressionSyntax;
  if(memberAccessExpr?.Name.ToString() != "Safely") return;

  var safelySymbol = context.SemanticModel.
                     GetSymbolInfo(memberAccessExpr).Symbol
                     as IMethodSymbol;

  var safelyContainingType = safelySymbol?.ContainingType;
  var safelyTypeName = safelyContainingType?.MetadataName;
  if(safelyTypeName != "To`1") return;
```

Emitting diagnostics

```
var safelyContainingType = safelySymbol?.ContainingType;
var safelyTypeName = safelyContainingType?.MetadataName;
if(safelyTypeName != "To`1") return;
```

Compile-Time errors

```
interface I1
{ }
interface I2
{ }
class C1 : I1
{ }

class Program
{
  static void Main(string[] args)
  {
      I1 o1 = new C1();
      I2 q1 = To<I2>.Safely(o1);
      object v = To<string>.Safely("aa");
  }
}
```

Error List

Entire Solution | 2 Errors

Search Error List

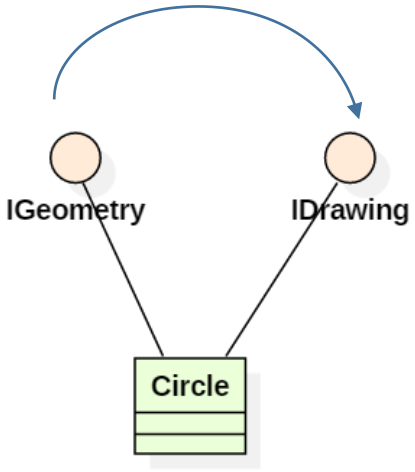
	Code	Description
	InterfaceCheck	Safe cast error: 'To' type parameter must be an interface but is String
	InterfaceCheck	Safe cast error: 'Safely' argument must be typed as an interface, but its type is String

Extending C#?

Libraries cannot introduce “inabilities” and constraints

Code-aware libraries can!

=> Inability to make an unsafe cross-cast



Inception

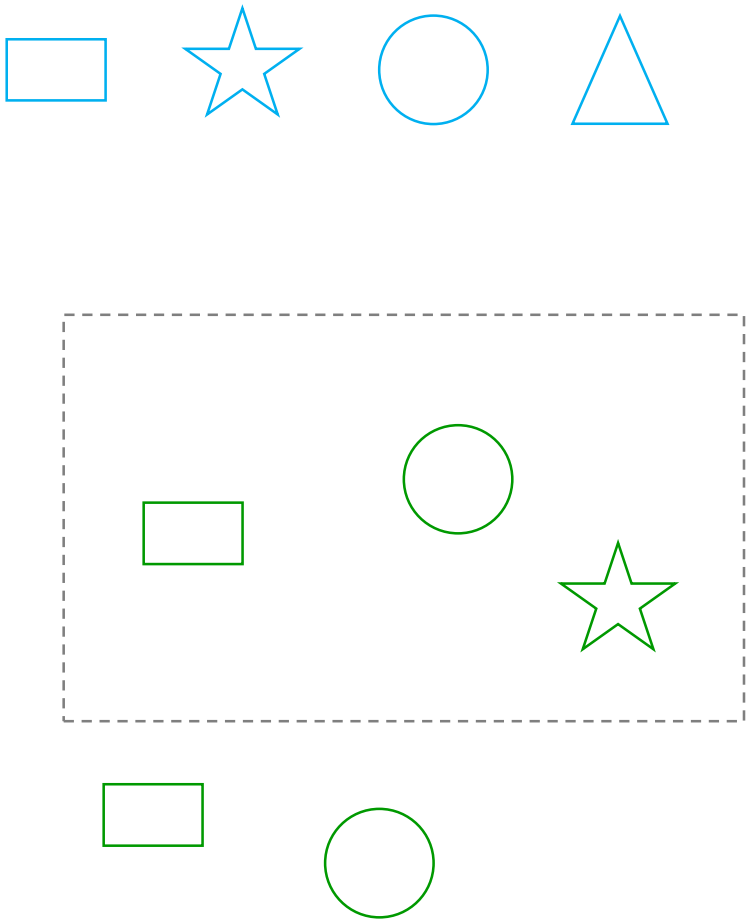
On Growth and Software

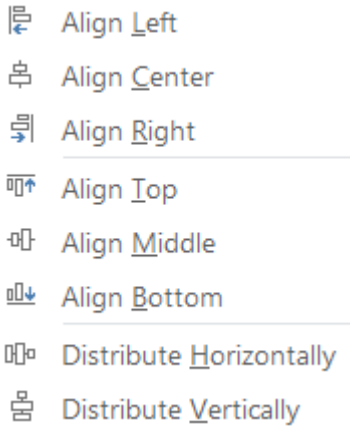
nCrafts 2016

Paris

http://videos.ncrafts.io/video/167699028

Vector drawing for dummies



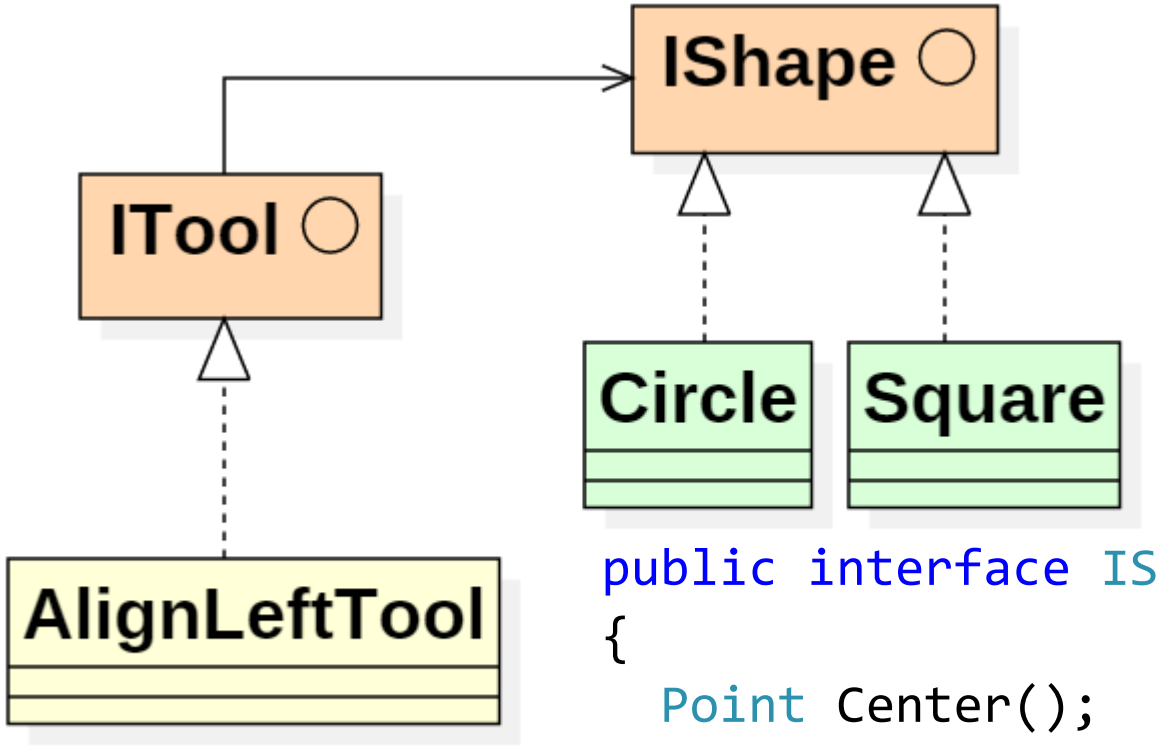


Growth Model

- New shape types will come over time
- New behaviors will come over time
- Behaviors can be composed out of a fixed core
  - That entire menu only requires {center, bounding box, move}

I'm dealing only with geometry right now

```
public interface ITool
{
  IEnumerable<IShape> ApplyTo(List<IShape> source);
}
```




```
public interface IShape
{
  Point Center();
  Box BoundingBox();
  IShape Move(Point newCenter);
}
```

Extensible? Sure!

- New shape type => new class
- New tool => new class
  - Given the right core methods in IShape
  - Geometric core methods will saturate anyway

Breaking the growth model

We only have geometry so far

I want rendering (new core methods!?)

I may want only geometry

I may want only rendering

I may want both

=> Geometry and Rendering in distinct artifacts?

```
interface IShape
{
  IShape Move(Point newCenter);
}
```

```
interface IDrawing
{
  void Render();
}
```

```
partial class Circle : IShape
{
  public IShape Move(Point newCenter)
  {
  return new Circle(newCenter, radius);
  }
}
```

```
partial class Circle : IDrawing
{
  public void Render()
  {
  Console.WriteLine(
    "I'm a Circle with radius " +
    radius.ToString());
  }
}
```

```
partial class Circle
{
  private readonly Point c;
  private readonly double r;

  public Circle( Point center, double radius )
  {
  c = center;
  r = radius;
  }
}
```

```
constraint
  IShape => IDrawing
```

```
static void Main(string[] args)
{
  IShape c = new Circle(new Point(10, 10), 5);
  IDrawing d = c as IDrawing;
  d.Render();
}
```

1) Reframe as Library

```
IDrawing d = c safely_as IDrawing;
```

```
IDrawing d = To<IDrawing>.Safely(c);
```

```
constraint  IShape => IDrawing
```

```
[module: Constraint.Implies<IShape,IDrawing>]
```

```
[module:
  Constraint.Implies(typeof(SafelyAsSample.IShape),
                     typeof(SafelyAsSample.IDrawing))]
```

2) Define the rules

```
To<type>.Safely(expression);
```

- type must be an interface (I2)
- expression must be typed as an interface (I1)
- there must be a constraint I1 => I2

```
[module:Constraint.Implies(typeof(type1),typeof(type2))]
```

- type1 and type2 must be interfaces
- Every class implementing type1 must implement type2

3) Turn rules into analyzer

Local checks (easy)

- constraint parameters must be interfaces
- To and Safely parameters must be interfaces

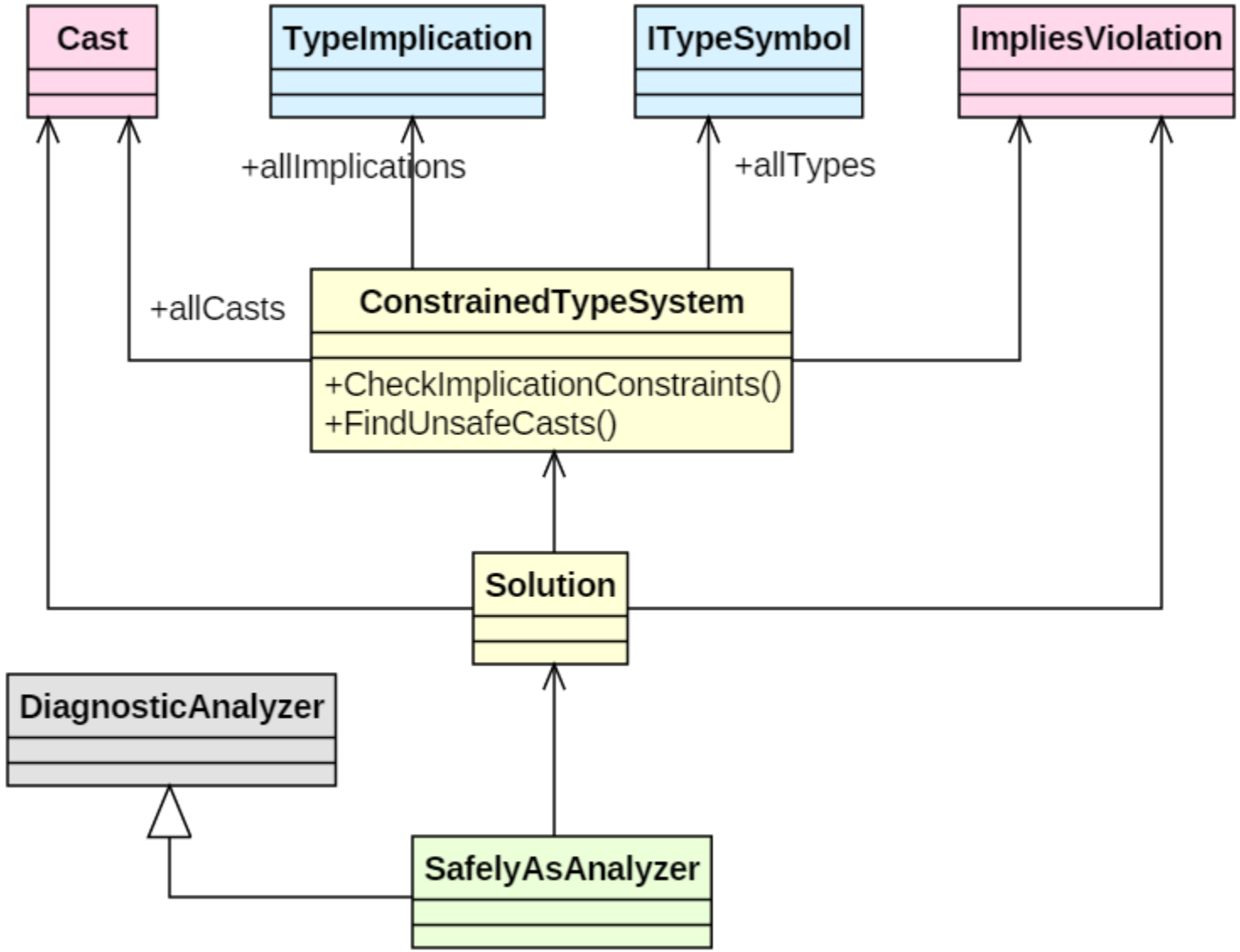
Global checks (“harder”)

- “every type implementing type1 must implement type2”
- “there must be a constraint I1 => I2” (somewhere)

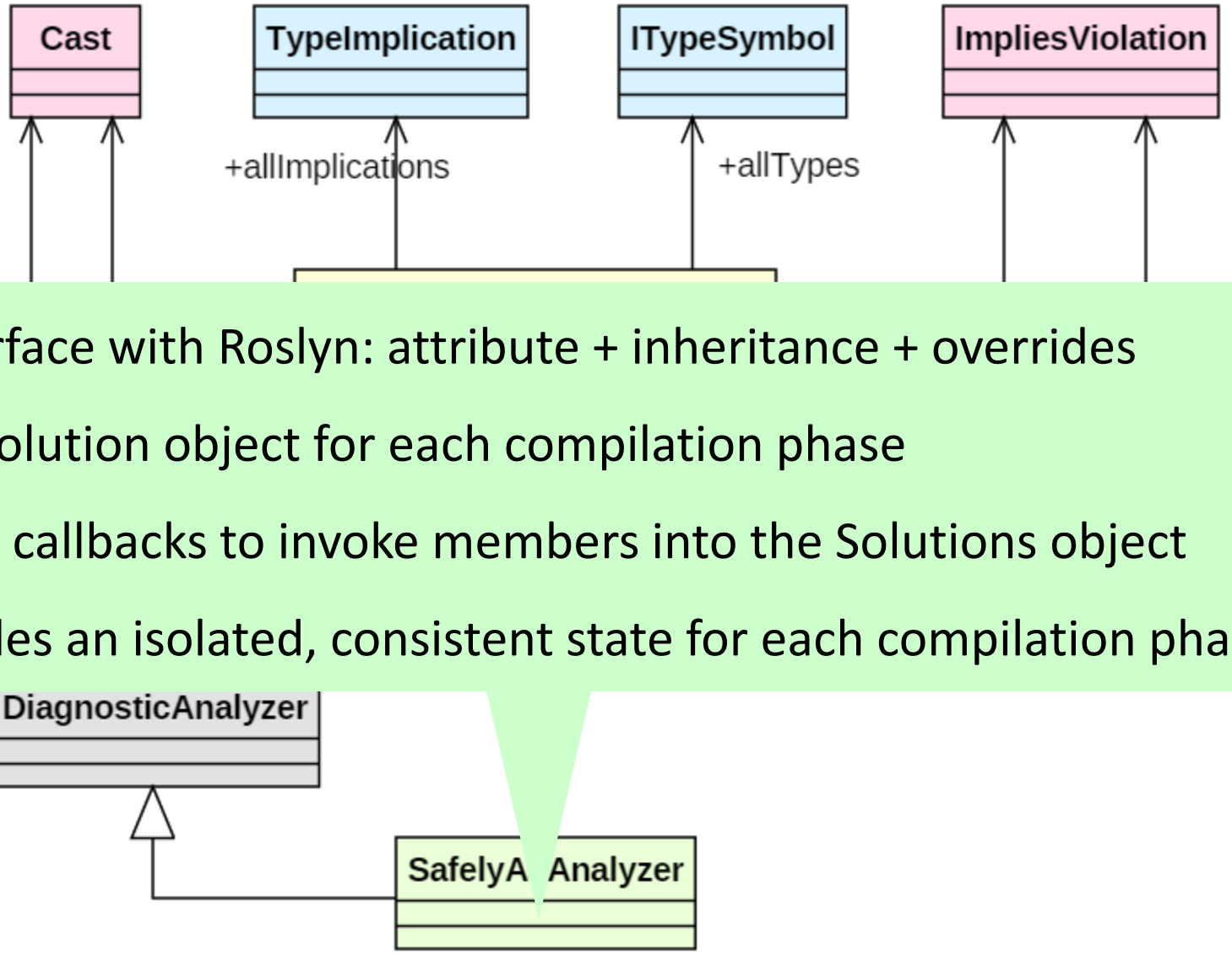
Analyzer callbacks

```
public override void Initialize(AnalysisContext context)
{
  context.RegisterCompilationStartAction(
    compilationContext =>
    {
      Solution wholeThing = new Solution();
      compilationContext.RegisterSyntaxNodeAction(
         wholeThing.AnalyzeImpliesDeclaration,
         SyntaxKind.Attribute);
      compilationContext.RegisterSyntaxNodeAction(
         wholeThing.AnalyzeSafelyCall,
         SyntaxKind.InvocationExpression);
      compilationContext.RegisterSymbolAction(
         wholeThing.RegisterNamedTypes,
         SymbolKind.NamedType);
      compilationContext.RegisterCompilationEndAction(
        wholeThing.AnalyzeConstraintsOnTypes);
    });
}
```

Prototype structure



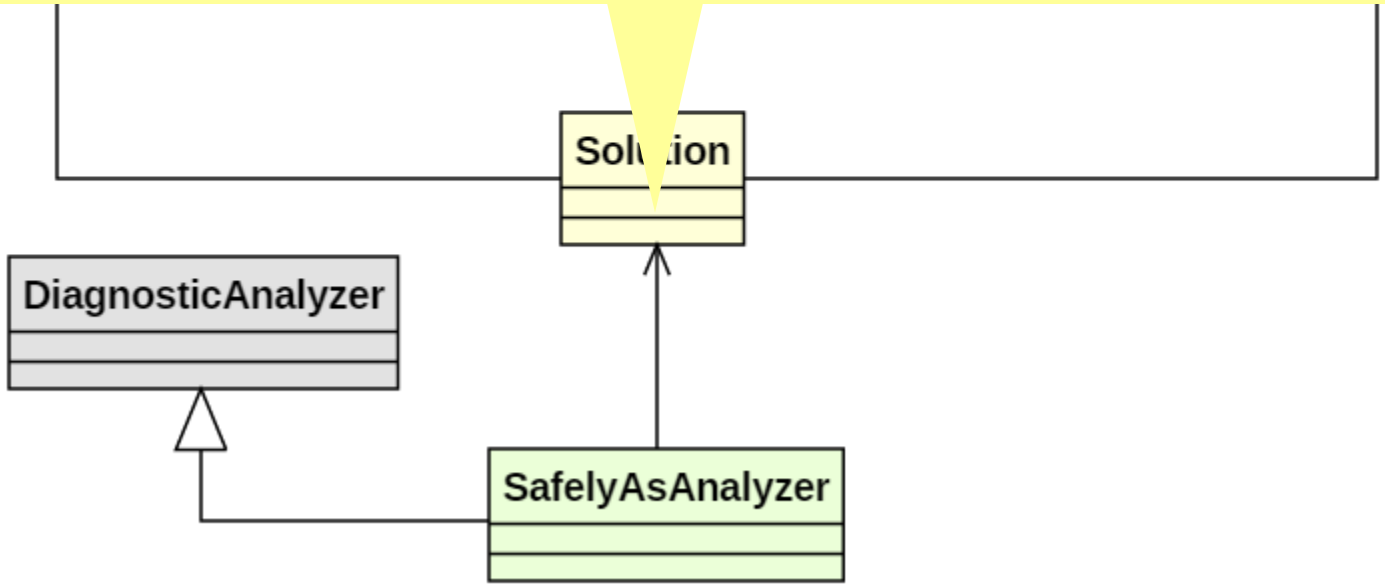
Responsibilities



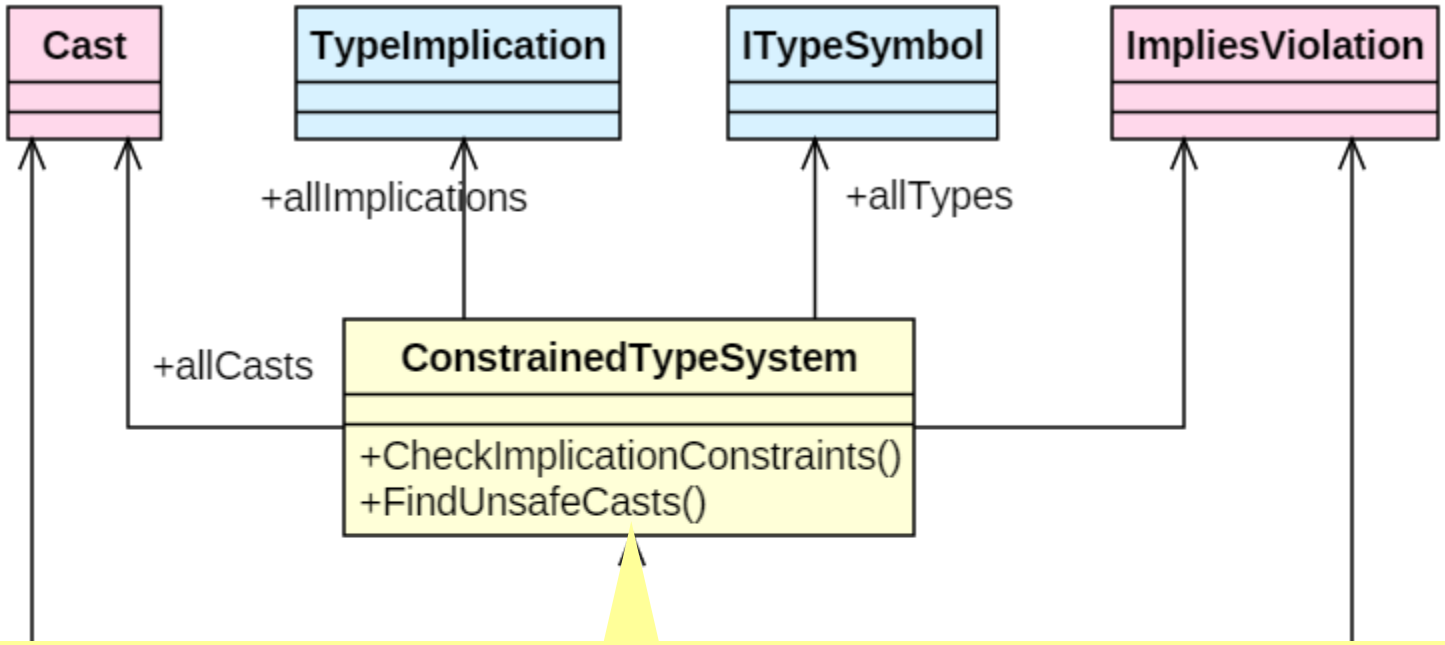
- Contact surface with Roslyn: attribute + inheritance + overrides
- Creates a Solution object for each compilation phase
- Sets up the callbacks to invoke members into the Solutions object
- That provides an isolated, consistent state for each compilation phase

Responsibilities

- Scans types, attributes, function calls
- Does all the simple local checking
- Stores casts, types and constraints in the CheckedTypeSystems
- Merges and reports local and global diagnostics
- Currently a bit fat



Responsibilities



- Keeps track of all types, constraint nd safe casts
- Does all the global type checking
- Returns diagnostics object (does not emit)
- Only contact with Roslyn is through ITypeSymbol

… and it works ☺

```
[module: Constraint.Implies(typeof(SafelyAsSample.IShape),
                            typeof(SafelyAsSample.IGeometry))]
[module: Constraint.Implies(typeof(SafelyAsSample.IShape),
                            typeof(SafelyAsSample.IDrawing))]
[module: Constraint.Implies(typeof(SafelyAsSample.Circle),
                            typeof(SafelyAsSample.Circle))]

public partial class Square : IGeometry { }

static void Main(string[] args)
{
  IShape s1 = new Circle(3);
  IDrawing d1 = To<IDrawing>.Safely(s1);

  Circle d2 = To<Circle>.Safely(s1);
  IDrawing d3 = To<IDrawing>.Safely(d2);

  INotImplemented n = To<INotImplemented>.Safely(s1);
}
```

Conclusions

- Code aware libraries open new possibilities
- Create better, safer libraries
- Do some “language hacking” without changing the compiler
  - And therefore making your language extensions useful for everyone

Resources

Dustin Campbell talk at Microsoft Build:
Analyzers and the Rise of Code-Aware Libraries
https://www.youtube.com/watch?v=Ip6wrpYFHhE

Alex Turner, *Use Roslyn to Write a Live Code Analyzer for Your API*
https://msdn.microsoft.com/en-us/magazine/dn879356

Code on github

https://github.com/carlopescio/safe_cast

Get in touch

carlo.pescio@gmail.com

@CarloPescio

http://eptacom.net

Domande?

Materiale su

http://www.communitydays.it/