LINQ – Beyond Queries

LINQ for Better Business Logic

pluralsight
see what you can learn

Repurposing LINQ Features

- **Extension methods**
  - For better APIs
- **Expression trees**
  - For static reflection
- **Funcs and Actions**
  - For functional, declarative programming
- **Demos**
  - Functional validation
  - Increasingly complex validations
  - Building a LINQ powered rules engine

Example Scenario

- **Scheduling tasks for periodic execution**

```
public class ScheduledTask {
    public ScheduledTask(ITask task,
                         TimeSpan interval,
                         TimeSpan expiration) {
        Task = task;
        Interval = interval;
        Expiration = expiration;
    }

    public ITask Task { get; protected set; }
    public TimeSpan Interval { get; protected set; }
    public TimeSpan Expiration { get; protected set; }
    ...
}
```

```
var task = new ScheduledTask(
                new AccountSynchronizationTask(),
                new TimeSpan(0, 0, 2, 0),
                new TimeSpan(2, 0, 0, 0));
```

?

Goals

- **Readability**
  - Easier to maintain
- **Essence over ceremony**
  - Remove language clutter

```
var task = new ScheduledTask(
                new AccountSynchronizationTask(),
                new TimeSpan(0, 0, 2, 0),
                new TimeSpan(2, 0, 0, 0));
```

Named parameters

- **Only a small step forward**
  - Particularly useful when combined with optional parameters
  - Gives reader a clue when using constants

```
var task = new ScheduledTask(
                Tasks.AccountSynchronization,
                runEvery: new TimeSpan(0, 0, 2, 0),
                expiresIn: new TimeSpan(2, 0, 0, 0));
```

Extension Methods

- **Extend types!**
  - Even sealed types, generic types, and interfaces

```
public static class StringExtensions
{
    public static int ToInt32(this string value)
    {
        return Int32.Parse(value);
    }
}
```

```
int value = "32".ToInt32();
```

Fluent APIs

- **A readable API**
  - Often uses method chaining

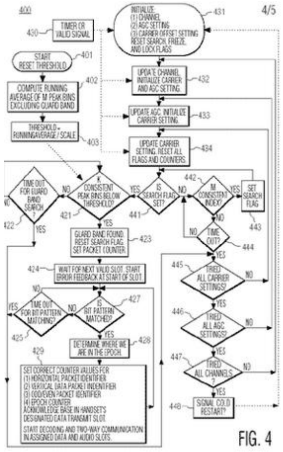
```
var then = 2.Minutes().Ago();
```

```
public static TimeSpan Minutes(this int value)
{
    return new TimeSpan(0, 0, value, 0, 0);
}

public static DateTime Ago(this TimeSpan value)
{
    return DateTime.Now - value;
}
```

Validation Example

- **Dealing with requirements in the form of complex flowcharts**
  - Model them with procedural if/else code?


400
4/5
430
TIMER OR VALID SIGNAL
INITIALIZE (1) CHANNEL (2) AGC SETTING (3) CARRIER OFFSET SETTING RESET SEARCH, FREEZE, AND LOCK FLAGS
431
START RESET THRESHOLD
401
402
UPDATE CHANNEL INITIALIZE CARRIER AND AGC SETTING
432
COMPUTE RUNNING AVERAGE OF M PEAK BINS EXCLUDING GUARD BAND
UPDATE AGC. INITIALIZE CARRIER SETTING
433
THRESHOLD = RUNNINGAVERAGE / SCALE
403
UPDATE CARRIER SETTING. RESET ALL FLAGS AND COUNTERS
434
442
TIME OUT FOR GUARD BAND SEARCH ?
NO
K CONSISTENT PEAK BINS BELOW THRESHOLD?
YES
IS SEARCH FLAG SET?
NO
M CONSISTENT INDEX?
YES
SET SEARCH FLAG
422
421
441
443
YES
YES
NO
GUARD BAND FOUND. RESET SEARCH FLAG. SET PACKET COUNTER
423
NO
TIME OUT?
444
YES
424
WAIT FOR NEXT VALID SLOT. START ERROR FEEDBACK AT START OF SLOT
445
TRIED ALL CARRIER SETTINGS?
NO
NO
427
YES
TIME OUT FOR BIT PATTERN MATCHING?
NO
IS BIT PATTERN MATCHED?
YES
446
TRIED ALL AGC SETTINGS?
NO
425
YES
DETERMINE WHERE WE ARE IN THE EPOCH
428
429
447
TRIED ALL CHANNELS ?
NO
SET CORRECT COUNTER VALUES FOR (1) HORIZONTAL PACKET IDENTIFIER (2) VERTICAL DATA PACKET INDENTIFIER (3) ODD/EVEN PACKET IDENTIFIER (4) EPOCH COUNTER ACKNOWLEDGE BASE IN HANDSETS DESIGNATED DATA TRANSMIT SLOT
START DECODING AND TWO-WAY COMMUNICATION IN ASSIGNED DATA AND AUDIO SLOTS
YES
448
SIGNAL COLD RESTART?
FIG. 4

```
public bool IsValid(Movie movie)
{
    if(string.IsNullOrEmpty(movie.Title))
    {
        return false;
    }

    if(movie.Length < 60 || movie.Length > 400)
    {
        return false;
    }

    if(movie.ReleaseDate.Value.Year < 1903)
    {
        return false;
    }

    return true;
}
```

Functional Validation

- **Using lambda expressions for a declarative approach**
  - Keep the code in a data structure for passive evaluation

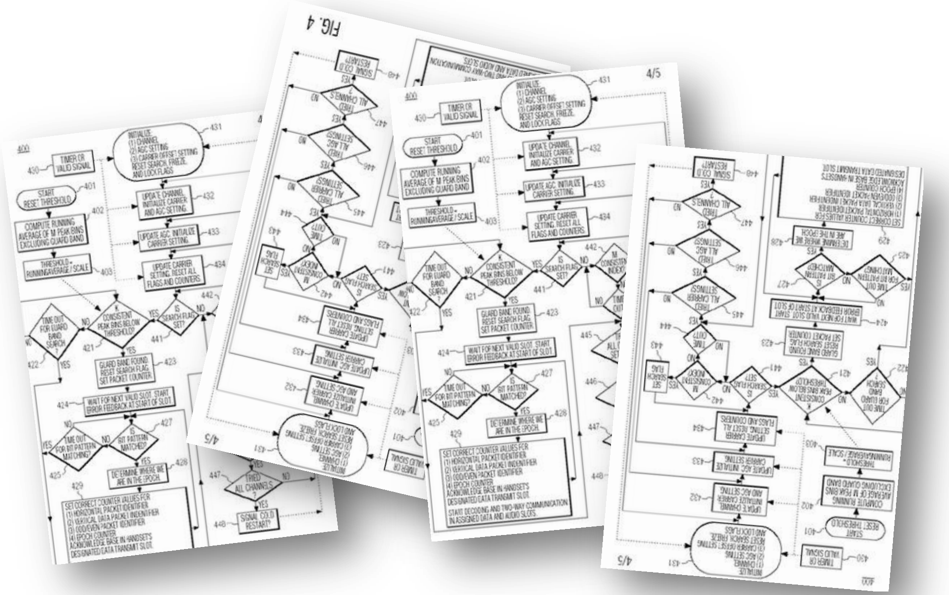
```
public bool IsValid(Movie movie)
{
    Func<Movie, bool>[] rules =
        {
            m => string.IsNullOrEmpty(m.Title),
            m => m.Length < 60 || m.Length > 400,
            m => m.ReleaseDate.Value.Year < 1903
        };


    return rules.All(rule => rule(movie) == false);

}
```

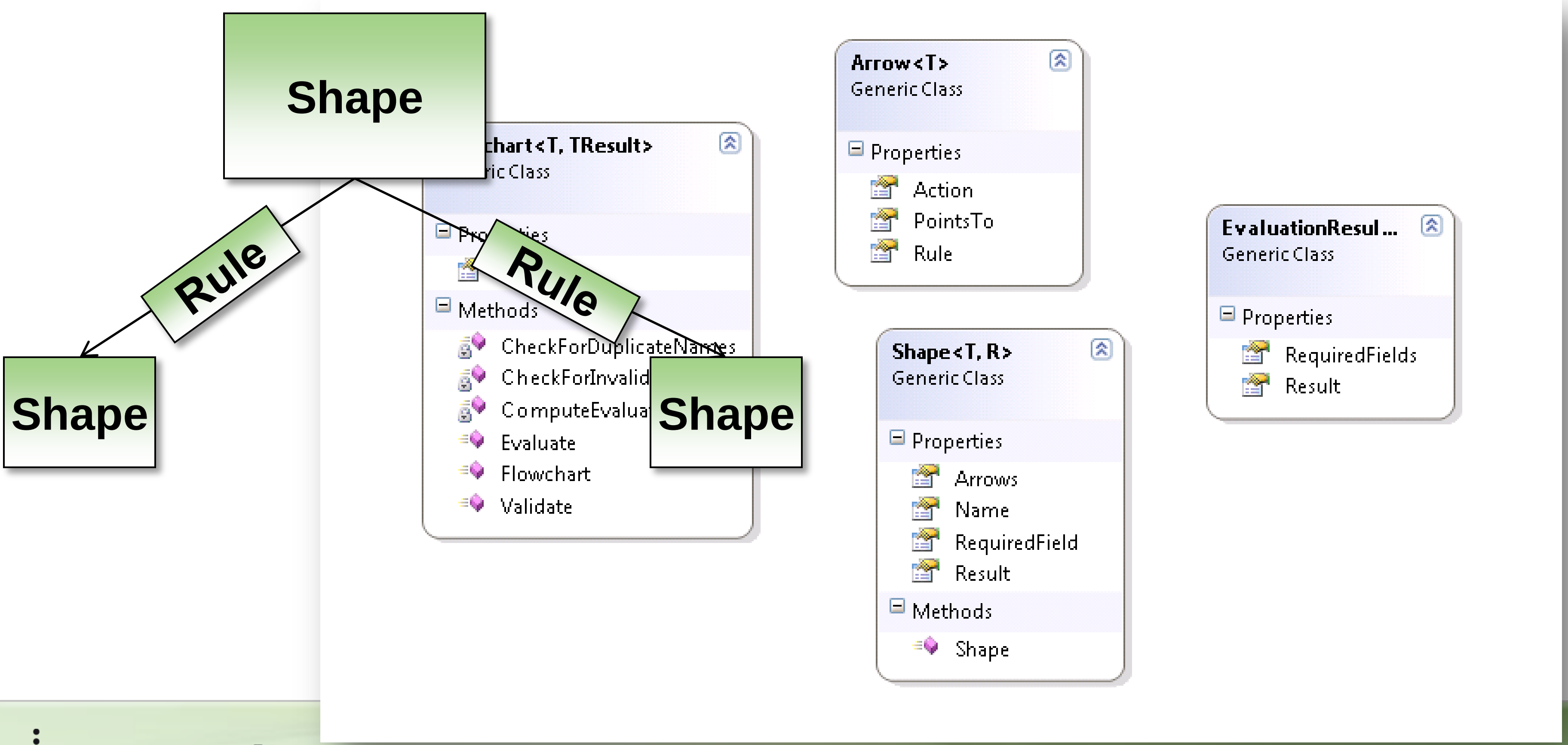
A More Complex Scenario

- **Instead of validation, we'll perform a rules evaluation**
  - Required to compute more than just a binary result
  - Should scale up to manage hundreds of rules
  - Should be able to complete evaluation even with incomplete data
  - Required to provide information about what properties are inspected



Domain Model

- **Models the business flowcharts**
  - Shape, arrows, rules, results



chart<T, TResult>
ric Class
Pro ies
Methods
CheckForDuplicateNames
CheckForInvalid
ComputeEvalua
Evaluate
Flowchart
Validate

Arrow<T>
Generic Class
Properties
Action
PointsTo
Rule

Shape<T, R>
Generic Class
Properties
Arrows
Name
RequiredField
Result
Methods
Shape

EvaluationResul...
Generic Class
Properties
RequiredFields
Result

Creating the Flowchart

- **Tedious!**

```
var chart = new MovieFlowchart();
chart.Shapes.Add(
    new Shape<Movie, MovieResult>()
    {
            Name = "CheckTitle",
            Arrows =
            {
                new Arrow<Movie>
                {
                    PointsTo = "CheckLength",
                    Rule = m => !String.IsNullOrEmpty(m.Title)
                }
            },
            RequiredField = new PropertySpecifier<Movie>(m=>m.Title)
    }
    // ... and so on
);
```

Building the Fluent API / Internal DSL

- **Heavy use of extension methods**

```
public static Flowchart<T, R> AddShape<T, R>(
                    this Flowchart<T, R> chart, string shapeName)
```

```
chart.AddShape("CheckTitle")
          .Requires(m => m.Title)
          .WithArrowPointingTo("CheckLength").AndRule(TitleNotNullOrEmpty)
      .AddShape("CheckLength")
          .Requires(m => m.Length)
          .WithArrowPointingTo("BadMovie").AndRule(LengthIsTooLong)
          .WithArrowPointingTo("GoodMovie").AndRule(LengthIsJustRight)
          .WithArrowPointingTo("CheckReleaseDate").AndRule(LengthExists)
      .AddShape("CheckReleaseDate")
          .Requires(m => m.ReleaseDate)
          .WithArrowPointingTo("BadMovie").AndRule(TooOld)
          .WithArrowPointingTo("GoodMovie").AndRule(HasReleaseDate)
      .AddShape("BadMovie").YieldsResult(MovieResult.BadMovie)
      .AddShape("GoodMovie").YieldsResult(MovieResult.GoodMovie);
```

Taking Advantage of Expression<T>

- **Expression<T> can yield rich meta-data about a piece of code**
  - “Static” reflection

```
public PropertySpecifier(Expression<Func<T, object>> expression)
{
    if(expression.Body is MemberExpression)
    {
        var me = expression.Body as MemberExpression;
        _propertyName = me.Member.Name;
    }
    else if(expression.Body is UnaryExpression)
    {
        var ue = expression.Body as UnaryExpression;
        var me = ue.Operand as MemberExpression;
        _propertyName = me.Member.Name;
    }
}
```

Summary

- **LINQ features – more than just data access**
  - Extension methods provide a shim for alternate APIs
  - Use lambdas and Func<> for expressive, functional programming
  - Leverage Expression<T> for metadata about code