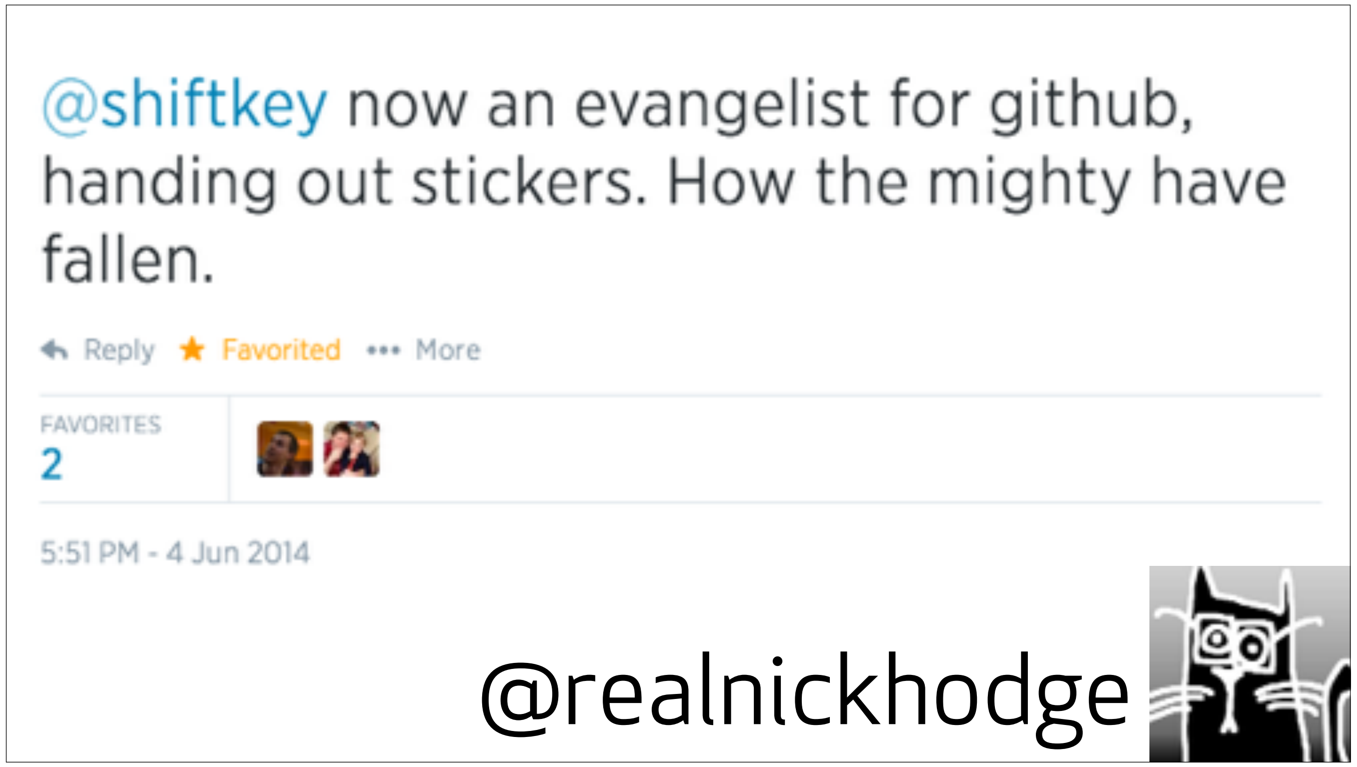
Let Me Tell You About Our Lord And Saviour FRP



@shiftkey now an evangelist for github, handing out stickers. How the mighty have fallen.

Reply ★ Favorited ••• More

FAVORITES
2

5:51 PM - 4 Jun 2014

@realnickhodge

MVVM

Functional Reactive Programming

ReactiveUI

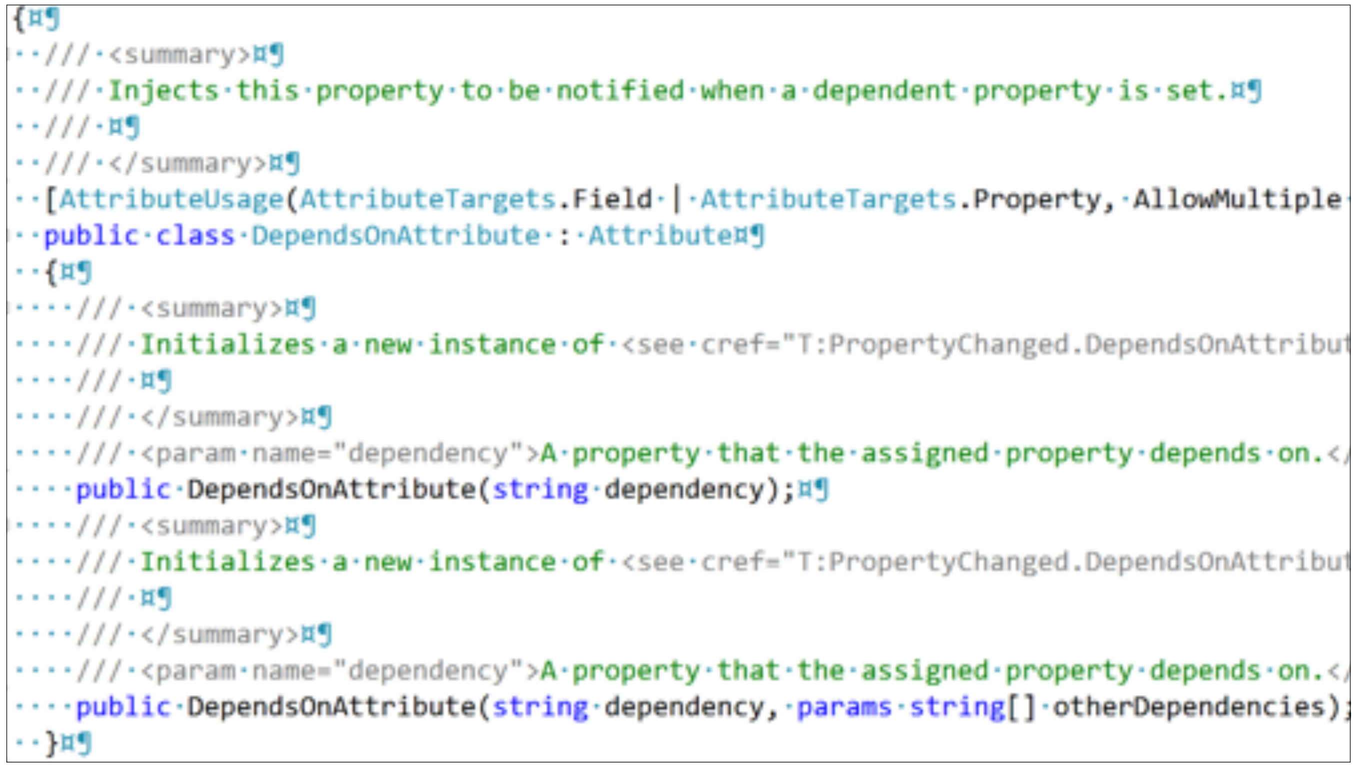
MVVM

change notifications and interactions

```
/* [ImplementPropertyChanged] */

[DependsOn("Card")]
public bool HasMembers
{
    get
    {
        if (Card == null) return false;
        return Card.IdMembers.Any();
    }
}
```

https://github.com/brendankowitz/AgilityWall/blob/1b9a737c9378ceb7dd6b1126c050dfa6fd1ac370/src/AgilityWall.Core/Features/TaskBoard/CardSummaryViewModel.cs#L99-L107



```
{¤¶
··///·<summary>¤¶
··///·Injects·this·property·to·be·notified·when·a·dependent·property·is·set.¤¶
··///·¤¶
··///·</summary>¤¶
··[AttributeUsage(AttributeTargets.Field·|·AttributeTargets.Property,·AllowMultiple·
··public·class·DependsOnAttribute·:·Attribute¤¶
··{¤¶
····///·<summary>¤¶
····///·Initializes·a·new·instance·of·<see·cref="T:PropertyChanged.DependsOnAttribut
····///·¤¶
····///·</summary>¤¶
····///·<param·name="dependency">A·property·that·the·assigned·property·depends·on.</
····public·DependsOnAttribute(string·dependency);¤¶
····///·<summary>¤¶
····///·Initializes·a·new·instance·of·<see·cref="T:PropertyChanged.DependsOnAttribut
····///·¤¶
····///·</summary>¤¶
····///·<param·name="dependency">A·property·that·the·assigned·property·depends·on.</
····public·DependsOnAttribute(string·dependency,·params·string[]·otherDependencies);
··}¤¶
```

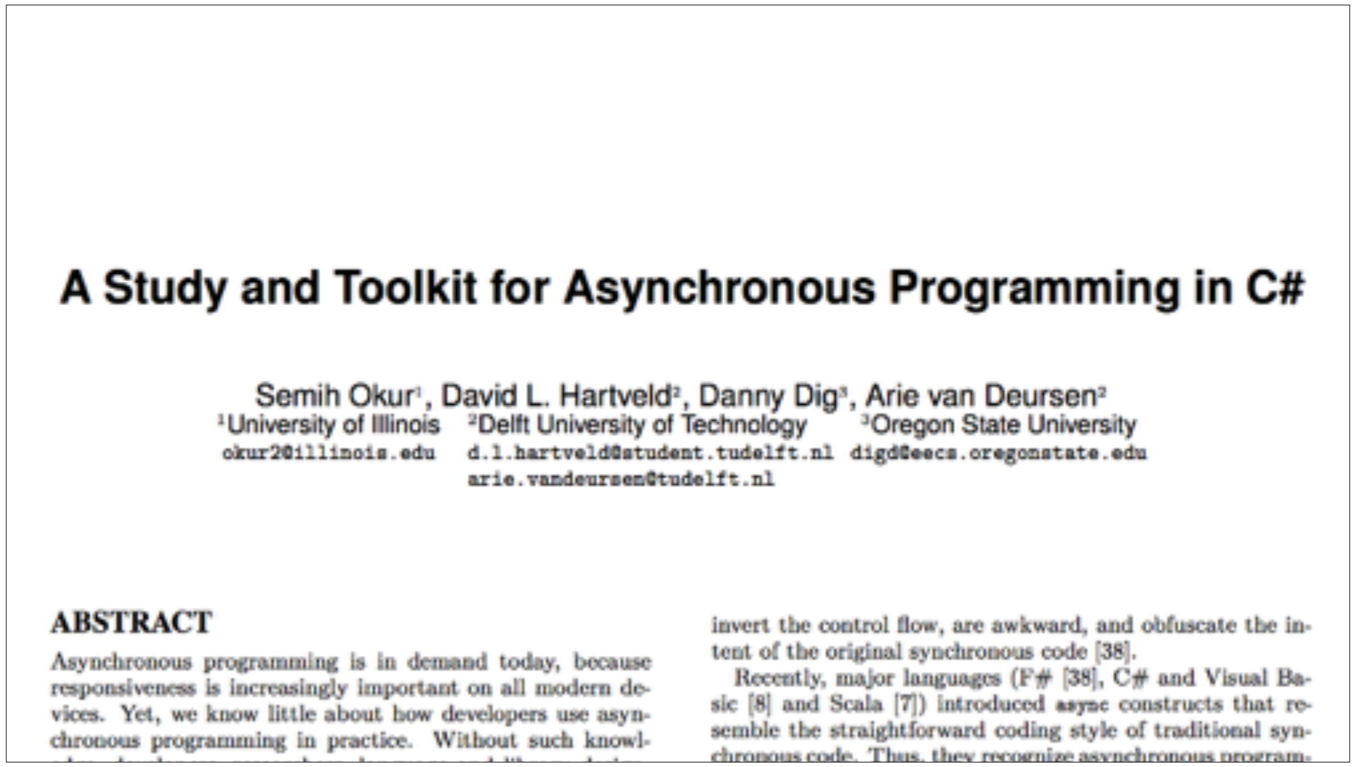
first-class support for asynchrony

```
public DirectoryPickerViewModel(
    ISelectDirectoryService selectDirectoryService,
    IProcessService processService)
{
    Argument.IsNotNull(() => selectDirectoryService);
    Argument.IsNotNull(() => processService);

    _selectDirectoryService = selectDirectoryService;
    _processService = processService;

    OpenDirectory = new Command(
       OnOpenDirectoryExecute,
       OnOpenDirectoryCanExecute);
    SelectDirectory = new Command(OnSelectDirectoryExecute);
}
```

https://github.com/Orcomp/Orchestra/blob/develop/src/Orchestra.Core/Orchestra.Core/ViewModels/DirectoryPickerViewModel.cs#L23-L33 isn’t async/await a perfectly fine abstraction for doing this?



A Study and Toolkit for Asynchronous Programming in C#

Semih Okur[1], David L. Hartveld[2], Danny Dig[3], Arie van Deursen[2]
[1]University of Illinois okur2@illinois.edu
[2]Delft University of Technology d.l.hartveld@student.tudelft.nl arie.vandeursen@tudelft.nl
[3]Oregon State University digd@eecs.oregonstate.edu

ABSTRACT

Asynchronous programming is in demand today, because responsiveness is increasingly important on all modern devices. Yet, we know little about how developers use asynchronous programming in practice. Without such knowl-

invert the control flow, are awkward, and obfuscate the intent of the original synchronous code [38].

Recently, major languages (F# [38], C# and Visual Basic [8] and Scala [7]) introduced async constructs that resemble the straightforward coding style of traditional synchronous code. Thus, they recognize asynchronous program-

http://swerl.tudelft.nl/twiki/pub/Main/TechnicalReports/TUD-SERG-2013-016.pdf

"We analyzed 1378 open source Windows Phone apps, comprising 12M SLOC produced by 3376 developers."

14% of async/await methods were unnecessary

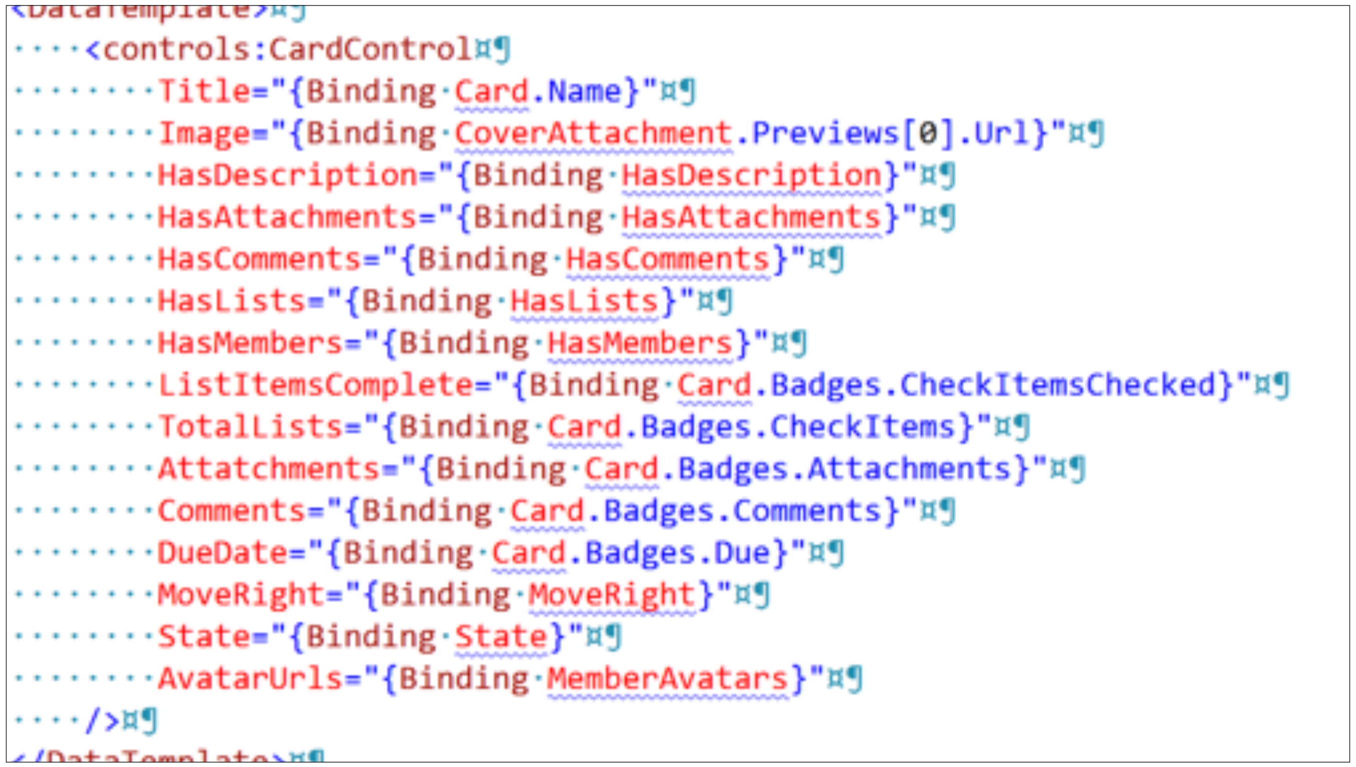
(just return a Task!)

1 in 5 apps miss opportunities in async methods to be more async

99% of async/await methods did not specify .ConfigureAwait(false) when it was needed

bindings are expensive

complex bindings
are just the worst



```xml
<DataTemplate>
    <controls:CardControl
        Title="{Binding Card.Name}"
        Image="{Binding CoverAttachment.Previews[0].Url}"
        HasDescription="{Binding HasDescription}"
        HasAttachments="{Binding HasAttachments}"
        HasComments="{Binding HasComments}"
        HasLists="{Binding HasLists}"
        HasMembers="{Binding HasMembers}"
        ListItemsComplete="{Binding Card.Badges.CheckItemsChecked}"
        TotalLists="{Binding Card.Badges.CheckItems}"
        Attatchments="{Binding Card.Badges.Attachments}"
        Comments="{Binding Card.Badges.Comments}"
        DueDate="{Binding Card.Badges.Due}"
        MoveRight="{Binding MoveRight}"
        State="{Binding State}"
        AvatarUrls="{Binding MemberAvatars}"
    />
</DataTemplate>
```

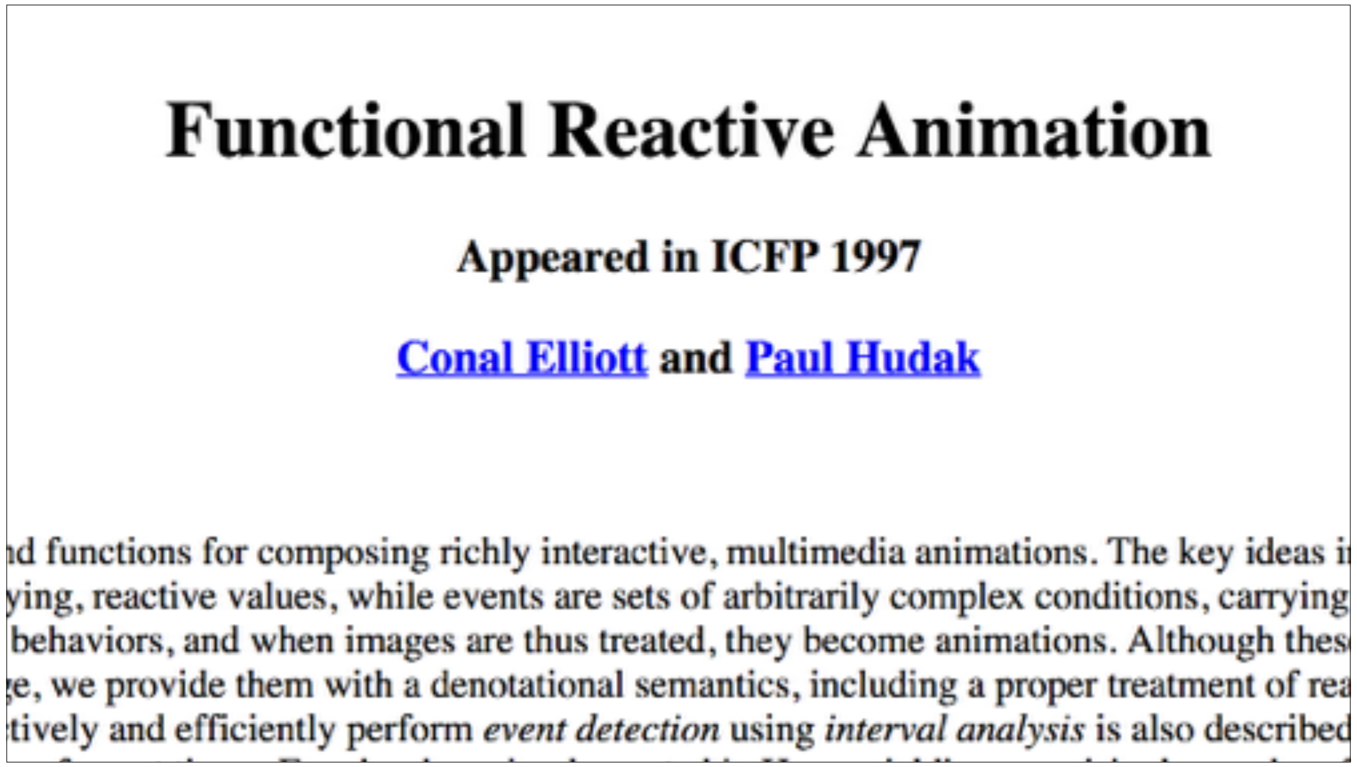
https://github.com/brendankowitz/AgilityWall/blob/dddbaa298784524a9da8d53b85670e0cea3c7209/src/AgilityWall.WinPhone/Features/TaskBoard/BoardView.xaml#L44-L62

```
System.Windows.Data Error: 40 : BindingExpression path
error: 'NonExistingProperty' property not found on
'object' ''Grid' (Name='pnlMain')'.
BindingExpression:Path=NonExistingProperty;
DataItem='Grid' (Name='pnlMain'); target element is
'TextBlock' (Name=''); target property is 'Text' (type
'String')
```

deep breath

Functional Reactive Programming

Winamp is released April 21, 1997.


Functional Reactive Animation

Appeared in ICFP 1997

Conal Elliott and Paul Hudak

ıd functions for composing richly interactive, multimedia animations. The key ideas i
ying, reactive values, while events are sets of arbitrarily complex conditions, carrying
behaviors, and when images are thus treated, they become animations. Although thes
ge, we provide them with a denotational semantics, including a proper treatment of rea
tively and efficiently perform *event detection* using *interval analysis* is also described

http://conal.net/papers/icfp97/
"In 2007, this paper was awarded as the most influential paper of ICFP '97."

“Values, called behaviours, that vary over continuous time are the chief values of interest” signals

“Values, called ~~behaviours~~, that vary over continuous time are the chief values of interest”

"Events may refer to
happenings in the real world
(e.g. mouse button presses),
but also to predicated based on
animation parameters (e.g.
proximity or collisions)"

“We would like to have a general way of “lifting” functions defined on static values to analogous functions defined on behaviours.”

```
wiggle = sin (pi * time)
wiggle = sin (pi * time)

wiggleRange lo hi =
lo + (hi-lo) * (wiggle+1)/2
```

```
paintBall = withColor red
(bigger (wiggleRange 0.5 1) circle)
```

```
paintBall = withColor red
(bigger (wiggleRange 0.5 1) circle)

rotateBall =
  move (vectorPolar 2.0 time)
       (bigger 0.1 paintBall)
```

```
followMouse shape t0
    = move (mouse t0) shape
```

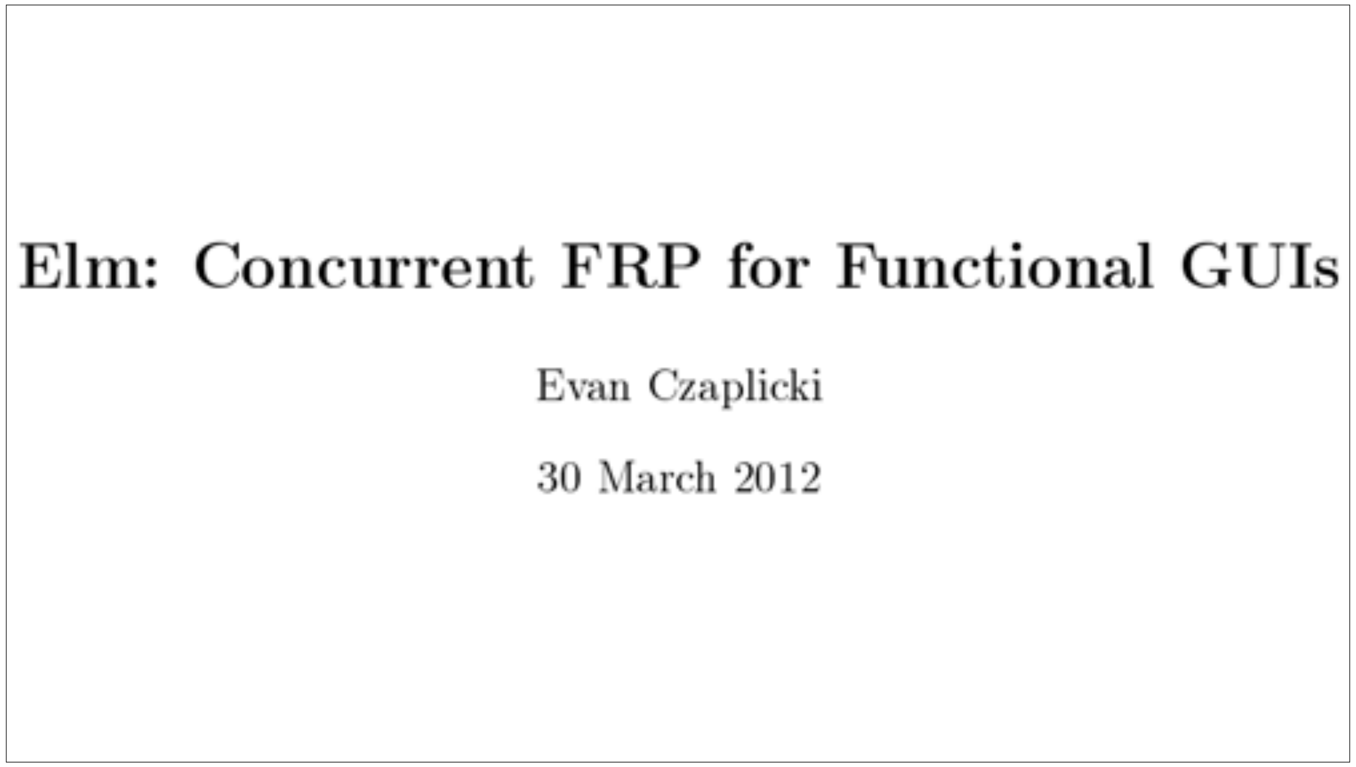
Demand-Driven Sampling

aka “pull”

Data-Driven Sampling

aka “push”

“Continuous Signals”
“Discrete Signals”


Elm: Concurrent FRP for Functional GUIs

Evan Czaplicki

30 March 2012

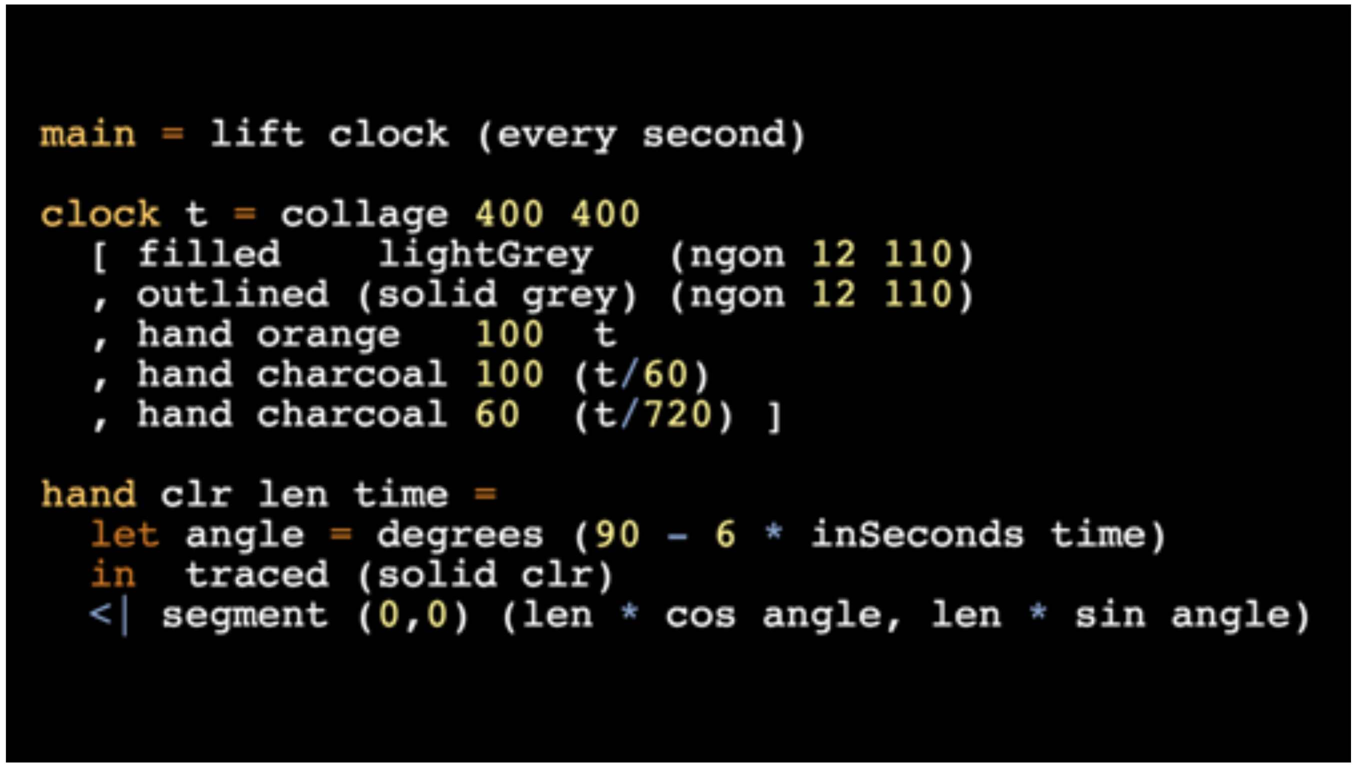
https://www.seas.harvard.edu/sites/default/files/files/archived/Czaplicki.pdf
http://engineering.prezi.com/blog/2013/05/21/elm-at-prezi/

Classical FRP

Real Time FRP

Event-Driven FRP

Arrowized FRP



```
main = lift clock (every second)

clock t = collage 400 400
  [ filled    lightGrey   (ngon 12 110)
  , outlined (solid grey) (ngon 12 110)
  , hand orange   100  t
  , hand charcoal 100 (t/60)
  , hand charcoal 60  (t/720) ]

hand clr len time =
  let angle = degrees (90 - 6 * inSeconds time)
  in  traced (solid clr)
  <| segment (0,0) (len * cos angle, len * sin angle)
```

http://elm-lang.org/edit/examples/Intermediate/Clock.elm

But What About .NET?

Reactive Extensions

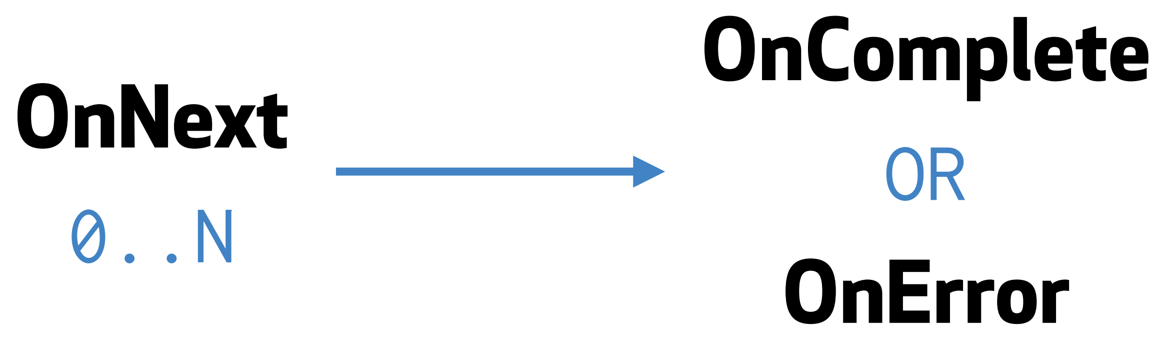
Really Really

Quick Explanation

IObservable<T>

IObserver<T>

IObservable<T>



$$\begin{array}{c} \text{Task} \\ \text{Task<T>} \end{array} \longleftrightarrow \begin{array}{c} \text{IObservable<Unit>} \\ \text{IObservable<T>} \end{array}$$

```
someObservable
    .Subscribe(
        result => /* do something */)
```

```
someObservable
   .Subscribe(
      result => /* do something */,
      () => /* no more results */,
      ex => /* error occurred */)
```

```
someObservable
    .Catch(Observable.Empty<bool>())
    .Subscribe(
      result => /* do something */,
      () => /* no more results */)
```

Schedulers

```
someObservable
    .ObserveOn(DispatcherScheduler.Current)
    .Subscribe(result => /* update UI */)
```

```
Observable.Start(
  () => DoLongRunningThing(),
  TaskPoolScheduler.Current)
    .ObserveOn(DispatcherScheduler.Current)
    .Subscribe(result => /* update UI */)
```

LINQ

```
someObservable
    .Skip(1)
    .Where(x => x > 0)
    .Subscribe(num => /* positive numbers */)
```

```
Observable.Combine(
  someObservable,
  otherObservable,
  (some, other) => some > 0 && other))
    .Subscribe(x => /* true or false */)
```

```
Observable.Combine(
  someObservable,
  otherObservable,
  (some, other) => some > 0 && other))
    .DistinctUntilChanged()
    .Subscribe(x => /* true or false */)
```

cold observables:
inactive when no
observers subscribed

hot observables:
always active, even when no observers subscribed

ReactiveUI

So what does an FRP codebase actually look like?

Immutable core

Signals for external inputs

Signals for interactions

and that’s basically it

"One huge benefit to this, especially important in production code, is **greatly enhanced readability.** When changes, events, and values are modeled as interchangeable streams, code locality is much better—you can keep all your logic for doing a particular task in one place, instead of spread across a bunch of spaghetti-like event handlers and state variables."

@jspahrsummers

http://www.quora.com/What-is-it-like-to-use-reactive-programming-in-production/answer/Justin-Spahr-Summers?srid=3zh0&share=1

$$\begin{tikzcd}\text{Observables} \arrow[d, leftrightarrow] \\ \text{Properties}\end{tikzcd}$$

ReactiveObject

INotifyPropertyChanged

```
this.WhenAny(
  x => x.SelectedAccount,
  x => x.SelectedAccount.IsStale,
  (account, isStale) => account.Value)
     .WhereNotNull()
     .Where(account => account.IsStale)
     .Subscribe(
         x => x.LoadRepositories.Execute(null));
```

```
readonly ObservableAsPropertyHelper<int>
  progress;

public RepositoryCloneViewModel() {
 progress = this.WhenAny(
  x => x.Model.CloningProgressValue, x => x.Value)
    .ToProperty(this, x => x.Progress);
}

public int Progress {
    get { return progress.Value; }
}
```

ReactiveCommand

ICommand

```
ReactiveCommand.Create();

ReactiveCommand.Create(
  this.WhenAny(x => x.SelectedUser, x != null));

ReactiveCommand.CreateAsyncObservable(
  o => RefreshSelectedUser());

ReactiveCommand.CreateAsyncTask(
  o => RefreshSelectedUser());
```

```csharp
var command =  ReactiveCommand.Create();

command.Subscribe(_ => /* callback */);
```

```
var viewModel = new MyViewModel();

await viewModel.Refresh.ExecuteAsync();

// assert something
```

```
var command =  ReactiveCommand.Create();

command.ThrownExceptions.Subscribe(
  _ => /* log errors */);
```

```
var refreshCommand =
  ReactiveCommand.CreateAsyncObservable(/* */);

var isRefreshing =
  refreshCommand
    .IsExecuting
    .ToProperty(this, x => x.IsRefreshing);
```

```
readonly ObservableAsPropertyHelper<bool> isRefreshing;

public bool IsRefreshing
{
  get { return isRefreshing.Value; }
}
```

View Bindings

XAML

Monotouch

Monoandroid

Monomac

```
public class ShellView : UserControl {

 public ShellView() {
    /* TODO */
  }

}
```

```
public class ShellView : UserControl,
                         IViewFor<IShellViewModel> {

  public ShellView() {
    /* TODO */
  }

  public IShellViewModel ViewModel
  { /* dependency property */ }
}
```

```
public ShellView() {
   this.Bind(
     ViewModel,
     vm => vm.Name,
     v => v.name.Text);
 }
```

```
public ShellView() {
   this.OneWayBind(
     ViewModel,
     vm => vm.IsRefreshing,
     v => v.refresh.Visibility);
 }
```

```
public ShellView() {
   this.BindCommand(
     ViewModel,
     vm => vm.RefreshCommand,
     v => v.refresh);
 }
```

type-safe bindings

compile-time validation

advanced selectors

```
this.WhenAny(
  x => x.ViewModel.SelectedRepositoryItem,
  x => x.ViewModel.IsFiltered,
  (x, y) => new {
    SelectedRepositoryItem = x.Value,
    IsFiltered = y.Value
  })
  .Where(x => x.IsFiltered)
  .Subscribe(x => /* focus on item */);
```

THE SUCK?

"Of course, there are tradeoffs. The biggest downside in practice is that **it's harder to debug reactive code**, since you're usually dealing with multiple levels of indirection in the call stack, instead of the very straightforward backtraces generated by imperative code."

@jspahrsummers

http://galleryhip.com/shark-fin-out-of-water.html