# ReactiveUI

## It's Pretty Neat

**Brendan Forster**

@shiftkey

**ReactiveUI**

MVVM

Reactive Extensions

Functional Reactive Programming

ReactiveUI

MVVM

Reactive Extensions

**Functional Reactive Programming**

# Functional Reactive Animation

**Appeared in ICFP 1997**

**Conal Elliott and Paul Hudak**

nd functions for composing richly interactive, multimedia animations. The key ideas in
ying, reactive values, while events are sets of arbitrarily complex conditions, carrying
behaviors, and when images are thus treated, they become animations. Although these
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

Functional Reactive Programming (FRP) integrates time flow and compositional events into functional programming.

# Functional + Reactive?

Immutability
Composition
Declarative

# Elm: Concurrent FRP for Functional GUIs

Evan Czaplicki

30 March 2012

```
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)
```

ReactiveUI

MVVM

**Reactive Extensions**

Functional Reactive Programming

**IObservable<T>**

IObserver<T>

**IEnumerable<T>**

IEnumerator<T>

**IObservable<T>**

IObserver<T>

# Duality and the End of Reactive

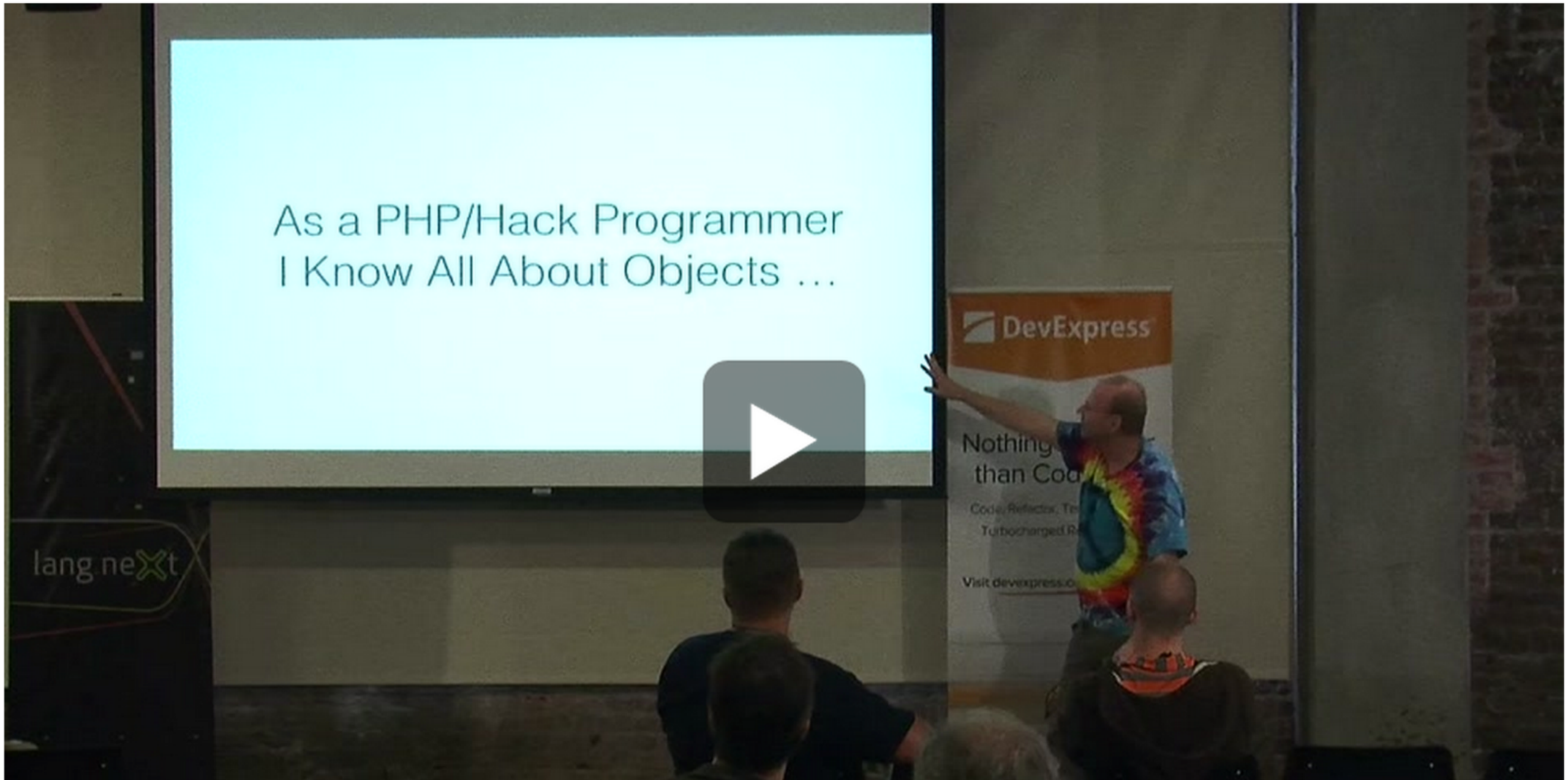
Date: May 30, 2014 from 9:00AM to 10:15AM | Day 2

Speakers: Erik Meijer

(17) | 49,342 Views | 28 Comments

Avg Rating: 4.5

79 pts. Tweet 238 Like 79



**Pull** **IEnumerable<T>**

IEnumerator<T>

**Push** **IObservable<T>**

IObserver<T>

[audience participation]

```
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 */)
```

cold observables:
inactive when no
observers subscribed

hot observables:

always active, even when no observers subscribed

# Schedulers

```
someObservable
      .ObserveOn(DispatcherScheduler.Instance)
      .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 */)
```

# Y U NO
# Task Parallel Library?

Task

Task<T>

IObservable<Unit>

IObservable<T>

```
public Task DoSomething()
{
    var obs =
DoSomething(TaskPoolScheduler.Instance);
    return obs.ToTask();
}
```

```
public IObservable<bool> DoSomething()
{
    var task = Task.Run(() => false); // lol
    return task.ToObservable();
}
```

```
public async bool AwaitAnObservable()
{
    var obs = Observable.Start(() => false); // lol
    var result = await obs;
    return result;
}
```

```
public async Task AwaitASeriesOfTasks()
{
   await firstTask();
   await secondTask();
   await thirdTask();
}
```

```
public async Task AwaitASeriesOfTasks()
{
   await Task.WhenAll(
      firstTask(),
      secondTask(),
      thirdTask()
   );
}
```

```
public IObservable<Unit> AwaitObservables()
{
    return Observable.Merge(
        doSomethingFirst(),
        doSomethingSecond(),
        doSomethingThird()
    ).Catch<Unit,Exception>(ex => {
        /* log error message */
        return Unit.Default;
    });
}
```

**ReactiveUI**

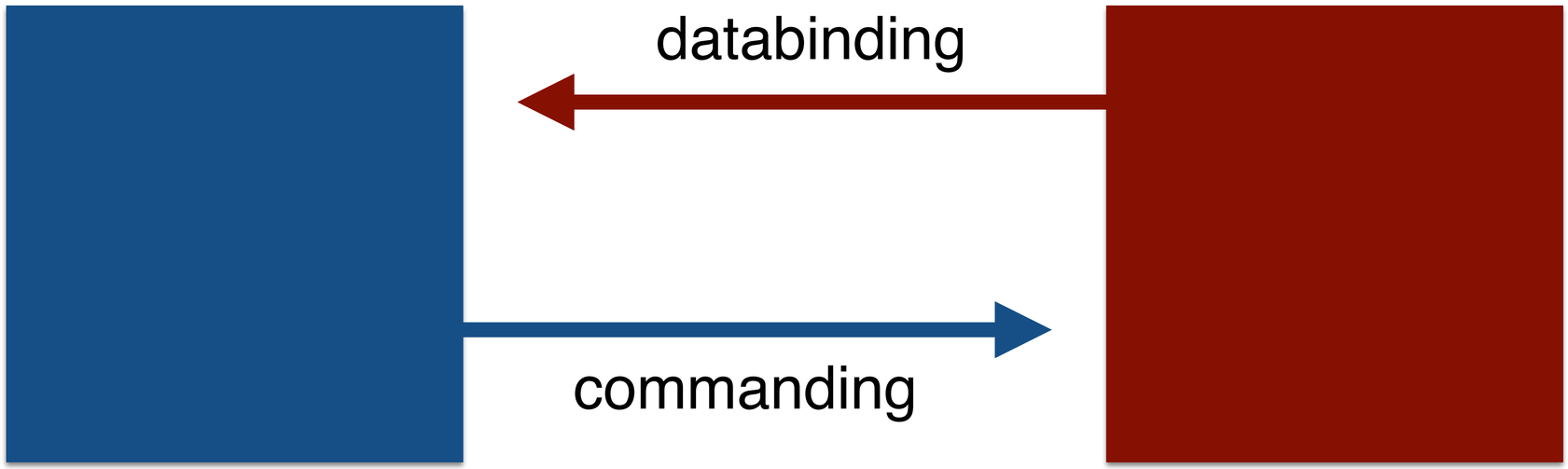
**MVVM**

Reactive Extensions

Functional Reactive Programming

“The essence of a **Presentation Model** is of a fully self-contained class that represents all the **data** and **behavior** of the UI window, but without any of the controls used to render that UI on the screen”

View ViewModel



MvvmLight

Caliburn.Micro

MvvmCross

Catel

Cinch

Okra

...

[asynchrony]

# ReactiveObject

# INotifyPropertyChanged

Observables

Properties

# ReactiveObject

this.WhenAny

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

# ObservableAsPropertyHelper

```
readonly ObservableAsPropertyHelper<int>
  progress;

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

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

# declarative ViewModels

decompose complexity

```
foreach(var item in allItems)
{
    if (item.LastName.StartsWith(“S”))
    {
        results.Add(item);
    }
}
```

```
var results = allItems
   .Where(x => x.LastName.StartsWith(“S”));
```

# 10 things I hate about ~~you~~ I Command

# ICommand

```
bool CanExecute(object o)
void Execute(object o)
```

# ICommand

```
bool CanExecute(object o)
void Execute(object o)
```

# ReactiveCommand

ICommand

```
ReactiveCommand.Create();

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

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

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

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

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

```
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 */);
```

# what is challenging about RxUI?

dude, where's my event?

get off my ~~lawn~~

main thread

async everywhere

warps the mind

# ReactiveUI

reactiveui.net

?