The First Substantial Line of Business Application in F#

Alex Peake - TFC
Adam Granicz - IntelliFactory

Commercial Users of Functional Programming
CUFP 2009 – Edinburgh, Scotland
What this talk is about

What does TFC do?
What does IntelliFactory do?
Why F#?
What is MarketingPlatform?
What were some of the key issues faced during development?
How did F# make those issues easier to address? (will see some code here)

Project status
Future developments
Conclusions
What this talk is about

What does TFC do?

What does IntelliFactory do?

Why F#?

What is MarketingPlatform?

What were some of the key issues faced during development?

How did F# make those issues easier to address? (will see some code here)

Project status

Future developments

Conclusions
About TFC

Based in Silicon Valley, TFC helps marketers increase their efficiency and their effectiveness by providing MarketingPlatform™, a comprehensive marketing technology suite that integrates marketing activities across the full marketing cycle.

These logos are registered trademarks of Microsoft Corporation.
What this talk is about

What does TFC do?

What does IntelliFactory do?

Why F#?

What is MarketingPlatform?

What were some of the key issues faced during development?

How did F# make those issues easier to address? (will see some code here)

Project status

Future developments

Conclusions
About IntelliFactory

**IntelliFactory** offers enterprise-grade F# development and training services, and a suite of functional web development tools.

These logos are registered trademarks of Microsoft Corporation.
About IntelliFactory

At IntelliFactory, we specialize in:

- Building robust .NET applications in F#
- Migrating to and extending legacy .NET applications in F#
- Customer-friendly, agile management of software development projects
- F# trainings, from basic to advanced, from individual to enterprise-wide
- Designing and implementing domain-specific languages
- Building tools for functional web application development
IntelliFactory – in a nutshell

At IntelliFactory, we firmly believe in:

- **Expertise**: constantly seeking to push the limits and apply FP to the fullest
- **Diversity**: we bring talents from all around the world; currently we have staff from Hungary, USA, Sweden, Ukraine, Colombia.
- **Solid academic and FP professional background**
- **Bridging academia and industry** -
  - Interns – EPFL, Caltech, EPITA, Eafit
  - Sponsorship – Central European Summer School in FP (CEFP 2009)
  - Industry partners – Microsoft, local and multi-national firms
- **A challenging place to work at** – but with lots of freedom
What this talk is about

What does TFC do?

What does IntelliFactory do?

Why F#?

What is MarketingPlatform?

What were some of the key issues faced during development?

How did F# make those issues easier to address? (will see some code here)

Project status

Future developments

Conclusions
F#

- Is a functional programming language developed by Microsoft
- Is an ideal vehicle for rapid and robust software development
- Packs more functionality in less code
- Yields code that is easier to extend and maintain
- Is a standard front-end in Visual Studio
- Has full access to the .NET APIs and components
- Runs within the .NET CLR, making it possible to use within existing .NET projects
Why F#?

Key benefits:

- Application code is **considerably shorter** than in C#, Visual Basic or Java
- Dramatically reduces development time by providing **better abstractions**
- Ideal for a wide range of domains including finance, science and technology, and those with heavy numerical and symbolic computation
- Language support for developing **distributed, parallel, asynchronous and reactive applications**
What this talk is about

What does TFC do?
What does IntelliFactory do?
Why F#?

What is MarketingPlatform?
What were some of the key issues faced during development?
How did F# make those issues easier to address? (will see some code here)

Project status
Future developments
Conclusions
What is MarketingPlatform™?

TFC MarketingPlatform™ allows marketing folks to

**Design** and

**Execute** marketing campaigns,

**Visualize** and

**Measure** their effectiveness

It is deployed at numerous large organizations to drive marketing campaigns.
Snippet- the Marketing Dashboard

Marketing Dashboard
KPIs
  Response rate
  Best customers
  Age groups
  Etc.
Campaign summaries
Team messages
Quick links/Favorites
Key system features

TFC MarketingPlatform™ has a number of relevant features:

Managing contents and assets
- Creating content and inventory
- Managing contents and their properties and relationships
- Uploading contents and related assets

Administering
- Users and groups
- Permissions
- Categories (content, list, etc.)
Key system features

**Campaign / Wave Designer**
- Selecting the contents of the campaign wave
- Identifying the target audience
- Customizing and previewing wave contents
- Selecting deployment type
- Payment

**List Management**
- Create lists based on rules
- Create lists based on pivot selection → from analysis data
- Upload fixed lists
Related system components

Payment subsystem

Triggering subsystem – service application

Rendering wave contents – service application

Managing the printing of marketing material – thick client on Windows

Shipping subsystem – administration of shipments, service application

Data loading – processing raw customer feeds, service application
What this talk is about

What does TFC do?

What does IntelliFactory do?

Why F#?

What is MarketingPlatform?

What were some of the key issues faced during development?

How did F# make those issues easier to address? (will see some code here)

Project status

Future developments

Conclusions
Some implementation issues

How can we speed up data access?

How do we deal with the huge amounts of information from the data cube?

How can we isolate implementation from various revisions to the data schema?

How can we allow administrator users to fine tune application logic?
What this talk is about

What does TFC do?

What does IntelliFactory do?

Why F#?

What is MarketingPlatform?

What were some of the key issues faced during development?

How did F# make those issues easier to address? (will see some code here)

Project status

Future developments

Conclusions
F# at work – a few examples

Memoization
To reduce database load where data is constant (following named FKs)

Active patterns
To form a conceptual layer on the top of the O/R mapping

Lazy computation (sequences)
To build and traverse large prospect lists
To save memory on computed sequences → compute on demand

Domain-specific languages
To express rule-based lists
To express triggering rules
Database interaction

Each deployed instance comes with a predefined and potentially different set of seed data for the key entities of the application. These entities are intertwined with the rest of the entities.

We use Linq to perform the O/R mapping → lends itself for functional-style data querying and manipulation.

F# allows to easily:

1. Build an abstraction around memoization
2. Use it for notable pieces of database data
3. Erect a conceptual layer around entities and their properties
4. Write code using the conceptual layer
Memoization

1. Building an abstraction around memoization

```ocaml
type DataMap<'a, 'b> =
  abstract Item : 'a -> 'b with get
  abstract Clear : unit -> unit
  abstract Invalidate : 'a -> unit
```
Memoization

1. Building an abstraction around memoization

```csharp
let Memoize f =
    let internalTable = new Dictionary<_, _>()
    { new DataMap<_, _> with
        member self.Item
            with get (n) =
                lock internalTable (fun () ->
                    if internalTable.ContainsKey n then
                        internalTable.[n]
                    else
                        let v = f n
                        internalTable.Add(n, v)
                        v)
    ...
```
Memoization

2. Using memoization for notable pieces of database data

```fsharp
module WorkflowStatus =
    let private workflowStatuses = Data.Memoize (fun s ->
        let db = Db.NewDbContext_MP ()
        try
            query <@ seq {
                for wt in db.WorkflowStatus do
                    if wt.Title = s then
                        yield wt.WorkflowStatusId
            } @>
        |> Seq.hd
        with
        | _ ->
            failwithf "Can not find workflow status [%s]" s)

module ID =
    let Initiated () = workflowStatuses.["Initiated"]
    let Approved () = workflowStatuses.["Approved"]
    let Denied () = workflowStatuses.["Denied"]
```
Active patterns

3. Erecting a conceptual layer around entities and their properties

```
module WorkflowStatus =
    let (|Initiated|Approved|Denied|Unknown|) (ws: WorkflowStatus) =
        if ws.WorkflowStatusId = States.WorkflowStatus.ID.Initiated() then
            Initiated
        elif ws.WorkflowStatusId = States.WorkflowStatus.ID.Approved() then
            Approved
        elif ws.WorkflowStatusId = States.WorkflowStatus.ID.Denied() then
            Denied
        else
            Unknown
```
Active patterns

4. Writing code using the conceptual layer

let w = CampaignWave.FindById (db, waveId)  // Does the wave have an approval record?
match w with
| Model.CampaignWave.Approval.Some aw ->

  match aw.WorkflowStatus with
  | Model.WorkflowStatus.Initiated -> "Awaiting Approval"
  | Model.WorkflowStatus.Approved -> statusByRTQ personId w
  | Model.WorkflowStatus.Denied -> "Approval Denied"
  | Model.WorkflowStatus.Unknown ->

  // Wave has no approval record
  | Model.CampaignWave.Approval.None ->
  ...

Copyright © 2004-2009 IntelliFactory
Results

Dramatic speed improvements for reading/writing database data (from seconds down to a fraction of a second for constructing and saving a large network of data)

Original code converted shortened and became much easier to work with, and it became independent of schema changes → these showed up as compiler errors.
Lazy computation

Building and traversing prospect lists

A sequence of raw cube data transformed into XML on the fly and streamed to the relational database.

Usage of traditional .NET streaming (provides buffering) and lazy computed sequences (provides a nice conceptual model to work with).
Lazy computation

Building values by traversing and processing streams/sequences of data.

A natural way to deal with sequences of raw data, group pieces together to build information packages.
Domain-specific languages

**Building rule-based lists**
- How can we identify prospects?

**Building triggering rules**
- When do we execute waves?

**Cost calculation**
- How much do waves cost?

Implementation via
- FsLex/FsYacc – efficient but slower to implement
- Active patterns – relatively efficient but a breeze to develop
More F# help

Units of measure

**Quotations** to model embedded DSLs
  Allows to express alternative execution mechanisms

**Computation expressions** to manipulate stateful objects
  Build asynchronous computation
What this talk is about

What does TFC do?

What does IntelliFactory do?

Why F#?

What is MarketingPlatform?

What were some of the key issues faced during development?

How did F# make those issues easier to address? (will see some code here)

Project status

Future developments

Conclusions
Project status

Main application and a number of related subsystems (triggering, rendering, shipping, etc.) developed and delivered.

Has been deployed to a number of TFC customers and largely replaced the legacy application that existed before.

New development taking place to accommodate new customers.
What this talk is about

What does TFC do?
What does IntelliFactory do?
Why F#?
What is MarketingPlatform?
What were some of the key issues faced during development?
How did F# make those issues easier to address? (will see some code here)

Project status

Future developments

Conclusions
Further developments

More UI enhancements

Internationalization – Spanish and French version

Further backend system integration – feedback source, exact wave status, etc.

New feature requests from existing customers

Exploiting functional web development
What this talk is about

What does TFC do?
What does IntelliFactory do?
Why F#?
What is MarketingPlatform?
What were some of the key issues faced during development?
How did F# make those issues easier to address? (will see some code here)

Project status
Future developments

Conclusions
Conclusions

Functional programming makes it easier to develop enterprise applications by giving quick, concise and elegant solutions to real and complex problems

F# is a great language to **prototype features** and to **write robust code**

F# **interoperates** with other .NET languages **seamlessly**, so using the right language for the right task is straightforward.

ASP.NET as a technology **benefits little** from F# and functional programming → a better, different, and functional approach is needed
Conclusions

Selling FP to commercial people needs a mindset that doesn’t ignore the realities of developing software.

What NOT to say:

“FP is a whole different way to think about problems and develop code, and it requires a new foundation to build applications.”

What works:

“FP allows to take what you have and extend it much quicker with new functionality.”

→ Needs the appropriate development platform → F# is a great choice
What this talk is about

What does TFC do?
What does IntelliFactory do?
Why F#?
What is MarketingPlatform?
What were some of the key issues faced during development?
How did F# make those issues easier to address? (will see some code here)

Project status
Future developments
Conclusions

Taking it further
Our offering for web development

View applications as **primarily client-based**
- Avoid state issues on server, no scaling issues

**Write code in F#** - no more HTML/CSS, JavaScript, etc.
- Type-safe, statically checked code that is guaranteed to run
- No more runtime errors
- Much shorter implementation time and code
- Pagelets $\rightarrow$ compose into larger pagelets/pages
- Formlets $\rightarrow$ compose forms programmatically and take their typed data

Mark functions that are client-based
- Get those automatically translated to JavaScript
- Pagelet dependencies are managed
The IntelliFactory WebSharper™ Platform
Available soon at:
http://www.intellifactory.com