Automasjon 2016

The analysis of C# to F#

Jostein Andreassen, Michael Blomli and Mikkel Eltervg

Automation

12. May 2015

Prosjektrapport – Side 2

Universitetet i Tromsø
Institutt for ingeniørvitnskap og sikkerhet
9037 TROMSØ

Studieretning: Automasjon	**År:** 2006

Tittel: The analysis of C# to F#	Dato: 12. Mai Gradering:
Forfattere: - Jostein Andreassen - Michael Blomli - Mikkel Eltervåg	Antall sider: Vedlegg:
Fortrolighet:	
Veileder: **Puneet Sharma**	
Oppdragsgiver: **Serit IT Partner Tromsø**	Oppdragsgivers kontaktperson: Jonas Juselius

Sammendrag:
Stikkord: F#, C#, fungsjonell programmering, analyse, programmering, .net

Summary

Preface

Contents

1 Introduction **5**
- 1.1 Background . . . 5
- 1.2 Problem for discussion . . . 6
- 1.3 Formulations of objectives . . . 7
- 1.4 Project specification . . . 8
  - 1.4.1 Where Serit is now . . . 8
  - 1.4.2 What Serit wants . . . 9
  - 1.4.3 Method of translation . . . 10

2 F# programming language **11**

3 Imperative and functional programming **12**

4 Project specification **13**

5 sTranslate **14**
- 5.1 How it works . . . 15
  - 5.1.1 Inputs . . . 16
- 5.2 Solution . . . 17
  - 5.2.1 Direct translation . . . 18
  - 5.2.2 Functional approach . . . 19
- 5.3 Analysis . . . 20

6 Complete analysis **21**
- 6.1 Development time . . . 21
- 6.2 Readability and clarity . . . 22
- 6.3 Debugging and error handling . . . 23
- 6.4 Performance . . . 23

7 Conclusion **24**

8 Reference list **25**

9 Attachments **26**

1 Introduction

1.1 Background

One of the biggest problems in modern application development is the rapidly growing complexity of all major software systems. This complexity makes it almost impossible to ensure the quality and accuracy of the code. It also becomes harder and harder to make changes to existing code without introducing new errors. All these difficulties multiply when you also want utilize modern computers with many CPU (central processing unit) cores for increased performance.

The imperative object-oriented programming paradigm has been dominant in software development for over 20 years. In the imperative paradigm the state variables will be handled explicitly, which can quickly give too much complexity. The functional programming paradigm has been known since the 1930s, but has not been popular with professional developers because of the slightly lower (single core) performance and greater resource use. Today these obstacles are long gone, and functional programming is experiencing a new renaissance due to significantly better control over complexity and parallelism.

We've received an assignment from Serit to translate parts of an existing project from C# code to F# code. C# is meant to be simple, modern, flexible and object oriented programming language. It is developed and maintained by Microsoft and is inspired by previous popular object-oriented languages like C++ and Java. F# is a hybrid language that supports both the familiar object-oriented method and functional programming. F# is also developed by Microsoft, and like C# also has access to Microsofts .NET framework. Serit wants us to find out the benefits of switching from development in the programming language C# development to F#.

1.2 Problem for discussion

C# and F# works in different ways, they both have benefits and downsides. The main question is if it is worth it for a company to change their main programming language. We have to look at what the company wants to achieve by making the change, and that boils down to making quality programs for a low price.

A modern IT company uses a lot of time developing, changing and fixing code. If we can use a programming language that takes less time to develop and at the same time works better without generating errors, that could be very cost saving.

The programming language F# claims to be a solution to these problems by using less code, be more simple and have better error handling than other programming languages. Our task is to find out if those claims are true by answering these questions:

- What are the benefits of switching from C# to F#?
- To what degree can we reduce the number of lines written in the program code?
- How much time is saved in the debugging stage?
- How much time is saved in the development of the code?

1.3 Formulations of objectives

To work on an analysis (compare code) of the old code compared to the new code, where we will look at how compact the code is, how many errors there are, how self-explaining the code is and how easy it is to develop the code.

1.4 Project specification

1.4.1 Where Serit is now

- They have an ASP.NET Web application in C# where the user interface is based on ASP Web Forms. All code is written in English, as well as all the text in the user interface.
- Language support is dissolved in a separate module sCore.Translation which is called from the application and performs translation according to data recorded in a translation table.
- Translation tables are located in a SQL database.

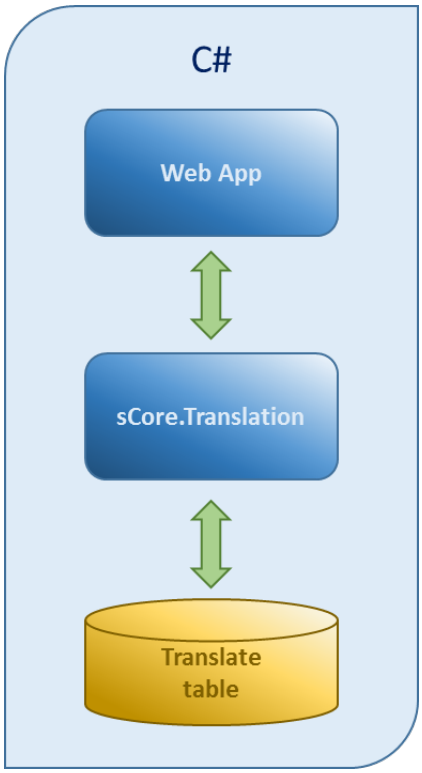

Figure 1: How the communication of Serit's sCore.Translation application looks now.

1.4.2 What Serit wants

- They want to have the existing translation module sCore.Translation developed as a separate module in the functional language F#. This should be able to be called from the present imperative program (C#) and from functional programs (F#).
- With the translation from C# to F# done, both languages and programming paradigms can be compared analytically. By this we can evaluate benefits (and possible disadvantages) with the functional paradigm in relation to an object-oriented imperative paradigm. The analysis will provide a better basis in the choice of programming language in future development projects.

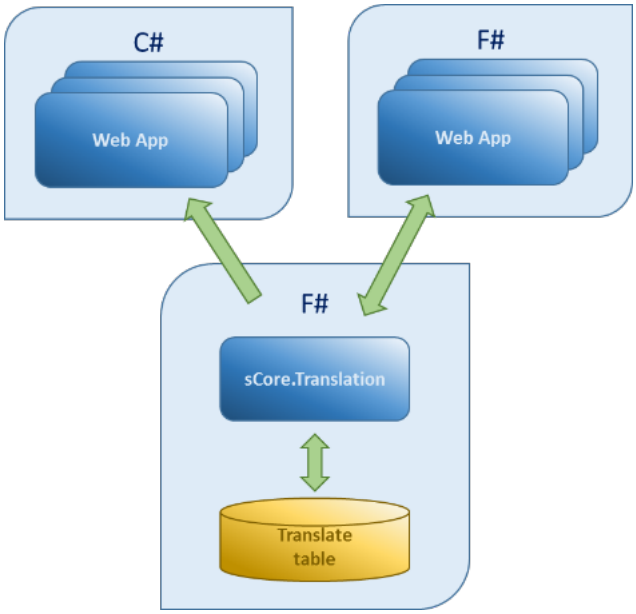

Figure 2: How they want the sCore.Translation application to communicate.

1.4.3 Method of translation

F# for fun and profit describes three levels of sophistication for porting code from C# to F#. The basic level is simply a direct port. Since F# supports imperative programming, we can translate directly. At the intermediate level, the code is refactored to be fully functional. The advanced level takes advantage of F#s data type system.

There are two paths to achieve this goal: Either by first porting to F# and then refactoring to functional code, or by converting to functional code in C# before porting that to F#.

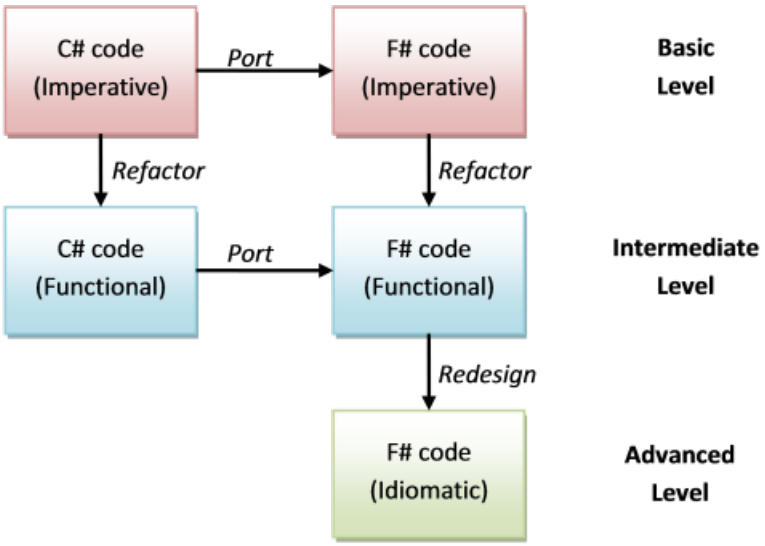

Figure 3: Method's of translating from C# to F#.

2 F# programming language

3 Imperative and functional programming

4 Project specification

5 sTranslate

sTranslate are a module that are part of a bigger program. The purpose module is to take in a word as an argument and then use a language database to translate the word.

The module is written in C# and contains 2 major function, one is designed to use when you are only going to translate one word and the other one is designed to be better translating multiple words. Even though the function works different they have the same inputs and outputs, that means that they do the same thing if you look at it from outside. You can see a illustration of the module in figure 4.

Our task from Serit is to take this C# code and translate it into F# with 2 different approaches. The first one we translate line for line and make the code the same way that the C# is coded, the other one is to translate the program the way F# is meant to be written. After we have translated it we have to a lot of optimizing to make the code better, faster and shorter. Then look at performance, difficulty of programming and all other obvious differences.

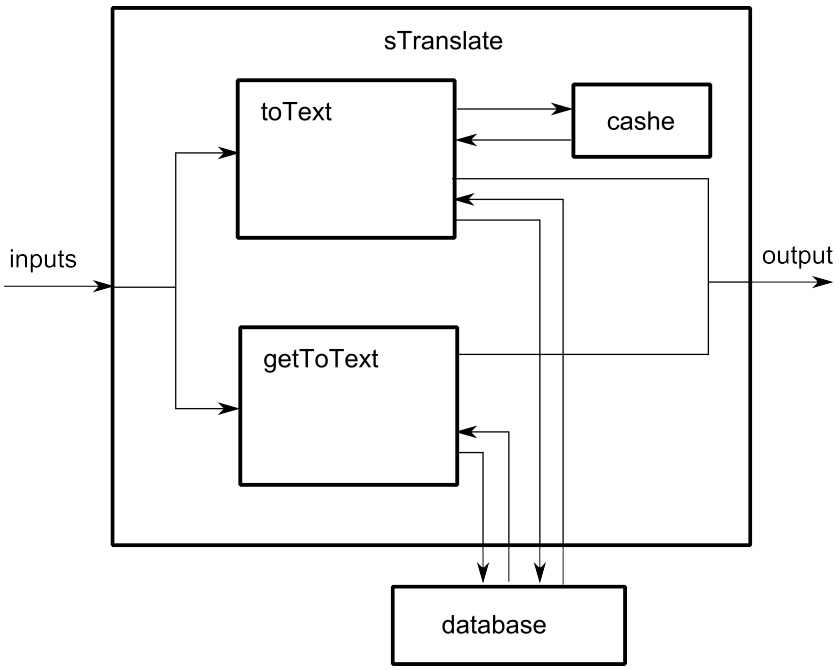

Figure 4: sTranslate module illustrated.

5.1 How it works

sTranslate module is supposed to be used with both C# and F# programs. It is important that the module is taking the same inputs and same outputs in both languages. Figure 5 show this communication.

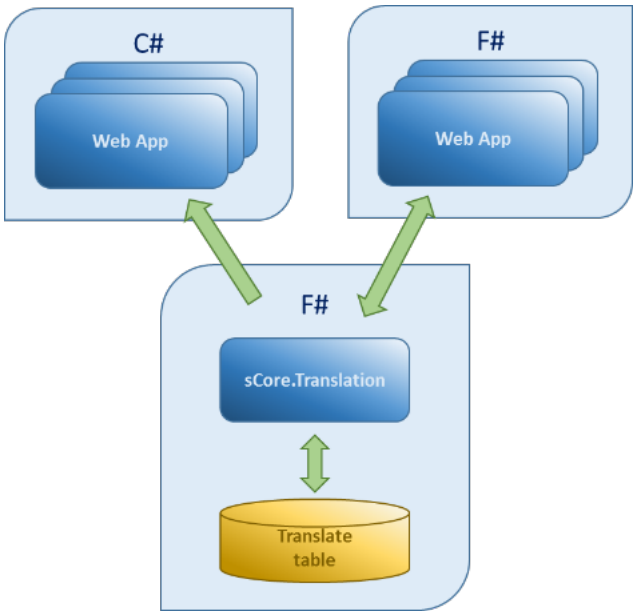

Figure 5: How Serit want the sCore.Translation application to communicate.

The main part of the module is two functions: toText and getToText, you can see this in figure 4. Both functions do the same thing but there are one major difference in them. In getToText you have to open the language database each time you want a word but in toText the database is cashed so it does not need to reopen the database.

5.1.1 Inputs

The sTranslate module takes in 5 inputs:

- **fromText**, the language the module is going to translate from. examples: no, en, ge.
- **context**, information over how the word is used, examples: string, title, lable.
- **property**, what the word have a property against, examples: id, text, tooltlip...
- **criteria**, the criteria that the search is going to get a match, examples: startWidth, endWidth, Contains.
- **toLang** the language the module is going to translate from.

5.2 Solution

There are two ways of translating code, the first one is to directly translate the program and the other is to translate it how it shod be written in the new language. Here we look at both, the direct translation and the more functional approach and look at pros and cons of the methods.

```
if (a = b){
  Console.WriteLine("equal");
}
else{
  Console.WriteLine("not equal");
}
```

Figure 6: C# code example

```
if a = b then
  Console.WriteLine("equal")
else
  Console.WriteLine("not equal")
```

Figure 7: Direct translation code example

```
match a with
    | b -> printfn "equal"
    | _ -> printfn "not equal"
```

Figure 8: Functional approach code example

5.2.1 Direct translation

The direct translation is maybe the most easy and the fastest method to translate the program. But i maybe have more flaws and have more poor performance to do it this ways. Below we have some examples from the translation that show the direct translation.

```
Insert code here!!!
```

Figure 9: Direct translation code

5.2.2 Functional approach

The functional approach is the more proper way to do it but it can take longer time and be harder to do if the person is not fluent in the new language. But the benefits can be better preference, less flaws and bugs. Below we have some examples from the translation that show the functional approach.

```
Insert code here!!!
```

Figure 10: Functional approach code

5.3 Analysis

6 Complete analysis

6.1 Development time

6.2 Readability and clarity

Readability and clarity is really important when writing code. The reason for this is that it is much more easy to find bugs, remove bugs and add future content to the code. If it is a bad written code a programmer can use a lot of time just familiarize themselves with the code before he can do changes to it.

For the most part i is up to the programmer to write readable and clear code, but the program language can have a lot to do with helping the programmer in this matter. Some programming languages like for example Assembly can be seen as programming language that makes it hard for the programmer to writhe readable code. Other like for example Python make it really easy for the programmer.

So the question is how does F# do in this matter? Is it easy or hard to write readable and clear code. How bad would a bad or new programmers code look? And how readable is a perfect code?

6.3 Debugging and error handling

6.4 Performance

7 Conclusion

8 Reference list

9 Attachments