"Vasile Alecsandri" University of Bacău
Faculty of Sciences
Scientific Studies and Research
Series Mathematics and Informatics
Vol. 19 (2009), No. 2, 459 - 470

F# MODULES INTEGRATION IN DISTRIBUTED APPLICATIONS DEVELOPMENT PROCESS

COSMIN TOMOZEI and MARIUS VETRICI

Abstract. The objective of this paper is to analyze distributed systems development process with special emphasis on F# modules integration.

Functions and classes written in F# prove to be valuable resources for a higher level of security, execution speed and accuracy. The .NET integration support makes the selection of this functional language for distributed applications development rather straightforward. Examples of architectures which implement modules written in F# will be given.

1. PRELIMINARY ASPECTS REGARDING DISTRIBUTED SYSTEMS

Our paper is focused mainly on describing the way we reengineer distributed systems, consisting of hardware, software and communication resources in order to obtain a higher level of quality and efficiency.

In [9] we have mentioned that a great number of specialists from the industry and research are being involved in studying and implementing software engineering and reengineering applications and strategies. The results of their research are both theoretical and methodological.

The development and reengineering of distributed applications have to follow a set of rules regarding the objectives, the techniques, the programming and modeling languages and testing and auditing processes.

Due to the increasing level of software systems complexity, it is practically impossible to start the development process from green field over and over again. Fast changing of the objectives which have to be accomplished by computer programs and IT systems in general presume that *maintainability, scalability and robustness* must become the most significant metrics.

Keywords and phrases: distributed applications, F# modules, forward engineering, reverse engineering, reengineering, C#, SOA
(2000) Mathematics Subject Classification: 68N01

In the same time, when talking about dimensions, volume of operations, or necessity of computing large volumes of information, the *distribution* of software, hardware and data come as a logical consequence.

Maintaining an adequate level of integrity and security involve distribution as well, hence being subjected to reengineering.

In [1], [9] there are presented some aspects regarding distributed systems that we mention in the following:

- *multiple nodes*, connected in a computer network; distributed systems suppose that multiple computers are connected and share tasks in order to realize the objectives; parallel processing is not to be confused with distributed computing; while parallel processing involves many processors on the same machine, distribution means at the outset task sharing;
- *concurrency*; each nod has independent functionalities apart from the other nodes and operates concurrently with the other nodes; it is likely to exist many processes on each node, and on each process there are multiple threads;
- *message passing* through protocols, such as TCP/IP over modems or Ethernet;
- *heterogeneity* of nodes; each node being dissimilar regarding hardware and software;
- *multiple protocols*, mainly asynchronous;
- *openness;* in comparison to sequential programs, which are mainly closed and do not change their configuration during execution, in distributed systems we can add nodes while the whole ensemble is functioning; the openness presumes that each node satisfies a set of conditions and protocols to ensure *interoperability* with the components added or modified;
- *fault tolerance and transparency* allow users to cooperate with the software system without knowing if there are components that don't work in a certain moment of time; this fact should not affect the general functioning of the system;
- *persistence;* data is stored in a persistent, non volatile environment, such as databases, data warehouses and storage servers;
- *security;* each user should interact with the system according to the rights he has; elements such as *authentication servers*, *firewalls*, *antivirus* technologies are to be used;
- there is *no central server,* but there are multiple servers which cooperate in order to achieve nonstop, uninterrupted operation of the system;

Distributed applications consist of the software components that run on the distributed systems, and have the following characteristics:

- they are built in distinct development environments, they operate in diverse environments, on different operating systems, on platforms that are connected in a computer network;
- they are built on two tiers (client - server), three tiers (client - middleware - server) or multitier (client - multiple middleware - multiple servers);

2. *SOA - SERVICE ORIENTED ARCHITECTURE* PERSPECTIVE

The applications mentioned above are mainly based on the Service Oriented Architecture, which is also known as SOA. The object - oriented programming paradigm is the one which proves to be the most appropriate for this architecture. We have JAVA RMI and CORBA from the Java based environments and Windows Communications Foundation part of the .NET Framework for Windows .NET Framework environments. ASP.NET web service architecture is described by the model from Figure 1 [3].

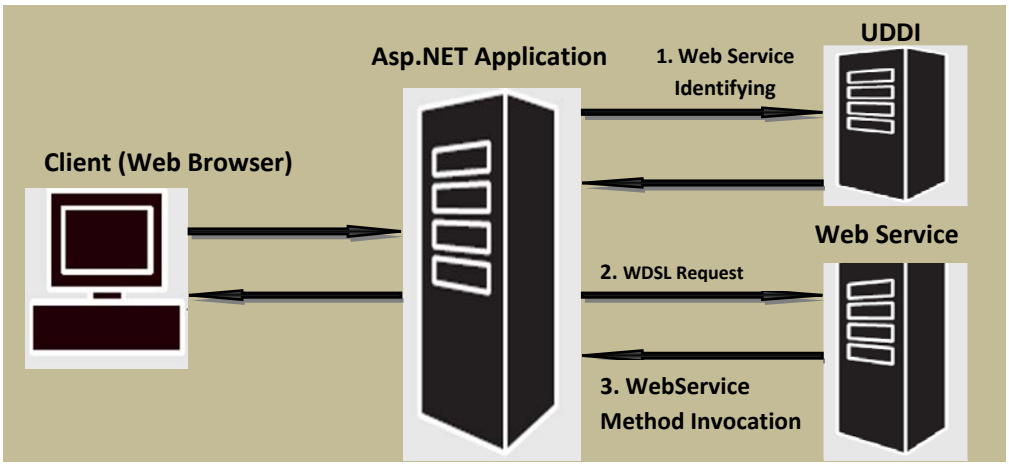

Figure 1. Asp.NET Web Service Life Cycle

First of all, in order to use the web service, the user application must find it. This operation is realized by the help of web address or by a UDDI address. UDDI means Universal Resources Data Integration. After that, the client applications find the WDSL file, which is the description file for the web service. The protocol which implements this process is SOAP and its initials come from Simple Object Application Protocol.

Client applications may utilize more than one web service, depending of their necessity. A web service may be accessed as well by many client applications, not just one.

In figure 2 [3] we describe how client applications use proxy classes in order to communicate with web services. Proxy classes' role is to translate the demands of heterogeneous client applications in SOAP requests. Message passing is realized on the HTTP protocol, which is firewall friendly and efficient.

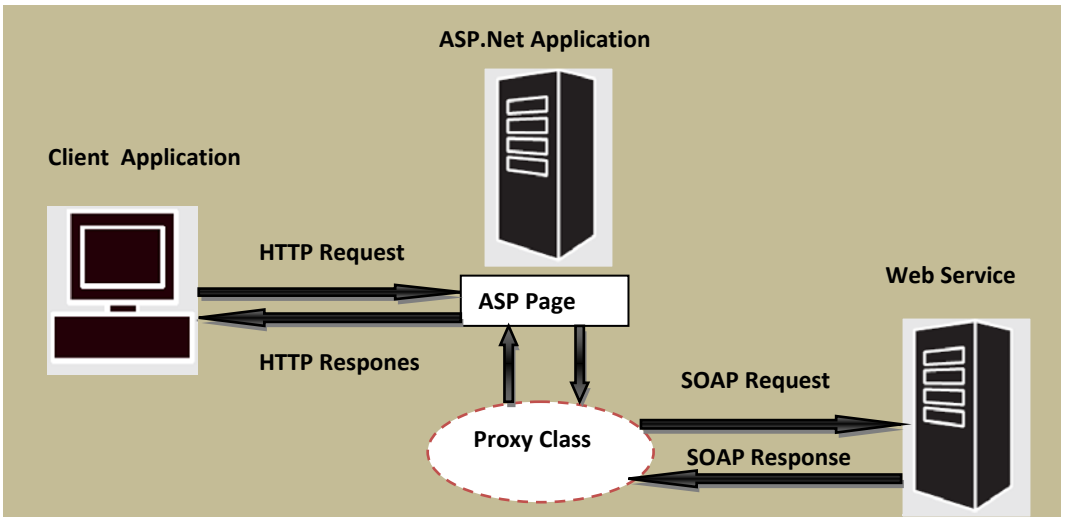

Figure 2. WebService methods invocation by proxy classes

The number of clients is described by the following formula:

$$NrUsers_k = \sum_{i=1}^{n} us_i \quad (1)$$

$$NrServ_k = \sum_{i=1}^{n} Serv_i \quad (2)$$

where:

- $NrUsers_k$ represents the amount of user applications, not humans, for the service *k;*
- us_i represents the user application;
- *n* the last index of user application;
- $NeServ_k$ represents the number of web services implemented by the client application;
- $Serv_i$ represents a certain web service implemented by the client tie;

If we consider that each web service has web methods which provide functionalities for the distributed applications, each one may be placed in formal representations.

$$NrServFunct_k = \sum_{i=1}^{n}\sum_{j=1}^{m} Serv_i * Funct_{ij} \tag{3}$$

Time evolution metrics about distributed applications are easily defined, with the help of indices. Depending on the fast changing demands and requests, the same web service may have distinctive configurations and different functionalities within hours.

The following indices are to be implemented in order to compare the number and operations implemented by client applications and web services after the process of reengineering:

$$I_{NrServFunct_k} = \frac{NrServFunct_{k1}}{NrServFunct_{k0}} \tag{4}$$

Functionalities of software application are described in a collaborative perspective by [5]. We have in [5] an array of quality characteristics, $C_1,C_2,...,C_n$ and for each one of them established the normal areas in which are enclosed, delimited like subintervals $[b_i, 1]$ with $0 < b_i < 1$, $i=1..n$, on represent on the nomogram the standard diagram of the collaborative system *functionality*:

$$IF = \frac{\min\{S_1, S_2\}}{\max\{S_1, S_2\}} \text{ [5]} \tag{5}$$

where:

- S_1 and S_2 are the surfaces delimited in the nomogam [5];

We may improve the (5) formula, by referring to the quality characteristics reflected in functionalities. The formula may have a predicted role, if the expectations of each functionality achievement regarding the quality characteristics C_i.

$$IF_2 = \frac{\min\left(\sum_{i=1}^{n}\sum_{j=1}^{m} C_i * Funct_{ij}\right)}{\max\left(\sum_{i=1}^{n}\sum_{j=1}^{m} C_i * Funct_{ij}\right)} \tag{6}$$

where:

- C_i represents the quality characteristic *i;*
- $Funct_{ij}$ represents an element from the functionality matrix;
- m is the number of rows in the functionality matrix;

- *n* is the number of columns in the functionality matrix;

Tables of association are used in order to show a relationship between each quality characteristic and the functionality matrix. In the first action, the actual value of the achievement of quality characteristics any functionality is determined. In the second action the matrix of predicted values is built. The matrix of predicted values consists of maximizing the level of quality achieved by each of the functionalities.

3. PRACTICAL IMPLEMENTATION OF WEB SERVICES IN .NET

Web methods which are implemented by the web service and return an ADO.NET component, such as Data Sets is written in the following source code lines.

```
namespace ServWebStud
{
    [WebService(Namespace = “http://cursfs.ub.ro/”)]
    [WebServiceBinding(ConformsTo =
WsiProfiles.BasicProfile1_1)]
    [ToolboxItem(false)]
    [System.Web.Script.Services.ScriptService]
    public class ServiciuWebStud :
System.Web.Services.WebService
    {
        [WebMethod]
        public string Welcome()
        {
            return “Web Service for The 1st year students”;
        }
        [WebMethod]
        public DataSet liste_studenti()
        {
           SqlConnection scon = new
SqlConnection(ConfigurationManager.ConnectionStrings[“conn1”
].ConnectionString);
string  st =  @”select nume,prenume, medie_ects from
                studenti1 order by medie_ects”;
           SqlDataAdapter adpt1 = new SqlDataAdapter();
           SqlCommand scmd1 = new SqlCommand(st, scon);
           System.Data.DataSet ods1 = new
System.Data.DataSet();
           adpt1.SelectCommand = scmd1;
           adpt1.Fill(ods1, “tabela_studenti_ADO”);
```

```
            return ods1;
        }
    }
}
```

Datasets are representing objects stored in the internal memory. Datasets bring information from Data Sources, which are placed in the external memory. Both of them are defined as .NET Classes.

We find in the code from above the following important aspects:

- the web service consists of a class, named *ServiciuWebStud* which is inherited from the System.Web.Services.WebService;
- it has a connection string for a DataBase connectivity technology, such as JDBC, ODBC or SQLCLIENT;
- the connection string is placed in the Web.Config file, which makes the application easy to reconfigure without future rebuilding;
- we also have a DataAdapter which fills the table *tabela_studenti_ADO,* from the dataset with data.

Reengineering applied to the source code from above will transform the code into a Visual Basic.NET code sequence. Even though the .NET platform offers interoperability, sometimes is necessary to translate from one language to another. For example, if one inherited application was developed in Visual Studio.NET from 2002, written in Visual Basic.NET, the new developers might want it rewritten in C#.

```
<System.Web.Script.Services.ScriptService()> _
<System.Web.Services.WebService(Namespace:=”http://cursfs.ub
.ro”)> _
<System.Web.Services.WebServiceBinding(ConformsTo:=WsiProfil
es.BasicProfile1_1)> _
<ToolboxItem(False)> _
Public Class ServWebStudBasic
    Inherits System.Web.Services.WebService
    <WebMethod()> _
    Public Function Welcome() As String
        Return “Web Service for the 1st year students”
    End Function
    <WebMethod()> _
   Public Function liste_studenti() As DataSet
   Dim sconn As New
SqlConnection(ConfigurationManager.ConnectionStrings(“conn1”
).ConnectionString)
        Dim st As String = “select nume,prenume,medie_ects
        from  studenti2 order by medie_ects”
        Dim adpt1 As New SqlDataAdapter()
        Dim ods As New DataSet()
```

```
        Dim scmd1 As New SqlCommand()
        scmd1.Connection = sconn
        scmd1.CommandText = st
        adpt1.SelectCommand = scmd1
        adpt1.Fill(ods, "tabela_studenti2_ADO")
        Return ods
    End Function
End Class
```

Reengineering based on objective's modification will permit web services to provide new functionalities to client applications, such as fragmentation of relational tables vertically or horizontally. New rules of accessing the data are also to be implemented due to reengineering, each web service or web method accessing only a dedicated fragment of a relational database.

We can add new web methods as well, or just transform the existing web methods in order to fulfill the new objectives. In our application, we will provide to the C# written web service access on the table containing information about the first year students and to the Visual Basic written web service access to the table containing information about the second year students.

In order to increase the speed of data accessing and actualization time stored procedures are also to be implemented. If the inherited software system did not have stored procedures, they will be implemented during the reengineering process.

The greatest advantage of web services is that functionalities are developed in distributed environments, such as SOAP with XML.

4. REPRESENTATIONS REGARDING F# INTEGRATION AND REENGINEERING

In this section we focus on the integration of functional modules written in F# in Windows 2008 server nodes. The process of software reengineering is entitled to add new software modules in distributed environments. In the following source code, we will put into practice a web service which returns also an ADO.NET dataset which was filled with data from the database.

In figure 3 [6] we present the functional architecture model of distributed reengineering. The process starts iteratively from the initial objectives which are realized by the current software application. In the earlier stages, the application had been subjected to *forward engineering,* which implemented the abstract requirements and objectives into actual software modules.

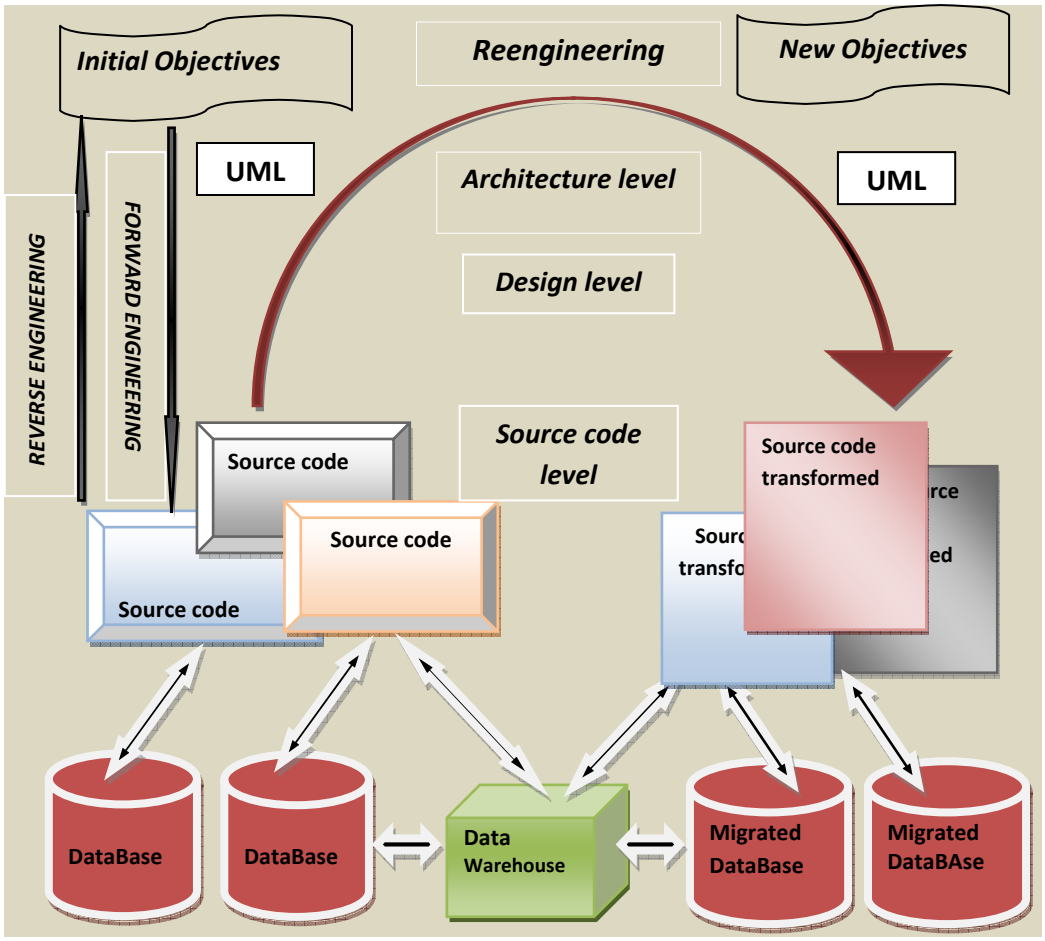

Figure 3. Functional architecture of distributed reengineering [9]

Forward engineering [4] consists of moving from the high level of abstraction levels of architecture and design, which are implementation independent to the physical implementation of the system. By forward engineering, we will get the result of source code and database implementations.

On the other hand, reverse engineering presumes the turn round way, from the existing implementations to the abstract architectures. Reengineering involves both of the processes, forward and reverse engineering.

Functional languages written modules, such as F# modules are to be integrated due to reengineering. In [7], we have mentioned that F# is appropriate for distributed computing, because of the power and robustness of functional programming languages and the offering the possibility of solving in a scientific conduct a set of very difficult problems.

The contribution of this new programming language to Computer Science research is very significant, joining together the simplicity of a functional language with the power, robustness and generality of the .NET framework.

The following source code presents a module written in F#, described in a distinct manner in [9], which represents a query from the same database as the above mentioned web services. It is straightforward to see that the entire software application is enriched with interoperability.

```
#light
open System.Collections.Generic
open System.Data
open System.Data.SqlClient
open System.Data.Common
open System
let connectionString1 = "Data Source=PROGRAMARE03-
PC\SQLEXPRESS;Initial Catalog=utilizfdiez;Integrated
Security=True;"
let fraza_sql = "select * from utiliz1"
let connect = new SqlConnection(connectionString1)
using (connect)
      (fun  connection ->
        let command = connection.CreateCommand()
        command.CommandText <-fraza_sql
        command.CommandText <- fraza_sql
        command.CommandType <- CommandType.Text;
        connection.Open()
        using (command.ExecuteReader())
            (fun reader ->
                let id_utiliz =
reader.GetOrdinal("id_utiliz")
                let nume = reader.GetOrdinal("nume")
                let prenume = reader.GetOrdinal("prenume")
                while reader.Read() do
                  Console.WriteLine (id_utiliz)
                  Console.WriteLine (nume)
                  Console.WriteLine (prenume)
                  ))
```

To conclude, we may state that working with many database management systems and with several programming languages is a key issue in distributed systems. Integrating distinct functional modules could help in that direction.

5. CONCLUSIONS

Distributed systems and consequently distributed applications are the mainstream development approach in the IT&C industry today. Web services have proved an important factor of evolution in software development, reunited by SOAP and other protocols in order to make possible machine to machine communication in distinct environments.

In this paper, we described the benefits of implementing such technologies; we stated and proved that the integration of functional languages, correspondingly F#, is appropriate and efficient.

We also compared similar applications written in different languages in order to integrate the functional modules.

Software metrics regarding web services were presented and formal aspects about functionalities and quality characteristics have been talked about.

References

[1] Edwin D. REILLY - **Concise Encyclopedia of Computer Science**, Wiley, ISBN 0470090952, 2004

[2] Cosmin TOMOZEI - **Security Engineering and Reengineering on Windows 2008 Server Based Distributed Systems** - Journal of Information Technology & Communication Security, SECITC 2009, Bucharest, pag. 63 - 73 ISBN 978-606-505-283-3

[3] Matthew MACDONALD, Mario SZPUSZTA – **Pro Asp.Net 2.0 in C# 2005**, Apress2005 ISBN-13 (pbk): 978-1-59059-496-4

[4] Serge DEMEYER, Stephane DUCASSE, Oscar NIERSTRASZ - **Object-Oriented Reengineering Patterns**, Elsevier Science, Square Bracket Associates, 2008, ISBN 978-3-9523341-2-6

[5] Ion IVAN, Cristian CIUREA **- Quality Characteristics of Collaborative Systems**, The Second International Conference on Advances in Computer-Human Interfaces, ACHI 2009, 1-7 February 2009, Cancun, Mexico, pg. 164-168, ISBN 978-0-7695-3529-6

[6] Cosmin TOMOZEI - **N-Tier Distributed Applications Dependable Construction**, Journal of Information Technology & Communication Security, SECITC 2008, Bucharest, pag.65-71, ISBN 978-606-505-139-7

[7] Cosmin TOMOZEI - **Quality Characteristics of Business – Oriented Open Source Community Projects**, Open Source Scientific Journal, Vol.1, no.1, 2009, ISSN 2066 – 740X**,** pag. 179 – 188.

[8] Robert PICKERING - **Foundations of F# - Apress 2007**, ISBN - 978-1-59059-757-6

[9] Cosmin TOMOZEI, Bogdan PĂTRUȚ - **Assessment on Distributed Collaborative Applications Research and Reengineering,** The Journal of Applied Collaborative Systems, Vol.1, no. 2, 2009, pag. 120- 128, ISSN **ISSN 2066-7450**

Cosmin TOMOZEI

Department of Mathematics and Informatics,
University "Vasile Alecsandri" of Bacău
Spiru Haret 8, 600114 Bacău, ROMANIA
cosmin.tomozei@ub.ro

Marius VETRICI

Economic Informatics Department
University of Economics, Bucharest
mariusvetrici@softmentor.ro