Configure and refactor cloud applications with Enterprise Library Integration Pack for Microsoft Azure using Aspect.NET

Vladimir O. Safonov, Dmitry A. Grigoriev, Adel N. Safonova, and Anastasiya V. Grigorieva

Citation: AIP Conference Proceedings **1648**, 310011 (2015); doi: 10.1063/1.4912564
View online: http://dx.doi.org/10.1063/1.4912564
View Table of Contents: http://scitation.aip.org/content/aip/proceeding/aipcp/1648?ver=pdfcov
Published by the AIP Publishing

Configure and Refactor Cloud Applications with Enterprise Library Integration Pack for Microsoft Azure Using Aspect.NET

Vladimir O. Safonov[1, a] , Dmitry A. Grigoriev[1, b] , Adel N. Safonova[1, c] , and Anastasiya V. Grigorieva[1, d]

[1]*St. Petersburg State University*
Faculty of Mathematics and Mechanics
28 Universitetsky prospect
Peterhof, St. Petersburg, 198504, Russia

[a] vosafonov@gmail.com
[b] gridmer@yandex.com
[c] adel_safonova@mail.ru
[d] nastya001@mail.ru

Abstract. The paper is a summary of already published results on the Aspect.NET project, our aspect-oriented programming toolkit for the .NET platform. The latest of our results up to now have been only discovered in Russian journals, so the primary goal of the paper is to widen the scope of publicizing our results. The paper overviews goals, key ideas and history of the Aspect.NET project and its use in our research on refactoring cloud applications for Microsoft Azure. The Enterprise Library Integration Pack for Microsoft Azure is a solution by Microsoft to separate cross-cutting concerns in developing cloud applications. Using this library implies modification of the target application's source code. In practice, there appear situations when updating the sources is undesirable or impossible, since they can just be confidential. The paper presents a method of seamless integration of aspects and the target application using the Aspect.NET toolkit which allows the developers to avoid changing the source code of the target application when weaving aspects.

Keywords: aspect-oriented programming, cloud computing, Microsoft Windows Azure, aspect-oriented refactoring, seamless integration, Aspect.NET
PACS: 68N19 Other programming technologies

INTRODUCTION

Cloud computing with Microsoft Azure is a rapidly evolving, state-of-the-art collection of software technologies. The architecture of cloud applications is very complicated; it consists of cloud Web services that should be developed, keeping in mind their trustworthiness, configurability, and elasticity.

Another very important approach to software development is trustworthy computing [1, 2] that unites software security, reliability, and privacy of confidential information.

The main goal of the paper is to present a method of refactoring and configuring cloud application projects and solutions for Microsoft Azure with application blocks of implementations of *cross-cutting concerns* (caching, cryptography, data access, exception handling, logging, security, validation) provided by Microsoft Enterprise Library Integration Pack for Microsoft Azure. Refactoring and configuring is implemented using aspect-oriented programming (AOP) and the Aspect.NET [2, 3] AOP toolkit for the .NET platform currently used for research and teaching in 26 countries. The latest version of Aspect.NET compatible to Visual Studio 2013 [11] contains a novel feature of replacing the base class of the target application and the calls of all its methods which is a powerful mechanism for *aspect-oriented refactoring* of cloud applications.

The Enterprise Library Integration Pack for Microsoft Azure is a solution by Microsoft for separation of cross-cutting concern in developing cloud applications. Using this library implies modification of the source code of the target application. In practice, there appear situations when the source code of the target application is just unavailable for confidentiality or any other reasons, so AOP tools relying on the source code are unapplicable. Our basic idea [11] is a method of seamless integration of aspects and the target project with Aspect.NET that allows us to avoid changing the source code of the target application.

Aspect-oriented programming [2] is a relatively new (invented in the 1990s) software development and modification paradigm intended to support modularization of cross-cutting concerns. Typical examples of such concerns are: security, privacy, logging, error handling, configuring. In general, most of the tasks to be solved with AOP are those of *trustworthy computing* [2]. Also, AOP is well suitable for solving the tasks of software configuration, since it deals with multiple automated code injections and replacements. *Aspect* in AOP is an implementation of a cross-cutting concern specified as a new kind of module. Aspect specification, in our terms [2], consists of a set of *actions* and their *weaving conditions*. *Weaving* is the way to use aspects – injecting the code of aspect actions into the *join points* of the target application code, selected as satisfying the corresponding aspect's weaving conditions. Weaving is performed automatically, based on the aspect specification, by an AOP toolkit, namely, by its component referred to as *weaver*. The aspect action may catch and process the context of the join point. So AOP is a powerful mechanism to make systematic changes and enhancements of software. In particular, AOP is well suited for configuring and securing Web services in the cloud, however the use of AOP in cloud computing is still at the starting point, so we hope our project will help to leverage it.

OVERVIEW OF THE ASPECT.NET PROJECT

Our Aspect.NET toolkit [2 – 9, 11, 13] is an AOP toolkit for the .NET platform. Its ideas were first presented in 2003 in [4]. The main goal of the project is to support AOP for Visual Studio.NET as one of the ubiquitous technologies available with Visual Studio, as a Visual Studio add-in.

We are grateful to Microsoft Research for their research grant support of the Aspect.NET project in 2002, 2004 and 2006. We have presented Aspect.NET in a number of conferences [6, 7, 9].

In 2005, the first working version Aspect.NET 1.0 was developed and published on Microsoft Faculty Connection Web site [16]. Since that time up to now, we have developed four more versions of Aspect.NET: Aspect.NET 1.1 [17], Aspect.NET 2.0 [18], Aspect.NET 2.1 [19], Aspect.NET 2.2 [20] published on Microsoft Faculty Connection Web site, and an experimental version of the weaver published on our project site yet.

Aspect.NET is based on a number of novel principles of its architecture and implementation (at least, those principles were novel when Aspect.NET appeared): multi-language AOP (currently Aspect.NET allows to combine aspects and target applications written in C# or VB); use of very simple language-agnostic AOP meta-language Aspect.NET.ML to specify aspects with *AOP annotations*; use of custom attributes to implement aspects; integration to Visual Studio; user-controlled weaving [2 – 9].

All versions of Aspect.NET, since Aspect.NET 1.0 till Aspect.NET 2.2, have the following architecture:

- *Aspect.NET Framework* - GUI for aspect visualization, weaving, and calling the updated target applications after weaving, implemented as Visual Studio add-in;
- *Converters* from Aspect.NET.ML AOP annotations to source code in C# (VB) that defines our specific AOP custom attributes used by the Aspect.NET weaver – *AspectAction* and *AspectDescription*;
- *Weaver* that performs, as the first stage, search of potential join points in the target application, and, as the second stage, actual weaving of the selected aspects into the selected join points, after the user's approval (the user can deselect some undesirable join points). Weaving is performed at CIL binary intermediate code level.

Aspect.NET does not have any specific extended language compiler, unlike the classical AOP tool AspectJ, since the scheme we use to process aspects is different: first, convert Aspect.NET.ML annotations to the aspect implementation source code, with AOP custom attributes injected, and, next, use the appropriate common Visual Studio compiler to generate the binary CIL code of the aspect. So the binary code of the aspect is just a normal .NET assembly, and our AOP attributes don't prevent from normal processing of .NET assemblies by commonly used tools – compilers, debuggers, profilers, etc. No specific XML configuration files, etc. are needed with our approach.

Our latest results on the project are: *Aspect.NET 3.0*, still at the experimental stage, compatible to Visual Studio.NET 2013 published on the Aspect.NET project site yet [3], and our new *Aspect4Cloud* aspect library [3] for refactoring Microsoft Azure cloud applications with the help of Microsoft Enterprise Library Integration Pack for Microsoft Azure.

Since Aspect.NET 3.0 is an experimental version, it only contains a weaver and a set of scripts to integrate aspect weaving into the Visual Studio assembly build process. The version of Aspect.NET Framework GUI compatible to Visual Studio 2013 is still under development and testing.

Here is an example of Aspect.NET aspect:

```
%aspect CloudServiceSecurity
 public class CloudServiceSecurity
   %rules
   %before %call MyCloudService*  %action
   public static void SecurityCheck() { ... }
}
```

This aspect injects a call of the *SecurityCheck* method before each call of any cloud service whose name starts with *MyCloudService*, i.e., makes the use of the service more secure. So, using AOP, there's no need to perform such code modifications manually, which would be unsafe. All code updates (weaving) will be performed by the weaver and controlled by the user via GUI. The advantage of AOP in this respect is as follows: automated modification of the code controlled by aspects and by AOP toolkit will be made safely, regardless of the size of the code that may be very large, and regardless of the required (probably big) number of aspect action injections. It is allowed for a developer to avoid using Aspect.NET.ML meta-language and instead use the AOP custom attribute *AspectAction* to specify the weaving rule for an aspect action. For example, the above aspect's action code written "in attributes" would look like this:

```
["AspectAction" %before %call MyCloudService*" ]
public static void SecurityCheck() { ... }
```

REFACTORING MICROSOFT AZURE CLOUD APPLICATIONS WITH ASPECT.NET

Now let's consider the feature of Aspect.NET that helps to refactor cross-cutting functionality provided by Enterprise Library Integration Pack for Microsoft Azure. A simple example is «Hands-on Lab 1: Using the Logging Application Block with Microsoft Azure Storage» [23], where by adding a reference to EL assemblies a logging application block is used in the project, and its method is called for passing a message into the WAD cloud repository of diagnostics information. It enables to tune the parameters of collecting and keeping debug messages via the graphics interface of the Logging Application Block of via its configuration files:

```
 //Web role on whose page the Logging Application Block is being tested:

public partial class Default : System.Web.UI.Page {
     //The message is sent in the handler of the mouse click on the page button
     protected void LogButton_Click(object sender, EventArgs e) {
        Microsoft.Practices.EnterpriseLibrary.Logging.
          Logger.Write("Message from the Logging Application Block");
     }
  }
```

Our task with this example is to move all the dependencies from EL and logging method calls to a separate project implemented by the aspect. Then, on applying by Aspect.NET the given aspect to the original project, we obtain its seamless integration with the Logging Application Block.

Traditionally, in Aspect.NET and other AOP tools, similar tasks are solved by placing the logging code into the aspect as aspect actions, and injecting the calls of those actions before, after, or instead of the call of the target method in the original project. In our case, the target method is the handler of the button click event *LogButton_Click()* of the *Default* Web page class. The object of that class is created and the events are passed to it by the ASP.NET and the IIS. It means that the code of the call of the target method is located outside of the assembly of the original project, and is not available to Aspect.NET. So, in our opinion, intercepting the external events handling method calls can be implemented via the inheritance of classes. If, in the aspect project, we define a class that inherits from the base class, and then to replace by that aspect class the original base project class, the desirable interception can be implemented in the overridden virtual method:

```
//The project with the replacing aspect descendant

[AspectDotNet.ReplaceBaseClass]
   public class AspectClass : Default {
       protected void LogButton_Click(object sender, EventArgs e) {
             Microsoft.Practices.EnterpriseLibrary.Logging.
             Logger.Write("Message from the Logging Application Block");
            base.LogButton_Click(sender, e);
       }
 }

//The original project, after eliminating the dependence from Logging Application Block
public partial class Default : System.Web.UI.Page {
          protected void LogButton_Click(object sender, EventArgs e) {}
}
```

The special custom attribute *[ReplaceBaseClass]* requires the Aspect.NET weaver to replace the target class by its aspect replacement descendant. More exactly, the weaver performs the following actions:

- Replace in the original assembly all calls of the methods of the base target class (including the constructors) by the appropriate calls of methods from its aspect replacing descendant.
- Make virtual all methods of the target class that are overridden in its replacing aspect descendant. If they are private, make them protected.
- If the calls of those methods in the original assembly are implemented by MSIL instructions of *call* or *ldftn*, replace them by *callvirt* and *ldvirtftn*, accordingly.
- Unite by the *ILRepack* tool (from the project Mono.Cecil [14]) the assemblies with the aspect and the original assembly.
- Assign some service name to the original base class, and assign its original name to the replacing aspect descendant.

Please note that the AOP refactoring features described above, to our knowledge, are not implemented at least in any other AOP toolkit for .NET.

The paper [11] contains some other examples of using the Enterprise Library for Microsoft Azure and further AOP refactoring of the code by Aspect.NET using the appropriate replacing aspect descendants which helps to separate the calls of EL methods from other functionality.

RELATED WORK

The most commonly used AOP toolkit for .NET is PostSharp [21]. Aspects are defined in PostSharp by custom attributes. PostSharp provides the following kinds of join points: method calls, accessing and updating properties, fields of a class, generation of an event or of an exception. To weave aspects, it is necessary to add to the target project some references to PostSharp service assemblies, to add the source code of the aspects to the target project, and to mark by special attributes the join points in the code of the target project. All of that requires storing aspect definitions and weaving rules together with the source files of the target project, so too high coupling is created between them. If the AOP tool is to be changed from PostSharp to other tool, it would be difficult to do. To our knowledge, there is no documented way of seamless integration of aspects with PostSharp.

A technique somewhat similar to our *ReplaceBaseClass* functionality is provided in CaesarJ [22] – virtual classes that solve similar task of giving more flexibility to object-oriented scheme. However, CaesarJ is implemented for the Java platform, and its specifics are to use a special extra keyword *cclass* (for CaesarJ class). We provide our replace base class functionality for .NET, without introducing any extra keywords, driven by a specific AOP custom attribute.

As already noted above, using AOP for cloud computing in general is now in the starting stage yet. The Enterprise Library Integration Pack for Microsoft Azure [10] itself contains a feature named Unity Application Block to enable some kind of limited weaving the EL's cross-cutting functionality into target applications. But weaving in Unity is implemented by a kind of interceptors controlled by special Unity containers, which is not so comfortable and does not allow the developers to perform seamless integration of the code. The advantages of our approach are, first, to enable powerful mechanism of aspect-oriented refactoring using the replacing aspect descendant, and, second, to enable seamless integration of the cross-cutting code and the basic cloud application functionality.

CONCLUSIONS

As stated above, the paper is actually a summary of already published results on the Aspect.NET project. Those results up to now have been already published in Russian journals, so one of the goals of the paper is to widen the scope of our results we consider important for the IT community.

Using the method proposed above, we developed the *Aspect4Cloud* aspect library for Aspect.NET to support refactoring and configuring cloud applications for Microsoft Azure using Enterprise Library Integration Pack. The code of the library is published on the Aspect.NET project site [3].

Seamless integration of aspects into target applications enables the developers to efficiently solve the problems of maintaining software projects without modification of the target source code. The proposed method allows us to use service libraries to implement cross-cutting functionality but to avoid explicit dependences on those libraries in the source code of the target project. In further project development, it is possible to drop using the library or to replace the library by another one. So we decrease the risk that a mistaken AOP decision would cause substantial updates of the target project. Implementing the proposed method with Aspect.NET allows the developers to use the common Visual Studio 2013 IDE, and to follow our simple code patterns implemented in Aspect4Cloud library to create aspects that implement access to library services.

Possible further research directions can be targeted to development of more aspect libraries to make easier to refactor Microsoft Azure cloud applications by adding service libraries functionality with the help of aspects and Aspect.NET.

REFERENCES

1. V.O. Safonov. Trustworthy Compilers. Wiley Interscience. John Wiley & Sons, 2010.
2. V.O. Safonov . Using aspect-oriented programming for trustworthy software development. Wiley Interscience. John Wiley & Sons, 2008.
3. Aspect.NET Web site. http://www.aspectdotnet.org
4. V.O. Safonov (2003). Aspect.NET – a new approach to aspect-oriented programming. .NET Developers Journal, 1(4), 36-40.
5. V.O. Safonov (2004). Aspect.NET – concepts and architecture. .NET Developers Journal, 2(9), 44-48.
6. V.O. Safonov. Presentation on Aspect.NET at the Web site of Microsoft Research SSCLI (Rotor) Seminar, Redmond, September 2005: research.microsoft.com/en-us/events/sscli2005/safonov.ppt. Checked May 30, 2014.
7. V.O. Safonov Presentation on Aspect.NET at Microsoft Faculty Summit, Redmond, July 2005. http://research.microsoft.com/en-us/um/redmond/events/fs2005/presentations/FacultySummit_2005_Safonov.ppt. Checked May 30, 2014.
8. V.O. Safonov and D.A. Grigoriev (2005). Aspect.NET: aspect-oriented programming for Microsoft.NET in practice. .NET Developer's Journal, 3(7), 28-33.
9. V.O. Safonov, M.K. Gratchev, D.A. Grigoriev and A.I. Maslennikov (2006). Aspect.NET – aspect-oriented toolkit for Microsoft.NET based on Phoenix and Whidbey. In: J. Knoop, V. Skala (Eds.), .NET Technologies 2006 International Conference. Univ. of West Bohemia Campus Bory, May 29 – June 1, 2006, Pilsen, Czech Republic. Full Paper Proceedings (pp. 19-29). http://dotnet.zcu.cz/NET_2006/NET_2006.htm .
10. The Web site of Enterprise Library 5.0 Integration Pack for Microsoft Azure // http://entlib.codeplex.com/wikipage?title=EntLib5Azure . Checked May 30, 2014.
11. D.A. Grigoriev, A.V. Grigorieva., V.O. Safonov. Seamless integration of aspects to cloud applications based on the Enterprise Library Integration Pack for Microsoft Azure and Aspect.NET // Computer Tools in Education, 2012, No. 4, 3 – 15 (in Russian).
12. V.O. Safonov. The Microsoft Azure Cloud Computing Platform. – Moscow: BINOM Publishers, 2011, 235 pp. (in Russian).
13. V.O. Safonov. Aspect-Oriented Programming and Aspect.NET as security and privacy tool for Web and 3D Web programming. Chapter 11 (40 pp.) – In: Security in Virtual Worlds, 3D webs and Immersive Environments: Models for Development, Interaction and Management, IGI Global Publishers, 2010.
14. The Mono.Cecil project site: http://www.mono-project.com/Cecil . Checked May 30, 2014.
15. V.O. Safonov. New Features of Microsoft Azure Cloud Computing Platform. – Moscow: BINOM Publishers, 2013, 304 pp. (in Russian).
16. Aspect.NET 1.0. September 2005. https://www.facultyresourcecenter.com/curriculum/6219-AspectNET-10.aspx?c1=en-us&c2=0. Checked May 30, 2014.
17. Aspect..NET 1.1. March 2006. htttps://www.facultyresourcecenter.com/curriculum/6334-AspectNET-11.aspx?c1=en-us&c2=0 . Checked May 30, 2014

18. Aspect.NET 2.0. September 2006. https://www.facultyresourcecenter.com/curriculum/6595-AspectNET-20.aspx?c1=en-us&c2=0 . Checked May 30, 2014
19. Aspect.NET 2.1. April 2007. https://www.facultyresourcecenter.com/curriculum/6801-AspectNET-21.aspx?c1=en-us&c2=0 . Checked May 30, 2014
20. Aspect.NET 2.2. April 2010. https://www.facultyresourcecenter.com/curriculum/8658-AspectNET-22.aspx?c1=en-us&c2=0 . Checked May 30, 2014
21. The PostSharp project site. http://www.postsharp.net/ Checked May 30, 2014.
22. I. Aracic, V. Gasiunas, M. Mezini, K. Ostermann. An overview of CaesarJ // Transactions on Aspect-Oriented Software Development I, Berlin - Heidelberg – New York: Springer, 2008, 135-173.
23. The Web site on Hand-on Labs for Enterprise Library Integration Pack for Microsoft Azure. http://www.microsoft.com/en-us/download/details.aspx?id=28785 . Checked May 30, 2014.