IJCSE International Journal of Computer Sciences and Engineering Open Access
Review Paper Vol.-7, Issue-1, Jan 2019 E-ISSN: 2347-2693

Aspect Oriented Programming Tools for .Net Framework

P.R. Sarode[1*], R.N. Jugele[2]

[1]Inter Institutional Computer Centre, Nagpur University Campus, RTM Nagpur University, Nagpur, India
[2]Department of Computer Science, Shri Shivaji Science College, Congress Nagar, Nagpur, India

[*]*Corresponding Author: priya.s1011@gmail.com, Tel.: +91-9503869986*

Available online at: www.ijcseonline.org

Accepted: 16/Jan/2019, Published: 31/Jan/2019

***Abstract*—** Aspect oriented programming is a young concept in Computer science. It is succeeding from research projects towards commercial applications. Most of the current AOP tools suitable for commercial projects are proposed for Java platform only, which bounds their applicability. AspectJ is the leading tool for Java technology, the only way to implement a new programming paradigm such as Aspect Oriented Programming is either to extend the Java language or to develop Java API to support it. For .NET, the situation is different — it is a multilanguage programming environment. Today, Aspect Oriented Programming is supported in most languages and platforms. For Microsoft .NET, PostSharp is the most advanced and mature framework, and has been used commercially for several years. There are various known Aspect Oriented Programming tools for Microsoft.NET also. This paper present the analysis and overview of the all various popular AOP tools for .Net framework in detail.

***Keywords*—**Aspect Oriented Programming, .Net, AOP tools, CLR, MSIL.

I. INTRODUCTION

Aspect oriented programming (AOP) is a young concept in Computer science. Gregor Kiczales lead a team of researchers who first described AOP in 1997. As any engineering discipline, software engineering learned that simplicity is achieved by pursuing a good separation of concerns. A separation of concerns is what allows designers to componentize solutions to recurring problems – whether in software engineering, mechanical engineering, electrical and electronic engineering, or architecture. Although object-oriented programming (OOP) made significant progress in this direction, there is still a large category of issues that cannot be addressed appropriately using traditional programming styles: technical requirements and other features grouped under the term of crosscutting concerns. AOP is an extension of OOP and is specifically designed to address crosscutting concerns in software development. AOP enables developers to write a formal expression of solutions to repeating problems and automatically implement them rather than manually doing so with conventional programming techniques. Just as object-oriented programming did not replaced structure programming, AOP is not a replacement of OOP but rather an extension of it. AOP's positive impact on software development is measurable in both controlled studies and real-world development efforts. Today, AOP is supported in most languages and platforms. AspectJ allows programmers to have the advantage of modularization for cross cutting concerns[1], the reference AOP implementation, is the leading tool for Java and the tool called AJATO 0is an assessment tool used to provide the quantitative analysis of software artifacts. It helps in computing the Aspect-Oriented metrics and also supports the use of heuristics [2]. For Microsoft .NET, PostSharp is by far the most advanced and mature framework, and has been used commercially for several years by Fortune 500 companies.

II. AOP FOR MICROSOFT .NET

Firstly, when .NET appeared in 2000, some scientists intended to implement AOP for .NET, following the success of AspectJ — simply considered AspectJ syntax and semantics and tried to present them some way into C#, either as a language extension or as an addition to C# code in the form of XML. However, the environment and essentials of .NET have some important differences from Java. To better understand these differences, most sensitive for AOP implementation, let us look at two important notions that [3] used when designing and implementing Aspect.NET.

i) .NET is a Multilanguage programming platform, more open in style than Java. For Java technology, the only way to implement a new programming paradigm such as AOP is either to extend the Java language or to develop Java API to support it. For .NET, the situation is different

— it is a Multilanguage programming environment. Therefore, the first idea proposed in 2002 when starting to do AOP for .NET was that AOP for .NET should preferably be language - agnostic. It would contradict the spirit of .NET and its principles of Multilanguage interoperability if AOP features would enhance C# language only and would not be available in Visual Basic.NET or in Managed C++.NET. However, further reasoning on this subject can lead to implementing too radical a solution: Let us make .NET's Common Language Runtime, the basis for Multilanguage interoperability (or at least its shared – source version, Rotor), "aspect - aware," to support AOP in the strongest way. Hundreds of AOP tools for .NET, discussed below, were developed. Many attempts have been made to join approaches to AOP for .NET and to implement within the CLR some common "AOP engine." The intent of the very interesting seminar "AOP goes .NET," organized at Microsoft in November 2005, was specifically to develop a common approach to AOP for .NET. There are many reasons for that primary ones are a variety of different approaches to AOP, and the strategic importance of the CLR, which is too significant and critical to any changes to accept the risk of updating for the needs of a new programming paradigm. Implementation of AOP for .NET should not be tied to any programming language (say, C#). It should be language - agnostic. However, it should not prevent from doing normal work by common - use .NET tools: CLR, debuggers, profilers, and so on. In other words, Implement AOP for .NET using the extension mechanisms it provides, but do not require that common - use .NET tools be "AOP - aware" This principle leads to the second notion [3] formulated in 2002 and described in early 2003 [4].

ii) Second notion is that .NET has a general and comfortable extension mechanism: custom attributes. They ideally fit to implementing any type of annotation: formal specifications, comments on the code, and for annotating some code as belonging to some aspect. Annotations in the form of custom attributes are understandable by the appropriate specialized tools — formal verifier and theorem prover, aspect weaver, and so on — the tools those annotations are intended for. Other common - use .NET tools (CLR, debugger, etc.) just ignore those custom attributes. However, as compared to any separate XML configuration file, .NET makes possible custom attributes that are an inherent part of the binary code and portable executable file, are traveling in the same file with the MSIL code, and cannot be "lost" when the code of the assembly is passed over the network. Aspect.NET was the first AOP tool for .NET that used custom attributes. A few years later, other AOP researchers arrived at similar beliefs and started using custom attributes to represent aspects for .NET. Another key question that get up when implementing aspects for .NET is - at what level should the weaving process take place? The following answer looks evident: Since .NET is a Multilanguage platform and all .NET languages compiled into MSIL code, MSIL should become the basis for weaving. There are two problems in MSIL related to weaving. First, MSIL postfix - style code has a linear structure and is not easy for weaving: adding, inserting, or replacing pieces of code. Therefore, an MSIL - level weaver should first convert MSIL code of the aspect and the target application into a more proper tree like or graph like format, then complete the weaving and convert the resulting target code back into MSIL. Second [4], if the AOP tool uses custom attributes to mark aspects and the results of their weaving, the attributes can only mark the named entities in the code: assembly, class, method, and field definitions. They cannot mark executable statements of MSIL code (assignments, branches, calls, etc.). Therefore, after weaving some aspects into a target code, it will not be, in general, possible to "color" the fragments that relate to different aspects. However, if all aspect code injections contain calls of aspect methods or access to aspect fields, the aspect visualizer can use them to identify the code fragments related to weaving of each aspect [3].

III. AOP TOOLS FOR .NET FRAMEWORK

Keeping the previous considerations in mind let us overview and analyze the most popular AOP tools for .NET. Names and references for some AOP - for - .NET tools are as follows:

[1] LOOM.NET

LOOM.NET [5] implements both runtime weaving (by Rapier - LOOM.NET weaver) and static weaving (by Gripper - LOOM.NET weaver). The ultimate goals of the project are to help in developing and enhancing dynamically reconfigurable multithreaded software that sometimes cannot be restarted for reconfiguration, for reliability reasons. The Rapier - LOOM.NET weaver is used to generate and weave dynamic proxies. Aspects in LOOM.NET defined via aspect classes. LOOM.NET implements some AspectJ - like features for the commercial version of .NET and for Mono [6]. Advice custom attributes are used in aspect class definitions to mark aspect classes, their methods, and to explicitly specify the weaving conditions. The dynamic weaver uses its factory methods to generate interwoven objects. The important restriction on this dynamic weaving approach is as follows: The methods to be interwoven should be virtual or should be defined via interfaces. For purposes of the project, it is acceptable. However, for general use of the system, this restriction looks serious. LOOM.NET also used to implement design - by - contract aspects. The authors of the approach pay a lot of attention to performance and reliability [4].

[2] AspectDNG

AspectDNG [7] is a multilanguage aspect weaver that works at the MSIL level. It allows us to weave any .NET 1.1 or .NET 2.0 assemblies. The project was inspired by AspectJ and is implemented in C#. Because the weaver works with MSIL code, the aspects and the target application can be written in any language compiled in MSIL. The weaver accepts two forms of join point languages. The first is XPath, the language widely used for navigation in XML namespaces. The second is the language of regular expressions. AspectDNG is definitely a helpful tool, but it cannot yet be regarded as a full - fledged AOP framework: It lacks a proper aspect specification language and an IDE [3] consider XPath is a proper language for AOP specification, but AspectDNG can be useful as part of a larger AOP development platform.

[3] Aspect#

Aspect# [8] is an AOP tool for C#. It consists of a simple (non - XML) aspect configuration language and an aspect engine (weaver). The advices supported are limited to interceptors only. The aspect engine (specified by an instance as the Aspect Engine class) is called from C# using its Wrap method, whose argument is the type to be extended by weaving interceptors. The list of interceptors is provided in the configuration file. The result of the Wrap method is the proxy to the extended type. Pointcuts are also limited to a primitive form of regular expressions. To identify the separately defined elements of the aspect (interceptors) in the aspect definition, string keys (like "interceptor1") are used. Aspect# has no IDE; neither is it integrated to Visual Studio or another existing IDE for .NET.

[4] PostSharp

PostSharp [9] is a postcompiler for the .NET platform. It works as an MSIL code transformer and uses custom attributes. This tool can be useful for many purposes, including AOP. To illustrate the AOP capabilities of PostSharp, a high - level aspect weaver, PostSharp Laos, was developed on top of PostSharp. An aspect in PostSharp, intended for use with the PostSharp Laos weaver, should be defined as a special type of custom attribute, which can be used with any class or method to extend its functionality. The project site contains the following example of such aspect that sheds more light on the AOP cuisine of PostSharp:

```
public sealed class RequiresRoleAttribute:
OnMethodBoundaryAspect
{
string[] roles;
public string[] Roles
{
    get { return this.roles; }
    set { this.roles = value; }
}
public override void
OnEntry(MethodExecutionEventArgs e )
{
    ((ISecurable) e.Instance).RequireRoles( this.roles);
}
}
```

This class defines an aspect as a custom attribute that adds security functionality of checking the user's roles on entry to some given methods (say, methods performing deletion of some information). The aspect can be applied to any assembly using the ordinary custom attribute definition in C#:

```
[assembly: RequiresRole ( Roles=new string[] {
"Delete"}, TargetMethods= "Delete*" )]
```

When processing the MSIL code of the application, the PostSharp Laos weaver understands this custom attribute as an instruction to weave into any method in the assembly whose name fits the Delete * wildcard, an interceptor aspect injecting the security action on entry to the method. This type of approach is typical for many AOP tools for .NET: They don't provide a single conceptually clear AOP framework with its own aspect specification language and set of concepts (more challenging to implement), but instead, use the existing features (classes, methods, custom attributes) to identify aspects directly by means of the implementation language (C# in most cases). The smartest and most complicated part of such AOP tools is a weaver whose work and semantics are considered as some kind of tricky "magic" can characterize such an approach to AOP as "experimental," which is normal for research projects.

[5] DotSpect

DotSpect (aka .SPECT) is a tool for aspect .NET assemblies. The advice code written in any language as long as we have a compiler for that language and a mechanism to inject in the targeted assembly. By default, .SPECT provides support for writing aspects in C# and VB.NET. Currently the assemblies are aspect at Compile time (Static injection) [10].

[6] Encase

Encase is an aspect oriented programming framework for the .NET Platform written in C#. Encase is unique in that it provides aspects to be applied during runtime in managed code rather than being configuration file based that other AOP frameworks rely upon. The advantage of applying aspects in this way is that it promotes productivity to developers new and/or unfamiliar with AOP. Extensions is both currently in early development [11].

[7] Compose*

Compose* is a modular extension tool/framework used for composition filters. This architecture supports multiple targets, repository-based set of analysis tools. Compose* (pronounced "Compose-star") is a project that aims at enhancing the modularization capabilities of component and

object based programming. In particular, compose* offers aspect-oriented programming through the composition filters model [12].

[8] Weave.NET

Weave.NET [13] is a project and AOP tool targeted to language - independent aspects. This term is understood by the users in the sense that [3] explained earlier in the section: MSIL - level aspects. The input for the weaver is a set of .NET assemblies and an XML file to specify weaving directives in the style and spirit of AspectJ. Weaving is performed at load time.

[9] Wicca and Phx.Morph

Wicca [14] is an AOP framework that implements various kinds of weaving strategies: both static and dynamic weaving, at the source code and MSIL code levels. In particular, MSIL - level weaving in Wicca is supported by Phx.Morph, which is based on Microsoft ' s Phoenix [15] back - end compiler infrastructure (as well as the weaver in Aspect.NET). Dynamic weaving in Wicca is implemented using another Microsoft technology and tool: the Microsoft Managed Debugger (mdbg) [16]. Wcs, a source code compiler and weaver, implement C #source code – level weaving in Wicca. The wcs tool implements a small extension of C#: statement annotations. Here is an example from the Wicca Web site:

```
public SimpleDraw()
{
    [Log (Sev.Info, ”Creating a line ”)]
    Shape s = new Line(new Point(1, 9), new Point(9,1));
}
```

Such annotation before the assignment statement is converted by wcs into ordinary C# code like this (again, the example is from the Wicca Web site):

```
[Statement(19, ”LogAttribute”, Sev.Info, ”Creating a line”)]
public SimpleDraw()
{
    Shape s = new Line (new Point(1, 9), new Point(9,1));
}
```

The latter code is compiled by common - use C# compiler. This is an original decision of the problem of annotation lack for statements, although the issue is resolved at the C# source code level rather than at the MSIL code level. The only disadvantage of this technique is lack of readability. The user who thinks in terms of the extended C# may not understand what happened to his or her code, since in debugging the user will see the latter fragment of the source code or its image in MSIL code. As from the examples, AOP implementation in Wicca is based on custom attributes. As for many other AOP projects for .NET, the goal of the project authors is to implement all functionality of AspectJ. However, the project is notable by three advantages: a wide spectrum of weaving techniques is offered to users, an interesting method of statement annotations (applicable to Java as well as to C#) is utilized, and the latest Microsoft technologies and tools are used.

[10] Seasar.NET

S2Container.NET is a lightweight DI container supporting AOP. It is a port of Java version Seasar2 to .NET framework [17].

[11] Spring.NET Framework

Spring.NET AOP is implemented in pure C#. There is no need for a special compilation process - all weaving is done at runtime. Spring.NET AOP does not need to control or modify the way in which assemblies are loaded, nor does it rely on unmanaged APIs, and is thus suitable for use in any CLR environment. Spring.NET currently supports interception of method invocations. Field interception is not implemented, although support for field interception could be added without breaking the core Spring.NET AOP APIs [18].

[12] HyperC#

HyperC# tool supports Multidimensional Separation of Concerns for C#. Adopting some of the techniques from Hyper/J (which is MDSOC for Java), the HyperC# tool, and implemented using C #. With HyperC#, a graphical user interface system is created that provides AOP features to C# developers or programmers by using MDSOC approach. Using this tool, C #developers can create new concern classes and new hypermodules. Using integration relationships, the system integrates concern classes together by using the pre-processor developed to create a new C# class that can be further used in other integration files, or can be compiled in an executable file using the C# compiler. HyperC# System demonstrates how the AOP extension to C# can enhance software evolution, modularity and reusability [19].

[13] CrosscutterN

CrossCutterN [20] is a free and lightweight AOP tool for .NET using IL weaving technology. As other AOP tools like postsharp, it helps developers to inject AOP code into their programs. The advantages of CrossCutterN comparing with other AOP, technologies include:

Free: CrossCutterN is open source and free under MIT license.

Light Weight: Instead of adding compile time dependency to projects, CrossCutterN injects AOP code after project assemblies are built. This approach allows AOP code injection into assemblies whose source code are not available, and decouples project code from AOP code as much as possible.

IV. LIMITATIONS OF AOP TOOLS FOR .NET

The following limitations of the existing AOP tools for .NET listed as follows:

1) Lack of multilanguage interoperability; dependence on C# or MSIL.

Most AOP tools for .NET are tied to C# only, so they do not use properly one of the main features of .NET: multilanguage interoperability. The only approach used to achieving interoperability in some other AOP tools for .NET is reliance on MSIL (which is surely not a high - level language).

2) Lack of proper aspect specification language.

Most projects prefer to use C#, MSIL, and .NET "as is," explicitly defining aspects as C# classes inherited from some specific "core aspect implementation classes " using custom attributes or " side - door " aspect configuration fi les not easy to understand. They just model AOP and AspectJ features in a lower - level implementation language. With such an approach, AOP for .NET becomes simply a kind of discipline, not quite reliable (since when explicitly defining complicated AOP custom attributes, it is easy to make a mistake), without any support at the aspect design stage. Note that for Java AOP tools unlike .NET AOP tools, [3] mentioned as one of the limitations quite a different thing, too - complicated aspect specification languages. It looks as if most AOP researchers for .NET experience such a deep influence of AspectJ that they make one of their major goals modeling AspectJ features by C# classes and XML configuration fi les rather than discovering new, more general, and open - style AOP features that may be supported by .NET.

3) Lack of aspect manipulation and visualization GUI.

Most AOP tools for .NET are still in an early stage of development, they only provide AOP supporting API and (some of them) AOP configuration files in XML, but they lack GUI to manipulate aspects, to visualize them, to debug them, and so on — everything accustomed to doing with "non - AOP " applications, due to modern integrated development environments.

4) Lack of integration to Visual Studio.NET or other IDE for .NET.

It is widely appreciated that Microsoft Visual Studio.NET is the most convenient and powerful IDE for .NET. Millions of users are accustomed to its rich set of features: source code navigation, highlighting and refactoring, project templates, automatic code generation for the most complicated kinds of programs (such as Web services), multilanguage debugging at the source code level, profiling, automated unit test generation, teamwork, and so on. No code is developed from scratch. The user always has a skeleton (and a visual image, if appropriate) if his or her code is at the design stage. So when learning and assessing a new technology such as AOP, users are not going to lack any of these features. They do not want to use command - line interface to start AOP tools. They do not want to suffer when typing complicated XML configuration files for an AOP tool in notepad, without the support of templates. They would like to see their aspects in proper images: what they look like, which join points they have with the target application, what results they should expect from weaving (in terms of the source code), and so on. Many AOP tools for Java (e.g., AspectJ) are integrated to Eclipse — one of the most popular Java IDEs, and offer the features just described. But unfortunately, most existing AOP tools for .NET are not yet integrated to Visual Studio.NET, although this IDE provides two integration methods: as an add - in and through Visual Studio Integration Program API.

V. OBSERVATIONS AND CONCLUSION

The aim of this paper is to contribute to the brief summary of the AOP tools for .NET. Each tool is unique and featured in their own way. In this section, we present the special feature, which is observed during the study of each tool.

Loom.Net is a tool, which implements runtime as well as static weaving. AspectDNG is a multilanguage aspect weaver that works at MSIL level. Aspect# is an AOP tool for C#, which consist of simple aspect language and aspect weaver. PostSharp is used as a postcompiler for .Net platform at MSIL level by using custom attributes. DotSpect is run for C# and VB.Net by using .Net assemblies at compile time. Encase is for C#.Net, that provides the aspects applied during runtime and by this feature it promotes productivity for new users of AOP. Compose* tool is specially designed for composition filters to increase the modularization of the component. Wave.Net tool is designed to language independent aspects and it is working on MSIL Level. Wicca and Phx.Morph implements at multi-level weaving strategies (static, runtime, source code level, MSIL level). HyperC# tool demonstrates how the AOP extension to C# can enhance software evolution, modularity and reusability. Seasar.Net is a lightweight tool with DI (Dependency Injection) technology. Spring.Net tool is purely implemented in C# with runtime weaving only and it is suitable for CLR environment. CrosscutterN is an open source (under MIT License) lightweight tool that uses IL (Intermediate Language) weaving.

All these tools contain many original approaches to implementing AOP. A lot of them use custom attributes. Some of them have experience of use for real tasks, some of them implemented only in C#; some are implements at MSIL level. Therefore, we can conclude that this paper provides the rich set of AOP tools for .Net platform with their unique features.

VI. ABBREVIATIONS AND ACRONYMS

Following abbreviations and acronyms used in the given paper listed below.

[1] AOP- Aspect Oriented Programming
[2] CLR- Common Language Runtime
[3] MSIL- Microsoft Intermediate Language
[4] API- Application Programming Interface
[5] GUI- Graphical User Interface

ACKNOWLEDGMENT

We thank the Babasaheb Ambedkar Research and Training Institute (BARTI), Pune for funding our research.

REFERENCES

[1] Jatin Arora, Jagandeep Singh Sidhu and Pavneet Kaur, "Applying Dependency Injection Through AOP Programming to Analyze the Performance of OS", International Journal of Computer Sciences and Engineering, Vol.3, Issue.2, pp.45-50, 2015.
[2] Geeta Bagade, Shashank Joshi, "Analysis of Aspect Oriented Systems: Refactorings using AspectJ", International Journal of Computer Sciences and Engineering, Vol.4, Issue.5, pp.76-80, 2016.
[3] Safonov, V. O. (Vladimir Olegovich) Using aspect-oriented programming for trustworthy software development /Vladimir O. Safonov. p. cm. ISBN 978-0-470-13817-5QA76. 64. S253 2008.
[4] Safonov V. Aspect.NET: a new approach to aspect - oriented programming, .NET Developer's Journal 2003 ;(4): 36 – 40.
[5] LOOM.NET Web pages. Available at http://www.rapier - loom.net/.
[6] Mono. Available at http://www.mono - project.com.
[7] AspectDNG Web pages. Available at http://sourceforge.net/projects/aspectdng/.
[8] Aspect# Web pages. Available at http://www.castleproject.org/aspectsharp/.
[9] PostSharp Web pages. Available at http://www.postsharp.org/.
[10] DotSpect Web pages. Available at http://dotspect.tigris.org/.
[11] Encase Web pages. Available at http://theagiledeveloper.com/articles/Encase.aspx.
[12] Compose* Web pages. Available at http://composestar.sourceforge.net/.
[13] Weave.NET. Available at http://www.dsg.cs.tcd.ie/dynamic/?category_id= - 26.
[14] Wicca and Phx.Morph Web site. Available at http://www.cs.columbia. edu/eaddy/wicca.
[15] Microsoft Phoenix. Available at http://research.microsoft.com/phoenix.
[16] Microsoft Managed Debugger (mdbg) Web pages. Available at http://msdn. microsoft.com/msdntv/episode.aspx?xml=episodes/en/20060302clrjs/manifest.xml.
[17] Seasar.Net Web pages available at http://s2container.net.seasar.org/en/index.html
[18] Spring.Net Framework for Aop Available at http://www.springframework.net/doc-latest/reference/html/aop.html.
[19] Angela Hantelmann, Cui Zhang: "Adding Aspect-Oriented Programming Features to C#.NET by using Multidimensional Separation of Concerns (MDSOC) Approach", in *Journal of Object Technology*, vol. 5 no. 4 Mai-June 06, pp. 59-83.
[20] CrosscutterN Available at https://www.codeproject.com/Tips/CrossCutterN-A-Light-Weight-AOP-Tool-for-NET

Authors Profile

Priyanka Sarode awarded B.Sc. degree in 2006 and MCA degree in 2009 from Rashtrasant Tukdoji Maharaj Nagpur University, Nagpur. Currently she is pursuing Ph.D. in Computer Science and a research fellow at Inter Institutional Computer Center, Rashtrasant Tukdoji Maharaj Nagpur University, and Nagpur. Her research work is acknowledged by BARTI, Pune. Her main research work focuses on Programming languages, Aspect Oriented Programming. Mail Id: priya.s1011@gmail.com Mobile No. 950386998

Ravikant Jugele is a M.Sc. in Computer Science from Marthwada University, Aurangabad in 1993. He also completed Ph. D. in Computer Science from Rashtrasant Tukdoji Maharaj Nagpur University. He is currently working as an Associate Professor in Department of Computer Science, Shivaji Science College, Congress nagar, Nagpur. Since 1995, his research interests include Multimedia and Hypermedia, cloud computing, Programming languages, Artificial Intelligence, Deep Technology and so on. Mail Id: rn_jugele@yahoo.com.

