Aspect.NET — aspect-oriented toolkit for Microsoft.NET based on Phoenix and Whidbey

Vladimir Safonov
St. Petersburg State University, Russia
v_o_safonov@mail.ru

Mikhail Gratchev
St. Petersburg State University, Russia
9r@mail.ru

Dmitry Grigoryev
St. Petersburg State University, Russia
gridmer@mail.ru

Alexander Maslennikov
St. Petersburg State University, Russia
khan@tepkom.ru

28 Universitetsky prospect, Petrodvorets, St. Petersburg, 198504 Russia

ABSTRACT

Aspect-oriented programming (AOP) methodology is evolving from research projects towards commercial applications. Most of the existing AOP tools suitable for commercial projects are intended for Java platform only which limits their applicability. Known AOP tools for Microsoft.NET such as Aspect#, Loom.NET, etc. are still at experimental stage. Most of them lack flexibility and comfortable user interface.

Aspect.NET, our AOP framework for Microsoft.NET, offers a new approach taking the best of Microsoft .NET specifics. Aspect.NET allows to define aspects using any language implemented for .NET that supports the concept of attribute. For aspect specification, we developed very simple and compact language-agnostic AOP meta-language - Aspect.NET.ML. At the source code layer, aspect definition in Aspect.NET looks like the code of a compilation unit annotated by Aspect.NET.ML constructs. The AOP annotations are converted into specific AOP custom attributes used by the Aspect.NET tool. Thus, an aspect assembly keeps all necessary information for aspect weaving whose result is represented as an augmented assembly.

Aspect.NET implementation is based on Microsoft Phoenix – state-of-the-art multi-targeted optimizing infrastructure for developing compilers and other language tools, in particular, comfortable for creating and editing .NET assemblies. The weaver uses Phoenix IR for scanning target applications and weaving aspects.

Aspect.NET Framework (GUI and aspect editor) is implemented as add-in to Microsoft Visual Studio.NET 2005 (Whidbey) and is seamlessly integrated into it. Important features of Aspect.NET Framework are: visualization of join points at source code layer, and user-controlled filtering potential join points before weaving.

Keywords

Aspect-oriented programming, Microsoft.NET, AOP meta-language, join point, weaving, Phoenix, Visual Studio.NET 2005, add-in.

1. INTRODUCTION

Modern AOP approach to software development is intended to solve a lot of issues related to increasing complexity of architecture, development and maintenance of software products. Aspect-oriented approach is helpful to simplify the business logic of an application, due to explicit separation of its cross-cutting concerns.

Well known examples of cross-cutting concerns are MT safety, security and logging.

More complicated example close to the authors' area of expertise is the task of extending a compiler by implementation of a new source language feature – e.g., generics in C#. It is clear that all the phases of the compiler should be updated for this purpose. So it is not enough to add new modules to the compiler but is it also necessary to insert into its code a number of tangled fragments to glue the new modules of the compiler to the existing ones.

Theoretical foundations of AOP are well defined by a variety of researchers [1, 14]. However, even basic AOP concepts are still understood and interpreted different way by different researchers and developers. Except for widely known AOP tools for Java – AspectJ [8] integrated into Eclipse, there are no AOP tools yet that could be easily integrated to the existing software development environments.

The goal of the Aspect.NET project [9, 10, 42] described in this paper is to create an AOP tool for Microsoft.NET which would be flexible, language-agnostic and integrated to the latest Microsoft software development environment – Visual Studio.NET 2005. A version of Aspect.NET for academic shared source .NET implementation - SSCLI / Rotor is also developed.

Aspect.NET allows to visualize the result of weaving at source code level, and to manually select or unselect potential join points.

The paper describes Aspect.NET principles, architecture, components, functionality, perspectives and ideas of future work on Aspect.NET.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

.NET Technologies 2006
Copyright UNION Agency – Science Press,
Plzen, Czech Republic.

2. RELATED WORK

The AOP methodology founded by Gregor Kiczales [1] is similar to a number of approaches already used in software technologies for a few years - subject-oriented programming [21], composition filters [22] adaptive programming [12], intentional programming [23], generative programming and transformational programming [13].

The papers [24, 25] provide introduction to AOP and describe pluses and minuses of different approaches to AOP. In [26] the most popular AOP tool – AspectJ is described in detail. The Web site [19] contains a variety of information on all AOP approaches and tools. The paper [18] describes one of possible approaches to AOP for .NET based on interceptors. Papers [27, 28] show the applicability of AOP approach for implementing object communication protocols' design patterns. In [29], design-by-contract foundations are described, as a reliable software development technology. In our opinion, design-by-contract principles can be applied using AOP tools, Aspect.NET in particular. The paper [28] proposes an approach to handling using AOP, provides some examples and gives some recommendations of AOP applicability at this software lifecycle stage.

Since Java was the most advanced software development platform in mid-1990s, the first AOP tools were developed for the Java platform. In particular, AspectJ [8] provides the following Java extensions

- *Aspects* – implementations of cross-cutting concerns;
- *pointcuts* – collections of patterns for join points selection and aspect weaving;
- *advices* – actions to be performed on reaching the aspect's joinpoints;
- inter-type declarations (*introduce*) — definitions of aspect members to be inserted into a target application in aspect weaving, but visible by the aspect only, rather than by the target application;.
- dynamic updates of control flow before, after or instead the code of a join point.

One of the key ideas of AspectJ - to perform a given action on reaching a given join point in the code – can be considered as an enhancement of the concept of breakpoint used in debuggers. But the most fundamental principle of AspectJ is to define a new kind of modules for aspect definitions. The paper [14] provides a systematic look at the existing AOP tools – AspectJ, HyperJ [32], Demeter, DemeterJ, and AOP models – PA, TRAV, COMPOSITOR and OC.

Another group of problems related to AOP is *aspect mining* [15 – 17], or as we call it *aspectizing* — extracting aspects from non aspect-oriented applications. Aspectizing can be very helpful to improve readability and maintainability of applications. There are several research projects and tools for aspectizing implemented for the Java platform: Aspect Mining Tool (AMT) [15], Aspect Browser [16] and FEAT [17].

When the Microsoft.NET platform was developed, it appeared necessary to implement multi-language aspects, in the spirit of .NET language interoperability, rather than to limit aspects to be only extensions of Java or any other concrete programming language.

There are lots of examples of real cross-cutting concerns and their implementation, both in commercial and in research software projects, in particular for Microsoft.NET platform and its non-commercial SSCLI implementation. When looking at the code developed by the SSCLI team to port Rotor to MacOS, or at the code developed by Gyro (generics for Rotor) team, it is quite clear that both of these are actually aspects.

Currently there are a number of research projects to support AOP for Microsoft.NET. Among them are: Aspect#, Loom.NET, Weave.NET, Wicca [44], Compose* [45]. The existing approaches to implementation of AOP for Microsoft.NET can be divided into four groups, according to the ways of representing aspects [47]:

- Using XML schemes for defining AOP specifications, e.g. SourceWeave.NET [43], Weave.NET [7], first versions of AspectDNG [38].
- Using COM+ style interceptors for dynamic weaving and activating AOP functionality. The configuration of the whole AOP system is described by XML files [41].
- Using Composition Filters Model (CF) as extending special classes – Compose* [45].
- Using both custom attributes and XML - Aspect# [5]
- Using custom attributes - Aspect.NET [9], Phx.Morph [46], AspectDNG [38].

So our Aspect.NET approach relates to the fourth group, according to the above classification.

The most advanced of the existing AOP integrated development environments (IDE) for the Java platform is referred to as *AspectJ Development Tools (AJDT)* [36] developed by the AspectJ team as a plug-in to the Eclipse IDE to support using AspectJ tools.

There is another tool similar to AJDT for Eclipse — *AspectJ Development Environment (AJDE)* [33-35] that can be plugged into Emacs, JDEE, Sun Studio, NetBeans and JBuilder.

Phx.Morph is another research AOP project which uses the same weaving techniques based on MS Phoenix [11]. In this tool, weaving is performed using attribute-based annotations. The tool does not offer any AOP meta-language for aspect specifications. Lack of AOP meta-language makes readability of aspects and specification of non-trivial join points much more complicated.

Currently none of the existing AOP IDE for Microsoft .NET, prior to our Aspect.NET tool, has comfortable GUI. We think this is because of initial stage and research nature of the majority of AOP implementations for Microsoft.NET.

3. ASPECT.NET BASICS

An aspect in Aspect.NET [9, 10] is defined as a source code of a class (more generally speaking, a compilation

unit) in C# or other .NET language, annotated by our simple AOP meta-language (referred to as *Aspect.NET.ML*) statements to highlight the parts of aspect definition. They are: *aspect header* (with optional *parameters*), (optional) *aspect data*; *aspect modules* (methods or functions), and *aspect weaving rules* which, in their turn, consist of *weaving conditions* and *actions* (to be woven into a target assembly, according to these rules). The structure of Aspect.NET.ML meta-language is so simple and self-explanatory that we decided to explain it using our examples given below, rather than provide its precise EBNF definition.

The aspect weaving rules determine the *join points* within a target application where the actions of the aspect are to be woven. The aspect actions provide the aspect's functionality.Speaking in terms of knowledge management, aspect weaving rules can be considered as special kind of knowledge (a rule set) defining how to apply the aspect to a target application.

There can also be weaving rule sets separate from concrete aspects, similar to pointcuts in AspectJ. Unlike AspectJ, a Java extension for AOP, in Aspect.NET, due to use of language-agnostic AOP annotations, it becomes possible to avoid the issue of extending each of the .NET languages by its own AOP extensions specific of that language.

The *Aspect.NET pre-processor* converts the AOP annotations to definitions of AOP custom attributes (*AspectDef*), specially designed for Aspect.NET, to mark classes and methods as parts of the aspect definition (see fig. 1). Next, an appropriate common use .NET compiler transforms the AOP custom attributes to the aspect assembly's metadata stored together with the MSIL code.

Join points in Aspect.NET are determined by weaving rules which are parts of aspect definition, or are defined in a separate rule set module. The weaving rules contain: *conditions* of calling aspect actions (*before, after,* or *instead*); *context* of the action call (a *call* of some method, *assign* to a variable (field), or *use* of some variable (field); *wildcard* to find the context of the aspect action's call.

The process of aspect weaving consists of *two phases* – *scanning* (finding join points within the target application) and *inserting (weaving)* the calls of the aspect actions into the join points found.

Unlike many other AOP tools, Aspect.NET allows the user to *select or unselect* any of the possible join points using Aspect.NET Framework GUI, to avoid "blind" weaving that could make the resulting code much less understandable and actually non-debuggable.

4. ASPECT.NET DESIGN

The Microsoft.NET platform is based on the principles of peer-to-peer multi-language programming. For any of the .NET languages, a very comfortable toolkit for software development and maintenance is provided – Microsoft.NET Framework and Visual Studio.NET.

The need to support multi-language programming makes the task of weaving and locating aspects more complicated, as compared to the Java platform. For example, AspectJ [8] is actually an implementation of Java extension by AOP constructs and concepts. AspectJ consists of the extended Java language compiler and a set of specific utilities that can work with this Java extension only.

To avoid developing a separate compiler for each of the .NET languages for the purpose of implementing multi-language AOP, Aspect.NET uses custom attributes to represent information on aspects. Due to that, an aspect definition in Aspect.NET is a syntactically and semantically correct source code of a compilation unit, with AOP custom attributes added to annotate parts of aspect. Typically, an Aspect.NET aspect is converted to a class with its fields and methods, marked by AOP custom attributes, intended for compilation into a .NET assembly by the appropriate common use .NET Framework compiler. The AOP custom attributes are usable and understandable by Aspect.NET only. They are stored together with the rest of the aspect assembly and don't prevent from normal functioning of the other .NET tools. Due to our approach, there is no need to make a special "AOP-aware" version of the .NET Framework or Visual Studio.NET.

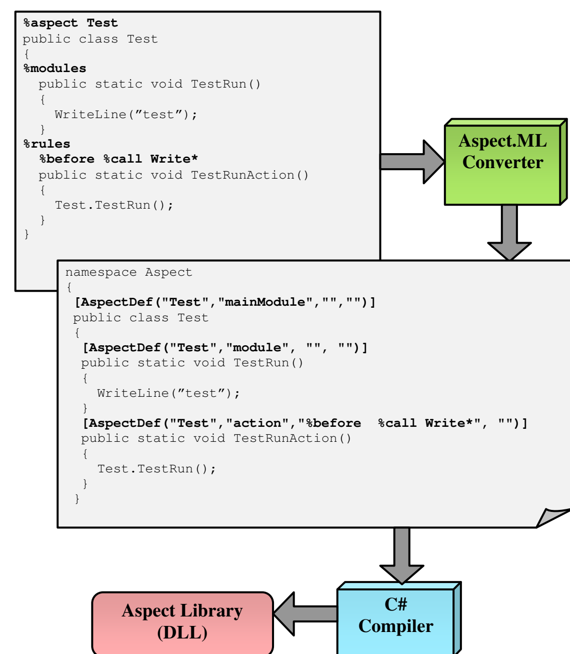

Figure 1. Aspect.NET.ML conversion to custom attributes

Full compatibility of Aspect.NET aspects to all the .NET tools makes it possible to use all the code refactoring, analysis, profiling and other features of .NET tools, while working with an aspect definition. Moreover, all the existing OOP quality criteria and metrics are applicable to .NET aspect-oriented applications based on Aspect.NET.

Aspect weaving is performed "statically" (see fig. 2), at the layer of .NET intermediate representation language (MSIL) and metadata, rather than at source code layer. All weaving-related transformations are made by the Aspect.NET toolkit. There is no need to transform in any way either source or intermediate code of a target application before weaving Aspect.NET aspects.

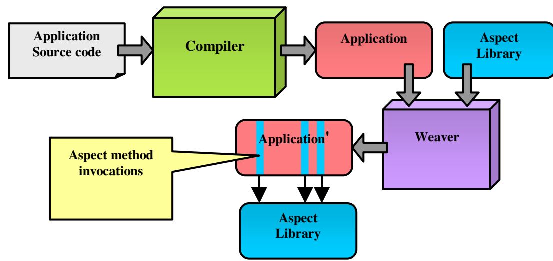

Figure 2. Static weaving in Aspect.NET

The advantages of static aspect weaving in Aspect.NET, as compared to dynamic weaving (e.g., in LOOM.NET [2]) and load-time weaving (e.g., in Weave.NET [7]), are higher performance and better understandability of a target application with the aspects woven. Dynamic weaving is usually implemented with the help of some debugging API which makes the operating system perform checking of each executable code instruction to satisfy some specific conditions, and to enable jumping to some other appropriate part of code when the condition is satisfied. Such dynamic checks may dramatically decrease performance. On the contrary, due to Aspect.NET approach, when using MSIL code of the resulting assembly it is quite possible to track the results of aspect weaving in vast detail by .NET utilities (*ilasm/ildasm,* debuggers, etc.) Thus, a developer who uses Aspect.NET is guaranteed to get a predictable and understandable resulting application after weaving. So the user does not need to use any kind of tricky checks of the results of weaving aspects, any non-trivial kinds of debugging, etc.

Up to the present moment, the main reason why similar AOP toolkits haven't yet been developed for .NET was the lack of adequate common use tools for analyzing and updating .NET assemblies (whose structure is very complicated) at the layer of MSIL intermediate code and metadata. To handle assemblies, some of the developers had to use RAIL [41] or to reinvent a wheel by developing their own, limited toolkit for this purpose.

Our Aspect.NET tool is based on Microsoft Phoenix [11] – a multi-targeting optimizing compiler back-end development environment. Phoenix provides a convenient high-level API to create, handle and update .NET assemblies by transforming it into high-level Phoenix IR (HIR) suitable for any program transformations like weaving. The resulting assembly unit is converted back to MSIL and metadata format. The latest version of Phoenix is dated November 2005 and is available within the framework of Phoenix Academic Program [11].

One of main shortcomings of the existing experimental AOP tools for .NET (Aspect# [5], AOP.NET [3], etc.) is the lack of functionality for analyzing and debugging the results of weaving aspects.

As for Aspect.NET, all its components have the central part, Aspect.NET Framework, implemented as an add-in to Microsoft Visual Studio.NET 2005. Due to that, the user can, for example, visualize the results of aspect weaving at the source code level. Aso, a version of Aspect.NET compatible to the Shared Source Common Language Infrastructure (Rotor) is developed. Currently it is based on command-line interface using Perl scripts. This version also uses Phoenix and is based on the same weaver.

The main components of Aspect.NET (see fig. 4) are as follows:

- Weaver
- Meta-language converter
- Aspect.NET Framework

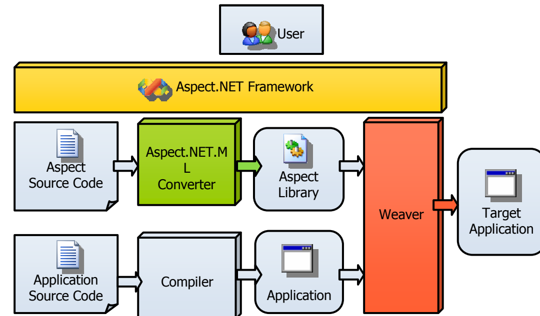

Figure 4. Components of Aspect.NET

Aspect.NET Framework allows the user to define aspects in Aspect.NET.ML meta-language, by creating a new kind of project (*Aspect*) and using a skeleton of the aspect source code generated by our wizard (see fig. 3), to map potential join points into the original target assembly's source code, and to visualize the results of weaving.

To collect information on the potential join points in the target assembly, as well as to perform aspect weaving itself, Aspect.NET Framework uses the functionality of the weaver. At the *scanning* phase, the weaver matches the code of the target assembly against the aspect (using its weaving rules), and creates the list of the potential join points. At the *weaving* phase, the actions of the aspect are woven into the target assembly. The user can edit the list of the (potential) join points, based on visualizing the join points within the target assembly's source code.

5. ASPECT.NET.ML CONVERTER

Aspect.NET.ML converter transforms user-defined aspects from AOP meta-language into source code fully written in the aspect's implementation language, annotated by AOP custom attributes. Also, the converter calls the appropriate common use .NET language compiler to compile the resulting source code into a .NET assembly.

Implementation of the converter is based on *CodeDom* – a set of .NET Framework classes for generating and handling object-style representation of a .NET source code. The aspect definition is transformed into a CodeDom graph which allows to modify the source code and to use language-independent form of aspect definition inside Aspect.NET.

Specifically for Aspect.NET, we introduced a new kind of Visual Studio.NET project – *Aspect* that includes a code pattern for aspect definition and all related resources. Thus, seamless integration into the Visual Studio IDE is enabled, and aspect reuse becomes easier.

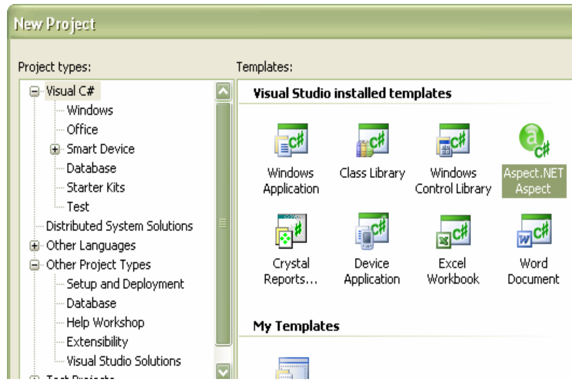

Figure 3. Creating a new Aspect.NET aspect project

On creating a correct aspect definition by the user, it is converted, then compiled into an assembly, and automatically passed to the *aspect browser* for its subsequent use within the Aspect.NET Framework.

In the aspect example in AOP meta-language (please see Appendix A), the keyword *%aspect* starts the *aspect header* that contains its *name* (in this example - *Politeness*), and can also contain *parameters* (lacking in this example). Then goes the *%modules* part where the aspect modules (methods) are defined. In the *%rules* part, the aspect *actions* are defined, each of them preceded by its *weaving rule*. In this example, the first action is to be inserted *before* calling each method of the target application, the second one - *after* each of its method calls.

In the next listing (see appendix B), the source code of the *Politeness* aspect generated by the converter is presented. All the members of the aspect's implementation class are marked by appropriate AOP custom attributes.

6. WEAVER DESIGN APPROACH

In Aspect.NET, weaver is implemented as a separate application, which allows to distinguish between weaving itself and its mapping into the source code. So, access to source codes of a project is not mandatory for subsequent weaving which is performed at the level of binary representations of the target assembly and the aspect assembly.

To find and analyze join points, the weaver uses high-level intermediate representation (HIR) of the binary target assembly generated by Phoenix [11]. Each executable module of the assembly is represented by a graph of high-level instructions which enable access to their source and destination arguments, debugging information, information on the parent unit, etc. The Phoenix API enables, on loading a MSIL assembly represented by a PE file, to get access to control and data flow, to the list of modules and instructions, to detailed information on types and symbols, to information on variable dependencies, etc. This makes possible to find a variety of the kinds of join points, and makes the weaving independent of concrete aspect implementation language. In scanning mode, the weaver scans this instruction stream, finds the join points (based on the weaving rules), and passes their coordinates in the target application to the Aspect.NET Framework add-in which presents them to the user.

Next, on getting from the framework the list of the user-selected join points, the weaver starts its weaving mode, scans the instruction stream of the target application, and finds the user-selected join points. Then, the weaver generates instructions for calling aspect's actions with the appropriate arguments. The arguments of the action can be the target method name and the pointer to the target object whose method is called. The weaver injects the generated aspect's action call instructions into the join point specified by the weaving rule, - before, instead or after the target call.

7. CASE STUDY: ASPECT.NET IN ACTION

Now let's consider in more detail the scenario of using Aspect.NET and the principles of its functioning.

1. The user defines an aspect in AOP meta-language and passes the source code of the aspect (as part of the Visual Studio's *Aspect* project) and the target application's source code (also a Visual Studio project) into the Aspect.NET Framework (see fig. 5).

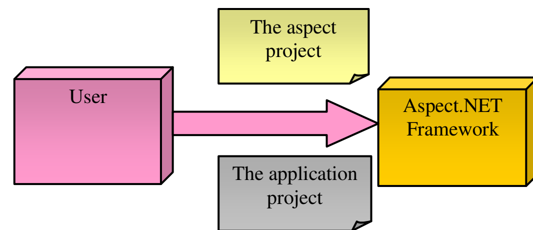

Figure 5. Creating the aspect and the application projects

2. Aspect.NET Framework initiates the compilation of the source code of the target application by the .NET compiler from the appropriate language, to create the target assembly with its debugging information (.pdb file). Also, Aspect.NET Framework passes the source code of the aspect to the AOP meta-language converter which, in its turn, converts the source code with meta-language annotations into a source code with AOP custom attributes, and generates a ready-to-use aspect assembly (by calling the .NET compiler). See fig. 6.

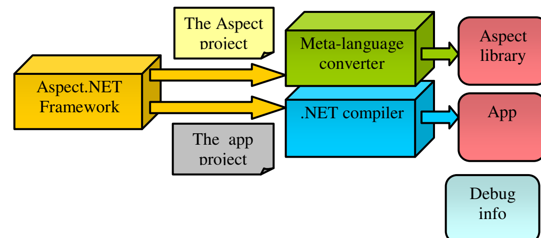

Figure 6. Preparing the aspect and the target application for weaving

3. To create a list of all possible join points within the target application, Aspect.NET Framework invokes the scanning phase of the weaver. To map the join points to the source code of the target application, Aspect.NET Framework provides the weaver with its debugging information (for Microsoft .NET Framework – represented as .pdb file, for Rotor – as .ildb file) The weaver performs

the scanning and generates the join points list as an XML document (see fig. 7).

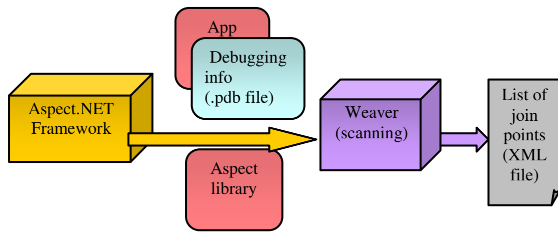

Figure 7. Generating the list of joinpoints

4. Based on the XML file, Aspect.NET Framework creates a GUI representation of the join points list, so that the user could visualize each of the join points within the editor of the code of the target application. The user can also filter the set of the join points by unselecting any of them. Then, Aspect.NET Framework passes the updated list of the join points and the other relevant working files to the weaver for the phase of weaving itself. As the result of weaving, the user obtains the updated target application's assembly (see fig. 8).

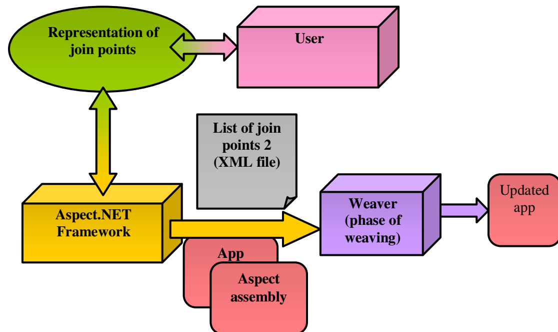

Figure 8. Join points filtering and weaving

Due to such scenario, the phases of scanning and weaving are separated. This opens great opportunities for software maintenance and configuration. For example, instead of passing to the client an updated version of a big monolithic application, it will be enough to pass the list of join points (internally represented in Aspect.NET as an XML file), the assemblies of the aspects implementing new functionality, the weaver application itself, and a simple script to initiate weaving on the client side. Thus, if the user would like to create a new version of the application with extended functionality, she just needs to configure the weaving of the appropriate aspects.

8. ASPECT.NET FRAMEWORK: FUNCTIONALITY OVERVIEW

Aspect.NET Framework provides user-level functionality for examining, studying and understanding Aspect.NET aspects. It contains:

- *Aspect browser*, to examine aspect DLLs, their weaving rules, and comments to them provided at aspect design stage in Aspect.NET.ML.

- *Join points tree*, displaying the hierarchy of namespaces, classes and methods of the target assembly's project, whose leaves are the possible join points.

- *Visualizer*, to display the mapping of the join points onto the source code of the target assembly.

8.1 Aspect browser

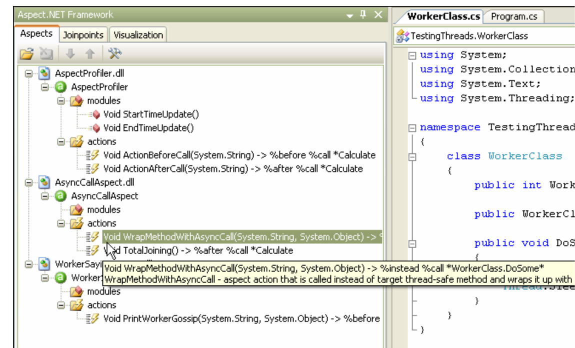

Figure 9. Aspect browser

Fig. 9 illustrates the Aspect.NET aspect browser functionality. The user can take a look at any of the available aspects, their modules and actions, and comments to them. The functionality of the browser is similar to the Outline View in the AJDT for Eclipse [36] (see fig. 10).

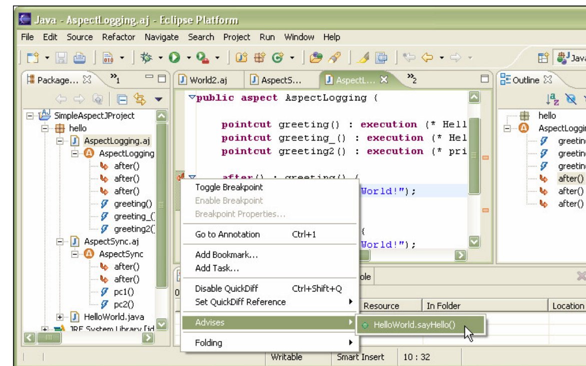

Figure 10. The aspect browser in AJDT for Eclipse.

The browser allows to change the order of the aspects, to resolve possible conflicts related to the order of weaving aspects to an application. So, if actions of the two aspects affect the same join points in the application, the rules of the aspect displayed higher will be applied before the rules of the one displayed lower.

8.2 Join points tree

On completion of scanning the target assembly by the weaver, the Aspect.NET Framework creates a join points tree, and displays it for the user (see fig. 11). The join points are represented by information on their actions to be called, and on how they will be woven according to the weaving rules – before, after or instead the join point code. By clicking at the join point leaves of the tree, the user can take a look at the appropriate points in the source code of the target application.

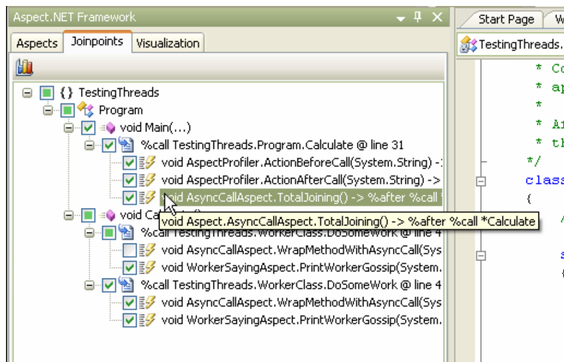

Figure 11. Join points tree

Due to the join points tree, the user can get full information on possible effect of weaving, and visually check the correctness of possible weaving into each join point before the weaving is actually done, so that undesirable join points could be unselected.

So, as opposed to AJDT for Eclipse, in Aspect.NET the user can visualize and control the process of join points filtering. In AJDT, the user can affect the selection of join points only by changing pointcut definitions in AspectJ language, which is not so comfortable and promptly, since it requires recompilation.

“Blind” weaving on the basis of wildcards only (i.e., based on lexical level of the source code instead of its semantic level) can be very dangerous. For example, if the user of an AOP tool would like to insert some actions before and after updating some common global resource to be synchronized on, and expresses the pattern for seeking the operation that updates the resource just by the *Set** wildcard for the name of the method, the result of weaving could be also inserting the aspect’s actions before and after the calls of “harmless” methods like *SetColor*.

So, we do think our design decision and functionality for manual filtering join points could be beneficial, until an appropriate semantic level approach is invented for this purpose, which should be the matter of a further research.

8.3 Visualization of aspect weaving effect

In order to help user understand aspect weaving effect on the target application, a specific component of Aspect.NET Framework was developed - *aspect weaving visualizer* (see fig. 12).

Each aspect woven into the target application is indicated by its own color that can be reselected by the user. Visualization of each of the aspects can be turned on or off.

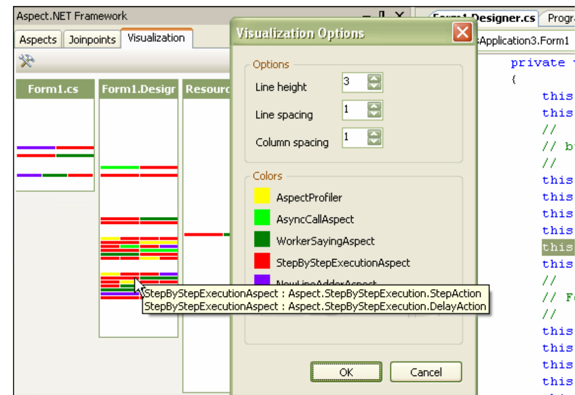

Figure 12. Aspect weaving visualizer

In Aspect.NET visualization is implemented similar to AJDT for Eclipse (see fig. 13) which, in its turn, inherited it from Aspect Browser [16].

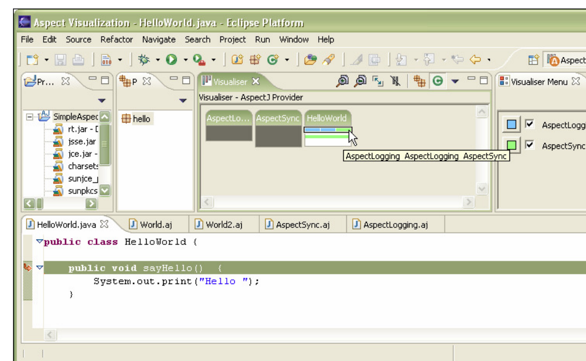

Figure 13. Aspect weaving visualizer in AJDT for Eclipse.

In accordance to the Seesoft graphic notation [30], each of the vertical columns represents one of the source files of the application. The height of the column is proportional to the size of the file. Colored marks inside the columns correspond to the join points where aspects are woven. Each mark corresponds to one action of an aspect. One horizontal line with one or more marks corresponds to a line in the source code. When clicking at any of the marks, a popup window is displayed with comments to the corresponding action of an aspect. By clicking at a horizontal line, the user can view the corresponding source code lines in the common use editor of the source code. Filtering join points with the help of join points tree is synchronized with the functionality of visualizing the effect of aspect weaving.

9. FUTURE WORK

9.1 Aspect debugger

In near future, we plan to add to the common use Visual Studio debugger an add-in for full-fledged debugging of Aspect.NET applications in terms of aspects. Due to that, the user will be able not only detect bugs in the aspect code, but to also trace and watch step by step the behavior of the resulting target application in terms of aspects.

9.2 Display changes in the source code

As our research shows, the main difficulty of applying AOP for commercial projects is the impossibility to estimate interaction of the woven aspects and business logic code. Aspects are woven at compile time or dynamically, and the result is a ready-to-use binary assembly. It is currently not possible to predict the behavior of the target application after aspect weaving. Aspect.NET can help to solve this task, either by aspect coloring in terms of the source code, or by creating unit tests by Visual Studio development environment. We also plan to provide the user with an opportunity to "finger" how his source code has been changed after aspect weaving, by visualizing appropriate fragments of decompiled code of the modified target assembly.

9.3 Interactive generator of weaving rules

Logical enhancement of the idea of weaving rules wizard could be a functionality to support generation of weaving rules in interactive mode, based on the existing code of the target application, for example, by clicking at the points of the source code of the target application to be affected by the aspect weaving rule being designed. This task requires a separate research.

9.4 Refactoring

In Visual Studio.NET 2005, advanced code refactoring functionality is supported - automated renaming members of an application, extracting interfaces from classes, transforming fragments of code into separate methods, etc. For more enhanced support of Aspect.NET, this set of refactoring transformations could be extended by actions like "transform a method to an aspect's action", or "convert a data definition in an application into an inter-type declaration". Supporting these two functions would be actually equivalent to a basic built-in aspectizer [9].

9.5 Aspects repository and aspect knowledge

To enable enterprise or higher level reuse of aspects, an aspect repository could be created and maintained. Aspect.NET Framework could perform searching in this repository, based on the problem domain and other parameters. When finding a suitable aspect, Aspect.NET Framework could weave it into the target project.

In longer perspective, we think a separate research could be helpful to investigate more formal and semantic-level representation, extraction and use of aspect knowledge, since in our viewpoint aspects can be regarded as special kind of knowledge on how to transform, enhance and maintain software projects and applications.

11. CONCLUSIONS

The growth of popularity of Microsoft .NET among software developers stimulates development of AOP tools for that platform. But the "single language" approach to AOP, i.e. implementing AOP features as extensions to some concrete language, may dramatically limit their applicability, and their integration to common use .NET software development tools and technologies. Other shortcomings of the single-language approach are lack of tools for visualizing the results of aspect weaving, and low performance of the resulting target applications.

The goal of our group is further developing of Aspect.NET which we hope is an adequate AOP tool for Microsoft.NET. Due to our general and simple approach, it provides comfortable mechanism for ubiquitous use of AOP as part of one of the most advanced software development environments – Visual Studio.NET. The proposed approach is based on AOP custom attributes, on static aspect weaving at .NET assembly level, and on using the source code of target projects to visualize the results of weaving. The proposed simple, expressive and powerful AOP meta-language enables language-agnostic AOP for the .NET platform.

Aspect.NET Framework, the user-oriented part of our system, provides a rich set of features to analyze and understand aspects and target applications subject to weaving. We think the functionality of Aspect.NET Framework is approaching to that of the most advanced AOP tool - AspectJ Development Tools for Eclipse [36], and has no analogs for Microsoft.NET platform.

The first working prototypes of Aspect.NET versions for Visual Studio.NET 2005 and for Rotor, with the Aspect.NET articles and examples, are available at [42]. The pre-requisites of using Aspect.NET are to install Visual Studio.NET 2005 and Phoenix.

12. REFERENCES

[1] G. Kiczales, J. Lamping, A. Mendhekar, etc. Aspect-oriented programming.- In: Proceedings of the European Conference on Object-Oriented Programming (ECOOP). Finland, Springer-Verlag LNCS 1241. June 1997

[2] The LOOM .NET Project: ttp://www.dcl.hpi.uni-potsdam.de/research/loom/

[3] M. Blackstock.. Aspect Weaving with C# and .NET. http://www.cs.ubc.ca/~michael/publications/AOPNET5.pdf

[4] Y. Xiong, F. Wan. CCC: An Aspect Oriented Intermediate Language on .NET Platform http://www.fit.ac.jp/~zhao/waosd2004/pdf/Xiong.pdf

[5] Aspect# home page: http://aspectsharp.sourceforge.net/

[6] AOP.NET home page: http://sourceforge.net/projects/aopnet/

[7] Weave.NET: www.dsg.cs.tcd.ie/sites/Weave.NET.html

[8] The AspectJ Project, www.aspectj.org

[9] V.O.Safonov. Aspect.NET: a new approach to aspect-oriented programming. - .NET Developer's Journal, 2003, #4.

[10] V. O. Safonov. Aspect.NET: concepts and architecture. - .NET Developer's Journal, 2004, # 10.

[11] Microsoft Phoenix home page. http://research.microsoft.com/phoenix

[12] K.Leiberherr. Component Enhancement: An Adaptive Reusability Mechanism for Groups of

Collaborating Classes. – In: Information Processing'92, 12th World Computer Congress, Madrid, Spain, J. van Leeuwen (Ed.), Elsevier, 1992, pp.179-185

[13] Krzysztof Czarnecki, Ulrich Eisenecker Generative Programming: Methods, Tools, and Applications, Addison-Wesley, Paperback, Published June 2000.

[14] Masuhara, J., Kichales, G. Modeling Crosscutting in Aspect-Oriented Mechanisms. Proceedings of ECOOP'2003

[15] Hannemann, J., Kichales, G. Overcoming the Prevalent Decomposition in Legacy Code. Proceedings of Workshop on Advanced Separation of Concerns, International Conference on Software Engineering (May 2001, Toronto, Canada)

[16] Aspect Browser: Bill Griswold's Web pages (University of California, San Diego): www.cs.ucsd.edu/users/wgg

[17] FEAT: Martin Robillard's and Gal Murphy's Web pages (University of British Columbia, Canada): www.cs.ubc.ca/~mrobilla/feat/index.html

[18] Shukla, D., Fill, S. and Sells, D. Aspect-Oriented Programming Enables Better Code Encapsulation and Reuse. MSDN Magazine, March 2002.

[19] Aspect-oriented software development Web site: www.aosd.net

[20] K.Czarnecki. Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models. PhD thesis, Technische Universitat Ilmenau, Germany, 1998.

[21] Homepage of the Subject-Oriented Programming Project, IBM Thomas J. Watson Research Center, YorktownHeights, New York, http://www.research.ibm.com/sop/

[22] Homepage of the TRESE Project, University of Twente, The Netherlands, http://wwwtrese.cs.utwente.nl/; also see the online tutorial on Composition Filters at http://wwwtrese.cs.utwente.nl/sina/cfom/

[23] Ch. Simony. The Death of Computer Languages, The Birth of Intentional Programming, Microsoft Research, 1995, http://research.microsoft.com/pubs/view.aspx?tr_id=4.

[24] G. Kiczales, J. Lamping, A. Mendhekar, etc. Aspect-oriented programming. *P*ublished in proceedings of the European Conference on Object-Oriented Programming (ECOOP). Finland, Springer-Verlag LNCS 1241. June 1997.

[25] E. Zhuravlev, V. Kiryanchikov. On the opportunity of dynamic aspects integration in aspect-oriented programming. – Proc. of Electro-Technical University, Informatics, control and computing technologies, 2002, vol, 3, pp.. 81 — 86 (in Russian)

[26] The AspectJ Programming Guide, 1998-2002, Xerox Corporation

[27] J. Hannemann, G. Kiczales. Design pattern implementations in Java and AspectJ. In: OOPSLA 02, New York, USA, November 2002. P. 161 — 173.

[28] M. Lippert, C Videira Lopes. A Study on Exception Detection and Handling Using Aspect-Oriented Programming. Xerox PARC Technical Report P9910229 CSL-99-1, Dec. 99

[29] B. Meyer, Applying Design by Contract, Prentice Hall, 1992

[30] Eick, S.G., J.L. Steffen, and E.E. Sumner, Seesoft – A Tool For Visualizing Line Oriented Software Statistics. IEEE Transactions on Software Engineering, 1992. 18 (11).

[31] Aspect-Oriented Programming with AJDT, Andy Clement, Adrian Colyer, Mik Kersten http://www.comp.lancs.ac.uk/computing/users/chitchya/AAOS2003/Assets/clemas_colyer_kersten.pdf

[32] HyperJ: www.research.ibm.com/hyperspace/HyperJ/HyperJ.htm

[33] AJDE for Emacs and JDEE: http://aspectj4emacs.sourceforge.net/

[34] AJDE for SunONE/NetBeans: http://aspectj4netbean.sourceforge.net/

[35] AJDE for JBuilder: http://aspectj4jbuildr.sourceforge.net/

[36] Eclipse AspectJ Development Tools project: http://www.eclipse.org/ajdt

[37] Eclipse.org - Main Page: http://www.eclipse.org

[38] AspectDNG: http://sourceforge.net/projects/aspectdng/

[39] PostSharp: http://gael.fraiteur.net/postsharp.aspx

[40] EOS: http://www.cs.virginia.edu/~eos

[41] RAIL: http://rail.dei.uc.pt

[42] Aspect.NET: http://www.msdnaa.net/curriculum/?id=6219

[43] SourceWeave.NET : http://www.dsg.cs.tcd.ie/index.php?category_id=438

[44] Wicca: http://www1.cs.columbia.edu/~eaddy/wicca/

[45] Compose*: http://composestar.sf.net/

[46] Phx.Morph: http://www.columbia.edu/~me133

[47] AOP goes .NET Community Site http://janus.cs.utwente.nl:8000/twiki/bin/view/AOSDNET/CharacterizationOfExistingApproaches

APPENDIX

A. ASPECT DEFINITION SAMPLE

```
//aspect header, contains aspect name
%aspect Politeness
using System;
using AspectDotNet;

public class Politeness
{
//aspect modules
%modules
   public static void SayHello ()
   {
      Console.WriteLine("Hello");
   }
   public static void SayBye ()
   {
      Console.WriteLine("Bye");
   }
//aspect rules and actions
%rules
   %before %call *
   %action public static void SayHelloAction() { Politeness.SayHello();}
   %after %call *
   %action public static void SayByeAction() { Politeness.SayBye();}
}
```

B. CONVERTED ASPECT SAMPLE

```
namespace Aspect {
   using System;
   using AspectDotNet;

   [AspectDef("Politeness", "MainModule", "")]
   public class Politeness {

      [AspectDef("Politeness", "module", "")]
      public static void SayHello() {
         Console.WriteLine("Hello");

      }

      [AspectDef("Politeness", "module", "")]
      public static void SayBye() {
         Console.WriteLine("Bye");

      }

      [AspectDef("Politeness", "action", "%before %call * ")]
      public static void SayHelloAction() {
         Politeness.SayHello();
      }

      [AspectDef("Politeness", "action", "%after %call * ")]
      public static void SayByeAction() {
         Politeness.SayBye();
      }
   }
}
```