инженерия программного обеспечения

Сафонов Владимир Олегович

ASPECT.NET – ИНСТРУМЕНТ АСПЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ ДЛЯ РАЗРАБОТКИ НАДЕЖНЫХ И БЕЗОПАСНЫХ ПРОГРАММ

ВВЕДЕНИЕ

Аспектно-ориентированное программирование (АОП) [1, 2] – новый перспективный подход к разработке программ. Суть данного подхода – поддержка разработки и модификации *сквозной функциональности (cross-cutting concerns)* в больших программных системах.

При проектировании, реализации и модификации любой программы ее архитектура, как правило, описывается в виде *иерархии модулей* – классов, процедур, функций, реализующих различные функциональные возможности (*функциональности*) программы. Простейшая из таких возможностей, например, вычисление какой-либо математической функции по известной формуле, может быть реализована всего одним модулем в классическом смысле этого слова – функцией, процедурой, статическим методом, макросом, имеющим, по терминологии Г. Майерса [3], *функциональную прочность*. Более сложная по семантике функциональность реализуется в виде *иерархии классов, библиотеки функций* и др. Такова, например, любая компонента большой программной системы, реализующая часть ее *бизнес-логики*, то есть решающая конкретную задачу из прикладной области, например, расчет зарплаты, расчет курсов акций, планирование распределения заданий между сотрудниками.

Такого рода функциональность в программе, реализация которой выразима в виде одного модуля или взаимосвязанной совокупности модулей, будем называть *модульной функциональностью (modular concern)*.

Как показывает анализ архитектуры программных систем, существует также *сквозная функциональность*. Данное понятие объединяет такие виды функциональных возможностей, идей, принципов, методов, реализация которых принципиально не осуществима в виде лишь одной иерархии взаимосвязанных модулей, а требует, в дополнение к этому, вставки в физически рассредоточенные точки программы фрагментов нового кода (либо исполнения новых фрагментов кода в рассредоточенных точках целевой программы без их явной вставки). Как правило, фрагменты кода, реализующие новую функциональность, являются исполняемыми конструкциями (операторами), чаще всего – вызовами, однако они могут быть и описаниями (определениями) данных, также являющихся частью реализации данной сквозной функциональности.

Типичные классы проблем, решение которых требует сквозной функциональности, относятся к *надежному и безопасному программированию (trustworthy computing)* [4, 5]:

– *безопасность (security)* – аутентификация пользователей и программ; авторизация (проверка полномочий для выполнения

тех или иных действий); криптографические операции над данными с целью обеспечения их конфиденциальности и т. д.;

– *надежность (reliability)* – проверка выполнения пред- и постусловий в модулях и инвариантов в классах; обработка ошибок и др.;

– *безопасность многопоточного выполнения кода (multi-threaded safety)* – синхронизация по ресурсам или по событиям, выделение критических участков кода, взаимное исключение доступа к ним и др.;

– *протоколирование и профилирование работы программы (logging and profiling)* – трассировка начала и окончания выполнения каждой функции (каждого метода), вывод их аргументов и результатов, сбор и вывод статистической информации об исполнении различных фрагментов программы и т. д.

Для решения подобных задач – в особенности, в характерных ситуациях, когда система в целом уже разработана, и ставится задача улучшения ее безопасности и надежности, – необходимы весьма трудоемкие и небезопасные, с точки зрения вероятности внесения ошибок, групповые вставки и модификации кода, обеспечивающие добавление в программу требуемых функциональных возможностей. Например, вставка вызова операции закрытия семафора перед любой операцией, изменяющей некоторый глобальный ресурс, и вставка вызова парной ей операции открытия семафора после каждого изменения данного глобального ресурса.

...необходимы весьма трудоемкие и небезопасные, с точки зрения вероятности внесения ошибок, групповые вставки и модификации...

Выполнение данной нетривиальной модификации существующей программной системы требует глобального анализа всего исходного кода программы, размер которого может составлять несколько десятков тысяч или даже несколько миллионов строк. Если подобные действия выполняются вручную, с использованием возможностей применяемой при разработке универсальной интегрированной среды, например, Visual Studio.NET или Eclipse, но без применения каких-либо специализированных технологий и инструментов, то весьма велик риск внесения в программу ошибок. Например, разработчик может вставить в код лишь вызов операции закрытия семафора, но по каким-либо причинам забыть вставить парную ей «синхронизирующую скобку» – вызов операции открытия семафора после модификации ресурса, что может привести к тупику (deadlock) при исполнении программы, который впоследствии весьма сложно обнаружить и устранить.

Аспектно-ориентированное программирование (АОП) – один из новых подходов к программированию, предназначенный для *модуляризации сквозной функциональности* и ее автоматизированного, безопасного и надежного *добавления* в целевую программу, а также *поиска (локализации)* и *модификации* в целевой программе некоторой уже реализованной сквозной функциональности.

АОП расширяет традиционную концепцию модуля по Г. Майерсу [3] понятием *аспекта (aspect)*. *Аспект* – это реализация сквозной функциональности, выполненная в виде специального модуля, содержащего фрагменты кода (*действия* аспекта), активируемые в заданных точках целевой программы как часть новой функциональности. Определение аспекта содержит также спецификацию *разреза* (*pointcut*) кода целе-

вой программы – совокупности *правил поиска* в ней *точек присоединения (join points)* для последующего *внедрения* (*weaving*) в них действий аспекта (слово *weave* буквально означает *ткать, плести*). Внедрение аспекта – способ его использования как новой разновидности модуля, в отличие от традиционных модулей «по Майерсу», используемых путем их явного вызова.

Основатель АОП – Г. Кичалес (G. Kiczales), Университет Британской Колумбии, Канада [2], а наиболее популярный инструмент АОП для платформы Java – разработанная под его руководством система AspectJ [6], первая версия которой была создана в 1995 г. в фирме Xerox Palo Alto Research Corporation, США. Подробный обзор подходов к АОП и инструментов АОП приведен в работе [5].

Для российских специалистов особенно важен тот факт, как мне довелось убедиться, совершенно не известный современным основоположникам АОП в США и Канаде, что подход к разработке программ, близкий к АОП, был предложен еще в конце 1970-х годов проф. А.Л. Фуксманом (Ростовский университет) [7] под названием *технология рассредоточенных действий,* сокращенно – *РД-технология* (другое название, использованное автором подхода, – *технология вертикального слоения*). Согласно данной технологии, *вертикальный слой (срез)* – совокупность *рассредоточенных действий,* фрагментов кода, реализующих некоторую *расширяющую функцию*, а процесс разработки и модификации программы представляет собой последовательность операций добавления или изменения расширяющих функций.

Даже при беглом изложении основных идей РД-технологии очевидно ее сходство с АОП. Более того, даже в самом названии монографии [7] присутствует слово *аспект*. К сожалению, РД-технология не получила своего коммерческого воплощения, но осталась в истории отечественного и мирового программирования пионерской идеей, преемственная связь которой с АОП несомненна.

В настоящее время АОП все шире распространяется среди программистов, как в академической среде, так и в коммерческих фирмах, во многом благодаря энтузиазму и активности разработчиков AspectJ и его значительному влиянию на развитие и применение инноваций в области инженерии программ. Ежегодно проводится конференция Aspect-Oriented Software Development [8]. Обучающие курсы (tutorials) и исследовательские доклады по АОП неизменно входят в программу широко известных в мире конференций в области ИТ, например, OOPSLA. Имеются сотни инструментов АОП для платформ Java и .NET, расширения средствами АОП языков программирования JavaScript, PHP и многих других [1].

В России АОП пока недостаточно распространено, несмотря на то, что его концепции известны уже более 12 лет. Созданы русскоязычные страницы Web-энциклопедии Wikipedia и краткий wiki-учебник по АОП на русском языке [9]. Литература по АОП на русском языке, как правило, ограничивается переводами и обзорами зарубежных работ [10]. Но уже появляются исследования молодых авторов, посвященные применению известных инструментов АОП (например, JBoss [11]) в различных областях, например, для разработки графических пользовательских интерфейсов [12]. Отечественная монография [13], посвященная методам расширения программ, определяет концепцию *рассредоточенного набора*, весьма близкую идеям технологии вертикального слоения и АОП.

Хотелось бы выразить надежду, что статья послужит более широкому распространению АОП в России, а также уточнению русскоязычной терминологии по АОП, насколько это возможно в настоящий момент: вследствие разнообразия подходов к АОП, даже англоязычную терминологию в столь новой области, как АОП, нельзя считать устоявшейся.

Aspect.NET [5, 14–21] – инструментарий АОП для платформы .NET – разработан в Санкт-Петербургском государственном университете коллективом в составе: *проф. В.О. Сафонов* – научный руководитель и главный архитектор проекта; *аспирант Д.А. Григорьев* – автор *подсистемы*

внедрения аспектов (*weaver*); *аспирант М.К. Грачев* – автор Aspect.NET Framework, подсистемы графического пользовательского интерфейса; *аспирант А.И. Масленников* – автор конвертора метаязыка АОП в язык C#.

Проект Aspect.NET поддержан Microsoft Research. На данный момент выпущены четыре версии Aspect.NET; текущая версия имеет номер 2.1 [19]. Система Aspect.NET и реализованный в ней подход к АОП получили широкую известность в мире. Система Aspect.NET свободно распространяется как результат исследовательского проекта через академический сайт Microsoft [19] и имеет пользователей в 19 странах: Австралии, Аргентине, Бразилии, Великобритании, Германии, Дании, Египте, Индии, Индонезии, Испании, Италии, Канаде, Казахстане, Колумбии, Корее, Нидерландах, России, США, Тайване. Дистрибутив системы [19] содержит подробное руководство пользователя и демонстрационные примеры.

Системе Aspect.NET и ее применению для разработки надежных и безопасных программ посвящена монография [5] на английском языке, которая будет издана в июне 2008 г. в США издательством John Wiley & Sons.

Принципы и возможности Aspect.NET описаны в серии статей [14–16] в журнале «.NET Developer's Journal» (США) по программным средствам и технологиям для платформы .NET.

...в виде специализированных модулей – аспектов, с целью ее ... внедрения в программу...

В силу исторических особенностей развития системы Aspect.NET, большинство публикаций по ней – на английском языке. Имеются также две русскоязычных публикации [20, 21] – тезисы докладов на конкурсе-конференции «Технологии Microsoft в теории и практике программирования», на которой проект Aspect.NET в 2006 и 2007 гг. занял первое место.

1. ОСНОВНЫЕ КОНЦЕПЦИИ АОП

Согласно определению АОП в Wikipedia [22], «программные парадигмы *аспектно-ориентированного программирования* (АОП) и *аспектно-ориентированной разработки программ* (АОРП) предназначены для *разделения функциональностей (separation of concerns)*, прежде всего – *сквозных функциональностей (cross-cutting concerns)*, и являются расширением концепций *модуляризации*. АОП обеспечивает это преимущественно путем реализации расширений используемого языка программирования; АОРП использует сочетание языка, окружения и метода».

Авторы статьи по АОП в Wikipedia противопоставляют концепции *АОП* и *АОРП*. В частности, если буквально следовать классификации, предложенной в [22], то и AspectJ, и Aspect.NET относятся к инструментам АОРП, а не АОП. Однако в данной статье будем использовать единый, более распространенный термин *АОП*, чтобы не усложнять терминологию и обеспечить более общую точку зрения на данный круг проблем. На мой взгляд, разделение на АОП и АОРП не вполне целесообразно – на данном этапе развития АОП объединение идей и усилий различных школ более важно, чем их противопоставление.

Таким образом, *аспектно-ориентированное программирование (aspect-oriented programming)* – это технология разработки *сквозной функциональности (cross-cutting concern)* в виде специализированных модулей – *аспектов (aspects)*, с целью ее последующего *внедрения* (*weaving*) в программу путем активизации фрагментов кода аспекта – *действий (advices)* в выделенных ин-

струментом АОП в целевой программе *точках присоединения* (*join points*). Спецификация точек присоединения, задаваемая в определении аспекта либо в виде отдельного модуля, на который могут ссылаться различные аспекты, получила название *разрез* (*pointcut*).

По сути дела, аспект можно представлять себе как совокупность *правил* вида *Условие* → *Действие*, где *Условие* специфицирует разрез целевой программы, а *Действие* задает фрагмент кода, активизируемый в целевой программе при выполнении данного условия.

Помимо основной задачи реализации и внедрения новой сквозной функциональности, АОП решает две других не менее важных задачи:

– *аспектизация* (*aspectizing* [14], *aspect-oriented refactoring* [23], *aspect mining* [24]) – выделение аспектов из не аспектно-ориентированных программ с целью их последующего многократного использования;

– *модификация сквозной функциональности*, представленной определением аспекта и всеми его внедрениями.

Спецификация (определение) аспекта в АОП может быть выполнена на специальном *метаязыке АОП*, как в системе Aspect.NET, либо в терминах *расширений базового языка программирования конструкциями АОП*, как в системе AspectJ, которая по существу является реализацией аспектно-ориентированного расширения языка Java.

При использовании метаязыка АОП возникает проблема его отображения в базовый язык реализации, которая в Aspect.NET решается путем конвертирования конструкций метаязыка Aspect.NET.ML в определения введенных нами специализированных атрибутов АОП (custom attributes).

При втором подходе возникает зависимость инструмента АОП от базового языка и всех его изменений. Например, код на языке AspectJ требует для компиляции специализированный компилятор *ajc,* входящий в состав системы AspectJ, и при любом изменении базового языка Java приходится вносить соответствующие изменения в компилятор *ajc*.

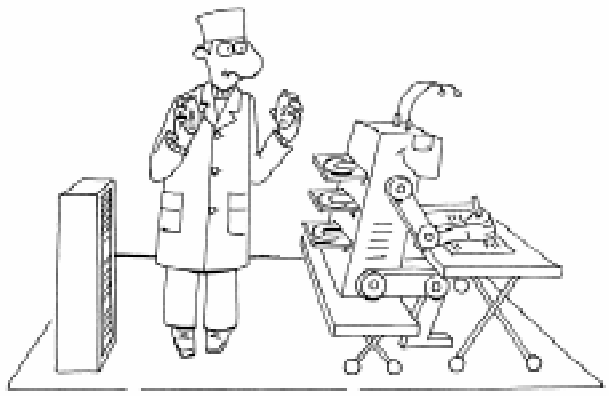
Но самое существенное, с концептуальной точки зрения, заключается в том, что при расширении базового языка средствами АОП традиционные для ООП понятия, конструкции и взаимосвязи (классы, поля, объекты, наследование) «сосуществуют» в одном расширенном языке с гораздо более общими по своей природе метапонятиями АОП (аспект, разрез, действие аспекта, внедрение, точка присоединения). При этом у пользователей возникает целый ряд проблем, которые можно охарактеризовать как «проблемы концептуальной путаницы»: могут ли аспекты наследоваться от классов, классы – от аспектов, как при этом изменяется семантика программы и т. д. Все это лишь затрудняет использование АОП, а не облегчает его, так как и без того непростой базовый язык реализации оказывается и вовсе запутанным, что может вообще оттолкнуть пользователей от освоения и использования АОП.

На мой взгляд, АОП и его концепции относятся к мета-уровню *методологии программирования, спецификации программ и их трансформаций*, более высокому, чем уровень конкретных известных программных парадигм – процедурное, объектно-ориентированное, функциональное программирование и др. АОП решает в общем виде задачи трансформации программ с целью

...при расширении базового языка ... традиционные для ООП понятия, конструкции и взаимосвязи... «сосуществуют» в одном расширенном языке с гораздо более общими... метапонятиями АОП...

добавления или модификации в них сквозной функциональности, а также спецификации таких трансформаций. При этом методы и подходы, характерные для АОП, на мой взгляд, практически не зависят от концепций базового языка реализации. Хотя при описании разрезов и правил внедрения и используются понятия базового языка, но по своей природе и семантике суть преобразований, выполняемых в АОП, – например, вставка перед всеми вызовами модуля (в его классическом смысле) с заданным именем некоторого нового кода – не зависит от базового языка реализации и воплощенной в нем парадигмы. Для процедурных языков Фортран, Паскаль и Си, либо для объектно-ориентированных языков C++, Java или C# суть АОП-преобразований программ аналогична; концепции и механизмы АОП ортогональны используемым в языках парадигмам. Поэтому в равной степени имеет смысл говорить о применении АОП к языкам процедурного, объектно-ориентированного или функционального программирования, хотя, разумеется, данный вопрос нуждается в тщательном исследовании и адаптации методов АОП для не объектно-ориентированных языков, поскольку на данный момент все известные инструменты АОП привязаны к системам объектно-ориентированного программирования.

С данной точки зрения, весьма важна также разработка методов *формальной спецификации аспектов и их формальной верификации* [5], которая в настоящий момент далека от своего решения.



Спецификацию аспекта можно рассматривать как представление знаний о методе решения задачи...

Следует также отметить *концептуальную связь АОП и инженерии знаний*. Спецификацию аспекта можно рассматривать как представление знаний о методе решения задачи и о методе преобразования целевой программы с целью расширения ее функциональности решением рассматриваемой задачи, описанным в виде аспекта. Даже сама форма определения аспекта в любой системе поддержки АОП – совокупность правил вида *Условие* → *Действие* – напоминает *продукционную систему*, где набор продукций (знаний) содержит как знания о требуемых преобразованиях целевой программы (*Условие*), так и знания о способе решения задачи аспектом (*Действие*). С данной точки зрения, представляет несомненный интерес исследование и представление *аспектно-ориентированных знаний*, которое только начинается. В частности, в нашей группе ведутся исследования по интеграции системы Aspect.NET с другой нашей системой – Knowledge.NET [25], инструментарием представления знаний для платформы .NET, содержащим средства представления и использования онтологий, фреймов и наборов правил.

С точки зрения семантики и реализации, существующие инструменты АОП [1] различаются своими *моделями внедрения (join point models)* – способами реализации аспектов и их внедрения. Основное различие имеет место между *статическим* и *динамическим внедрением*.

Инструменты АОП со *статическим внедрением* осуществляют преобразования целевой программы либо на уровне *исходного кода*, либо на уровне *промежуточного кода* (например, байт-кода Java, универсального промежуточного кода CIL платформы .NET или какого-либо собственного внутреннего представления, используемого только данным инструментом АОП), либо, наконец, на уровне *платформно-зависимого объектного кода (native code)*. Используется также подход, основанный на внедрении аспекта на этапе *just-in-time компиляции,* то есть в процессе динамической компиляции

метода из промежуточного в платформно-независимый объектный код при первом его вызове. В каждой из этих моделей внедрения, в том или ином смысле выполняется вставка кода аспекта в целевую программу. Преимущество – более высокая эффективность результирующего кода, недостаток – увеличение его объема, что не всегда приемлемо, например, для мобильных устройств с ограниченными ресурсами.

При *динамическом внедрении* инструмент АОП ведет себя подобно отладчику, в котором заданы контрольные точки. Для активизации аспекта перед исполнением каждого оператора целевой программы инструмент АОП проверяет выполнимость каждого из условий, заданных в аспекте, и при истинности какого-либо из условий выполняет соответствующее ему действие аспекта. Очевидно, что подобный подход удобен прежде всего для отладки программы, но совершенно неприемлем при ее эксплуатации, с точки зрения эффективности.

2. ПРИНЦИПЫ, ВОЗМОЖНОСТИ И ПЕРСПЕКТИВЫ ASPECT.NET

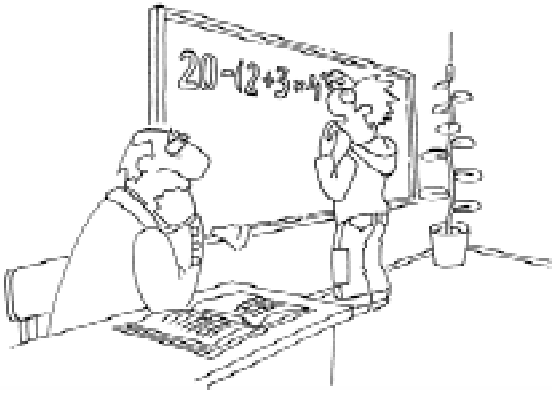
Мое знакомство с АОП началось в 2001 г. с номера журнала Communications of the ACM [26], посвященному АОП. Содержание его статей позволило мне неожиданно для себя увериться в том, что фактически я был сторонником АОП, развивал и использовал аналогичные принципы разработки программ в течение всей своей профессиональной деятельности.

В частности, цели ТИП-технологии [27], разработанной в 1980-х гг. и использованной нашим коллективом в течение ряда лет для разработки компиляторов и экспертных систем, состояли, выражаясь современным языком, в поддержке *шаблонов проектирования* и реализации операций над сложными структурами данных в виде предопределенного *набора уровней абстракции (уровень представления, уровень определения, концептуальный уровень)* и *вертикальных срезов (интерфейс генерации/удаления, интерфейс доступа, интерфейс модификации, интерфейс вывода),* реализуемых в виде информационно прочного [3] многовходового модуля – *технологического инструментального пакета (ТИП)*. Использование (внедрение) ТИП путем вызова его абстрактных операций, в отличие от внедрения аспекта в АОП, не было автоматизировано, не опиралось на какие-либо инструменты, отсутствовал метаязык определения разрезов и правил внедрения ТИП, однако элементами технологической дисциплины ТИП-технологии были семантические рекомендации по использованию операций ТИП [27] и мнемонические обозначения точек использования (присоединения) ТИП в виде комментариев, например:

```
%List% p := Cons (1, Nil);
```

это облегчало локализацию с помощью редактора или текстовых фильтров в исходном коде программы всех фрагментов (действий) ТИП. Таким образом, ТИП-технология [27], как и технология вертикального слоения [7], была шагом вперед на пути к АОП.

Как известно, значительную трудность для большинства программистов при сопровождении больших программных систем представляют *выделение и модификация сквозной функциональности*. Для исправления ошибки или недоработки в программе (например, отсутствие синхронизирующих скобок при модификации общего ресурса) программисту приходится, используя сред-



Для исправления ошибки или недоработки в программе (например, отсутствие синхронизирующих скобок при модификации общего ресурса)...

ства универсальной интегрированной среды, во-первых, найти в программе все вызовы операции изменения ресурса, например *SetResource (NewValue)*, во-вторых, заключить все эти вызовы в синхронизирующие скобки, например, вставить вызов операции *Wait (Semaphore)* перед каждым вызовом *SetResource*, а вызов операции *Signal (Semaphore)* – после каждого вызова *SetResource*. Указанные групповые модификации кода выполняются средствами текстового редактора интегрированной среды. При этом в лучшем случае в среде автоматизирован поиск всех вызовов заданного метода (например, в виде одной из опций функции *refactoring* [28]), либо и такая возможность отсутствует, и для поиска приходится использовать сторонние утилиты типа *grep*. Для надежности и безопасности вставки кода синхронизирующих скобок универсальная среда разработки не обеспечивает *никакой* поддержки, кроме базовых возможностей редактора (*Copy / Paste*). Таким образом, ситуация, когда программист вставил в код только вызов *Wait*, но забыл вставить вызов *Signal*, не контролируется технологией разработки.

Рассмотрим, как описанная в п. 1 задача решается в АОП, в системе Aspect.NET с помощью определения и внедрения простого аспекта:

```
%aspect SemaphoreBrackets
public class SemaphoreBrackets {
  %before %call *.SetResource
  %action
    public static void WaitAction ()
    { Wait (Semaphore); }
  %after %call *.SetResource
  %action
    public static void SignalAction ()
  { Signal (Semaphore); }
} // SemaphoreBrackets
```

Аспект **SemaphoreBrackets** состоит из двух действий – **WaitAction** и **SignalAction**, статических методов, вызовы которых, согласно правилам внедрения аспекта, вставляются в код целевой программы соответственно перед каждым вызовом и после каждого вызова метода **SetResource**.

Строки исходного кода аспекта, начинающиеся знаком процента (**%**), представляют собой *аспектно-ориентированные аннотации*, или, иначе говоря, конструкции *метаязыка АОП – Aspect.NET.ML*, являющегося частью нашей системы. Весь остальной код – это обычный синтаксически и семантически правильный код определения класса на C#. Конструкции метаязыка Aspect.NET.ML *конвертируются* в определения *специализированных атрибутов (custom attributes)* .NET на языке C#. Этап конвертирования является первым шагом процесса сборки проекта в Visual Studio. Например, код на метаязыке Aspect.NET.ML:

```
%before %call *.SetResource %action
```

будет конвертирован в следующий код на C#:

```
[AspectAction("%before %call
                    *.SetResource")]
```

– код, аннотирующий метод *SetResource* экземпляром специализированного атрибута АОП *AspectAction* (действие аспекта).

В действиях аспекта могут быть использованы конструкции для *анализа контекста точки присоединения*, что совершенно необходимо для реальных применений АОП. Например, вывод на консоль *имени метода в целевой программе*, перед вызовом которого вставлен вызов действия аспекта, выполняется следующим образом:

```
System.Console.WriteLine
  ("Target method name=" +
            %TargetMemberInfo.Name);
```

Аналогичным образом возможна *обработка («захват») аргументов и результатов целевого метода; объекта, к которому применяется метод в целевой программе; имени файла и номера строки точки присоединения аспекта в целевой программе и т. д.*

Пользователь Aspect.NET имеет возможность для разработки определения нового аспекта использовать любой из *двух видов проекта Visual Studio.NET*, которыми Aspect.NET расширяет стандартный набор типов проектов Visual Studio – *Aspect.NET.ML module* (код на метаязыке АОП) и *Aspect.NET module* – код на C#, в

котором специализированные атрибуты АОП можно задавать непосредственно и, таким образом, разрабатывать аспекты «в атрибутах», не используя метаязыка АОП вообще. Однако, в отличие от конструкций АОП в AspectJ, конструкции метаязыка Aspect.NET.ML очень просты, могут быть быстро освоены с помощью примеров и шаблонов кода, введенных в Visual Studio для проектов типа *Aspect.NET.ML module*.

Полученный в результате конвертирования исходный код на C# компилируется в универсальный промежуточный код CIL платформы .NET стандартными средствами Visual Studio. Бинарный код сборки (файл *переносимого кода, portable executable* платформы .NET) несет в себе информацию не только о коде класса, но и о принадлежности его фрагментов (действий, реализуемых методами) аспекту *SemaphoreBrackets*. Это не препятствует работе стандартных утилит .NET (например, отладчика), но позволяет системе Aspect.NET, используя атрибуты, «понимать» аспекты, представленные в виде двоичных кодов сборок. Использование атрибутов АОП более надежно, чем распространенная во многих других инструментах АОП спецификация аспектов в виде отдельных XML-файлов.

Вставка кода действий аспекта в Aspect.NET выполняется автоматически *подсистемой внедрения аспектов (weaver).* При этом гарантируется надежность и полнота выполнения данной групповой модификации: Aspect.NET не может «случайно забыть» выполнить вставку какого-либо действия аспекта. Подсистема внедрения аспектов вызывается через *интегрированную среду – Aspect.NET Framework*, являющуюся частью (*add-in*) универсальной интегрированной среды Visual Studio.NET 2005. Для обработки сборок .NET подсистема внедрения аспектов использует инструментарий Microsoft Phoenix [29] предназначенный для создания платформно-зависимых модулей (back-ends) компиляторов, пользователем которого в рамках программы Phoenix Academic Program [29], наша группа является с 2003 года.

Среда Aspect.NET Framework обеспечивает открытие аспектов, представленных двоичными сборками, запоминание открытых аспектов для последующих открытий той же целевой программы в Visual Studio, визуализацию аспектов, вызов подсистемы внедрения, визуализацию точек присоединения аспектов в целевой программе, а также пробный запуск одним нажатием кнопки полученного в результате внедрения аспектов двоичного кода сборки .NET.

Aspect.NET, единственный из известных на данный момент инструментов АОП, обеспечивает режим *внедрения аспектов, управляемых пользователем (user-controlled weaving*). Перед фактическим внедрением аспектов пользователь имеет возможность просмотреть все найденные Aspect.NET потенциальные точки присоединения аспекта в исходном коде целевой программы, проанализировать возможный эффект внедрения действий аспекта в каждую из этих точек, *отменить* (*deselect*) те из них, внедрение в которые нежелательно, и лишь затем выполнить внедрение аспекта. Таким образом, наша система позволяет избежать самой серьезной опасности, которую таит в себе АОП как мощный механизм групповых преобразований кода, – «слепого», бесконтрольного изменения поведения программы, последствия которого нетрудно предвидеть.

Вопреки распространенной точке зрения о неэффективности применения АОП, в отличие от «ручной» вставки аналогичных фрагментов кода, Aspect.NET *не ухудшает эффективность* результирующей программы [5]. Вставки или замены выполняются на уровне *бинарного кода сборки (assembly)*. При этом, как правило, кроме самих вызовов действий аспекта, в код целевой программы не вставляется никакого дополнительного «инструментовочного» кода, то есть результат внедрения аспекта в сборку ничем не отличается от результата ее аналогичной ручной модификации. Полученная сборка может использоваться всеми универсальными инструментами .NET – общей средой поддержки выполнения (CLR), отладчиком, профилировщиком (profiler) и т. д.

Более подробное описание и обоснование возможностей Aspect.NET приведено в монографии [5], статьях [14–16] и в руководстве пользователя [19]. Удобство и возможности Aspect.NET, простота использования системы, в отличие от AspectJ, JBoss и т. д., отмечены экспертами многих стран мира; Aspect.NET посвящены статьи и диссертации [5].

ЗАКЛЮЧЕНИЕ

Перспективы Aspect.NET – все более широкое использование нашей системы, ее адаптация для задач пользователей, разработка *библиотек (repositories) аспектов* широкого круга проблемных областей; развитие самой системы и реализация широкого круга новых, уже запланированных функциональных возможностей, например:

– реализация поддержки в Aspect.NET других основных языков .NET (Visual Basic, JScript, Managed C++, J#), заметим, что синтаксис и семантика метаязыка Aspect.NET.ML *не зависят от языка реализации* аспектов, что дает возможность использовать тот же метаязык для других языков реализации);

– реализация аспектизатора;

– интеграция Aspect.NET с Visual Studio.NET 2008;

– реализация дополнительных видов точек присоединения аспектов;

– реализация *сценариев (scripts)*, позволяющих, например, добавить в результате внедрения аспекта в целевую программу аннотации в виде атрибутов;

– реализация параметризованных аспектов (*generic aspects*), в том числе – аспектов, параметризуемых условиями внедрения [14].

Мы заинтересованы в контактах со специалистами и потенциальными заказчиками, осваивающими и использующими АОП и Aspect.NET Отзывы о нашей системе, пожелания и предложения просим присылать по электронной почте: *v_o_safonov@mail.ru*.

Литература

1. Web-сайт по аспектно-ориентированной разработке программ. http://aosd.net
2. *Kiczales G. et al.* Aspect-oriented programming. Xerox PARC, 1997.
3. *Майерс Г.* Надежность программного обеспечения. М.: Мир, 1980.
4. *Mundie C. et al.* Trustworthy Computing. Microsoft White Paper. October 2002. http://www.microsoft.com/mscorp/twc/default.mspx
5. *Safonov V.O.* Using aspect-oriented programming for trustworthy software development. Wiley Interscience. John Wiley & Sons, June 2008 (to be published). ISBN 978-0-470-13817-5
6. Web-сайт проекта AspectJ. http://www.aspectj.org
7. *Фуксман А.Л.* Технологические аспекты создания программных систем. М.: Статистика, 1979.
8. Web-страницы ежегодной конференции по аспектно-ориентированной разработке программ. http://aosd.net/conference
9. Русскоязычные Web-страницы и wiki-учебник на тему «Аспектно-ориентированное программирование». http://ru.wikipedia.org/wiki/
10. *Павлов В.* Анализ вариантов применения аспектно-ориентированного подхода при разработке программных систем. http://www.javable.com/columns/aop/workshop/01/
11. Web-сайт проекта JBoss AOP. http://labs.jboss.com/portal/jbossaop
12. *Михеев О.И.* Разработка сред пользовательских интерфейсов нового поколения с применением аспектно-ориентированного программирования / Дисс. на соиск. уч. степ. канд. техн. наук, СПб: Политехнический университет, 2007.
13. *Горбунов-Посадов М.М.* Расширяемые программы. М.: Полиптих, 1999.
14. *Safonov V.* Aspect.NET: a new approach to aspect-oriented programming. .NET Developer's Journal 2003; (4): 36–40.
15. *Safonov V.* Aspect.NET: concepts and architecture. .NET Developer's Journal 2004; (10): 44–48.
16. *Safonov V, Grigoryev D.* Aspect.NET: aspect-oriented programming for Microsoft.NET in practice. .NET Developer's Journal 2005; (7): 28–33.

17. *Safonov V. et al.* Aspect.NET – aspect-oriented toolkit for Microsoft.NET based on Phoenix and Whidbey. In: Knoop J, Skala V, editors. .NET Technologies 2006. University of West Bohemia. Campus Bory. Full papers proceedings; Pilsen, Czech Republic; 2006. P. 19–34.

18. *Safonov V , Grigoryev D.* Aspect.NET – an aspect-oriented programming tool for Microsoft.NET. In: 110th Anniversary of Radio Invention; St. Petersburg; 2006. P. 11–21.

19. Aspect.NET 2.1. http://www.msdnaa.net/curriculum/?id=6801

20. *Григорьев Д.А., Грачев М.К., Масленников А.И. Сафонов В.О.* Aspect.NET: инструментарий аспектно-ориентированного программирования для платформы Microsoft.NET // Тезисы докладов конкурса-конференции «Технологии Microsoft в теории и практике программирования». СПб.: изд-во СПбГПУ, 2006.

21. *Григорьев Д.А., Грачев М.К., Масленников А.И. Сафонов В.О.* Адаптация методологии АОП для практического применения на платформе Microsoft.NET // Тезисы докладов конкурса-конференции «Технологии Microsoft в теории и практике программирования». СПб.: изд-во СПбГПУ, 2007.

22. Статья по аспектно-ориентированному программированию в Wikipedia. http://en.wikipedia.org/wiki/Aspect-oriented_programming

23. *Binkley D. et al.* Automated refactoring of object-oriented code into aspects // Proceedings of the 21st IEEE International Conference onVolume, Issue , 26–29 Sept. 2005. P. 27–36.

24. *Hannemann J, Kiczales G.* Overcoming the prevalent decomposition of legacy code. In: Proceedings of Workshop on Advanced Separation of Concerns at the International Conference on Software Engineering; Toronto, Canada, 2001.

25. *Сафонов В.О. и др.* Интеграция методов инженерии знаний и инженерии программ. Система управления знаниями Knowledge.NET // Компьютерные инструменты в образовании, 2005, № 5.

26. *Elrad T. et al.* Discussing aspects of AOP. Communications of the ACM 2001; 44(10): 33–38.

27. *Сафонов В.О.* Языки и методы программирования в системе «Эльбрус». М.: Наука, 1989.

28. Web-сайт технологии «refactoring». http://www.refactoring.com/

29. Web-сайт Microsoft Phoenix. http://research.microsoft.com/phoenix

Сафонов Владимир Олегович,
доктор технических наук,
профессор кафедры информатики
СПбГУ, руководитель лаборатории
Java-технологии.