УДК 004.4'2

Григорьева Анастасия Викторовна

АСПЕКТНО-ОРИЕНТИРОВАННЫЙ РЕФАКТОРИНГ ОБЛАЧНЫХ ПРИЛОЖЕНИЙ MS AZURE С ПОМОЩЬЮ СИСТЕМЫ ASPECT.NET

Аннотация

Облачная платформа MS Azure предоставляет множество сервисов, которые позволяют разрабатывать производительные и надежные приложения. Управление этими сервисами производится из исходного кода облачного приложения, что приводит к проблеме «сквозной функциональности», когда реализация одной функции распределена по всему коду.

Данная работа предлагает подход, который позволяет устранить этот недостаток.

Ключевые слова: аспектно-ориентированное программирование, АОП-рефакторинг, унаследованный код, Aspect.NET.

ВВЕДЕНИЕ

Облачная платформа MS Azure [1] предоставляет множество сервисов, которые позволяют разрабатывать производительные и надежные приложения. К таким сервисам относятся СУБД MS SQL Azure, сбор диагностической информации, управление своей конфигурацией, разделяемый кэш и пр. В зависимости от фактической или предполагаемой нагрузки, приложение может увеличивать или уменьшать количество своих одновременно запущенных экземпляров (instances). При этом все сложности взаимодействия с аппаратной инфраструктурой скрыты от программиста. Сервисные компоненты Azure не зависят от вызывающего их контекста и, тем самым, предназначены для гибкого повторного использования в различном окружении.

Однако при этом возникает проблема «сквозной функциональности» (cross-cutting concerns), когда для реализации какой-либо функции с помощью сервисов Azure требуется вносить изменения во множество разных точек исходного кода. Если, например, в веб-роль (web-role) на основе ASP.NET MVC потребуется добавить кэширование результатов запроса некоторых данных из удаленного хранилища [2] и организовать управление кэшированием через веб-интерфейс, то программисту придется вносить изменения в следующие модули:

– каждый класс, ответственный за загрузку тех данных, которые необходимо кэшировать;

– модель (Model), на основе которой Вид (View) отобразит полученные данные и элементы управления кэшированием;

– контроллер (Controller), который инициализирует соответствующими настройками кэширования этот класс, применит его для заполнения данными модели и передаст ее для отображения в соответствующий Вид;

– Вид, который также должен содержать веб-разметку для элементов управления кэшированием.

Подобные ситуации возникают и в ряде других задач, таких как протоколирование, авторизация, проверка контрактов, профилирование и т.п. Сквозная функциональность затрудняет сопровождение проекта, ведь при смене способа решения задачи (например, авторизации), необходимо вне-

сти изменения во множество точек исходного кода. Если же на определенном этапе потребуется вообще удалить все, относящееся к определенной функции (например, профилированию), то исходный код снова поменяется на множестве уровней.

В данной работе предложена методика «сокрытия изменений» для облачных приложений MS Azure, которая позволяет выделить весь код, необходимый для реализации какой-либо функции, в отдельный модуль («аспект» в терминологии аспектно-ориентированного программирования, АОП). Кроме непосредственной реализации своей функции, аспект содержит правила, определяющие местоположение тех точек в целевом приложении, где необходимо использовать эту функцию. Затем аспект применяется к целевому приложению без фактического изменения исходного кода. Данный прием позволит сосредоточиться лишь на реализации бизнес-логики в целевом приложении. Слияние бинарных сборок аспектов и целевого приложения производится специальным инструментом – «компоновщиком» (weaver).

Применение этой методики иллюстрируется на АОП-рефакторинге обучающих примеров, созданных компанией Microsoft, для изучения различных сервисов MS Azure. Под АОП-рефакторингом понимается процесс аспектно-ориентированных преобразований программной системы, при котором не меняется внешнее поведение кода, но улучшается его внутренняя структура [3]. В рамках данной методики созданы повторно используемые аспекты для таких сервисов MS Azure (далее просто «служб»), как:

1. Протоколирование в диагностическую таблицу WAD.

2. Динамическое управление количеством одновременно запущенных экземпляров в зависимости от нагрузки.

3. Интеграция в целевой исходный код сервиса протоколирования IOC-контейнера MS Enterprise Library [4] .

4. Кэширование запросов к MS SQL Azure с помощью Microsoft.ApplicationServer.Caching.DataCacheFactory.

Влияние предложенной методики на улучшение внутренней структуры оценено с помощью архитектурных метрик качества кода. Программная реализация осуществлена на базе АОП-инструмента Aspect.NET [5] и среды разработки MS Visual Studio 2010.

ТЕХНОЛОГИЯ ASPECT.NET

Aspect.NET – это инструментарий АОП для платформы .NET, разработанный в лаборатории Java-технологии математико-механического факультета СПбГУ под научным руководством профессора В.О. Сафонова. С помощью Aspect.NET можно определять аспекты в отдельных библиотеках классов, а затем вплетать вызовы их методов в заданные места целевой сборки. Определения аспектов не зависят от конкретного языка, а разрабатывать их можно в любой среде разработки, поддерживающей платформу .NET. В качестве альтернативы и для более простого определения аспектов Aspect.NET позволяет описывать их на простом метаязыке Aspect.NET ML. В этом заключается его основное преимущество по сравнению с другими АОП-инструментами, обеспечивающими АОП-функционал только через пользовательские атрибуты (например PostSharp [6]).

Другой альтернативой могут служить IOC-контейнеры. Например, компания Microsoft позиционирует Enterprise Library (EL) как набор наиболее удачных архитектурных решений, применимых в большинстве программных систем. По сравнению с динамическим применением аспектов в EL, когда во время выполнения программы среда .NET создает прокси-объекты в заданных местах, Aspect.NET вплетает действия на уровне MSIL-инструкций после этапа компиляции целевой сборки, что влечет повышение производительности целевого приложения. Достоинствами библиотеки EL является встроенная поддержка со стороны среды выполнения .NET, а также возможность настройки аспектов без компиляции через конфигурационные XML-файлы. Однако следует учитывать, что при работе с облачными приложениями, перекоm-

пиляция для внедрения аспектов не вызывает трудностей, так как MS Azure поддерживает прозрачное развертывание изменившегося приложения путем постепенного перезапуска отработавших экземпляров. Более того, «пост-обработка» Aspect.NET дает возможность выбирать конкретные места применения действий аспектов.

Аспектом может быть любой класс, наследующий предопределенный класс Aspect [7]. Реализация аспекта осуществляется статическими методами («действиями»), которые затем будут вставлены компоновщиком в заданные точки внедрения (joinpoints) в сборке целевого приложения. Требуемое множество точек внедрения задается в пользовательском атрибуте AspectAction() своего действия. Любое действие можно вставлять перед (ключевое слово %before), после (%after) или вместо (%instead) вызова заданного целевого метода. Название целевого метода задается с помощью регулярных выражений относительно его сигнатуры. Местонахождение точки внедрения внутри конкретного метода или класса можно фильтровать по ключевому слову %within.

Внутри действий можно использовать свойства базового класса, предоставляющие доступ к контексту точки внедрения, например, «this» и «target» объектам целевого метода, его метаданным (типа MethodInfo), отладочной информации и пр. Аргументы целевого метода передаются через аргументы действия.

Наконец, если мы пометим пользовательским атрибутом ReplaceBaseClass класс в нашем аспекте, производный от какого-либо целевого класса (далее «аспектный производный класс»), то компоновщик подставит его вместо своего базового класса. Связь через наследование устанавливает сильную зависимость между аспектом и целевым классом, что не согласуется с теоретическими установками АОП. Однако без него не обойтись в тех случаях, когда аспект должен обеспечить совместимость интерфейса с целевым классом, а вынесение поведения целевого класса в отдельный интерфейс не желательно, так как потребует модификации его исходного текста.

Компоновщик аспектов – это отдельное консольное приложение. Его параметрами являются пути к сборкам аспектов и целевого приложения. При работе в MS Visual Studio требуется в свойствах проекта аспекта включить его вызов в события пост-компиляции (post-build events).

КАТАЛОГ АОП-РЕФАКТОРИНГОВ

После последовательного применения приемов из данного каталога к целевому приложению в его исходном коде останется лишь непосредственная бизнес-логика, а весь «сервисный» код будет вынесен в отдельные проекты аспектов. Будем считать, что аспект определен удачно, если для его применения в целевом проекте не требуется менять исходный код. При этом допустимы изменения конфигурационных файлов (например Web.config) и декларативный код разметки веб-страниц (например aspx), поскольку такие изменения не ухудшают модифицируемость программных компонентов целевого приложения.

1. Локализация в аспекте объектов службы.

Как правило, использование службы заключается в том, что один из целевых классов хранит ссылки на его объекты в виде полей, инициализирует их в своем конструкторе (или при старте роли), а затем вызывает их внутри своих методов при необходимости. Таким образом, целевой класс имеет ассоциацию с классами сервиса.

Если весь код по инициализации этих классов и вызову их методов будет перенесен в действия аспекта, то повысится зацепление (cohesion) и уменьшится связность (coupling) роли. Теперь весь код, связанный с функцией службы располагается в одном модуле.

Для этого в проект аспекта добавим ссылки на нужные служебные сборки MS Azure и перенесем объекты службы из полей целевого класса в поля аспекта. Затем перенесем все вызовы методов класса службы из каждого метода целевого класса в от-

дельные действия аспекта.

До АОП-рефакторинга [2] (листинг 1).

После АОП-рефакторинга (листинг 2).

2. Перенос всех вызовов объектов службы из целевого класса в аспектный производный класс, если созданием целевого класса занимается MS Azure.

В Aspect.NET достижимыми точками внедрения будут только те, у которых вызовы целевых методов происходят внутри целевой сборки. MS Azure является каркасом (framework), и зачастую он сам вызывает переопределенные методы целевого класса, например Page_Load() при загрузке веб-страницы. У Aspect.NET нет доступа к внутренним системным библиотекам, поэтому вставить вызов действия в этом случае невозможно. Однако, если мы унас-

Листинг 1

```
public class ProductsRepository : IProductRepository
{
//Объекты службы кэширования
private static DataCacheFactory CacheFactory;
private static DataCacheFactoryConfiguration FactoryConfig;

public List<string> GetProducts() {
    List<string> products = null;
      //… Загрузка данных из кэша
      //…Если их там нет, загрузка из БД…
      products = query.ToList();
      //…Сохранение данных в кэше
      return products;
}
}// class ProductsRepository
```

Листинг 2

```
public class ProductsRepository : IProductRepository {
public List<string> GetProducts() {
    List<string> products = null;
      //…Загрузка из БД…
      products = query.ToList();
      return products;
}
}// class ProductsRepository

public class Caching : Aspect {
private static DataCacheFactory CacheFactory;
private static DataCacheFactoryConfiguration FactoryConfig;

[AspectAction("%instead %call *IProductRepository.GetProducts()")]
public static List<string> GetProducts() {
     //… Загрузка данных из кэша
     //…Если их там нет, выполняем целевой метод
     products = (TargetMemberInfo as MethodInfo)
                .Invoke(TargetObject, null) as List<string>;
     //…Сохранение данных в кэше
     return products;
}
}// class Caching
```

ледуем от класса, содержащего нужный нам целевой метод, то Aspect.NET сможет подменить им свой базовый класс в целевой сборке. Теперь осталось лишь переопределить нужный целевой метод в этом аспектном производном классе.

До АОП-рефакторинга (листинг 3).

После АОП-рефакторинга (листинг 4).

3. Привязка действий аспекта, связанного с ролью, к переадресованным вызовам методов базового класса RoleEntryPoint.

В том случае, если обращения к методам сервисного объекта MS Azure содержатся внутри переопределенных методов роли и их необходимо перенести в действия аспекта, следует привязать их точки внедрения к переадресованным вызовам методов базового класса.

Предположим, в переопределенном методе роли OnStart() происходит инициализация сервисного объекта MS Azure. Тогда следует перенести эту инициализацию в действие, которое будет вставлено, например, перед вызовом RoleEntryPoint.OnStart(). Здесь не подходит решение с определением в аспекте наследника роли, переопределением у него метода OnStart() и подменой им своего базового класса с помощью пользовательского атрибута ReplaceBaseClass. Дело в том, что среда MS Azure автоматически создает экземпляры ролей, требуя при этом, чтобы непосредственным базовым классом роли был класс Microsoft.WindowsAzure.ServiceRuntime.RoleEntryPoint. К счастью, в большинстве случаев внутри переопределенных методов роли вызываются методы базового класса, поэтому мы можем привязать вызов действия аспекта к нему.

До АОП-рефакторинга [8] (листинг 5).

После АОП-рефакторинга (листинг 6).

4. Выделение визуальных, управляющих службой элементов (controls), в .ascx компонент аспекта.

Иногда управление службой производится на основе веб-интерфейса. В этом случае код разметки основной страницы с данными (.aspx) перемешан с разметкой веб-интерфейса службы. Если мы выделим

Листинг 3

```
public partial class _Default : System.Web.UI.Page {
//…
protected void LogButton_Click(object sender, EventArgs e){
   Microsoft.Practices.EnterpriseLibrary.Logging.
      Logger.Write("Message from the Logging App Block");
}
}// class _Default
```

Листинг 4

```
public partial class _Default : System.Web.UI.Page {
//…
protected void LogButton_Click(object sender, EventArgs e)
{}
}// class _Default

[ReplaceBaseClass]
//Подменяет целевой класс _Default данным наследником
public class ELLogger : _Default {
protected void LogButton_Click(object sender, EventArgs e){
   Microsoft.Practices.EnterpriseLibrary.Logging.
      Logger.Write("Message from the Logging App Block");
   base.LogButton_Click(sender, e);
}
}// class ELLogger
```

Листинг 5

```
public class WebRole : RoleEntryPoint {
public override bool OnStart() {
    DiagnosticMonitorConfiguration config =
        DiagnosticMonitor.GetDefaultInitialConfiguration();
   //…
    DiagnosticMonitor.Start("DiagnosticsConnectionString", config);
   //…
   return base.OnStart();
} //class WebRole
```

Листинг 6

```
public class WebRole : RoleEntryPoint {
     public override bool OnStart() {
             return base.OnStart();
     }
}// class WebRole

public class ElasticAzureAspect : Aspect {
[AspectAction("%before %call *RoleEntryPoint.OnStart() &&
                                                %within (*WebRole)")]
public static void StartDiagnostic() {
    DiagnosticMonitorConfiguration config =
        DiagnosticMonitor.GetDefaultInitialConfiguration();
    //…
    DiagnosticMonitor.Start(DiagnosticsConnectionString, config);
}
}// class ElasticAzureAspect
```

этот веб-интерфейс службы в отдельный .ascx компонент, то сможем повторно использовать его на других страницах. Предположим, что облачная веб-роль реализуется по принципам ASP.NET MVC [2]: Контроллер заполняет Модель данными и передает ее Виду (то есть .aspx странице) для визуального отображения. Тогда, чтобы выделить из Вида отдельный .ascx компонент, необходимо выполнить следующие действия:

a) перенести данные, требуемые для управления службой из общей модели в отдельную модель и сохранить ее класс в проекте аспекта;

b) выделить управляющие элементы веб-интерфейса службы в .ascx компонент и связать его с новой моделью (далее AscxControlViewModel);

c) скопировать этот .ascx компонент в проект аспекта;

d) показать содержимое .ascx компонента на .aspx странице Вида через подстановку команды;
<%= Html.Action(«AscxControlView») %>
в то место, где раньше находилась разметка веб-интерфейса управления службы, при этом AscxControlView – название этого компонента;

e) создать аспектный производный класс для Контроллера и определить в нем метод для обработки команды AscxControlView (листинг 7).

5. Получение информации от аспекта и управление им через «пустые» private свойства в целевом коде.

Зачастую мы используем сервисы MS Azure для получения какого-либо значения,

Листинг 7

```
[ReplaceBaseClass]
public class AspectController : TargetController {
     [ChildActionOnly]
     public ActionResult AscxControlView() {
          //…Вычисляем данные для модели управления службой
          var model = new AscxControlViewModel() {
              //… Инициализируем данные модели
          };
          //.ascx компонент визуализирует новую модель
          return PartialView(model);
     }
}
```

например для вывода пользователю на .aspx-странице текущего количества одновременно запущенных экземпляров (instances) веб-роли. Или же используем это значение для управления службой, например, устанавливаем требуемое число экземпляров веб-роли. Данное поведение полностью соответствует концепции свойств в .NET и имеет смысл заменить вызовы методов сервиса MS Azure на его пустое свойство – «представитель». Сами вызовы будут перенесены в аспект, действия которого привяжутся к get_ и set_ методам свойства и подменят их на этапе компиляции (с помощью правила %instead %call).

До АОП-рефакторинга [8] (листинг 8).

После АОП-рефакторинга (листинг 9).

6. Перенос изменений Web.config в действия аспекта.

Листинг 8

```
public partial class _Default : System.Web.UI.Page {
//…
protected void btnRefresh_Click(object sender, EventArgs e) {
    //Отображаем текущее кол-во активных экземпляров
    lblCurrentInstanceCount.Text = GetCurrentInstances();
}
protected string GetCurrentInstances() {
    //Используем Azure-службу AzureRESTMgmtHelper
    //…
    string InstanceCount =
        AzureRESTMgmtHelper.GetInstanceCount(svcconfig, "WebRole1");
    return InstanceCount;
}
protected void btnUpdateConfig_Click(object sender, EventArgs e) {
    //Меняем кол-во активных экземпляров на значение в поле
    //txtNumberInstances
    //…
    string UpdatedSvcConfig =
        AzureRESTMgmtHelper.ChangeInstanceCount(svcconfig,
          "WebRole1", txtNumberInstances.Text.Trim());
    AzureRESTMgmtHelper.ChangeConfigFile(UpdatedSvcConfig);
}
}// class _Default
```

Для того или иного сервиса MS Azure требуется внести изменения в Web.config целевого приложения. Если мы вызовем метод данного сервиса, а программный контекст соответствующим образом не будет сконфигурирован, то, в лучшем случае, ничего не произойдет, либо сгенерируется исключение. Вполне допустимо прописать изменения вручную, при этом необходимо продублировать их в комментариях аспекта, чтобы впоследствии быстро локализовать их местоположение в Web.config. Однако встречаются случаи, когда прямая модификация Web.config невозможна, например, отсутствуют исходные тексты целевого проекта.

В этом случае определяется отдельное действие аспекта, которое вставляется перед методом данного сервиса Azure. Внутри этого действия проводится проверка соответствующего состояния среды и при необходимости проводится ее программная конфигурация.

До АОП-рефакторинга (листинг 10).

После АОП-рефакторинга (листинг 11).

Для оценки полученных результатов использовалась интегральная метрика Maintainability Index, которая встроена в MS

Листинг 9

```
public partial class _Default : System.Web.UI.Page {
//…
//Свойство-представитель аспекта ElasticAzureAspect
private string CurrentInstances {
    get {return ""; }
    set {}
}
protected void btnRefresh_Click(object sender, EventArgs e)
{
    lblCurrentInstanceCount.Text = CurrentInstances;
}
protected void btnUpdateConfig_Click(object sender, EventArgs e) {
    CurrentInstances = lblCurrentInstanceCount.Text.Trim();
}
}// class _Default
public class ElasticAzureAspect : Aspect {
//…
[AspectAction("%instead %call *_Default.get_CurrentInstances()")]
public static string get_CurrentInstances() {
    //…
    string InstanceCount =
        AzureRESTMgmtHelper.GetInstanceCount(svcconfig, "WebRole1");
            return InstanceCount;
}

[AspectAction("%instead %call *_Default.set_CurrentInstances(string)")]
public static void set_CurrentInstances(string value) {
     //…
     string UpdatedSvcConfig =
         AzureRESTMgmtHelper.ChangeInstanceCount(svcconfig,
                                                 "WebRole1", value);
    AzureRESTMgmtHelper.ChangeConfigFile(UpdatedSvcConfig);
}
}// class ElasticAzureAspect
```

Visual Studio 2010 и определяет удобство сопровождения исходного кода отдельных классов и проекта в целом. Так, например, выделение из целевых классов приложения [2] функции кэширования в отдельный аспект позволило увеличить для них этот индекс на 33%. Для целевых классов другого приложения [8] аспектное управление конфигурацией дало прирост этого индекса на 24%.

ЗАКЛЮЧЕНИЕ

Таким образом, применение АОП и Aspect.NET в разработке облачного приложения позволяет повысить легкость его сопровождения, увеличить скорость разработки и снизить затраты за счет повторного использования универсальных аспектов. Все упомянутые аспекты доступны на сайте проекта Aspect.NET [9].

Листинг 10

```
<system.diagnostics>
  <trace>
    <listeners>
      <add type="Microsoft.WindowsAzure.Diagnostics.
                                        DiagnosticMonitorTraceListener,
Microsoft.WindowsAzure.Diagnostics, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35"
name="AzureDiagnostics">
      </add>
    </listeners>
  </trace>
</system.diagnostics>
```

Листинг 11

```
public class AzureTrace: Aspect {
//…
//Перед каждым вызовом TraceInformation() убеждаемся,
//что соотв. Azure-подписчик зарегистрирован
[AspectAction("%before %call *System.Diagnostics.Trace.
                                                TraceInformation")]
public static void AspectTrace() {
     //Если его еще нет в списке подписчиков, то добавляем в
     //этот список…
     if (!ContainsWADListener()) {
         System.Diagnostics.Trace.Listeners.Add(
                        new Microsoft.WindowsAzure.Diagnostics.
                             DiagnosticMonitorTraceListener());
     }
}
private static bool ContainsWADListener() {
    foreach(System.Diagnostics.TraceListener listener in
                                  System.Diagnostics.Trace.Listeners)
     if (listener is Microsoft.WindowsAzure.Diagnostics.
        DiagnosticMonitorTraceListener) return true;
     else return false;
}
}// class AzureTrace
```

Литература

1. Сайт проекта MS Windows Azure // http://www.windowsazure.com (дата обращения 25.02.12).
2. Windows Azure Training Course: Caching Data with Windows Azure Caching // http://msdn.microsoft.com/en-us/gg457898 (дата обращения 25.02.12).
3. *Фаулер М.* Рефакторинг. Улучшение существующего кода. СПб.: Символ, 2004. С. 14.
4. Сайт проекта MS Enterprise Library // http://wag.codeplex.com/ (дата обращения 25.02.12).
5. *Safonov V.O.* Using aspect-oriented programming for trustworthy software development // Wiley Interscience. John Wiley & Sons, 2008.
6. Сайт проекта PostSharp // http://www.sharpcrafters.com/ (дата обращения 25.02.12).
7. *Григорьев Д.А.* Реализация и практическое применение аспектно-ориентированной среды программирования для Microsoft .NET // СПб.: Научно-технические ведомости СПбГПУ, 2009. № 3. С. 225–232.
8. *J. Fultz.* Performance-Based Scaling in Windows Azure // http://msdn.microsoft.com/en-us/magazine/gg232759.aspx (дата обращения 25.02.12).
9. Сайт проекта Aspect.NET // http://aspectdotnet.org/ (дата обращения 28.02.12).

Abstract

The MS Azure cloud platform provides a set of services which allow to develop efficient and safe applications. Control of these services is made from the source code of cloud application that leads to a problem of «cross-cutting concerns» when implementation of one function is tangled on all code.

This paper describes an approach which allows to remove this disadvantage.

Keywords: aspect-oriented programming, AOP-refactoring, legacy code, Aspect.NET.

Григорьева Анастасия Викторовна,
аспирант кафедры информатики
математико-механического
факультета СПбГУ,
nastya001@mail.ru