РАЗРАБОТКА ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ПЛАТФОРМЕ .NET С ПРИМЕНЕНИЕМ АСПЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

УДК 004.4.2

Глеб Анатольевич Черменнов,
аспирант каф. Математического обеспечения информационных систем и инноватики, Московский государственный университет экономики, статистики и информатики (МЭСИ)
Тел.: 8 (916) 314-93-24
Эл. почта: thebitterend77@gmail.com

Елена Владимировна Ковалевская,
к.э.н., доц., проф. кафедры Математического обеспечения информационных систем и инноватики, Московский государственный университет экономики, статистики и информатики (МЭСИ)
Тел.: 8 (495) 442-80-98
Эл. почта: EKovalevskaya@mesi.ru

Целью данной статьи является разработка модели для построения устойчивых распределенных приложений. Задачами данной статьи являются выявление основных проблем при разработке распределенных приложений и предложение методов по их решению. Предлагаемое решение может быть применено в распределенных системах и позволит существенно снизить время на поддержку инфраструктурного кода.

Ключевые слова: аспектно-ориентированное программирование, разработка приложений, разделение обязанностей, механизм для объединения аспектов и кода приложения в рабочую систему, технические аспекты приложения.

Gleb A. Chermennov,
Post-graduate student, the Department of Software of Information Systems and Innovations, Moscow State University of Economics, Statistics and Informatics (MESI)
Tel.: 8 (916) 314-93-24
E-mail: thebitterend77@gmail.com

Elena V. Kovalevskaya,
PhD in Economics, Professor, the Department of Software of Information Systems and Innovations, Moscow State University of Economics, Statistics and Informatics (MESI)
Tel.: 8 (495) 442-80-98
E-mail: EKovalevskaya@mesi.ru

APPLICATION SOFTWARE DEVELOPMENT POWERED BY .NET USING THE ASPECT-ORIENTED PROGRAMMING

The authors' aim is to develop application software for building stable distributed applications. The objectives are to identify main issues in building distributed applications and to suggest effective methods to solve these problems. The proposed solution can be used in building distributed applications and it will allow to significantly reduce the time spent on supporting infrastructure code in such applications.

Keywords: aspect-oriented programming, applications programming, separation of concerns, mechanism for association of aspects and application code in a system, technical aspects of application.

1. Введение

Почему разработка программного обеспечения так сложна? Одна из основных причин тому – есть много вещей, которые нужно держать в голове при написании кода. С точки зрения руководителя проекта, нужно контролировать затраченное время, бюджет, ресурсы, профессиональные навыки сотрудников и ряд других аспектов. Чаще всего, у любого участника команды есть много задач, и выполнение некоторых из них выходит за рамки его непосредственных обязанностей. Вполне может оказаться, что программист отчитывается перед двумя разными людьми, и каждый из них ждет от него 100%-ной производительности, так что приходится выполнять работу на 200%. Программисты должны разбираться в приложении, которое разрабатывают, в предметной области и в особенностях используемой программной платформы [3]. При проектировании системы необходимо решить сразу несколько задач: например, как система выполняет предъявленные к ней требования с точки зрения бизнеса, как она достигает высокой скорости работы, масштабируется ли и как обходит ограничения платформы. В процессе разработки часто выясняется, что написанный код выполняет несколько задач сразу, что является примером низкокачественного проектирования. В этом случае необходимо улучшить разделение обязанностей (separation of concerns) [1]. Равно как и каждый член команды, каждый фрагмент написанного кода должен быть сфокусирован на выполнении ровно одной задачи в отдельно взятый момент времени.

К счастью, времена, когда количество профессиональных инструментов для разработки, тестирования и разворачивания приложений было невелико, безвозвратно ушли в прошлое. В наше время разработчик на любой программной платформе имеет в своем распоряжении впечатляющий арсенал, позволяющий ему сосредоточиться на исполнении своих обязанностей и свести время, уходящее на выполнение рутинных операций, к минимуму. Об одном из таких инструментов или, вернее, подходов, и пойдет речь в этой статье.

Новизна данного подхода состоит в следующем – используются возможности аспектно-ориентированного программирования (АОП) для сокращения объема инфраструктурного кода практически до нуля как на стороне клиента, так и на стороне сервера, а также возможность модульного тестирования написанного кода, что немаловажно в системах с большим количеством кода и сложным поведением.

2. Методика исследования

За последние 4 года, участвуя в разработке нескольких крупных распределенных приложений, мы сталкивались с тем, что в каждом приложении присутствовали определенные вещи, которые приходилось писать ежедневно (так называемый boilerplate code). Мы с коллегами стремились к сокращению количества таких участков, но, как выяснилось, в рамках стандартного инструментария возможности сосредоточения такого функционала в одном месте сильно ограничены. Так было обращено внимание в сторону альтернативных решений, в частности, PostSharp (библиотеки, реализующей аспектно-ориентированный подход при разработке под .NET Framework). Проведенное исследование показало, что почти весь функционал, распространенный по всей системе, можно сосредоточить в одном месте, а также предоставить возможность для настройки его поведения. После анализа опыта разработки таких проектов был составлен список типовых задач, которые приходилось часто решать, и большинство из них были реализованы в библиотеке, предназначенной для упрощения написания распределенных приложений на платформе .NET. Библиотека находится в процессе разработки, но уже успешно применяется в нескольких проектах.

3. Экспериментальная часть

Инфраструктурные требования к современным приложениям постоянно усложняются – нужно записывать ошибки, произошедшие во время выполнения приложения (logging), сохранять и изменять введенные пользователями данные (persistence), обрабатывать исключительные ситуации и учесть еще множество различных вопросов [4]. В связи с этим, все чаще складывается ситуация, когда разработчик пишет не код, отвечающий за непосредственный функционал приложения, а занимается обеспечением вышеперечисленных задач. Это не только не приносит пользы компании, на которую он работает, но и зачастую привносит в код приложения элементы, засоряющие проект.

Аспектно-ориентированное программирование (aspect-oriented pro¬gramming, AOP) называет такие чисто технические аспекты приложения сквозными обязанностями (cross-cutting concerns), потому что код для выполнения этих задач имеет обыкновение «расползаться» по всей системе. Этот факт особенно хорошо заметен при применении объектно-ориентированных языков, таких как Java и C#, так как сквозные обязанности в приложении плохо стыкуются с объектно-ориентированной парадигмой. АОП предлагает средства по вынесению вышеуказанного функционала из кода приложения в аспекты (aspects) – блоки кода, позволяющие компактно организовать реализацию сквозных обязанностей. Этот подход также предоставляет механизм для объединения аспектов и собственно кода приложения в рабочую систему во время компиляции или во время выполнения [8]. Таким образом, исходный код приложения освобождается от навязанных ему сторонних обязанностей, и команде разработчиков будет легче его поддерживать и расширять.

В каждом более-менее крупном приложении всегда присутствуют определенного рода соглашения внутри команды разработчиков – не только в плане стандарта кодирования, но и в плане подходов к реализации тех или иных компонентов системы. Каждому новому разработчику эти правила объясняют и показывают примеры использования. С ростом объема исходного кода осуществлять контроль за выполнением таких правил становится все сложнее и сложнее. Если одни и те же правила применяются на определенных частях системы, их можно описать в коде и автоматически применять для указанных компонентов. Автоматизировав эту задачу – скажем, проводя проверку на соответствие правилам каждый раз во время сборки приложения, можно сэкономить многие часы отладки и избежать сообщений об ошибках во время выполнения. Современные библиотеки, реализующие аспектно-ориентированный подход, позволяют справиться с такой задачей.

Многие библиотеки накладывают технические ограничения различного рода – к примеру, если информация об объекте передается по сети, он должен быть способен к сериализации (сохранению своего состояния в тот или иной формат передачи данных). Код, написанный только для удовлетворения этих ограничений, имеет тенденцию к разбуханию, – более того, написание и поддержка такого кода не несут непосредственной пользы для приложения [5]. В результате по всей системе распространяются знания о конкретной библиотеке или компоненте, что почти всегда ведет к жесткому связыванию (tight coupling) инфраструктурной части системы и бизнес-логики и затрудняет поддержку всего приложения. Эта проблема легко решается с применением АОП, и данное решение экономит разработчикам время и позволяет сконцентрироваться непосредственно на функционале приложения.

Разумеется, у аспектно-ориентированного подхода есть и слабые стороны. В качестве главных минусов можно назвать сложность технической реализации почти всех библиотек такого рода, что отпугивает от них большинство программистов, и сложность отладки приложения в случае возникновения непредвиденных ситуаций. Кроме того, аспектно-ориентированное программирование зачастую заставляет идти на технические компромиссы, одним из следствий которых является сложность тестирования компонентов, поведение которых изменяется или контролируется аспектами. Те из библиотек, которые преобразовывают промежуточный код по окончании процесса компиляции, зачастую немного замедляют процесс сборки приложения [7]. Таким образом, можно прийти к выводу, что в полной мере ощутить мощь аспектно-ориентированного подхода смогут лишь опытные программисты, которые хорошо представляют себе, что именно происходит «за кулисами» того или иного инструмента, знают его достоинства и недостатки и пользуются им в случаях, когда это действительно оправданно.

Применение аспектно-ориентированного подхода хорошо себя зарекомендовало в разработке сложных настольных приложений с применением технологии Windows Presentation Foundation(WPF). Возможности WPF дают разработчикам массу возможностей, но в то же время накладывают ограничения на написанный ими код – в первую очередь это относится к реализации интерфейса INotifyPropertyChanged, посредством которого происходит уведомление об изменениях данных, привязанных к элементам пользовательского интерфейса [6].

Большинство современных платформ для разработки обладает средствами для реализации аспектно-ориентированного подхода. В среде Java наиболее широко распространены AspectJ и SpringAOP, в .NETFramework пользуются популярностью PostSharp и Aspect.Net. Механизмы аннотаций (annotations) в Java и атрибутов (attributes) в .NET позволяют разработчикам не только использовать решения, предлагаемые авторами библиотек, но также реализовывать свои собственные и использовать их в своих приложениях.

С течением времени требования относительно надежности, масштабируемости, безопасности распределенных приложений постоянно возрастают. В связи с этим возникает ряд проблем при проектировании современных распределенных приложений:

- выполнять требования по масштабированию и отказоустойчивости становится все сложнее, так как с ростом количество мощностей растет аппаратная инфраструктура, необходимая для поддержания этих характеристик, и соответственно стоимость поддержки такого решения;
- чем крупнее приложение, тем сложнее осуществлять его настройку и развертывание;
- по мере изменения требований к приложению становится все сложнее модифицировать исходный код приложения;
- по мере разработки технические аспекты приложения (различные диагностические показатели) имеют

тенденцию смешиваться с кодом, отвечающим непосредственно за решение основных задач приложения.

Решением инфраструктурных сложностей служат облачные технологии, и в первую очередь платформа Windows Azure от Microsoft, предоставляющая широкий набор возможностей и инструментов для разработчиков [2]. С помощью Windows Azure можно решить проблемы масштабируемости распределенных приложений и повышения их отказоустойчивости за счет использования современных методик настройки промежуточной среды.

Сложности с настройкой и модификацией кода в соответствии с требованиями, а также переплетения при разработке технических и бизнес-аспектов приложения успешно решаются с помощью революционного подхода к разработке программного обеспечения – аспектно-ориентированного программирования. Аспектно-ориентированное программирование представляет собой парадигму, основанную на идее разделения функциональности для улучшения разбиения программы на модули. Аспектно-ориентированное программирование прекрасно вписывается в концепцию объектно-ориентированного программирования. Применение двух этих подходов позволяет существенно повысить гибкость и облегчить модификацию и поддержку программного обеспечения, тем самым сократив расходы на соответствующие этапы жизненного цикла программного продукта.

4. Заключение

В ходе исследования были выявлены проблемы при разработке распределенных приложений, такие как необходимость явно вызывать одни и те же участки кода, в то время как места их вызова четко определены.

Решение этих проблем основано на использовании аспектно-ориентированного программирования и позволяет сократить время на разработку частей системы, не имеющих прямого отношения к бизнесу. Таким образом, у разработчиков будет больше времени для решения задач бизнеса.

Литература.

1. М.Фаулер. Рефакторинг. Улучшение существующего кода. // М., Символ плюс, 2008.

2. J.Lowy. Programming WCF Services: Mastering WCF and the AzureApp-Fabric Service Bus. // O'Reilly, 2010.

3. T.Erl. Service-Oriented Architecture: Concepts, Technology, and Design // Prentice Hall PTR, 2005.

4. R.Seroter. Applied Architecture Patterns on the Microsoft Platform. // Packt Publishing, 2010.

5. И.Соммервилл. Инженерия программного обеспечения. // М., Вильямс, 2002.

6. MacDonald, Matthew. Pro Windows Presentation Foundation in C# 2010 (Expert's Voice in .NET). // Apress, 2010.

7. Ivar Jacobson, Pan-Wei Ng. Aspect-Oriented Software Development with Use Cases. // Addison-Wesley, 2005.

8. Vladimir Safonov . Using Aspect-Oriented Programming for Trustworthy Software Development. // Wiley-Interscience, 2008.

References

1. Fauler M. Refactoring. Improvement of an existing code. // M., Simvolplus, 2008.

2. J.Lowy. Programming WCF Services: Mastering WCF and the AzureApp-Fabric Service Bus. // O'Reilly, 2010.

3. T.Erl. Service-Oriented Architecture: Concepts, Technology, and Design // Prentice Hall PTR, 2005.

4. R.Seroter. Applied Architecture Patterns on the Microsoft Platform. // Packt Publishing, 2010.

5. Sommerwill I. Software engineering. // M., Williams, 2002.

6. MacDonald, Matthew. Pro Windows Presentation Foundation in C# 2010 (Expert's Voice in .NET). // Apress, 2010.

7. Ivar Jacobson, Pan-Wei Ng. Aspect-Oriented Software Development with Use Cases. // Addison-Wesley, 2005

8. Vladimir Safonov . Using Aspect-Oriented Programming for Trustworthy Software Development. // Wiley-Interscience, 2008.