Література

1. *Сальніков В. Н.* Бізнес-планування та організація фінансово-господарської діяльності підприємства: Навчальний посібник. — 3-е вид. доп. і випр. — К.: Альтера, 2006. — 520 с.
2. *Верченко П. І.* Багатокритеріальність і динаміка економічного ризику (моделі та методи): Монографія. — К.: КНЕУ, 2006. — 272 с.
3. *Верченко П. І.* Багатокритеріальні моделі в інтелектуальних системах прийняття рішень. — Моделювання та інформ. системи в економіці: Зб. наук. пр./відп. ред. В. К. Галіцин. — 2008. — Вип. 78. — Ст. 36—44.
4. *Верченко П. І.* Принцип максимального відхилення як критерій прийняття багатокритеріальних рішень. — Економічна кібернетика: міжнародний науковий журнал. — 2001. — № 1—2. — С. 16—24.
5. *Камінський А. Б.* Моделювання фінансових ризиків: Монографія. — К.: Видавничо-поліграфічний центр «Київський університет», 2006. — 204 с.
6. Бізнес-планування підприємницької діяльності в умовах невизначеності та ризику. — Х: Фактор, 2006 — 480 с.

УДК 621.391 (075.8)

А. В. Бєгун, канд. екон. наук,
ДВНЗ «Київський національний економічний університет імені Вадима Гетьмана»

АСПЕКТНО-ОРІЄНТОВАНА ТЕХНОЛОГІЯ ОПТИМІЗАЦІЇ ЗАХИСТУ ДОДАТКІВ WEB-ПОРТАЛУ

Анотація. В статті розглянуто аспектно-орієнтований підхід створення захищених програмних систем складної структури. Основу цього підходу складає аспектно-системний модуль, в який внесена наскрізна функціональність. Аспект відображає аудит, використання якого оптимізує захист додатків при їх розробці.

ANNOTATION. The article deals with Aspect-oriented approach to creating secure software systems with complex structures. The basis of this approach is Aspect-system module, which included cross-functionality. Aspect audit shows that optimizes protection applications in their development

Ключові слова. Web-портал, аспект, наскрізна функціональність, аудит, інтегратор аспектів, Java-сервлети.

Сучасний Web-портал неможливий без використання Web-додатків, на які часто спрямовують свої атаки зловмисники. Web-додатки стають усе більшь розповсюдженими й усе більше складними, відіграючи в такий спосіб основну роль у більшості онлайнових проектів. За оцінками фахівців, близько 70 % часу в проек-

тах витрачається на супровід і внесення змін у готовий програмний код. Тому досить важливою і актуальною стає роль захисту додатків та її оптимізація на базі новітніх технологій, наприклад, аспектно-орієнтованого програмування (АОП) і подібних трансформаційних підходів.

Як правило, будь-яка програмна система складається з кількох частин: основної (предметно-орієнтованої) і системної частини, які несуть у собі необхідну функціональність. Наприклад, ядро системи обробки кредитних карток призначено для роботи із платежами, тоді як функціональність системного рівня призначена для ведення журналу подій, цілісності транзакцій, авторизації, безпеки, продуктивності й т.д. Більшість подібних частин системи, відома як наскрізна функціональність, стосуються безлічі основних предметно-орієнтованих модулів. Тобто, складну програмну систему можна розглядати як комбінацію модулів, кожний з яких містить у собі крім бізнес-логіки частину наскрізної функціональності з набору вимог до системи (рис. 1).

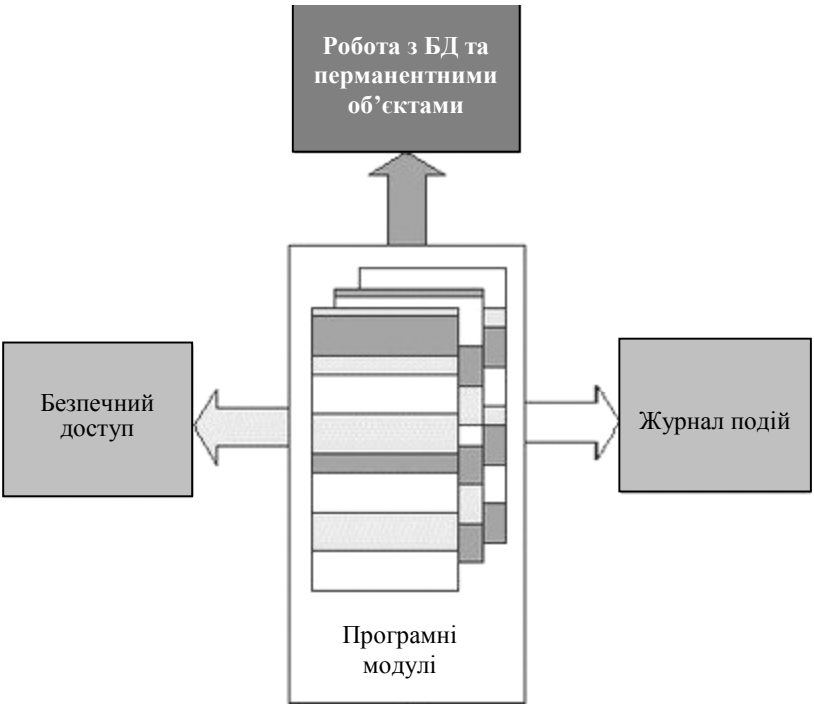

Рис. 1. Система як набір функціональних модулів

При розробці програмної системи з використанням існуючих методологій програмування наскрізна функціональність буде включена в усі модулі. Тоді система буде складною при проектуванні, розумінні, реалізації й підтримці.

Розробник створює програмну систему як результат обробки безлічі вимог. Можна явно виділити із цієї множини вимоги до логіки конкретного модуля й загальносистемні. Багато із системних вимог можуть бути ортогональними один до одного та вимогам конкретного модуля. Вимоги системного рівня мають тенденцію перетинатися з безліччю основних вимог. Наприклад, типова система рівня підприємства містить у собі такі види наскрізної функціональності, як ведення журналу подій, управління ресурсним пулом, адміністрування, аналіз продуктивності й керування носіями інформації. Кожне із цих вимог до системи торкається безлічі підсистем, наприклад, вимога з управління носіями інформації торкається зберіганню кожного бізнес-об'єкту.

Сучасні технології розробки програмного забезпечення на рівні мов програмування надають зручні засоби для виділення логіки функціонування програми в окремі модулі, але жодна з них не пропонує зручного способу локалізації в окремі модулі функціональності, яка повинна поширюватися на всю систему [1]. Через те, що реалізація наскрізної функціональності не може бути відособлена засобами мови програмування в окремому програмному модулі, елементи цієї реалізації присутні в тому або іншому виді у більшості модулів, що утворюють програмну систему. Розглянемо приклад java-класу, основним завданням якого є реалізація деякої логіки:

```
public class SomeBusinessClass {
  // Бізнес-дані
  // допоміжні дані
            public void
performSomeOperation(OperationInformation info) {
            // перевірити рівень доступу до даних
            // перевірити відповідність вхідних даних контракту
            // заборонити доступ до даних інших потоків виконання
            // перевірити стан кеша даних
            // занести в журнал оцінку про початок операції
            // ==== Реалізація логіки даного класу ====
            // занести в журнал оцінку про кінець операції
            // дозволити доступ до даних інших потоків виконання
            }
}
```

У цьому прикладі можна виділити дві проблеми. По-перше, обумовлені допоміжні дані не ставляться до вимог, що накладаються на даний модуль, а потрібні для роботи наскрізної функціональності. По-друге, реалізація performSomeOperation(..) виглядає більш навантаженою, чим просто «Реалізація логіки даного класу». Для правильної роботи даного методу за вимогами необхідно виконати ряд дій, що не ставляться конкретно до даного модуля — перевірити рівень доступу до даних, перевірити відповідність вхідних даних контракту, заборонити доступ до даних інших потоків виконання, перевірити стан кеша даних, занести в журнал оцінку — це вимоги системного рівня. До того ж, багато хто із цих загальних вимог повинні бути реалізовані в інших модулях.

Аспектом в АОП є системний модуль, у який винесена наскрізна функціональність. Аспектний модуль — це результат аспектної декомпозиції, на етапі якої виявляються які-небудь явища, поняття, події які можуть бути застосовані до групи компонентів, отриманих після об'єктної декомпозиції. Аспект являє собою язикову концепцію, схожу із класом, але тільки більше високого рівня абстракції [2, 3].

Аспекти можуть зачіпати багато компонентів і використають так називані крапки вставки для реалізації регулярних дій, які звичайно розосереджені по всьому тексту програми. В аспектному модулі описуються зрізи крапок — крапки виконання програми, у які вбудовуються інструкції мови, які повинні виконуватися до, після або замість строго певної крапки виконання програми. Подібні інструкції мови є функціональністю, що підтримує взаємодія компонентів. Крім того, в аспектному модулі можуть описуватися ролі компонентів, на які може впливати даний аспект. В окремих реалізаціях АОП за допомогою аспектних модулів можна впливати на існуючу схему спадкування. З погляду АОП аспект є сервісом, що зв'язує компоненти системи.

Для ілюстрації роботи інтегратора аспектів повернемося, наприклад, до обробки кредитних карт. Для стислості розглянемо тільки дві операції — кредит і дебет:

```
public classCreditCardProcessor {
   public void debit(CreditCard card, Currency amount)
      throwsInvalidCardException, NotEnoughAmountException,
          CardExpiredException {
      // логіка по дебету
   }
   public void credit(CreditCard card, Currency amount)
       throws InvalidCardException {
```

```
        // логіка по кредиту
    }
}
```

та інтерфейс журналу подій:

```
public interface Logger {
  public void log(Stringmessage);
}
```

Для одержання бажаної композиції потрібне застосування наступних правил, виражених звичайною мовою:

— записати в журнал початок кожної операції

— записати в журнал закінчення кожної операції

— записати в журнал кожну виняткову ситуацію, що може виникнути в процесі роботи цього модуля.

Інтегратор аспектів, застосовуючи такі правила, одержить код еквівалентний даному:

```
public class CreditCardProcessorWithLogging {
  Logger _logger;
  public void debit(CreditCard card, Money amount)
    throws InvalidCardException, NotEnoughAmountException,
    CardExpiredException {
    _logger.log("Starting CreditCardProcessor.debit(CreditCard,
    Money) "+ "Card: " + card + " Amount: " + amount);
    // Debiting logic
    _logger.log("Completing CreditCardProcessor.debit(CreditCard,
    Money) " + "Card: " + card + " Amount: " + amount);
  }
  public void credit(CreditCard card, Money amount)
    throws InvalidCardException {
    System.out.println("Debiting");
    _logger.log("Starting CreditCardProcessor.credit(CreditCard,
    Money) " + "Card: " + card + " Amount: " + amount);
    // Crediting logic
    _logger.log("Completing CreditCardProcessor.credit(CreditCard,
    Money) " + "Card: " + card + " Amount: " + amount);
  }
}
```

Автоматичне компонування аспектів і традиційних модулів програми — компонентів є ключовою властивістю АОП, що визначає основну перевагу даної технології: уможливлюють інкапсуляцію наскрізної функціональності в окремих програмних модулях.

Для підтвердження цих тез розглянемо приклад використання АОП при написанні **JAVA-servlets.**

Відомо, що Сервлети — це високопродуктивні платформенонезалежні server-side-додатки, які написані на мові Java і складають реальну конкуренцію таким технологіям, як CGI, PHP3, Perl і, звичайно, ASP.

Java-сервлети були створені в Sun. Сервлети схожі на CGI-сценарії тим, що це код, який створює документи. Проте, Сервлети, оскільки вони використовують Java, повинні бути скомпільовані перед запуском як класи, що динамічно завантажуються вебсервером при запуску сервлетів. Інтерфейс відрізняється від CGI, JavaServer Pages або JSP; це інша технологія, що дозволяє розроблювачам вбудовувати Java у веб-сторінки, на зразок ASP.

Аспект буде відображати аудит, як один з головних засобів захисту інформації. Аудит є самим наочним прикладом застосування АОП, тому що його реалізація звичайним способом зобов'язує розробників додавати той самий код занесення в журнал і в усі частини системи. Наприклад, для ведення журналу дій кожного користувача необхідно включити в кожен контролер виклик методу відповідального за занесення даних у журнал подій, знову ж гарантії що такі виклики включені в кожен контролер, забезпечити важко. Або припустимо, що перша версія системи підтримувала простий механізм аудита, у той час як для наступної версії вимоги ускладнилися, і тепер необхідно змінити всі включення викликів старого типу на виклики нової системи журналювання.

Перевага АОП рішення в тому, що наскрізна функціональність аудита ізольована в межах аспекту. Всі зміни в типі й деталізації подій можуть бути з легкістю проведені не торкаючись основної функціональності.

Для прикладу розглянемо аспект, який буде перехоплювати всі виклики контролерів, визначаючи користувача і записувати інформацію в журнал.

Створимо новий аспект з pointcut, який перехоплює усі виконання контролерів і захоплює об'єкт запиту. Після чого на основі створеного pointcut напишемо before advice, в якості повідомлення на System.out.

Код аспекту:

```
package aop.example;

import org.aspectj.lang.*;
import javax.servlet.http.*;
```

```
                    // Аспект, що забезпечує аудит
public aspect AuditAspect {
            // Всі методи, які необхідно заносити до журналу
  public pointcut controllerMethods
(HttpServletRequest request) :
                // Виконання усіх методів,
                  //  параметрами яких є HttpServletRequest
                  // і HttpServletResponse
                // для всіх класів спадкоємців
 HttpServlet
                execution(* javax.servlet.http.
 HttpServlet+.*(HttpServletRequest,
 HttpServletResponse)) && args(request,
 HttpServletResponse);
  /**
   * Advice який забезпечує друк
 на стандартний вивід
   * інформації який метод буде
 виконано й користувача
   * який виконує метод
   */
  before(HttpServletRequest request)
 : controllerMethods(request) {
                // Збираємо статичну інформацію
                Signature sig = thisJoinPointStaticPart.
getSignature();
                // Виводимо доступну інформацію
                System.out.println("User " +
                        request.getSession().getAttribute
(EntranceFilter.USER_KEY).toString() +
                        " executing " + sig.
getDeclaringType().getName() +
                        "." + sig.getName());
  }

}
```

У даному прикладі стандартний вивід системи буде відображати рядки виду:

User Anonymous executing aop.example.LoginServlet.doGet
User user1 executing aop.example.ViewServlet.doGet.

Як бачимо з прикладу, аспектна методологія оптимізує код Web-додатків та підвищує їх захищеність.

Таким чином, аспектно-орієнтований підхід при правильному використанні може наступне:

- зменшити обсяг коду системи (отже, знизити ймовірність програмних помилок);
- поліпшити дизайн системи з погляду реалізації наскрізної функціональності;
- спростити систему завдяки локалізації коду, який не ставиться до основної функціональності;
- спростити тестування системи (можна тестувати різні аспекти окремо, а тільки потім ті, що вплетені в систему). Поліпшити керованість коду, як наслідок — простота еволюції й супроводу;
- збільшити кількість повторно використовуваних модулів завдяки слабкій зв'язаності підсистем.

Отже, на Web-порталах використовується багато різноманітних додатків і захищеність їх від атак зловмисників повинна стояти на першому місці при побудові системи захисту інформації порталу. Якщо в коді додатків є «діри», які можуть використовувати потенційні нападники, то ніякі антивіруси, брандмауери та VPN-мережі вже не допоможуть. Отже, велику увагу захищеності додатків треба приділяти на етапі їх написання. При використанні аспектно-орієнтованого методу програмування спрощується код, поліпшується дизайн системи (шляхом виділення наскрізної функціональності), зменшується обсяг коду системи і ймовірність програмних помилок, які можуть завадити зловмиснику.

Література

1. *Низамутдинов М.Ф.* Тактика защиты и нападения на Web-приложения. — СПб.: БХВ-Петербург, 2005. — 432 с.

2. *I. Kiselev*. Aspect-Oriented Programming with AspectJ. Indianapolis, IN, USA: SAMS Publishing, 2002.

3. *J. Hannemann, G. Kiczales*. Design pattern implementations in Java and AspectJ OOPSLA 02, New York, USA, November 2002. — P. 161—173.