6th International Conference on Applied Informatics
Eger, Hungary, January 27–31, 2004.

Component collaboration in Web environment

Richárd Jónás

Department of Information Technology, Institute of Informatics,
University of Debrecen
H-4010 Debrecen, P.O.Box 12, Hungary
e-mail: richard.jonas@tsoft.hu

Abstract

The ever increasing importance of Web applications demands an effective way to develop and maintain such applications. There are a lot of frameworks that support the creation of Web applications but they require the knowledge of the famous "yet another scripting language" in several cases. These languages are not roboust enough so a general purpose programming language is needed—such as Java—which is capable of describing general activities. With that language, because of its generality, we have to manufacture a great deal of code to make such tools accessible which are embedded in the script languages mentioned before.

Making Web components in Java 40-80% of the codelines of a class will be the same for many cases which results in difficult-to-maintain applications. These codes with common structure cannot be put into a superclass or a utility class because they crosscut several classes. Code generation is a partial solution because it results in the desired code but it is also hard to modify. In this paper, I give aspect-oriented coding strategies to solve the problem of some recurring patterns (e.g., getting the value of request parameters, database operations, etc.). I use the AspectJ programming language to implement the described solutions to exploit the byte-code compatibility with the standard Java byte-code.

Categories and Subject Descriptors: D.1.5 [Programming Techniques]: Object-Oriented Programming; D.2.13 [Software Engineering]: Reusable software D.3.3. [Programming Languages]: Constraints;

1. Introduction

As application development is a major part of information technology an efficient software development philosophy is needed. Object-oriented programming is

a suitable paradigm to make large systems but it lacks functionality-based technologies (structure of object oriented software is based on the responsibility of components but not on the role of them). Although applications are for functionality.

Component-based software development ([1]) is an efficient way to make software from reusable pieces — either source or binary — thus it gives us a development strategy which deals with the cost, flexibility, usability, development and documentation issues.

In this paper I will analyse the components from the aspect of their functionality and their usability concerns. At first some functionaly-based problems of the object-orientation will be introduced then this paper gives a possible solution of component collaboration of frequently changing component-based systems.

2. Components in a changing world

To make software from reusable parts is a very cost-efficient way to build large systems. It gives the possibility to build software from products of independent teams, besides, the teams can develop software modules simultaneously. One can make components to reuse them in other applications, others emphasize the usability over reusability. We can make source components, or buy third-party components in the component market. Some way or another it is worth using them for several reasons.

Object-oriented programming supports the representation of real-world things (called objects) but the OOP tries to capture the responsibility of real-world things rather than their roles in functionalities of the system. In the real world, objects are changed because of the change of requirements against them (and functionalities raise requirements).

2.1. Separating functionalities

In many cases it is more comfortable to collect the functions that implement the functionalities of the system, in this way we can split the system into two parts — a static and a dynamic part.

The static part resists most functional and collaborational changes of a system because it represents the real-world thing itself. Hence the dynamic part can be frequently changed in a process, which can be described by the concepts of extreme programming ([2]). Besides, the dynamic part can describe the usability of the components e.g., whether they take part in a functionality or not. It is important because we have to install the component into a subsystem or module, so the appropriate interface is needed to put the component into the right place.

There are component frameworks which try to capture the communication between the components, but it is impossible to describe all types of the communication. You can invent such a framework but its usability will be surely weak,

because of its generality. In that case it is more useful to give hints, patterns how the communication can be described.

How can we detach and put together the functionalities featuring the static and the dynamic part of the components? To split the components into the two mentioned parts, a careful and deep software analysis is needed to find out which things belong to the invariant which are the ones the object has to keep, and what belong to the functionalities that can frequently be changed.

Some questions have arisen: how can the parts be described and how can those parts be assembled. The static part can be written in Java, the dynamic part can be written as separated aspects.

2.2. The world of aspects

Aspects are concerns or functionalities of a system. Aspect-oriented programming ([3]) tries to solve the problem of cross-cutting ([4],[5]). Let us suppose that we have a system with several functionalities—which can be captured by use cases—and those functionalities are the result of the collaboration of many objects. So there will be one or two methods in several objects that call the methods of other objects. To modify such a functionality, we have to modify the source of several objects and we have to deploy all those objects of course. We say those functionalities cross-cut the objects. It is a very important concept of software development, because in general the reason we have to modify objects is that either the requirement or the functionaltity of the system are changed. Why is it so important? Because we have to schedule the work of many teams developing the same software, and one team manages only a few functionalities of the system, so we have to synchronize their work.

Aspect-oriented programming is a language-neutral paradigm but one of the best languages—with which it can work—is Java. There is a well-known programming language called AspectJ ([6]) which is based on Java and realizes the concepts of aspect-oriented programming.

3. Applications

As mentioned before, the static part of the system can be described by object-oriented tools and the dynamic part by aspect-oriented tools. In order to demonstrate the usage of aspect-oriented programming—in this way—some applications are shown.

3.1. Bean-JSP communication

The first is a very useful one: it solves the problem of getting the value of request parameters to use them by a bean. Let suppose that we have written a bean called `Bean` which wants to know the value of the request parameter `invoiceId`. To solve the problem let us try to reduce it. Let us define an attribute `parameterInvoiceId`

whose value will be the value of the request parameter mentioned before. Getting its value is the second problem to solve. So we have a class with the following source:

```
public class Bean {
    private String parameterInvoiceId;

    public void execute(){
        ...
        ResultSet r = statement.executeQuery("select * from" +
            "invoice where id = '" + parameterInvoiceId + "'");
        ...
    }
}
```

To denote that a class wants to know the value of request parameters, an interface has to be created through which one can get or set the request object.

```
interface RequestGetter {
    public ServletRequest getRequest();
    public void setRequest(ServletRequest request);
}
```

In order to solve the problem of getting the request parameter values, let us define an aspect called `ParameterGetter`. At first (denoting by (1) in the comment) the aspect declares that the class `Bean` implements the `RequestGetter` interface. Secondly, starting from line denoted by (2), objects implementing the request-getter interface define an attribute called `_rq` type of `ServletRequest`. Then those objects define the getter and setter method of that attribute. One can implement interfaces automatically in this way. It is quite efficient because one can easily give the property of getting parameters to a bean with the modification of line denoted by (1).

```
aspect ParameterGetter {
    declare parents: Bean implements RequestGetter;    // (1)

    private ServletRequest RequestGetter._rq = null;   // (2)

    public ServletRequest RequestGetter.getRequest(){
        return _rq;
    }

    public void Request.setRequest(ServletRequest r){
        _rq = r;
    }
```

```
    after(Bean b, HttpServletRequest request):          // (3)
        execution(public Bean.new(..)) &&
        this(b) &&
        cflow(
            execution(public void *.HttpJspBase._jspService(
            HttpServletRequest, HttpServletResponse)
        ) &&
        args(request)
    ){
        b.setRequest(request);
    }

    String around(RequestGetter rg):                    // (4)
        get(String RequestGetter+.parameter*) &&
        this(rg) {
        String attributeName =
            thisJoinPointStaticPart.getSignature().getName();
        String parameterName = attributeName.substring(9);
        return rg.getRequest().getParameter(parameterName);
    }
}
```

With line (3) an advice is started which means the following: after creation of a `Bean` under the control flow of the execution of the `_jspService` method of the `HttpJspBase` class let us set the request object to the bean. The topical object is denoted by `b` and the first parameter can be accessed by the indentifier `request`. According to the JSP specification ([7]) the main functionality of any JSP page is described in the `_jspService` method of the `HttpJspBase` class, so by the `cflow` pointcut one can check that something is under the control of a JSP page. One can define a context-dependent behaviour of a component with the pattern introcuded.

The main functionality of the aspect is defined in (4). Instead of getting the actual value of the attributes starting with the prefix `parameter`, the value of the request parameter whose name is the name of the attribute without the prefix will be provided.

3.2. Bean-database communication

It is conventional that a bean gets the value of its attributes from a database, and can synchronize those values with a database. To write the methods dealing with addition, modification and deletion is quite tiresome work—because they contain very similar codes—so one can solve the problem with copy-paste or code generation. It solves the problem of creating those methods but the generated codes are hard to maintain, especially by several developers.

Instead of generating those methods, let us try to write them in AspectJ. The main idea is the following: instead of writing the data manipulation methods one

by one, let us describe them formally and implement those methods generally. To achive this let us define the following methods of the class `Invoice`:

- `add_invoice(String _invoiceId, Timestamp createDate)`
- `mod_invoice(String _invoiceId, Timestamp createDate)`
- `del_invoice(String _invoiceId)`

The prefixes add, mod and del stand for insertion, modification and deletion, respectively. It is clear that the data manipulation SQL statements can be derived from the signature of methods, for example the insertion can be written as follows:

```
add_invoice(String _invoiceId, Timestamp create_date)
insert into invoice (invoice_id, create_date) values (?, ?)
```

In the case of modification a condition is needed that selects the row to be updated. To make such a condition we have to know which values belong to the primary key. Due to the lack of concept of metadata in Java[1], the parameters, the name of which starts with the underline, will be the parts of the primary key. This can be viewed as a "poor man's metadata" feature but this example also emphasizes the importance of the metadata feature.

3.3. Binary components

How can third-party components be deployed into a system? Let us suppose that a JDBC driver was installed into the system and we would like to measure its performance and try to detect its bottleneck. In that case we need the API of the driver and we have to select the suspicious points as join-points. With the corresponding pointcuts the execution times can be detected. It can be done with AspectJ because AspectJ can weave aspects into binary components (Java Archives). But it can be considered as an opportunity rather than a pattern because it requires a little "hacking", so the process of making such an aspect can be very long.

4. Conclusion

Component-based software development is a progressive part of information technology but it lacks several solutions. At first it is not as powerful as in mechanical engineering where the industry tries to follow some standards in order to fabricates components that can be assembled. There are many concurrent software platforms which try to find the best solutions in order to sell them. I do not want to judge which systems are good or bad because it depends on several things. I would rather give some programming hints that help to solve the communication problem between components.

[1]During the research only the Java 1.4.2 was available in stable release.

During developing a web-application it is worth creating components which can discover their context, can adapt to other components (throw an exception when some of the run conditions are not satisfied), can be configurable, etc. Those properties make the component useable between contexts, so we have to write them only once.

I have been developing a web-application for the MythoLogic project, the product of which is a portal system with which one can build portals in a flexible and dynamic way. The patterns mentioned before were built into that system and it seems that it is worth using them. The source of the system is quite large, namely 4 megabytes. It contains about 50 database tables that can be maintained by the web-application. So it requires at least 50 types of business components which have to communicate in order to create transactions. Being a web-application, its requirements are frequently changed, so the communication graph of the components is changed again and again. But one can change the functionalities easily because the essence of the transactions are described in aspects.

Of all the aspect-oriented systems I chose the AspectJ because of its expressive pointcut system. Moreover, those languages are not so mature, so they are developed quickly and often contain new tools. New tools always give new thoughts.

References

[1] Component Software: Beyond Object-Oriented Programming, Clements Szyperski, Addison-Wesley, 1998.

[2] Extreme Programming: A Gentle Introduction, `http://www.extremeprogramming.org/`

[3] Aspect-Oriented Software Development Community, `http://aosd.net`

[4] Aspect-Oriented Refactoring: Part 1, Overview and Process, Ramnivas Laddad, `http://www.theserverside.com/articles/article.jsp?l=AspectOriented`

[5] Aspect-Oriented Refactoring: Part 2, The Techniques of the Trade, Ramnivas Laddad, `http://www.theserverside.com/articles/article.jsp?l=AspectOriented`

[6] AspectJ Project, `http://eclipse.org/aspectj`

[7] Java Server Pages, Sun Microsystems, `http://java.sun.com/products/jsp/`

[8] Java 2 Standart Editon, Sun Microsystems, `http://java.sun.com/j2se/1.5.0/`

[9] Java Specification Request: A Metadata Facility for the JavaTM Programming Language, `http://www.jcp.org/jsr/detail/175.jsp`