RESEARCH PAPERS OF THE WROCŁAW UNIVERSITY OF ECONOMICS
No. 8 (1208) 2008
Advanced Information Technologies for Management – AITM 2007

Cezary Hołub, Mieczysław L. Owoc

Wrocław University of Economics, Poland

ASPECT ORIENTED PROGRAMMING AS A NEW APPROACH TO SOFTWARE ENGINEERING

Abstract: The main goal of this article is to present new technology software engineering which expands Object Oriented Programming. This technology is called Aspect Oriented Programming (AOP). In general aspect programming is a paradigm of software engineering which assists the separation of concerns and helps to divide software in independent functional parts. This approach gave us better business requirements mapping development phase in software. Gregor Kiczales and his team at Xerox Corporation originated the concept of AOP in 1996. This team also developed the first and most popular AOP language, AspectJ in 2001. AOP was introduced to address crosscutting concerns such as security, logging, persistence, debugging, tracing, distribution, performance monitoring, and exception handling in a more effective manner. Unlike conventional development techniques, which scatter the implementation of each concern into multiple classes, aspect-oriented programming localizes them. AOP attempts to aid programmers in the separation of concerns, specifically cross-cutting concerns, as an advance in modularization. AOP improves the modularity of programmes, making the code much closer to the design. It can dramatically reduce the time taken to implement common features and functions, improve quality, and integrate old solutions with our product. AOP can give us better and cheaper computer systems. Such IT solutions are needed for the growing information society. Specific to this kind of society is the central position information technology has for production, economy, and society at large. Information technology AOP's core idea is to separating the business logic in an application from the common services that support it. Aspect programming will be probably so important technology as object programming is now.

1. Introduction

There is still in software engineering a need to develop more and more flexible technologies. They are able to support changing business challenges. The main goal of this paper is to present a relatively new technology which expands Object Oriented Programming. This technology is called Aspect Oriented Programming (AOP). Generally speaking, aspect programming is a paradigm of software engineering which assists separation of concerns and helps to divide software in independent functional components. This approach gives users better business requirements mapping in software development phase. Gregor Kiczales and his team at Xerox Corporation originated the concept of AOP in 1996. This team also developed the first and most popular AOP language, AspectJ five years later. AOP was introduced to address crosscutting concerns such as security, logging, persistence, debugging, tracing, distribution, performance monitoring, and exception handling in a more effective manner. Unlike conventional development techniques, which scatter the implementation of each concern into multiple classes, aspect-oriented programming localizes them. AOP attempts to aid programmers in the separation of concerns, specifically cross-cutting concerns, as an advance in modularization. AOP improves the modularity of programmes, making the code much closer to the design. It can dramatically reduce the time taken to implement common features and functions, improve quality, and integrate old solutions with our product. AOP can give us better and cheaper computer systems. Such IT solutions are needed for the growing expectations of information society. Specific to this kind of society is the central position information technology has for production, economy, and society at large. Information technology AOP's core idea is to separating the business logic in an application from the common services that support it. Aspect programming will be probably so important technology as object programming is now.

The crucial problem presented in the paper is focused on stressing the essential features of the approach and its confrontation with chosen software engineering streams of progress. The paper consists of the following sections. The first one presents the history and the background of the whole idea. The main reason of AOP usability is the separation of functionally independent components called aspects. The next part of this article shows a typical model of aspect programming and its implementation as Java "sub-parts" No doubts, the same general concepts of object oriented programming can be achieved. The third section gives some examples of usability of the described approach in practice. Banking and marketing applications confirm the specialty of this philosophy and prove new potential quality effects in such a context. The fourth part is devoted to setting AOP as a real supportive branch in software engineering progress. The analysis of current research in this domain and comparison of solutions in object-oriented approach is presented. The final conclusions end the paper.

2. Genesis of Aspect Oriented Programming

Information society – the term meaning such society that information becomes the basic market product more valuable even from material goods. The evolution of informative society is based on information technologies. Information technologies make possible processing and storing information across development of computer systems. The quality of used information technologies has a key impact on quality of information. Information systems "from always" played very important part in management of organisation. Their significance is proved by projects devoted to the development of usability information technologies models which in majority information technology is not only computer infrastructure supporting making decision processes but also the necessary condition of formation of long-wave strategy of organization development.

The paper presents one of these information technologies, namely **Aspect Oriented Programming**, AOP. In 2001 the editors of January/February "MIT Tech-nology Review" chose ten newly formed areas of technology, which soon – according to the authors – will have essential influence on economy as well as a way that people live and work. Aspect oriented programming was selected as one of such areas apart from biometrics, data mining, robotics or natural language processing. According to the authors, we need about 15 years to implement AOP as a standard in the commercial production of software.

We tend to expect computer systems should be more and more effective. They should less and less fail and be more simple from the constructing point of view and to solve complex problems of the real world. We may observe the tendencies of larger sizing of software products. With this increasing software complexity grows dramatically. It refers to all aspects of their functionality including management of resources, security or recording the events. From an economic point of view, system's division on several computers acting independent seems to be rational. Ideally, particular parts would communicate one with another using interface only, however, each would be treated as a black box. This is very difficult to produce large business systems in practice. We can reach just some level of system's modularisation. The higher level of system's modularisation assures fewer emergencies which are easier in maintenance and more transparent. Enlarging the level of system's modularity in design phase simplifies the cost of evaluation and time necessary to perform the system. Certainly, a system produced in such a way will be more useful for customers.

In systems with the low degree of modularisation all potential changes in software impact huge risk of generating new errors, not saying about costs of such changes. Nowadays the considerable majority of software is implemented with supporting object-oriented languages. Object-oriented approach helps to understand better the considered problem (objects reflect the elements of the real world). Created elements can be used many times and they can be adapted in new environments. Unfortunately, such an approach has also some disadvantages. In case of large applications created programming code is not quite transparent and well understood. On base the procedural and object-oriented programming, assuring the large modularity of projects, the concept of aspect-oriented programming is still developing. Its main goal is the improvement of so-called "separation of the concerns" that is distributing of certain aspects of programmes functionality (for example: the synchronisation of accessing the resources or tracing the programme execution) using separate, independent modules. It seems to be desirable that modularity of a system should better reflect what we think about a problem than how to solve it via computer tools. In case of complex systems the object-oriented programming does not allow for the modularisation of all system's quests. Some problems will always crosscut boundaries of different modules.

The idea of aspect-oriented programming is the delivering of mechanisms permitting on full system's modularisation. It should simplify a programming code, enlarge the speed of its creation and make development understanding, preservation and reuse easier. Modularised crosscutting concerns are called **aspects** [Słowikowski 2007].

A lot of decades were devoted to searching panacea on the perfect modularisation of computer systems. The aspect-oriented approach is a result of a lot of research in the fields of new methods programming which concerns on the limitations of analysis, design and programming of complex problems. One should be stressed in AOP methods of separation of problems' result of modularisation level growing and the same information systems could be cheaper and more reliable. Professor Gregor Kiczales from the university in Vancouver is "a father" of AOP thanks to his research in the 90s concerning an essence and principles of aspect oriented programming. The term Aspect Oriented Programming was introduced in 1996 during his work in the Xerox Corporation. The concept was accepted by a lot of people. Several investigations devoted to its applications in many areas started that time and the first conference on that subject took place in parallel [Walter 2007]. The paper "Aspect – Oriented Programming" [Kiczales 1997] written by Gregor Kiczales in 1997 is recognised as the first publication about aspect programming.

3. Aspect Model and Programming Language AspectJ

Aspect-oriented approach gives us new possibilities of modularisation. This is very important in large complex systems where insufficient level of modularisation is quite difficult. Like structural and objects techniques give a software developer a bigger level of modularization and a possibility of keeping big systems, AOP goes further in this matter. A computer system created with aspect techniques is friendlier for sudden changes in architecture or in creating a new line of product. Moreover, this makes possible cost and error reduction, good cohesion of long life cycle of product. Each support for better concerns separation in computer systems is measurable profit for a company. These advantages have important meaning for an organisation which applies complex computer systems and because of that the problem of crosscutting concerns is very popular there. A company from finance sector like banks or insurance companies are classical customers for new technologies because of their specificity [Bodkin 2003].

In the beginning phase a lot of applications of aspect approach were mainly created by innovative developers or system engineers which persuade this solution at management. However, for commercial success this technology is necessary that management should see advantages and needs for using it. We might assume that following requirements should fulfil to AOP on good settled in computer systems [Bodkin 2003]:

- business must know that this technology gives big advantage according to alternative solutions,
- business should precisely present reports from the application of this technology in already working systems,
- business must present clear future of this technology and companion technology to invest in AOP,
- business should be supported in IT industry for the part of software vendors, consultants and system integrator vendors.

Probably each software developer comes across a certain problem. His application consists not only in the implementation of its main target, but also other companion issues (we might say: orthogonal according to the main target). For example: the source of business application does not consist only in logic (e.g. they calculate the sum for shopping in an Internet shop) but also in implementation additional functionalities which are mentioned before logging or authorization should be applied. This is obvious normal behaviour which is the implication of the concerns of complexity of client requirements to design a system. A customer for sure does not accept an application which required orthogonal issues not to be implemented. This situation as described before is natural. The problem is in another issue: in the way how to transmit the number of orthogonal aspects of client requirements to make implementation concrete (or to design in a previous phase of software development).

The example of such a situation is also the part of banking system (showed at Fig. 1). The object Bank holds the collection of bank products: Accounts, Credits, and Deposits. Further Bank holds the objects of Report class which work on bank products: It is easy to see that every method of these classes needs common properties which are not connected with functional object division e.g. transact methods invocation. The trial of implementation of such a feature by invokes in all classes to source code which realized it might conduct to the mess and might reduce the code of maintenance [Walter 2007].

An object-oriented programme, where this problem is implemented, for sure will invoke in many places the methods to perform individual system aspects. In result it looks like spaghetti (see Fig. 2), in which the idea of modularisation is completely unclear. An aspect-oriented programming can solve this problem in more transparent and smart way. For a typical object mechanisms add the idea of aspect which grouping issues cut another fragment of code. Thanks to this it is possible to divide them logically in a separate modularised unit without touching the original separation giving legibility code, encapsulation etc. Classes representing business issues still cover only what they are responsible for. Aspects using classes make possible better multi-criteria decomposition computer system than using classes only [Walter 2007].

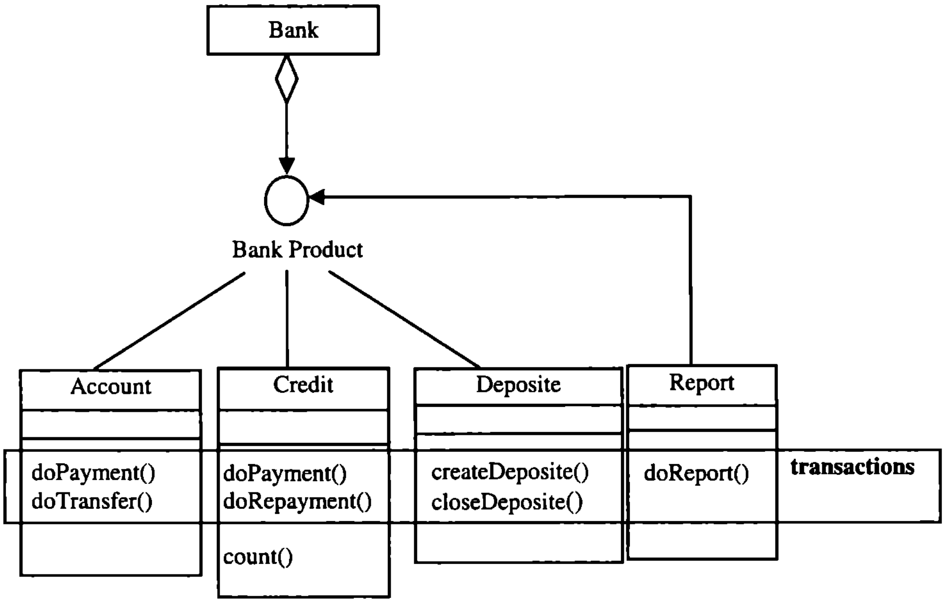

Figure 1. Part of banking system
Source: [Walter 2007].

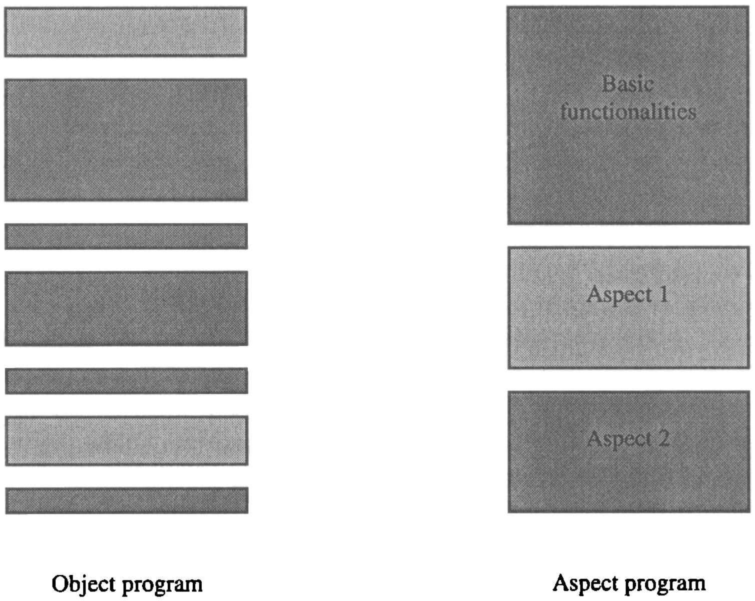

Figure 2. Spaghetti code and code well modularized on functional part
Source: [Walter 2007].

The authors of aspect approach notice that software design process and programming language are closely connection. As far as design process might be perceived as the process of extraction from the system of smaller and smaller modules, the programming language is a mechanism which makes possible to connect them and create more complex subsystems. The object and procedural programming have a common future that in a design phase we extract autonomous part from, and leave a developer how to join them. Aspect programming depends on extract templates called aspects. Aspect might be e.g. standard error handling, matrix representation or memory allocation. Trying to write an aspect in object representation is very inconvenient because self value (concrete instance of an aspect) depends on elements (components) on which it works. The connection aspects with components give us something which might be object or function according to object or structural approach. In result aspect approach gives us the clear representation of orthogonal aspects of client requirements on an orthogonal aspect of implementation [Stochmiałek 2007].

As we described above in developing programmes we should use better and better technologies for creating software. New technologies allow to save time and money and they also allow to pass competition. Computer systems become better and better for growing information society.

4. Aspect Oriented Programming versus Object Oriented Programming

The paradigm that dominated the world market of software is object-oriented programming. The difference among programming object-oriented (OOP) and aspect-oriented programming (AOP) does not rely on different aims (in both cases is about assembling the similar concepts and the separation differences), but in different selection of tools. A concept of class, encapsulation and heritance are basic tools in object-oriented programming. Typically, they permit to apply the assembling according to one criterion conception, which in some cases seems to be sufficient. It is the advantage of object-oriented programming which strengthens it and ensures its position on the market, as well as supports in wide scale its popular programming languages. The aspect programming, therefore, does neither stand in contradiction with foundations nor the tools of object-oriented programming. We can say that aspect-oriented approach is the super-set of object-oriented approach [Walter 2007].

The main assumptions of object-oriented paradigm are:

- the assembling of similar concepts and heritance,
- a class is a basic individual modularisation category,
- encapsulation allows for hiding the implementation of behaviour object implementation as well as it guarantees that a change of state of object can happen in result of revoking object method only,

- polymorphism – makes the development of particular behaviour for given object possible, (hidden under references),
- heritance – simplifies the process of defining class hierarchy. In new classes (subclasses) we may recall data and behaviour definition from superior classes.

Additionally, aspect programming offers new features:

- assembling of similar concepts in unrelated classes,
- specific mechanism of modularisation, namely aspect.

As a result aspects are responsible for:

- connection of assembling behaviour with defined crosscutting problem,
- qualification of principles of applying this behaviour in applications,
- suitable actions and circumstance of their executing,
- separation of crosscutting problems from business logic.

Object-oriented programming is more mature technology and formally defined. A lot of tools were created (for example Microsoft Visual Studio, Eclipse, Borland Together) supporting in object-oriented languages the development of information systems. Unified Modelling Language (UML) was defined as a standard that facilitates the building of modern information systems. An object-oriented paradigm is widely implemented and well documented. But this approach has also weak points. As technology existing over thirty-years it creates some problems in certain circumstances and additional efforts should be performed. Some examples where object-oriented paradigm does not fit the modularisation of user requirements in an easy way are [Szała 2006]:

- durability of data,
- authorization,
- authentication,
- transactional cohesion,
- recording events in journal information,
- multithreaded synchronisation,
- remembering (caching) the results of long-lasting operations for example,
- pooling (e.g. connection to a database),
- the assurance of correctness of data (checking the arguments of function for example).

They are typically sectional problems that occur at the same time in many modules of a programme and in extreme cases even in all (exposed in Fig. 1). Therefore, this approach can be evaluated very high – there are more and more mature solutions representing this technology.

5. Conclusions

The final findings of the paper can be expressed in the following way:

1) object-oriented programming apart from many advantages is not free from weak points, especially in modularisation,

2) aspect-oriented programming offers new features that overcome difficulties existing in the object oriented approach,

3) more flexible way of software engineering development proposed in AOP meet information society requirements; new dimensions of projects can be served better [Wyrwał 2007].

Aspect oriented programming as a new trend in software engineering will be intensively implemented and on the other hand some challenges can be observed. It is worth stressing that AOP fills the main expectations of software engineering in more flexible and efficient way and this approach can be regarded as a very important step in the production of software. This is most probable that this technology will stand up for some time so popular as an object-oriented approach is today.

References

Bieniasz S. *Programowanie aspektowe AOP*, retrieved February 20, 2008 from https://home.agh.edu.pl/~olekb/ wyklady/aop.pdf.

Bodkin R. *Commercialization of AOSD: The Road Ahead*, retrieved February 20, 2008 from http://www.jpmdesign.de/conferences/aosd/2003/papers/AOSD_Commercialization_Position_2003_ final. pdf.

Carver L. Next steps for Commercializing AOP, retrieved February 20, 2008 from http://www.jpmdesign.de/ conferences/aosd/2003/papers/Carver.pdf.

Colyer A., Andy Clement A., Harley G., Webster M. (2004), *Eclipse AspectJ: Aspect-Oriented Programming with AspectJ and the Eclipse AspectJ Development Tool*. Addison Wesley Professional.

Jacobson I., Ng P.W. (2004), *Aspect-Oriented Software Development with Use Casus*. Addison Wesley Professional.

Kiczales G. AspectJ Overview, retrieved February 20, 2008 from http://www.cs.ubc.ca/~gregor/papers/kiczales-ECOOP2001-AspectJ.pdf

Kiczales G. Aspect-Oriented Programming, retrieved February 20, 2008 from http://www.parc.com/research/projects/aspectj/downloads/ECOOP1997-AOP.pdf.

Maciaszek L.A., Liong B.L. (2004), *Practical Software Engineering*. Addison-Wesley.

Massaro A., Rosendahl J. *Agile Programming*, retrieved February 20, 2008 from http://www.idt.mdh.se/kurser/cd5130/msl/2005lp4/downloads/reports/agile_programming.pdf.

Miles R. (2005), *AspectJ Cookbook*. O'Reilly.

MIT Technology Review, retrieved February 20, 2008 from, http://www.technologyreview.com/ magazine/toc/58/.

Parnas D. *On the Criteria to Be Used in Decomposing Systems into Module*, retrieved February 20, 2008 from http://www.acm.org/classics/may96.

Słowikowski P. *Programowanie aspektowe*, retrieved February 20, 2008 from http://portal.ics.agh.edu.pl:8001/ papers/TR-01-1.pdf.

Stochmiałek M. *Wprowadzenie do programowania aspektowego*, retrieved February 20, 2008 from http://misto.e-informatyka.pl/papers/aop-intro.pdf.

Stochmiałek M. *Programowanie aspektowe: studium empiryczne*, retrieved February 20, 2008 from http://misto.e-informatyka.pl/papers/aop-thesis.pdf.

Szała Ł. *Programowanie aspektowe – wpływ metod testowania na jakość kodu i produktywność programisty w kontekście programowania aspektowego.* Praca magisterska. Politechnika Wrocławska, Wydział Informatyki i Zarządzania, Wrocław 2006, http://szala.eu/www/pub/aop-thesis.pdf.

Tworzenie aplikacji J2EE w oparciu o Spring Framework, retrieved February 20, 2008 from http://www.ploug.org.pl/ szkola/szkola_5/materialy/7_Spring.pdf.

Vanderperren W. *Combining aspect-oriented and component-based software engineering*, retrieved February 20, 2008 from, http://ssel.vub.ac.be/Members/wvdperre/thesiswim.pdf.

Walter B. *Advanced object design – aspect programming* (Polish translation: *Zaawansowane projektowanie obiektowe – programowanie aspektowe*), retrieved February 20, 2008 from, http://wazniak.mimuw.edu.pl/images/e/ea/Zpo-12-wyk.pdf.

Wampler D. *The future of aspect oriented programming*, retrieved February 20, 2008 from http://aspect programming.com/papers/The%20Future%20of%20AOP.pdf.

Wampler D. *Aspect-Oriented Design Principles: Lessons from Object-Oriented Design*, retrieved February 20, 2008 from, http://www.aosd.net/2007/program/industry/I6-AspectDesignPrinciples.pdf.

Wyrwał S. *Przy okazji zrób jeszcze to... czyli Programowanie aspektowe i Aspektowo zorientowane wytwarzanie oprogramowania"*, retrieved February 20, 2008 from http://www.ploug.org.pl/plougtki.php?action=read&p=32&a=8.