THE ASPECT-ORIENTED DEVELOPMENT OF CONCURRENT SYSTEMS

Victor BEŞLIU[1], Dumitru CIORBA[1], Anthony T. Chronopoulos[2]

[1] Technical University of Moldova, Information Technology Department
168, Stefan cel Mare, Chisinau, Republic of Moldova

San Antonio University, Texas, USA

The concurrent systems have some functionality that are not or cannot be localized in separate structural units because of classic approach of abstractization and modulation. This is realized through system's components, increasing the interdependence between structural units (procedures, functions, objects), that makes difficult their adaptation and reusability. The division of these functionalities in separate structural units is, of course, an important process but very difficult for concurrent systems, especially for those with evolutionist trends. In the work, there are specified the techniques that assure the aspect-oriented development of concurrent systems and the problems of this new technology.

Keywords: aspect-oriented programming, concurrent systems, crosscutting concepts

1. INTRODUCTION

Each system is determined by its functions that correspond to some basic requirements and to some general requirements of the system, named *functional*. If we consider that each requirement determines a dimension of the problem, than a problem can be represented in a multidimensional space, as in figure 1. But at the actual moment, the elaboration of systems is based on an abstractization and modulation in a single space that of solution, determined by the used technology and by the basic requirements. This approach surely leads at an imprecise reflection in realization or to an increase of the difficulty of maintaining of functional requirements.

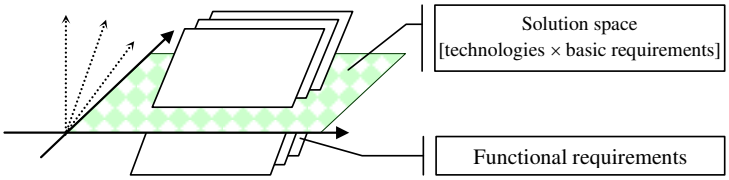

Figure 1. Multidimensional space of the problem

2. DEFICIENCY IN TRADITIONAL REALIZATION

In [5] there are described the criteria after which we can identify the problematical realization of functionalities. The criteria are referred especially to the source code, which can be:

1. **tangled**, when a modulus realizes more requirements. For example, business-logic of the system often is realized together with synchronizing and/or security code, that leads to a tangle in the code.
2. **scattered**, when the appeal of the functionality is scattered through the system, and the functionality itself is not localized in a separate modulus. For example, if the system has a function of following of access to the databases, then such functionality will be implemented in all the modulus that work with databases.

The consequences [11] resulted from the methods of projection and realization and widely used in actual software engineering can be the following:

1. **the bad understanding of the code**: the realization of more functions in a single modulus does not permit the unique establishment of functionality-modulus relation;
2. **the impossibility of reusability of the code**: a modulus cannot be reused if, besides the base logics, it contains other functionalities which are not necessary in another context;
3. **the difficulties in maintaining**: the modification of the requirements implies the modification of the functionalities realized through the system, the pursuit of the effects being very difficult;
4. **the large probability of appearance of errors**: according to the definition, there are possible sources of appearance of errors and the understanding of the system's code is difficult.

For the realization of functionalities one can resort to mix-in classes [1][10], patterns [6] and frameworks [3][4] (in general, object-oriented techniques), but their usage can be improved only through the removal of the inherited anomalies [7] and through a natural and explicit utterance (expression) of functionalities, that offers only an aspect-oriented development.

3. THE ASPECT-ORIENTED PROGRAMMING

At the end of the 90th there could not be discussed an aspect-oriented development, but only the appearance of some ideas of removal of defects of a special nature resulted from the complexity of systems and from the techniques offered by OOP. These ideas had been materialized through the

concepts of aspect-oriented programming suggested by Kiczales and his team [5].

AOP introduces the concepts **aspect** and **weaving** in order to treat unforced, such named **crosscutting** functionalities, the usage of which is scattered through the entire system. Another aspect is a structural element of program construction resulted from the system's modulation and which encapsulates states and behaviours, proper and/or of other elements. Weaving is a systematic process of combination of aspects with basic components (figure 2) and can be realized by a pre-processor, an interpreter or a compiler.

The determination of criteria of decomposition [9] is essential for the process of software development, but the question remains open: **how do we separate the units** in such a way that we could obtain a better understanding of the system, an easier fitting, a maintaining and reusability at a high level.

The OO techniques of decomposition had been asserted in time, but the nature of concurrent systems is much over their possibilities, because it is demanded a decomposition with more dimensions, one for each aspect.

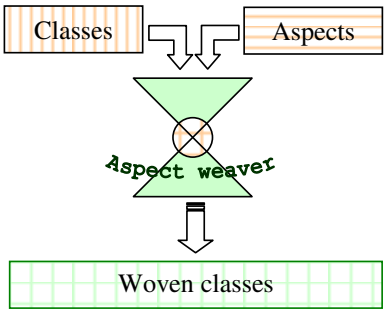

Figure 2. Aspect-weaving (aspects + classes)

It is also important to understand that the aspect-oriented programming must not be underplaced to object-oriented programming; it comes to retain its advantages and to offer a better separation of functionalities.

The evolution of AO techniques was determined and sustained by different university and research centers, each of them having a proper point of view over the modalities of implementation of AO principles which consist of techniques of aspects' separation, techniques of sliding of these aspects in language structures and instrument that would assure the understanding process. Although, a unique point of view does not exist, most of them correspond to AOP general principles. A concise list [8] of aspect-oriented techniques is presented in table 1.

Table 1. Aspect-oriented techniques

Nr.	Name	General description
1.	Meta-object protocol	Proposes a new level of abstractization for existent OO systems, and a program consisting of two parts: the first consisting of a usual code, and the second, of a meta-language code, which describes some additional behaviours of the basic program (logging, security, etc.). The technique realized in CLOS (Common Lisp Object System).
2.	Reflective programming	The general principles correspond to the system of B. Smith which is a framework that consists of a „tower" of interpreters, and each interpreter is an interpreted program of high

Nr.	Name	General description
		level (interpreter). It may be considered as a generalization of MOP technique.
3.	Adaptive programming	It is meant for the creation of evolutive software systems which are adapting to the context changes. The context may be behaviour, structures of synchronizing, structures of migration, etc. The adaptation is realized through a partial definition of the class, and the defining of a new relation is realized through the attachment of a new method, of course previous described in accordance with the proposed techniques. The technique realized in Demeter/C++.
4.	Subject-Oriented programming	This is an approach to building of OO software systems by composing several sub-systems, which are known as subjects, according to a composition expression which describes the rules that the subjects correspond to, and how should they be merged to implement required system functionality.
5.	Composition filters	The basic object model is extended modularly by introducing input and output composition filters that affect the received and sent messages, respectively. Composition filters have the following important characteristics modular extension, orthogonal extension, open-ended, aspect-oriented, and declarative.
6.	On-demand re-modularization	It is based on multi-dimensional separation of concerns. The IBM team has developed HyperJ that supports dividing components into independent modules known as hyperslices, which can be dynamically combined to form a ready application. The hyperslices consist of a standard code, but also of a set of configuration files that define each slice's behavior.

4. ESSENTIAL HINTS IN AO DEVELOPMENT OF CONCURRENT SYSTEMS

The concurrent systems are determined by a series of imperatives which must be taken into account in the process of development. These imperatives result from the nature of concurrent systems, from fundamental principles of elaboration of applications and/or from the specific of used technologies.

In order to have an AO development, the technology must offer [2]: the separation and localization of aspects, weaving mechanism, mechanisms of release of aspects and interaction.

From the fundamental principles of elaboration results that the software system must have the following properties that imply a better understanding:

1. **Low coupling and reusability**. Assures the quality, fastness and reduction of costs of elaboration through reusability of aspects and components. It is considered that the reusability may be applied to all phases of life-cycle of the system.
2. **Usability and maintainability.** There are factors to be ignored, which can determine the system's success in the last instance, especially which the system's study and maintaining are obligatory stages in the life-cycle of the system.

Authentication, fault tolerance, load balancing, scheduling, synchronization, testing and verification are functionalities due to which are realized the general objectives (numbered below) of the concurrent systems:

1. **Cooperation and competition**, which conveyed only through OO techniques, do not assure entirely the fundamental principles of development of software systems.

2. **Protection,** which comprises the integrity and the selectivity of the access. In the perspective of usage of AO techniques it is very important to understand the nature of appearance of some new imperfections in the system resulted from the logics of aspects, mechanism of blending or from the interaction aspect-class and/or aspect-aspect.
3. **Performance**, especially determined by the debit and time of answer which may be considered a crosscutting functionality.
4. **Adaptability and opening**, are important characteristics for the concurrent systems which are not obligatory guaranteed by OO techniques.

5. CONCLUSIONS

The development of concurrent systems is a long and difficult process which must correspond to some specific requirements of the given systems. The reusability is a modality of getting easier the elaboration, but the nature of concurrent systems determines the appearance of some anomalies described in many works.

Beyond doubt, the aspect-oriented programming is a solution to many problems appeared in the development of concurrent systems. But, is this solution efficient and sufficient mature to be used on the industrial stage? At the actual moment, we don't think that anybody can surely contend something.

There exist many works dedicated to the aspect-oriented development, also exist many instruments which somehow realize the principles of this technology, but however, that what is named **Aspect-Oriented Software Development** is on the stage of formation, because:

- It is not clear how do we analyse the aspects: it does not exist an unique standard of notation in UML;
- There are no effective methods of verification and testing of aspect-oriented programs;
- It is not implemented in all spheres of popular development;
- The IDE that still realize AOP do not assure the incremental compilation.

BIBLIOGRAPHY

1. Bracha G., Cook W., *Mixin-based inheritance*, In Proceedings of ECOOP, 1990
2. Constantinides C. A., and Elrad, T., *On the Requirements for Concurrent Software Architectures to Support Advanced Separation of Concerns*, OOPSLA, Workshop on Advanced Separation of Concerns, 2000
3. Fayad M., Schmidt D. *Object-Oriented Application Frameworks*, In Communications of the ACM, 40(10), 1997
4. Johnson R., *Frameworks=(components+patterns)*, In Communications of the ACM, 40(10), 1997
5. Kiczales G., Lamping J., Mendhekar A., Maeda C., Lopes Cr., Loingtier J.M., Irwin J. *Aspect-Oriented Programming*, In Proceedings of ECOOP, 1997
6. Lea D., *Concurrent programming in Java. Design principles and patterns*, http://gee.cs.oswego.edu/dl/cpj/, 2000
7. Matsuoka S., Yonezawa A., *Analysis of inheritance anomaly in object-oriented concurrent programming languages*, In Research Directions in Concurrent Object-Oriented Programming, 1993
8. Mickelsson M., *AOP compared to OOP when implementing a distributed, web-based application*, Master of Science thesis, Uppsala University, 2002
9. Parnas D. L., *On the criteria to be used in decomposing systems into modules*, In Communications of the ACM, 15(12), 1972
10. Strandh R., Durand Ir., Traité de Programmation en Common Lisp, http://dept-info.labri.u-bordeaux.fr/~strandh, 2003
11. Павлов В., *Аспектно-ориентированное прграмирование. Анализ вариантов применения аспектно-ориентированного подхода при разработке программных систем*, http://www.javable.com/columns/aop/, 2003

Kommentar [d2]: Accesibil pe http://bracha.org/mwp.htm

Kommentar [d3]: accesibil pe http://www.cs.wustl.edu/~schmidt/CACM-frameworks.html

Kommentar [d4]: accesibil pe http://www.idi.ntnu.no/grupper/su/courses/dif8901/papers2003/P-r22-johnson97.pdf

Kommentar [d5]: Accesibil pe http://www.parc.xerox.com/csl/groups/sda/publications/papers/Kiczales-ECOOP97/

Kommentar [d6]: Accesibil pe http://citeseer.ist.psu.edu/rd/28377816%2C71245%2C1%2C0.25%2CDownload/http://citeseer.ist.psu.edu/cache/papers/cs/3223/ftp:zSzzSzcamille.is.s.u-tokyo.ac.jpzSzpubzSzpaperszSzbook-inheritance-anomaly-a4.pdf/matsuoka93analysis.pdf

Kommentar [d7]: Accesibil pe http://labs.cs.utt.ro/labs/ip2/html/lectures/1/res/Parnas-CriteriaForDecomposingSystems.pdf