ON ADAPTIVE PRINCIPLES IN SOFTWARE SYSTEMS EVOLUTION

Jana BANDÁKOVÁ, Ján KOLLÁR
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice
E-mail: jana.bandakova@tuke.sk , jan.kollar@tuke.sk
Tel: 095/602 4148, 095/602 4148

SUMMARY

This paper deals with a problem of software system evolution. When the software system is developed the programmer tries to transform his ideas about final properties of the system in running application using some programming paradigm. Properties of the software system have an important role not only in development process but also in later phases of lifecycle, especially when the modification of an existing system is needed. Therefore it is necessary to express them formally as clearly as possible. When properties are expressed unambiguous then the behavior of the system can be controlled and modified effectively. In our research project we concentrate on formalizing the development process in form of aspect weaving and try to make the process automated. In addition, when the system is complex and complicated then it is appropriate to express it using multiple paradigms so we try to analyze the relations between functional, imperative, object-oriented and aspect-oriented paradigms that have crucial role for our goals. As can be seen, the principles of adaptive programming can be used in solution of software system evolution.

Keywords: *system properties, multiple programming paradigms, process functional paradigm, adaptive programming*

1. INTRODUCTION

An evolution of software system starts already in human mind in the form of abstract ideas. The programmer tries to express these abstract ideas in the form of running software system using one of an existing paradigm. In order to develop such system he follows some progresses that can be reused in the following development as well as in later phase of maintainance and evolution. Nowadays, the main problem in this area is how to express these progresses in formal way, how to express in formal way the properties that system should satisfy and how to make an effective transformation to an implementation. A multiparadigm approach seems to be an appropriate way not only at the implementation stage but also at the specification stage of software system development. In solution of this problem also the principles of adaptive programming can be useful.

2. MULTIPARADIGM APPROACH

An important role in process of software system development and evolution has a way in which the problem is solved. Different approaches that are used to solve the problem are defined by different paradigms.

While simple programs can be effectively expressed using only one paradigm for example procedural, functional, object-oriented paradigm etc., large and complex software systems are usually decomposed into smaller and easier problems and require the use of different points of view to solve these particular problems. This means, the use of different paradigms in the sense of their combination and integration. The principles of paradigm composition are described in [4] The use of different paradigms should contribute not only to a better and reliable software system development but also to a later process of evolution of an existing system.

A multiparadigm approach seems to be an appropriate way to achieve this goal [1, 2]. Nowadays, there are many programming paradigms and languages that are used to solve particular problems in different ways and from different points of view. The existence of these paradigms and the fact that every paradigm has its advantage against the other leads to their combination and integration in order to develop a multiparadigm language. The advantage or disadvantage of particular paradigm depends on problem that will be solved because not every paradigm is appropriate to solve a certain problem. Multiparadigm programming languages have been envisioned as a vehicle for constructing large and complex heterogeneous systems [11].

This work was supported by VEGA Grant No. 1/4073/07 Aspect-oriented Evolution of Complex Software Systems.

3. TAKING ADVANTAGE OF PROCESS FUNCTIONAL LANGUAGE

Multiparadigm approach leads to two main goals:

1. to overcome the specific limitations of each paradigm and
2. to take advantage of the most useful characteristics of each one through their combination

For our purpose of software system development and evolution a multiparadigm process functional language [2, 5] is be used. This language integrates and combines suitable properties of functional [6, 7], object-oriented, imperative and nowadays also aspect-oriented language [8, 9]. Generally, aspect-oriented paradigm has been built as an extension to object-oriented and procedural languages but also functional languages can benefit from positive properties of aspect-oriented approach. Aspect-oriented programming languages provide facility to interrupt the flow of control in application at specific points (join points), and insert new computation (advice) at this point without modifying the original source code. In order to triggered advice at specific join points, specified conditions defined by programmer should be met.

When conditions are met an advice is woven in join point by weaving mechanism. The weaving mechanism seems to be an appropriate mechanism for change and control the behavior of the software system during its run time and for automatic implementation that is based on process functional paradigm and enables the process of evolution to be automated.

The process functional paradigm is based on pure functional language Haskell [6, 7]. Primarily, the pure functional language was enhanced to store the values in memory cell through environment variable. The environment variable expects some data value (value of some type T), but may also contains an undefined value. Also unit value () is used and has a function of control value to access the value stored in memory cell. On one hand, large software systems can communicate through the environment variables exchanging or accessing its values. On the other hand, the environment variable serves as an attribute for arguments of pure function. Such function with attributed argument is called process and environment variable is used only in type definition of the process. There are two basic processes that can handle with values stored in environment variable – data process (*data*) a control process (*control*):

$$data :: v\ T \rightarrow T$$
$$data\ x = x$$

a.) data process

$$control\ \ () \rightarrow T$$
$$control\ () = ()$$

b.) control process

Using environment variables, state can be manipulated like in imperative language. Data process is used to update value in environment variable and control process is used to access the value that is stored in environment variable. Data and control processes as well as transformations that are supported in PFL are described in [2,5].

As mentioned above, a multiparadigm approach seems to be an appropriate way not only for the implementation stage of development process and evolution of software system but it has its application also at the specification stage.

4. PROBLEMS WITH MULTIPARADIGM APPROACH

In respect to use of multiple paradigms there exist some problems. On one hand, functional paradigm is simple and using mathematical formalism and constructions with a strict defined semantics the problem can be described effectively. Using functional languages, the reliability of the system, from the correct functionality point of view, can be increased. But on the other hand, using only functional languages not all real world events such states, input and output functions, error handling etc. can be expressed because of lack of side effects (lack of assignments). Therefore, functional language was extended using imperative, object-oriented and nowadays also aspect-oriented paradigm.

But also these paradigms have some disadvantages for our goal of software system development and evolution process. Even though the object-oriented paradigm is very useful at the implementation stage by developing complex software systems there have been arise some crucial problems. First, from the point of view of modularization, not every part of object-oriented program can be expressed in modular way, for example the process of login. Even though modular decomposition of software system there exist functionalities that cut particular parts of source code across and can not be expressed in modular way. This leads to a tangled source code what cause problems from the development and evolution process point of view. To solve such kind of problems, an aspect-oriented paradigm have been developed [8, 9, 10, 15] that can express crosscutting concerns effectively through the use of modular unit called aspect.

The main reason why we try to integrate this paradigm to PFL was described in section 3. The simple principle of weaving mechanism in aspect-oriented paradigm is illustrated in Fig. 1.

However, the process of weaving mechanism can be static or dynamic, in our research project we concentrate on dynamic weaving process because it can change and control the behavior of the software system during its run time. From the point of view of software system evolution, one of the most important aims is to solve the problem of separation of concerns. The problem of separation of concerns is described in following section.

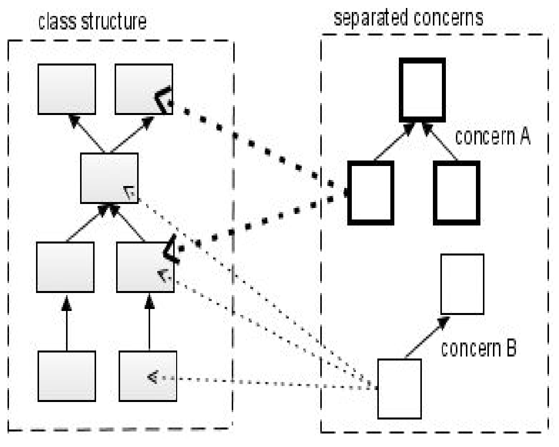

Fig. 1 Weaving mechanism principle

5. SEPARATION OF CONCERNS AND MODULARITY

The decomposition of the problem into smaller sub-problems has a key role in a solution of large and complex software systems. The decomposition leads to modularization and it is known that programs that are modularized are easier and better to understand. The approach of decomposition is applied in object-oriented paradigm and the modularity is obtained throughout the decomposition of system into objects which communicate to each other using message passing. There are three basic principles of object-oriented paradigm – encapsulation, inheritance and polymorphism that help the programmer to improve the specification of the language in regard of modularization and reuse. But even though modular decomposition of object-oriented programs there still exist concerns (functionalities) that cut the different modules cross as mentioned above in section 4. The concerns (features, requirements) can be changed hard because of their multiple locations in the program that need to be modified. There exist many special purpose concerns such concurrency, distribution, location control, failure recovery [9]. These concerns have a basic concern that is responsible for computational algorithm and basic functionality. These functionalities are called crosscutting concerns and can not be expressed in modular way so they leads to tangled and scattered code and reduce a program's overall modularity. They make a program harder and costly to understand, develop, maintain and evolve. As can be seen, the modularity of the system is limited by presence of crosscutting concerns. The existence of crosscutting concerns affects negative also the process of software system evolution. Another disadvantage of object-oriented programs from the evolution point of view is strong boundaries between class structure and system behavior. On one hand this strong boundary limits flexible adaptability of system to new requirements and on the other hand the reusability of system is limited because of cross boundaries between particular parts of the system. This problem can be solved using new programming approach called adaptive programming that principles will be described in the following section.

6. TOWARD ADAPTIVE PROGRAMMING PRINCIPLES

As mention above, the main disadvantage of object oriented and also aspect oriented paradigm are strong boundaries between class structure and the system behavior what limits flexible adaptability of the system to new requirements. New paradigm called adaptive paradigm tries to solve this problem so that it defines classes and methods for an application without its boundaries on class structure [12, 13, 14]. The main idea is to make boundaries between methods and data structures when it is needed only. The process of evolution is supported using class graph and propagation patterns. Class graph and propagation patterns represent a higher level of class specification and behavior. Based on this it is able to generate a source code. Adaptive program can be seen as a set of propagation patterns that define a restriction for classes. Adaptive program defines a set of programs that satisfy the restrictions defined by propagation patterns. Class structure that satisfies the restrictions is expressed as a class graph [14]. For such class graph propagation pattern generates an object oriented program. We can say that adaptive programs have also defined class structure and the behavior as object oriented programs, but the class structure is defined only particular using restrictions. The behavior is implemented also particular only. This means that methods are defined only when they are needed.

Propagation patterns support the behavior of the system on a higher level of abstraction and implements the functionality for a group of classes while this group is expressed using specification. In the following there is an example of such propagation pattern.

```
*interface* void function (Type* variable)
*from* ClassA
*to* ClassB
*primary* ClassB
(@ piece of source code; @)
```

Adaptive programming represents also a new form of parametric polymorphism. This means that adaptive program is used for different kind of classes and specifies the information that is defined implicitly. Such kind of parametric polymorphism enables boundary between method and class in the phase of adaptation process. The idea of adaptive programming has its denotation especially from the viewpoint of flexibility of software system. It represents a higher level of abstraction and one of the most important advantages is abstraction behavior and details of the system that has its amount in an evolution process.

Multiparadigm approach for software system design has been proposed in [1]. Based on multiparadigm software system development and the concept of paradigm a new method of multiparadigm design with feature modeling has been developed. Application and solution domain have been modeled using conceptual modeling technique – feature modeling.

In [3] a multiparadigm specification technique is presented. It is intended to help people organize and write complex specifications, exploiting the best features of several different specification languages. Complex systems have many heterogeneous aspects. It may not be possible to find a single specification language suitable for all aspects, and it may not be feasible (due to the complexity of each aspect) to specify some aspects in languages illsuited to them.

Adaptive principles are useful for many applications of real-time systems [16], in which the problem of run-time adaptation of a language, instead of a system itself will play great role in the future.

7. CONCLUSION

The essence problem of software system development and evolution are system's properties. Nowadays, a significant problem is that these properties are often expressed not clearly in specification of the software system. If the specification is not clear then the behavior of the system can not be changed effectively. Our future goal in evolution of software system is to express the properties of the system as clearly as possible in formal way using multiparadigm approach and try to predict these properties in the future applying decision rules in process of evolution. For the purpose of self evolving software system the principles of adaptive programming will be used because it seems that the idea and principles of adaptive programming can be used not only on an implementation stage but also at specification stage and not only in software system specification but also for the specification of the process of evolution.

REFERENCES

[1] Valentino Vranič: Multi-Paradigm Design with Feature Modeling. Computer Science and Information Systems Journal (ComSIS), june 2005, vol. 2, number 1, pp. 79—102.

[2] J. Kollár: Process Functional Programming. Proc. 33rd Spring International Conference MOSIS'99 - ISM'99, Information Systems Modeling, Rožnov pod Radhoštem, Czech Republic, ACTA MOSIS No. 74, pp. 41-48, April 27-29, 1999.

[3] P. Zave, M. Jackson: Where Do Operations Come From? A Multiparadigm Specification Technique. *IEEE Transactions on Software Engineering*, vol. 22, no. 7, pp. 508-528, jul 1996.

[4] P. Zave: A Compositional Approach to Multiparadigm Programming, IEEE Software vol. 6. 1989, pp. 15–25.

[5] Ján Kollár: Unified Approach to Environments in a Process Functional Programming Language. pp. 439–456, 2003, vol.23.

[6] Peyton Jones, S. L.: The Implementation of Functional Programming Languages. Prentice-Hall, Inc., 1987

[7] P. Wadler: The Essence of Functional Programming. In 19th Annual Symposium on Principles of Programming Languages, Santa Fe, New Mexico, January 1992, draft, pp.23

[8] Kiczales, G. and Hilsdale, E. and Hugunin, J. and Kersten, M. and Palm, J. and Griswold, W. G. An overview of AspectJ. Proc. ECOOP 2001, LNCS 2072, Springer-Verlag, jun 2001, pp. 327—353.

[9] Filman, R., Elrad, T., Clarke, S., and Aksit, M. Aspect-Oriented Software Development. Addison-Wesley Professional, 2004, p. 800.

[10] Greenwood, P. Dynamic Framed Aspects for Dynamic Software Evolution. 2004, pp. 101–110.

[11] T.A. Budd. Multiparadigm Programming in Leda. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994.

[12] K. J. Lieberherr, S.L. Ignacio, and X. Cun. Adaptive object-oriented programming using graph-based customization. Commun. ACM, 37(5):94–101, 1994.

[13] K. J. Lieberherr. Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns. PWS Publishing Company, Boston, 1996. ISBN 0-534-94602-X.

[14] K. J. Lieberherr. Demeter method. 2006. http://www.ccs.neu.edu/home/lieber/inside-impl.html.

[15] D. Rebernak, M. Mernik, H. P. Rangel, and M.J.V. Pereira. Aspectlisa: an aspect-oriented compiler construction system based on attribute grammars. In LDTA'06: 6th Workshop on Language Descriptions, Tools and Applications, Vienna, AT 2006.

[16] D. Zmaranda, and G. Gabor: Issues on Scheduling in Rof Crosscutting Modularity, Proc. 8th International Conference on Engineering of Modern Electric Systems, Felix Spa-Oradea, May 24 - 26, Oradea, Romania, University of Oradea, 2007, pp. 43-48, ISSN 1223-2106

BIOGRAPHIES

Jana Bandáková was born in 1982. She graduated at Technical University of Košice, Slovakia. She is working on her PhD. Degree at Department of Computer and Informatics FEEI, Technical University of Košice, Slovakia. Her scientific research area is adaptive software system evolution.

Ján Kollár (Assoc. Prof., M.Sc., Ph.D.) received his M.Sc. summa cum laude in 1978 and his Ph.D. in Computing Science in 1991. In 1978-1981 he was with the Institute of Electrical Machines in Košice.

In 1982-1991 he was with Institute of Computer Science at the P.J. Šafárik University in Košice. Since 1992 he is with the Department of Computer and Informatics at the Technical University of Košice. In 1985 he spent 3 months in the Joint Institute of Nuclear Research in Dubna, Soviet Union. In 1990 he spent 2 months at the Department of Computer Science at Reading University, United Kingdom. He was involved the research projects dealing with real-time systems, the design of (micro) programming languages, image processing and remote sensing, dataflow systems, implementation of programming languages. Currently he is working in the field of multi-paradigmatic languages with respect of aspect paradigm and on adaptable languages. He is the author of PFL – a process functional language.