1. Introduction

Until now, generative programming has been regarded as a discipline of object-oriented programming. However, in recent years there has been a world-wide surge of projects aimed at the development of scripting languages for application generators, such as Open Promol (Štuikys et al., 2001) and CodeWorker (Lemaire, 2003). Advantages of using scripting languages should be reached through avoiding certain weaknesses of object-oriented programming, primarily the (Ousterhout, 1998) rigidity of the object model, high level of standardization and the need to have a translation/compiling phase during the development of the program. In addition, we could single out the following scripting language characteristics useful in generative programming (Štuikys et al., 2001):

- scripting language abilities in character queue processing,
- connecting completed components written in target program languages and
- flexibility of scripting language syntax stemming from low standardization level.

The reason scripting language has not been used so far was the absence of applicable models, using which one could model the generators - ability not available to object oriented modelling based on UML diagrams. Therefore, in order to enable scripting languages for generative programming, the industry needed to develop the corresponding scripting model (Radošević, 2005). The scripting model represents the graphical model based on the implementation of aspects, i.e. characteristics not connected to individual organizational program units such as functions and classes, but showing up in various application parts, (Kiczales et al., 1997)(Lee, 2002). In that respect, the scripting model is a part of the group of so-called Join Points Model (Kandé et al., 2002)(Lieberherr, 2003). Also, the scripting model is a part of the type-free system group (Albano et al., 1989), since the connecting points do not represent classes and their objects, but only connections between metaprograms and characteristics defined in the application specification(Radošević, 2005).
The object and the scripting model are compared with regard to the following criteria: static and dynamic view within the processing level and the level of software development design, specifications of program requests, implementation of encapsulation and succession, and the relationship between base model elements (classes/metascripts).
Next came the analysis of compatibilities between the two models, but also important conceptual differences. The scripting model is a free-type system (and UML is not), which simplifies the connectivity of the aspects using the join points model. To wit, standardization represents a problem in realizing connective points, thereby also for the implementation of aspects within the join points model (Barca, 2003)(Roychoudhury et al., 2003) and for graphical modelling of aspects (Stein et al., 2003)(Gray et al., 2002). Additionally, using the scripting model , i.e. due to its simplicity, a higher flexibility in the development of generators and applications using the Boehm cyclical software development model is possible (Boehm, 1988).

2. LEVELS OF MODELLING

UML supports modelling of complex systems through various views in order to reduce the complexity of each such system, while also supporting various modelling levels (business level and IS level). If we approach the case from the standpoint of static and dynamic view (OMG, 2006), the architecture of the development of s complex software system supported by UML diagram techniques could be modeled as shown in Fig. 1. According to this architecture, we recognize the following levels: the USE CASE level; the processing level, the design level and the implementation level (Jacobson et al., 1999), on Fig. 1. The attempt to decrease the complexity of such elaborate systems is, therefore, made through modelling static and dynamic system components (static/dynamic view).

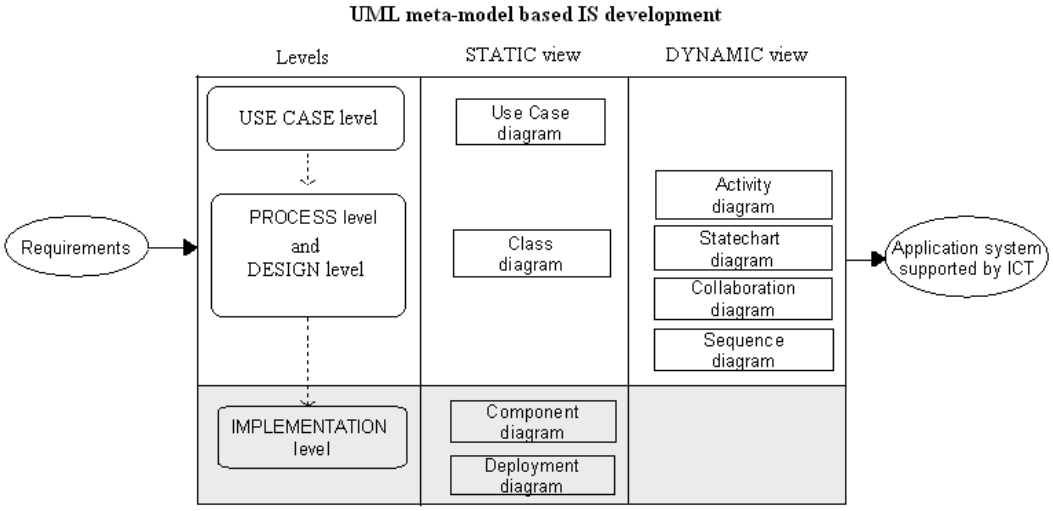

Fig. 1: Architecture of software system development supported by UML diagram techniques

The paper especially emphasizes the defining of relationships between the generator scripting model and base levels of the UML model (Use Case and Process).
The scripting model defines only the dynamic view, and includes two diagrams on two levels: the specification level and the processing level. The dynamic view is a part of the programming code template – *metascripts* (Fig. 2).

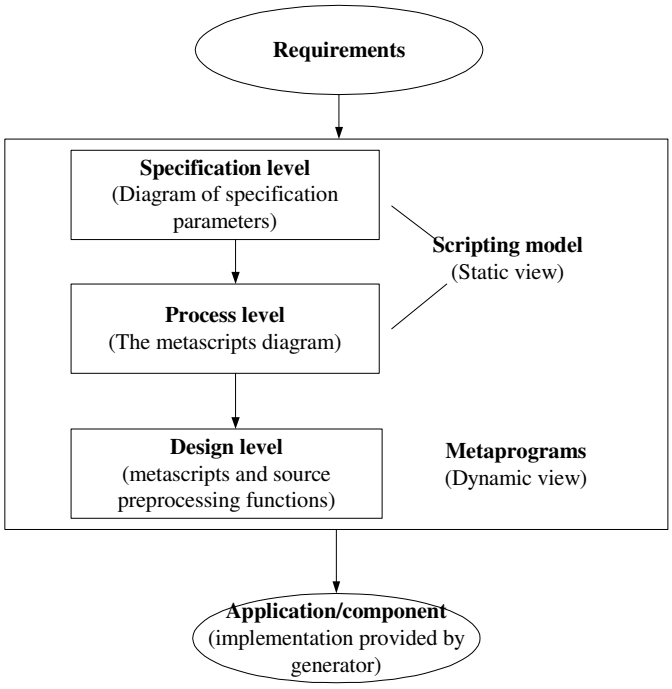

Fig. 2: Levels of scripting model architecture

3. SIMILARITIES BETWEEN THE OBJECT AND SCRIPTING MODEL

3.1. BASIC MODEL ELEMENTS

Class in the object model represents a cluster of attributes and operations (methods) used to describe the structure and behaviour of class objects. **Object** is an instance of the class, i.e. actualization of something which exists within time and space. Within the Class Diagram, class is represented by a rectangle containing the class name, attributes and methods, and rights of access (Fig. 3).

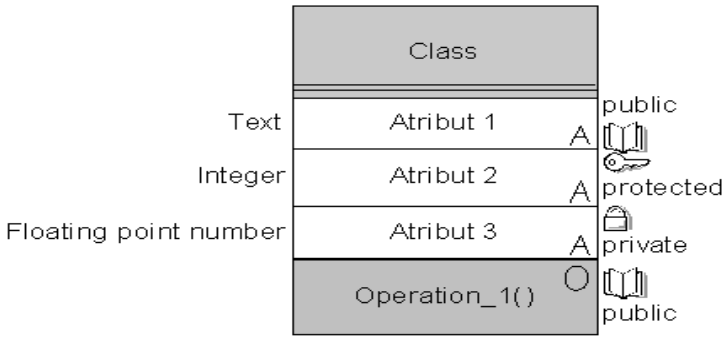

Fig. 3: The Class concept

Metascript from the scripting model represents a metaprogram, part of the code used for generating. The metascript is in the metascript diagram represented by a rectangle (Fig. 4) containing the metascript name, the source of the program code and (optional) exit from the program code (i.e. the name of the exit file).

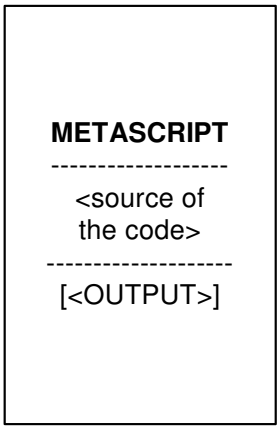

Fig. 4: Metascript element in the metascript diagram

Script (=application, generated program code) represents the product of generating, i.e. application or a part of an application defined through one or more metascripts and data connections/sources.

3.1.1. RELATIONS BETWEEN BASE ELEMENTS

Classes can be connected through the following connection types (OMG, 2006):

- *association* (structural relationship between classes or class instances),
- *aggregation* (special type of association showing the relationship between parts and the whole; aggregation by reference is a weaker type of relation/connection in which the class as a *whole* and class as a *part* are independent, while aggregation by value is the stronger connection type in which the class as a *whole* depends on the class as a *part* and vice-versa),
- *dependability* (one class uses the other during the execution of its operation),
- *generalization* (subordinate class is a specialization of a superior class; it has all its traits, but can also have additional characteristics) and
- *actualization* (of interface class).

The static class structure and their inter-relations can be shown using a certain type of diagram technique: UML *class diagram*. Relations between the metascripts and between other elements of the scripting model, connections and sources, can all be seen in the metascript diagram (Fig. 5). Each metascript diagram defines individual multi-level generator.

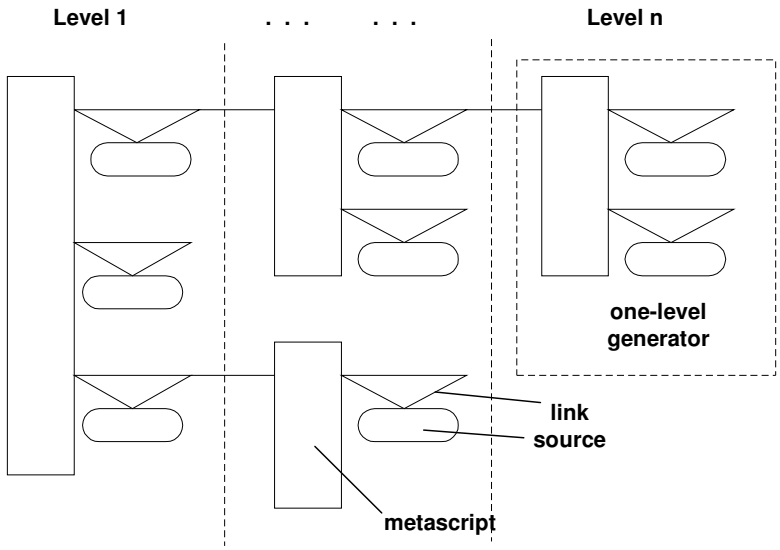

Fig. 5: Metascript diagram

Connections between the elements of the metascript diagram may be as follows:

a.) **Aggregation**. The object model differs aggregation by reference and aggregation by value (Fig. 6), while the metascript diagram defines aggregation by value only, as a basic model between different level generators. Metascript of a higher level grows using characteristics of lower-level metascripts, that is, a higher-level generator is made by superposition of several single-level generators (Fig. 5).

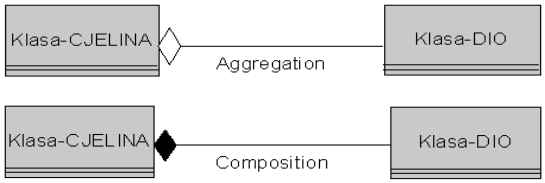

Fig. 6: Aggregation by reference and aggregation by value in class diagram

Aggregation relationship between the superior and subordinate generator is 1: 1..N (Fig. 7).

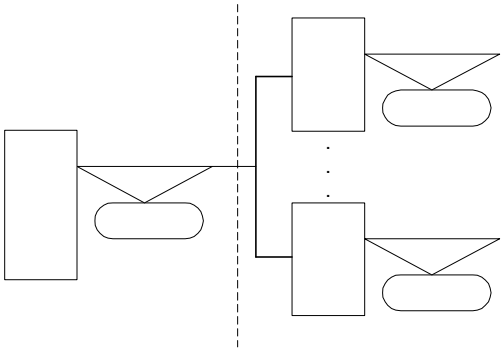

Fig. 7: Aggregation relation between generators

Aggregation in the metascript diagram is selective, i.e. its actualization in the final implementation depends on the specification of a particular application.

b.) **Association.** Association exists between the metascript and the source, within the individual single-level generator. Connections represent substitute symbols and physically present within the metascript in a way that each connection within the metascript may appear once or several times, so that the relation between the metascript and the source is 1..N:1 (Fig. 8).

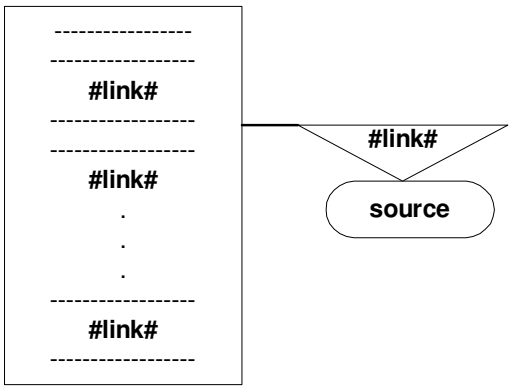

Fig. 8: Each individual connection may have more physical entries within any one metascript

3.2 BASIC CONCEPTS

Of all basic concepts of object programming (encapsulation, succession, polymorphism and data hiding) the scripting model supports the following:

Encapsulation in the scripting model exists within the application specification. Application specification encapsulates given application aspects on the level of individual branches of the application description parameter diagram. Characteristics may relate to data and functionality.
Succession within the scripting model functions in the following manner: the superior metascript gets increased by characteristics and functionalities of subordinate metascripts and data sources. This is completely different from the object model, where subordinate classes succeed the superior ones. Within the scripting model, succession is selective, determined by application specifications. This same way leads to **polymorphism** - several implementations are gained for one metascript, but base characteristics (within the metascript) remain unchanged.

3.3. SPECIFICATION OF PROGRAMMING REQUESTS/DEMANDS

Business function (Use Case) is a complex category which can be shown (modelled) using various diagram techniques, depending on the function components which we wish to include in the model. For example, for a certain function we can show

activities, events, messages, states, goals, resources, data and other components and their inter-connections. The static view of the function shows WHAT the function does, while the dynamic view shows HOW the function works, and represents the complete view of all its components. The text further features short descriptions of several UML diagram techniques.

UML Use-Case Diagram (Fig. 9): **static** view of system functionality and participant (actor) interaction with application cases. In other words, it is a user view of the functioning of the system (what the system does, not how it does it). Users who use the application system are tied into individual functions (i.e. cases of use) aimed at solving their tasks.

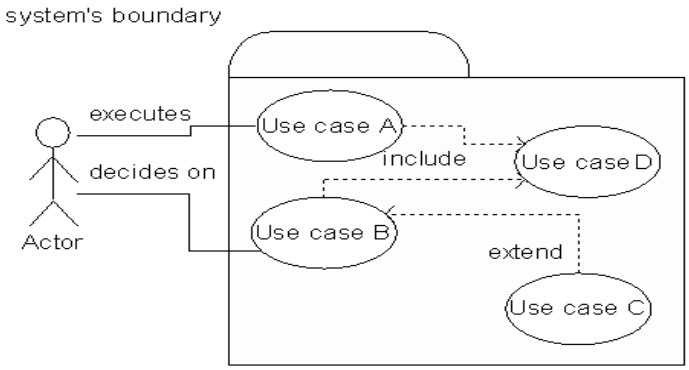

Fig. 9: Example of UML Use Case diagram

Use (application) cases are implemented within the metascripts, as program code templates which contain common characteristics of various applications within the problem domain. Crosscutting characteristics (aspects) of various application cases are singled out into application specification, i.e., separation of concerns (views) is done as presented by (Stein et al., 2003), Fig. 10.

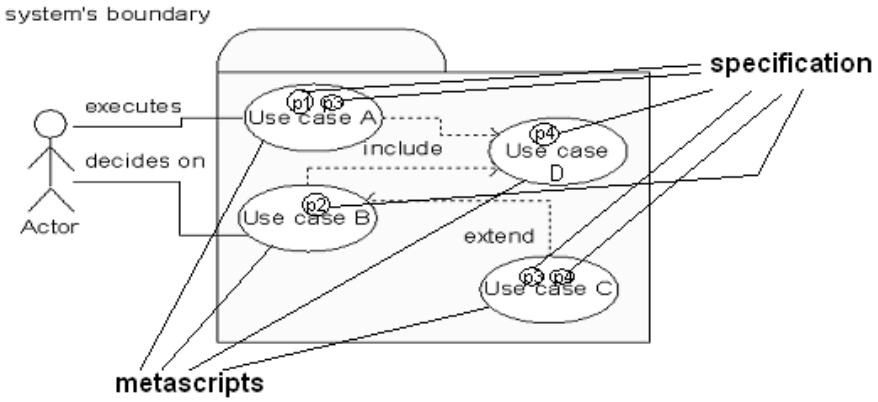

Fig. 10: Separation of crosscutting concerns

Application specifications contain values for each characteristic (aspect) and are defined by the diagram of application description parameters (Fig. 11), where specific corresponding tags are used for individual characteristics. Dispersion of characteristics onto various parts of the application is defined by the *metascript diagram*, where characteristics are represented by the element *source*.

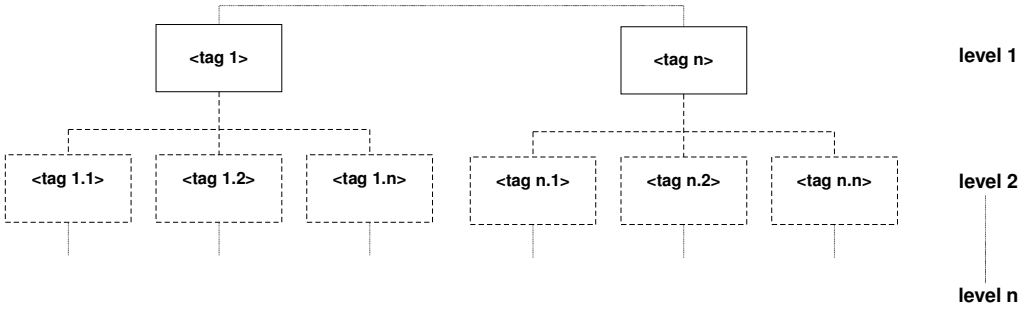

Fig. 11: Diagram of application description parameters

4. CONCEPTUAL DIFFERENCES

UML is a generic model because it defines generic components containing all characteristics which could be used by all applications from the problem domain. At the same time, UML is a model based on types (i.e. Type System), which makes the implementation of connecting points within aspect modelling harder. Connecting points in such a system represent complex types (classes) (Stein et al., 2002).
The scripting model represents the generative model, because individual characteristics (aspects) are included in the generated application according to its specification, i.e. characteristics are introduced into the goal application according to need, thereby achieving optimization with relation to the generic model. Furthermore, the scripting model is not based on types, i.e. it represents a type-free system (Albano et al., 1989). Connecting points in the scripting model do not represent classes and their objects, but only connections between metaprograms and characteristics defined in application specification (Fig. 12).

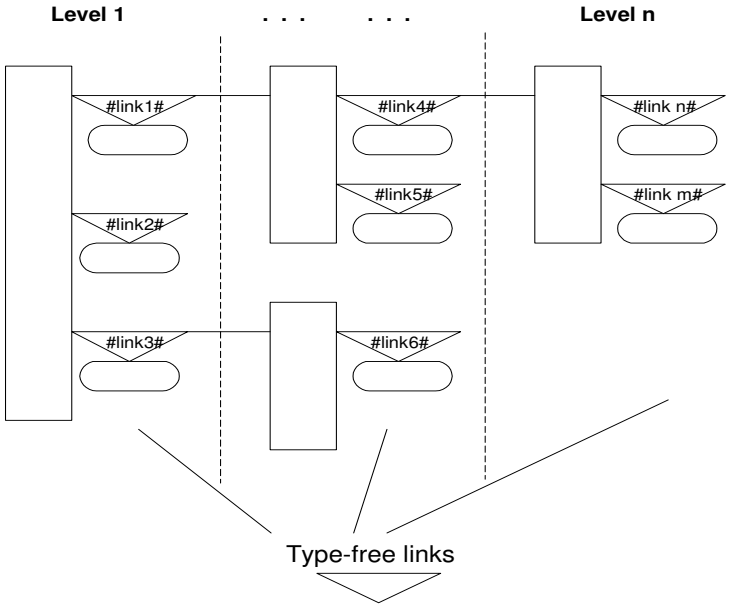

Fig. 12: Connection points in the metascript diagram

This significantly simplifies aspect implementation using connecting points, and the generative application development becomes more flexible, which allows for easier use of the Boehm spiral model of software development (Boehm, 1988) (Fig. 13).

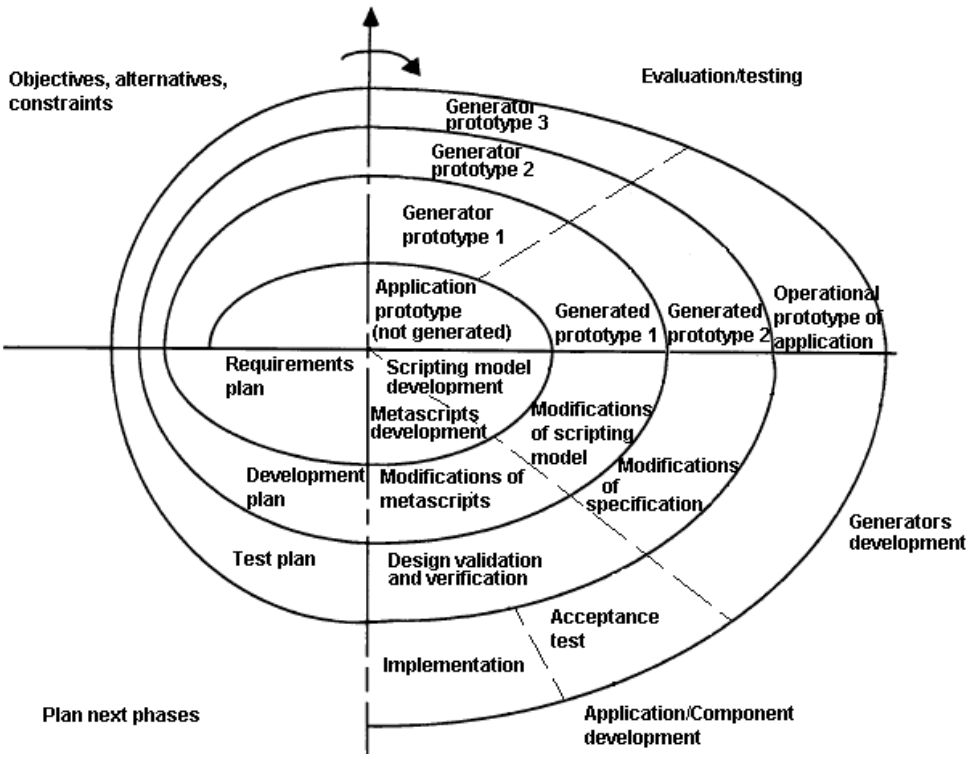

Fig. 13: Generative application development as spiral development using the Boehm (Boehm, 1988) model

Generative application development begins with the Requirements plan and the problem domain prototype application. This is followed by the Separation of concerns, so that specific characteristics of each individual application are contained within its specification, while common characteristics end up in metascripts. The scripting model defines the assembling of given application within the presented problem domain.

5. CONCLUSION

Comparison of the object and the scripting model has shown important analogies between the two models in view of basic model elements, their inter-relations and basic concepts within the processing level and the design level in the development of the software system. However, important conceptual differences have also been ascertained. Specifically, UML represents a generic model based on types (classes), and as such has significant problems in aspect modelling within the Join Point Model. On the other hand, the scripting model represents the generator model and is fully type-free. This eases generative application development, since it makes the

development system much more flexible, and also facilitates the use of the Boehm spiral model for software system development.

Bibliography

Albano, A., Dearle, A., Ghelli, G., Marlin, C., Morrison, R., Orsini, R., Stemple, D. (1989).A Framework for Comparing Type Systems for Database Programming Languages, http://citeseer.ist.psu.edu/albano89framework.html

Barca, M. (2003). Aspect Oriented Programming, With an Overview of AspectJ, CIS Department of Ohio State University, http://www.cse.ohio-state.edu/ ~barca

Boehm, B.W. (1988). A Spiral Model of Software Development and Enhancement, Computer, May 1988, v. 21 no. 5, pp. 61-72.

Gray J., Bapty, T., Sandeep N., Ariruddha G. (2002). Aspect-Oriented Domain-Specific Modeling, http://www.isis.vanderbilt.edu/projects/PCES/AODM.pdf

Jacobson, I., Booch, G., Rumbaugh, J. (1999). The Unified Software Development Process, Addison-Wessley

Kandé, M.M., Kienzle,J., Strohmeier, A. (2002). From AOP to UML - A Bottom-Up Approach, 1st International Conference on Aspect-Oriented Software Development, Enschede, The Netherlands, http://lglwww.epfl.ch/workshops/aosd-uml/Allsubs/kande.pdf

Kiczales, G., Lamping, J., Mendhekar, A., Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin. (1997). Aspect-Oriented Programming. In Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241. June 1997.

Lee, K.W.K. (2002). An Introduction to Aspect-Oriented Programming, COMP610E: Course of Software Development of E-Business Applications (Spring 2002), Hong Kong University of Science and Technology, 2002.

Lemaire, C. (2003). CODEWORKER Parsing tool and Code generator - User's guide & Reference manual, http://codeworker.free.fr/CodeWorker.pdf, CodeWorker.free.fr

Lieberherr, K.J. (2003). General AOP: Join Point Model, Demeter, Center for Software Sciences, 2003., http:// www.ccs.neu.edu/research/demeter/

Object Management Group (OMG) (2006). The Unified Modelling Language, http://www.omg.org

Ousterhout J. K. (1998). Scripting : Higher Level programming for the 21st Century, IEEE computer magazine, march 1998.

Radošević, D. (2005). Integration of generative programming and skripting languages, doctorate thesis, Faculty of Organization and Informatics, Varaždin, pages 41-82.

Roychoudhury, S., Gray, J., Wu, H., Zhang, J., Lin, Y. (2003). A Comparative Analysis of Meta-programming and Aspect-Orientation, Proceedings of 41st Annual ACM Southeast Conference, Armstrong Atlantic State University, Savannah, Georgia, March 7-8 2003

Stein D., Hanenberg, S., Unland R. (2003). Position Paper on Aspect-Oriented Modelling: Issues on Representing Crosscutting Features, International

Conference on Aspect-Oriented Software Development (AOSD 2003), Boston, 2003.

Stein D., Hanenberg, S., Unland R. (2002). On Representing Join Points in the UML, 2nd AOSD-UML Workshop at UML '02, Dresden, Germany, 2002.

Štuikys, V., Damaševičius, R., Ziberkas, G. (2001). Open PROMOL: An Experimental Language for Target Program Modification, Software Engineering Department, Kaunas University of Technology, Kaunas, Lithuania, 2001., http://soften.ktu.lt/~damarobe/publications/Vytautas_Stuikys.pdf

Corresponding Author Data:

Name and email address of corresponding author: Danijel Radošević, danijel.radosevic@foi.hr

Manuscript Data:

1. **Author(s) Name(s): Danijel Radošević, Melita Kozina, Božidar Kliček**
2. **Title of Manuscript:** Conceptual similarities and differences between Object Model and Generator Application Scripting Model
3. **Key words:** UML, scripting model, aspects, comparison
4. **Abstract:** The basic features of UML and scripting model of application generators are compared in this paper. The comparison between two models is performed according to the following features: static and dynamic view within the process level and design level, implementation of encapsulation and inheritance, relationships among basic model elements (classes/metascripts) and software requirements specification. Compatibilities of models are ascertained, but there are also conceptual differences. The scripting model actually represents a model of generators and it is based on aspects and their distribution on different program parts. UML is based on the generic approach which lacks efficiency in dealing with the modelling of aspects due to the features of the object model being a system based on types. The comparison shows that the scripting model is simpler and type-free. As such, it enables more flexibility in the development of application generators, and the application of Boehm's cyclic model of software development.
5. **Acknowledgment(s):**
6. **Thanks:**
8. **Number of Additional Copies of Scientific Book:** -
9. **Please send my copy/copies of Book to the following address:** Faculty of organization and informatics, Pavlinska 2, 42000 Varaždin, Croatia

DAAAM Authors Data:

1. Digital Photo:
2. First / Middle / Family Name: Danijel Radošević
3. Titles: PhD
4. Position / Since: Higher assistant/2005
5. Institution/Firm: Faculty of organization and informatics, University of Zagreb
6. Place and Date of Birth (yyyyy-mm-dd): Zagreb, Croatia, 1969-03-27
7. Nationality / Citizenship: Croatian/Varaždin
8. Field of interests (key words): generative programming, text mining
9. Hobbies:
10. E-mail address: danijel.radosevic@foi.hr
11. Home Page: http://www.student.foi.hr/~darados
12. Postal address: Pavlinska 2, Varaždin, Croatia
13. Phone & Fax #:385 042 390 834, 385 042 213413

1. Digital Photo:
2. First / Middle / Family Name: Melita Kozina
3. Titles: PhD
4. Position / Since: Assistant Professor/2006
5. Institution/Firm:Faculty of organization and informatics, University of Zagreb
6. Place and Date of Birth (yyyyy-mm-dd): Varaždin, Croatia, 1965-06-18
7. Nationality / Citizenship: Croatian/ Varaždin
8. Field of interests (key words): business systems modelling, ICT management
9. Hobbies: -
10. E-mail address: melita.kozina@foi.hr
11. Home Page: http://www.foi.hr/nastavnici/kozina.melita/index.html
12. Postal address: Pavlinska 2, Varaždin, Croatia
13. Phone & Fax #:385 042 390 800, 385 042 213413

1. Digital Photo:
2. First / Middle / Family Name: Božidar Kliček

3. Titles: PhD
4. Position / Since: Full professor/2004
5. Institution/Firm: Faculty of organization and informatics, University of Zagreb
6. Place and Date of Birth (yyyyy-mm-dd):1957-07-07
7. Nationality / Citizenship: Croatian/Varaždin
8. Field of interests (key words): AI, multimedia systems
9. Hobbies:
10. E-mail address: bozidar.klicek@foi.hr
11. Home Page: http://www.foi.hr/nastavnici/klicek.bozidar/index.html
12. Postal address: Pavlinska 2, Varaždin, Croatia
13. Phone & Fax #: 385 042 390 829, 385 042 213413