PHP Scripts Generator for Remote Database Administration based on C++ Generative Objects

Danijel Radošević, Tihomir Orehovački, Mario Konecki
Faculty of Organization and Informatics
University of Zagreb
Address: Pavlinska 2, 42 000 Varaždin, Croatia
Phone: +385 42 390 800 Fax: +385 42 213 413
E-mail: danijel.radosevic@foi.hr, tihomir.orehovacki@foi.hr, mario.konecki@foi.hr

Abstract - The scripting model of generators was originally developed for faster development of application generators in scripting languages. It was shown later that this model is not restricted only on scripting languages, so the appropriate library for generator development was developed in C++. For the purpose of this paper, the simple generator of PHP scripts for remote administration of database using web interface was developed. The application generators scripting model properties, like aspect orientation and the fact that the generators scripting model is typeless (which is the property of scripting languages like PHP) were important for this generator development.

I. INTRODUCTION

Generative programming is a discipline of automatic programming which started, under that name, in the late 90's of last century. According to this definition, generative programming represents "... designing and implementing software modules which can be combined to generate specialized and highly optimized systems fulfilling specific requirements", Eisenecker [5]. Aspiration to programming code optimization makes, according to [5], the main specific difference toward other techniques of automatic programming.

The main goals of generative programming are, according to Czarnecki [3]:

- decrease the conceptual gap between program code and domain concepts (known as achieving high intentionality)
- achieve high reusability and adaptability
- simplify managing many variants of a component, and
- increase efficiency (both in space and execution time)

Generative programming came out from the aspiration to increase productivity of making software, by producing it in a way comparable to industrial production. Because of some weaknesses, observed by Guerraoui [6], and Ousterhout [11], today's object-oriented programming is not able to fulfill those aspirations completely. That was the reason why some new disciplines of programming occurred in the middle of 90's in the last century, which should succeed this object paradigm. One of that disciplines is so called Aspect Oriented Programming (AOP) [6], which is one of the basic disciplines in generative programming.

According to Czarnecki [3], generative programming is based on the following disciplines:

- metaprogramming
- generic programming
- object-oriented programming
- aspect-oriented programming and
- domain engineering.

II. USAGE OF SCRIPTING LANGUAGES WITHIN GENERATIVE PROGRAMMING

Generative programming has been originally observed mostly as a discipline of object-oriented programming. But, as announced by Sells [14], some attempts of connecting generative programming to scripting languages appeared. Also, some projects of making specialized scripting languages for generative programming appeared. Some examples of them are Open Promol [15] in Lithuania and Codeworker [10] in France.

Advantages of using scripting languages should be reached through avoiding certain weaknesses of object-oriented programming, primarily the [11] rigidity of object model, large amount of typing and the need to have a translation/compiling phase during the development of the program. In addition, we could point out the following scripting language characteristics that are useful in generative programming:

- scripting language abilities in character queue processing [15][10],
- possibilities of connecting completed components written in target program languages [15] and
- flexibility of scripting language syntax which come out from low typing level [15][11].

Mentioned features are used in up-today's projects of scripting languages usage within generative programming, such as Open Promol and Codeworker.

III. THE GENERATOR SCRIPTING MODEL

Scripting model was developed for the needs of generative programming based on scripting languages [12]. Different to the object model, it's oriented to defining specified, specific aspects of the future applications within specified problem domain, not on all application functionalities, because these are defined on lower level (in program code templates; within the scripting model these

are called metascripts). Aspects, according to Kinczales [8], represent features that are not strictly connected to individual program organizational units like functions or classes, so they can appear within different application parts. Therefore the connection model is needed. In the basis, the offered model represents the kind of join point model, according to Kandé [7]. Furthermore, the scripting model is not based on types, i.e. it represents a type-free system [1], because connecting points in the scripting model do not represent classes and their objects, but only connections between metaprograms and characteristics defined in application specification [12].

The scripting model consists of two graphic diagrams (or equivalent textual specifications), so it's simpler in relation to the models based on UML [12]. The first diagram is called the ***specification diagram*** and it defines the structure of the application specification within the generation system (Figure 1.).

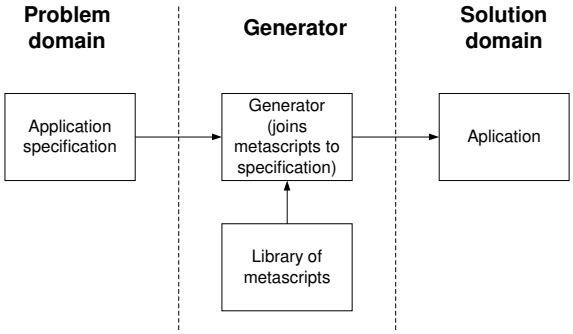

Figure 1. The generation system

The generation system generates the application within its problem domain, which is designated by program code templates (metascripts). The connection rules for connecting metascripts to application specification are defined in the second diagram - the ***metascripts diagram*** [12].

A. The specification diagram

The specification diagram of PHP scripts for remote database maintaining generator defines features (aspects) which make single application different from other within it's problem domain. In the example, specification defines used tables and fields in each table (Figure 2.).

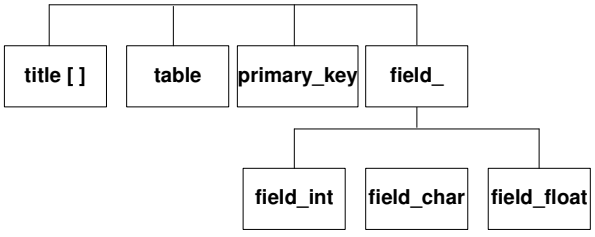

Figure 2. The specification diagram

B. The metascripts diagram

The metascripts diagram of PHP scripts for remote database maintaining generator defines connections between metascripts and application specification (Figure 3.).

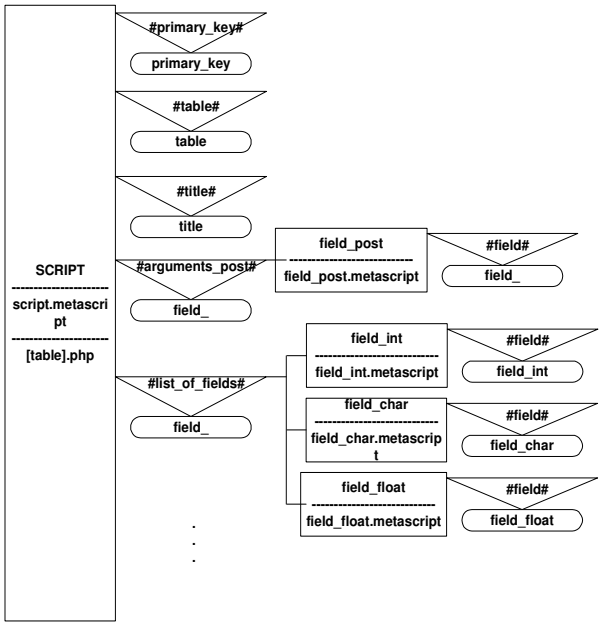

Figure 3. The metascripts diagram

IV. IMPLEMENTATION OF GENERATOR SCIPTING MODEL THROUGH GENERATIVE OBJECTS

The scripting model of generators was originally developed for the purpose of developing generators in scripting languages, like Perl. However, it was shown that model and its implementation don't depend on using scripting languages. Generators based on scripting model can be written in object-oriented programming languages like C++, as well as in scripting languages. Moreover, using object-oriented languages enables usage of object modeling techniques, like UML.

Generative objects are objects from classes which are included in programs in a form of libraries. For that purpose, the appropriate library for C++ is developed.

A. C++ library for generator development

The library defines two classes for generator development: *cgenerator*, and *cspecification*. The *cgenerator* class enables implementation of generating functions, while the *cspecification* class inherits *cgenerator*, adding methods for working with application specification.

B. Class cgenerator

The *cgenerator* class enables implementation of simple one-level generator in C++ language, which is shown in the next diagram (Figure 4.).

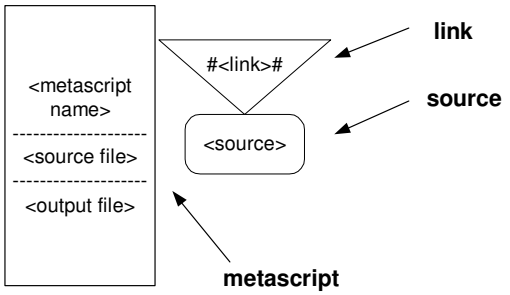

Figure 4. Single level generator

Operations supported by appropriate methods from *cgenerator* class are following:

- loading program code templates (metascripts)
- simple generating by exchanging links using appropriate exchange contents (sources)
- saving generated program code into output file
- different operations on character strings, like concatenation of generated code and assembling templates

C. Class cspecification

The *cspecification* class enables working with application specification. Application specification is proposed by specification diagram (Fig. 3). The *cspecification* class inherits *cgenerator* and enables implementation of specification linked list, all operations connected to application specification and implementation of more complex generating functions.

The application specification is in a simple textual file, in a form of label-value pairs, like the following example:

title:students
field_int:id
field_char:surname_name
field_float:average_mark

The linked list of application specification is formed by loading from textual specification (Figure 5.).

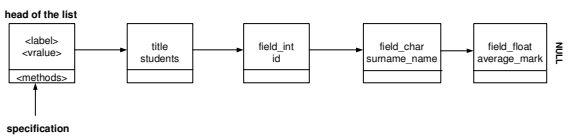

Figure 5. Linked list of specification

Operations supported by appropriate methods from *cspecification* class are following:

- loading specification to specification linked list
- implementation of simple single level generator (Figure 4.)
- selecting parts of specification, due to proper connecting sources to metascripts.

D. The structure of generator

The general structure of generator based on C++ generative objects is shown in Figure 6.

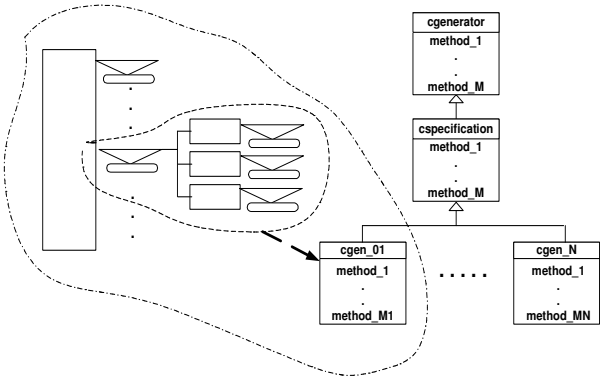

Figure 6. General structure of generator based on C++ generative objects

As shown in Figure 6., particular generators are implemented by appropriate classes, which are inherited from *cspecification*. Particular branches of metascripts diagram are implemented by appropriate methods.

V. THE APPLICATION PROTOTYPE

The prototype presents starting point of generator development because it's reingeenering results in programming code templates (metascripts). Also, prototype presents one of possible applications inside of problem domain which generator covers. In this case we examine remote database administration through web interface. The prototype is made of PHP scripts and HTML form templates which enable the main operations over „students" table (data review, new record entry, existing record modification and record deletion, Table 1.)

TABLE I
STRUCTURE OF PROTOTYPE DATABASE TABLE

Attributes	Data types
student_id (primary key)	integer
surname_name	varchar
year_of_study	integer
year_of_enrolment	integer

During the development of the application prototype the following infrastructure was used:

- Apache Web server
- mySQL database – for entity relationship model implementation
- PHP server-side scripting language – for user's view implementation

The main purpose of this prototype is generator development. This generator will be the virtue of prototype structure and functionality (generation of similar and more complex systems). Model of database-driven prototype which presents prototype functionality and used infrastructure is shown in Figure 7.

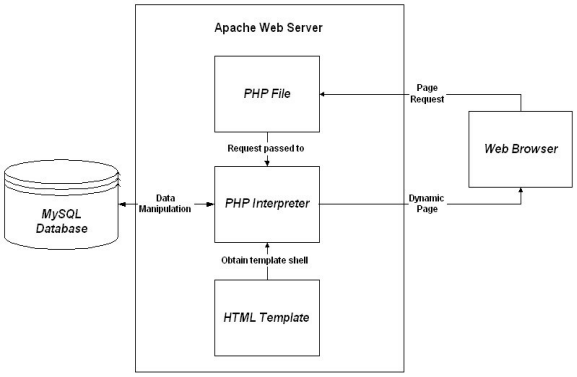

Figure 7. Model of a database-driven prototype

HTML templates of main operations (data review and existing record modification) which prototype maintains are shown in Figure 8. and Figure 9., respectively.

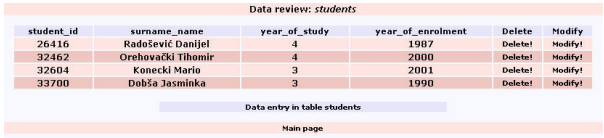

Figure 8. Data review

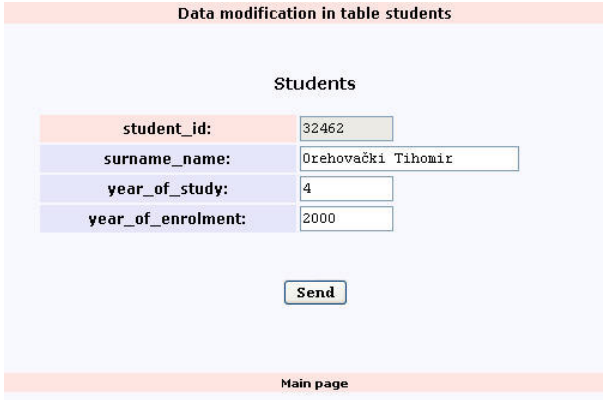

Figure 9. Data modification

VI. CONCLUSION

In this paper, it is shown that scripting model of generators can be implemented by appropriate object model by using object-oriented programming languages, like C++. For that purpose, the appropriate library for C++ was developed, as well as generator of PHP scripts for remote administration of database using web interface. Development of applications and their generators is adapted to Boehm's spiral model of software development [2].

By usage of C++ in generator development, some advantages in implementation of scripting model are achieved in comparison to scripting languages: better efficiency of program code, and its better protection (because of compilation into machine language), possibility of using object-modeling techniques like UML.

On the other hand, the flexibility in generators development using scripting languages was held.

REFERENCES

[1] Albano, A., Dearle, A., Ghelli, G., Marlin, C., Morrison, R., Orsini, R., Stemple, D.: "A Framework for Comparing Type Systems for Database Programming Languages", *http://citeseer.ist.psu.edu/albano89framework.html*

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

[3] Czarnecki, K.: "Generative Programming and GMCL", Technische Universität Ilmenau,Fakultät für Informatik und Automatisierung, 1999., *http://www-ia.tu-ilmenau.de/~czarn/gmcl/*

[4] Czarnecki, K.:"Generative Core Concepts - Generative Domain Model", Program-Transformation.Org, 2002., *http://www.program-transformation.org/Transform/GenerativeCoreConcepts*

[5] Eisenecker, U.:" .:"Generative Programming (GP) with C++", Proceedings of Modular Programming Languages (JMLC'97, Linz, Austria, March 1997.), Springer-Verlag, heidelberg 1997."

[6] Guerraoui R. : "Strategic directions in object-oriented programming", ACM Computing Surveys, Baltimore, december 1996.

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

[8] Kinczales, G., Lamping, J., Mendhekar, A., Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin. "Aspect-Oriented Programming". In Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241. June 1997. *http://citeseer.nj.nec.com/kiczales97aspectoriented.html*

[9] Lee, K.W.K.:"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.

[10] Lemaire, C.: "CODEWORKER Parsing tool and Code generator - User's guide & Reference manual", http://codeworker.free.fr/CodeWorker.pdf

[11] Ousterhout J. K. : "Scripting : Higher Level programming for the 21st Century", IEEE computer magazine, march 1998.

[12] Radošević, D.:"Integracija generativnog programiranja i skriptnih jezika", doctoral thesis, Fakultet organizacije i informatike, Varaždin, 2005.

[13] Radošević,D., Kozina,M., Kliček B.: "Comparison Between UML And Generator Application Scripting Model", Conference proceedings of "Information and Intelligent Systems 2005" (IIS 2005), Fakultet organizacije i informatike, Varaždin, 21.-23.09.2005.

[14] Sells, C.: "Generative programming: Modern Techniques to Automate Repetitive programming Tasks", MSDN Magazine, december 2001, http://msdn.microsoft.com/msdnmag/issues/01/12/GenProg/GenProg.asp

[15] Štuikys, V., Damaševičius, R., Ziberkas, G.: "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