1. Introduction

Generative programming has been recently observed mostly as a discipline of object-oriented programming. However, some projects that include scripting languages in generative programming appeared during last years. Some of these projects are for development of new scripting languages intended for generative programming, like Open Promol (Štuikys et al., 2001) and CodeWorker (Lemaire, 2003). Using scripting languages should result with some advantages in relation to object-oriented languages and their problems. According to (Ousterhout, 1998) these are the rigid structure of object model, high degree of typing and the need of compilation. On the other hand, scripting languages have some good properties that could be used in generative programming (Štuikys et al., 2001):

- possibilities of scripting languages in character string processing,
- gluing of target language components or their parts and
- flexibility of scripting languages syntax which is the result of low degree of typing.

Except these features, the potentially useful feature of scripting languages is possibility of character string evaluation, in the way that they could be interpreted as program code, and some features of high-level languages, like possibility of using fields unrestricted length.

However, mentioned languages, like the scripting languages generally, suffer from the lack of clear graphic models, like UML in the field of object-oriented programming. In this paper is offered simple graphic model intended for modeling of application generators based on program code modifications. In a difference to object model, offered scripting model is oriented to define specified, specific aspects of future applications in specified problem domain, but not on all application functionalities, because the other are specified on lower level - in program code templates (metaprograms), which are called *metascripts* within the scripting model. Such specified aspects are defined in application specification and represent all the specific properties which distinguish specified application from other applications within it's problem domain.

The model is tested on generators development in Perl language, and used in development of web applications in scripting languages.

2. Concept of generative programming based on scripting languages

The general model of generators

Fig. 1 shows the general model of generators according to the offered concept, which is based on scripting languages.

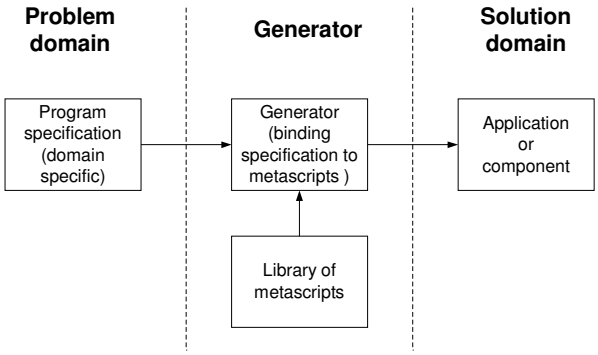

Fig. 1. General model of the generator according to the concept based on scripting languages

Program specification includes all specific properties of the application by which that application differs from the others within the same problem domain covered by generator. All common properties of different applications within the generator problem domain are defined inside metascripts. That results with better overview on the application as a whole. Also, the maintenance of applications become easier, especially for the applications which consist of many program modules, because changes in specification are updated in all program modules during the process of generation. That problem of updating many program modules is common to web applications in scripting languages, because that applications typically consist of large number of program modules and other files (e.g. HTML documents like forms for data input).

Levels of generation

Three levels of generation could be distinguished according to offered concept, based on scripting languages:

- level of metagenerator (generalized level of application generators),
- level of application generator (generalized application level) and
- application level (target level).

Relations among three levels of generation is shown on following diagram (Fig. 2):

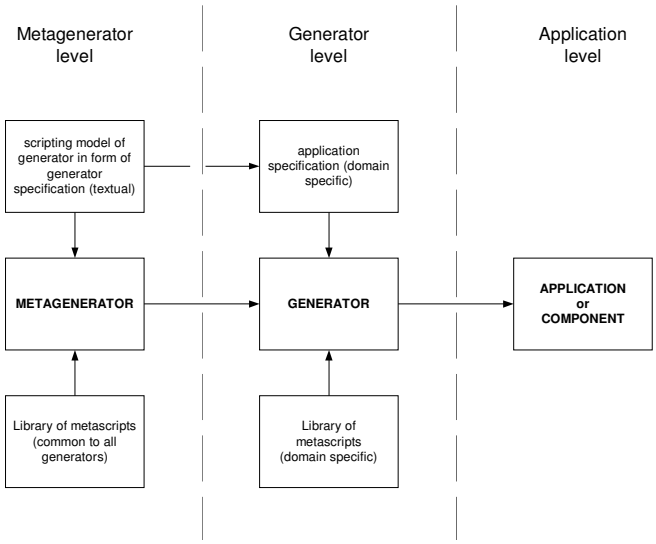

Fig. 2. Levels of generation

Generative application development

Scripting model of application generators is used within the generative application development, which is the process of parallel development of generators, together with target applications, as shown on Fig. 3:

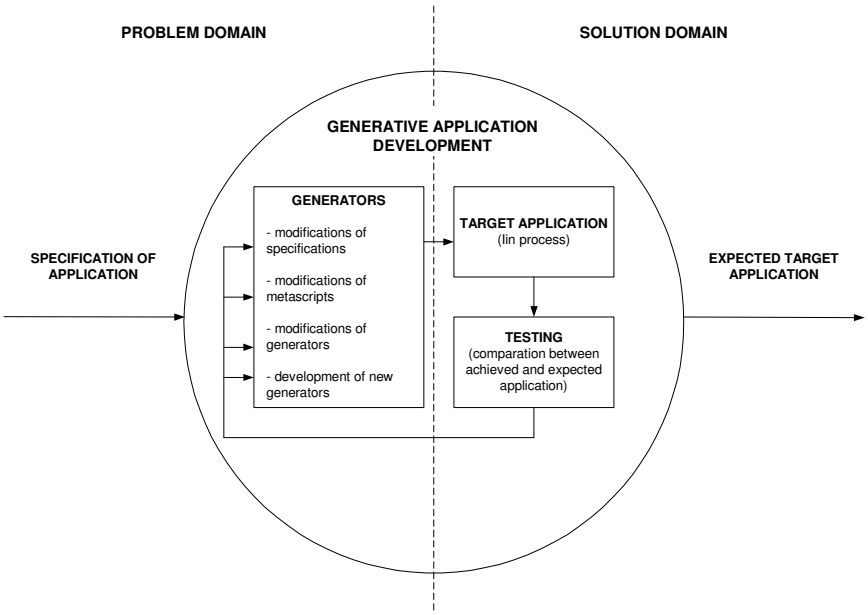

Fig. 3. Generative application development as a circular process

One of the basic features of generative programming in relation to other disciplines of automatic programming is the possibility of using more generators in application development (Czarnecki & Eisenecker, 2000). That means that generators can be made for generating of individual components, not necessary for complete applications. Generative programming according to offered concept, based on

scripting languages, enables even more flexibility in development of generators and applications, as a circular process (Fig. 3).

So, generative application development includes development and modifications of generators, according to the needs of target application. Developed generators became the knowledge base about their problem domains, so they can be used within the new projects. Of course, some modifications of generators could be necessary in some cases to adapt then to the problem domain of project task (Fig. 4).

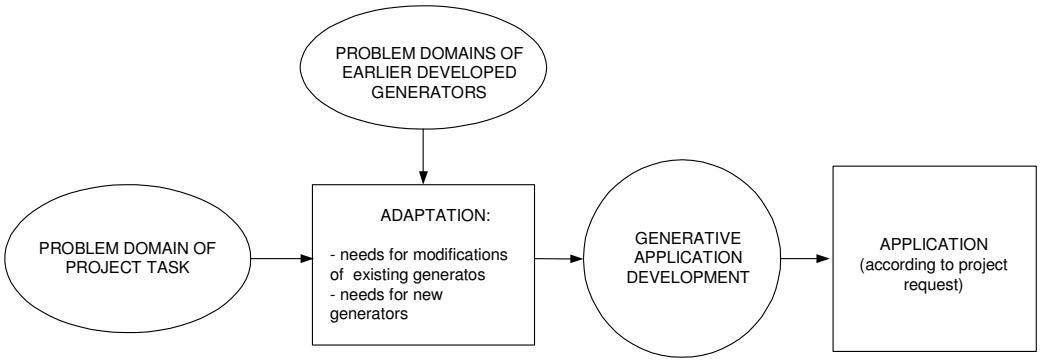

Fig. 4. Adaptation of problem domain of earlier developed generators to the problem domain of project task

So, the new project starts with establishing the usability of existing generators, and according to that, the needs for modifications of existing generators and development of new generators.

3. The scripting model of application generators

Scripting model is developed for the needs of generative programming based on scripting languages. 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 lever (in program code templates; within the scripting model these are called *metascripts*). Aspects represents features that are not strictly connected to individual program organizational units like functions or classes, so can appear within different application parts (Kinczales et al., 1997). So, the connection model is needed. In the basis, the offered model represents the kind of join point model (Gray et al., 2003).

The scripting model consists of two graphic diagrams, so it's simpler in relation to the models based on UML. Scripting languages are not based on classes and objects, but their basic properties like encapsulation, inheritance, data hiding and polymorphism could be achieved on higher level of scripting. Scripts on higher level within this model are called *metascripts*. Metascripts represent templates for generating program code in different program languages. However, important difference between object model and offered scripting model is that scripting model is based on aspects, while modeling of aspects is still a problem within the object model (Kühl, 2000)(Lee,

2002). Also, object model defines individual application, while scripting model defines application generator (for designated problem domain).

Diagrams of scripting model

Scripting model of application generators consists of two graphic diagrams:

- diagram of the application specification parameters - defines designated properties (aspects) of the future application and defines the structure of specification, which will be used in application generation.

- the metascripts diagram - defines distribution of specific properties, which are designated by diagram of application specification parameters, within application, and also their connecting to program code templates (metascripts).

Diagram of the application specification parameters is hierarchic diagram that defines designated application properties. Properties are defined by parameters that are used for specifying application from generator's domain. Parameters of specification have hierarchic structure and define individual application within designated problem domain, covered by generator, and can refer to properties like:

- data definition and
- process kind definition.

Diagram of the application specification parameters contain tags on different hierarchical levels (Fig. 5):

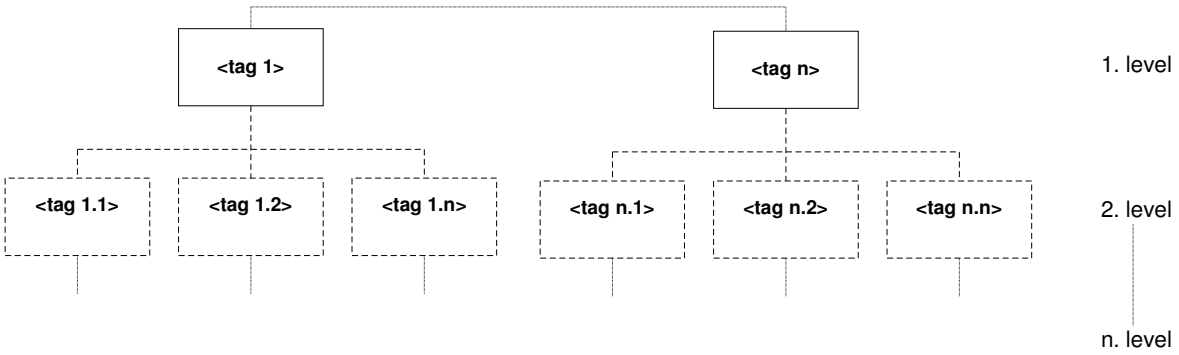

Fig. 5. Diagram of the application specification parameters

Appropriate structure of application specification is defined by diagram of the application specification parameters:

<tag 1>: <value 1>
<tag 1.1>: <value 1.1>
<tag 1.2>: <value 1.2>
<tag 1.n>: <value 1.n>
<tag n>: <value n>
.
.
<tag n.1>: <value n.1>
<tag n.2>: <value n.2>
<tag n.n>: <value n.n>

All tags are optional - could be omitted, or replicated by needed number of times. Also, tags with no values (or no used values) are possible, because the metascripts diagram defines the use of values.

The metascripts diagram defines distribution of specific properties, which are designated by diagram of application specification parameters, within application, and also their connecting to program code templates (metascripts). So, the metascripts diagram represents the connection scheme of the application.

The metascripts diagram consists of three basic elements (Fig. 6):

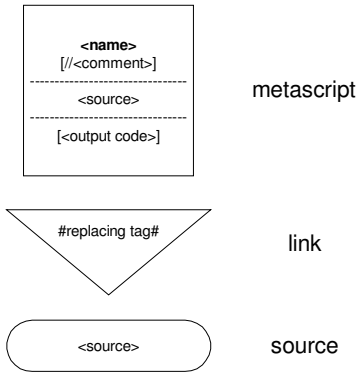

Fig. 6. Elements of the metascripts diagram

Metascript is a template used in generation of its implementation - program code in target programming language. It is represented by rectangle. There are some data within the element:

- metascript name,
- comment (optional)
- source (filename or name of other used source)

- output code (filename or name of other output which contain generated program
code; optional)

Metascripts contain replacing tags - tags that are being replaced with designated data or program code. Tags are designated within the metascripts diagram by the *link* element.

Link connects metascript to the replacing source, and (optional) to one or more metascripts of lower level, that are used in generation of replacing code. It is represented by triangle, with one of the corners oriented down, within that is the name of replacing tag. The replacement of code can be performed by one of two following ways:

- direct replacement of tag by data from source (if there no metascripts of lower level) and
- replacement of tag by metascript of lower level (which also contains its own links and sources)

Source represents particular parameter defined by appropriate tag in a diagram of the application specification parameters. Sources can be defined as group parameters (have their tree structure) - in that case their further use must be defined on lower levels of the metascripts diagram (by specifying particular sources).
Source is defined as particular parameter (defined by its tag) from diagram of the application specification parameters:

<tag> - tag name

Tag name may consist of common part only. In that case the tag name ends with '_' or '.' sign:

<tag>_ or <tag>.

Example:

<field_> - represents all tags with name beginning with 'field_' (e.g. field_number, field_real or field_char)

Also, some sources require preprocessing by appropriate process defined as a function, which is marked by '&' sign:

&<function_name>(<parameters>)

Parameters are group or particular sources.

The rationalization in defining of sources is possible in cases when the same metascript is used for more different parameters from the same group. In that case it is sufficient to put only common part of the source name, which ends with '_' or '.' sign:

<common part of source name>_

or

< common part of source name >.

The elements of the metascripts diagram are located in vertical columns. Each of the columns represents appropriate level of metascripts, which is shown on Fig. 7:

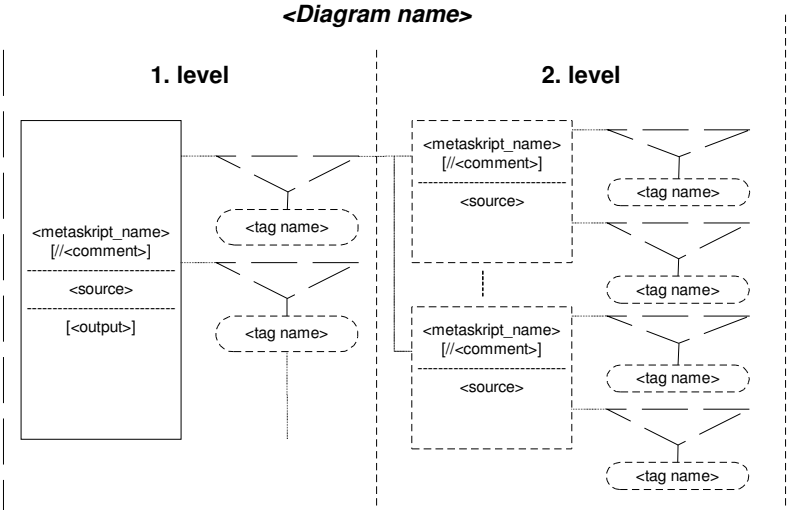

Fig. 7. Arrangement of elements on metascripts diagram

Fig. 9 shows that element 'link' connects metascripts to attached sources and (optionally) metascripts of lower level. Link to metascripts of lower level is 1:0..N type of link.
Number of levels in metascripts diagram determines the level of generator, so the levels of generators in scripting model are:

- one-level generator - simple generator that consists of only one metascript (its metascripts diagram has only one level) and
- multi-level generator - complex generator that includes more metascripts arranged on different levels (its metascripts diagram has more than one level). Each branch in the metascripts diagram defines appropriate generator of lower level. The lowermost level of each branch in metascripts diagram defines one simple one-level generator (Fig. 8). Multi-level generator is given by superposition of more one-level generators.

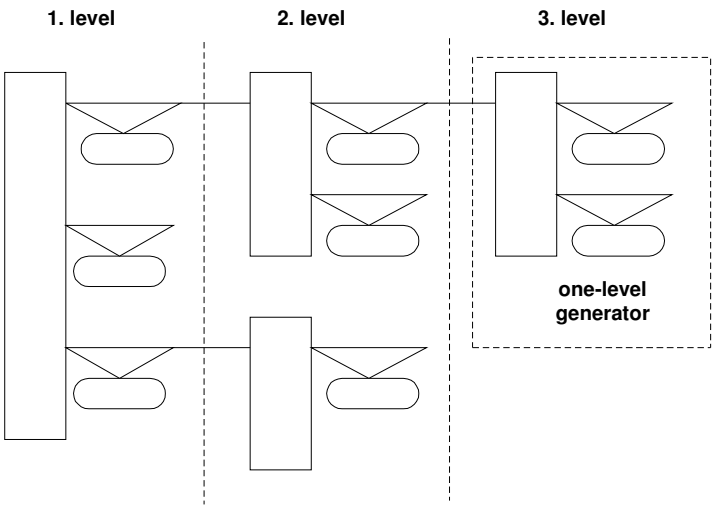

Fig. 8. Multi-level generator is given by superposition of more one-level generators

4. Example of simple two-level generator

Following example gives the scripting model of simple generator of applications in C++. The application enables data entry, their simple processing and console output. Individual applications within such defined program domain differs one from another according to data that they are using (could be of different names and types). Application specification is designated by diagram of the application specification parameters (Fig. 9):

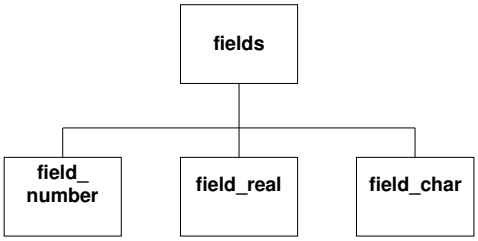

Fig. 9. Diagram of the application specification parameters of simple generator of applications in C++

It's shown on Fig. 9 that application specification consists of tag 'field' which includes tags 'field_number', 'field_real' and 'field_char' that define data types. Using of the values for particular tags is defined in the metascripts diagram.
Appropriate application specification looks like this:

fields:
field_<type >:<value>

where *type* represent one of three offered possibilities: number, real and char
Example of application specification:

fields:
field_number:first
field_real:second
field_char:third

Tags for types will be replaced in generated application by types from appropriate metascripts (according to target program language of application) which is shown in the metascripts diagram (Fig. 10).

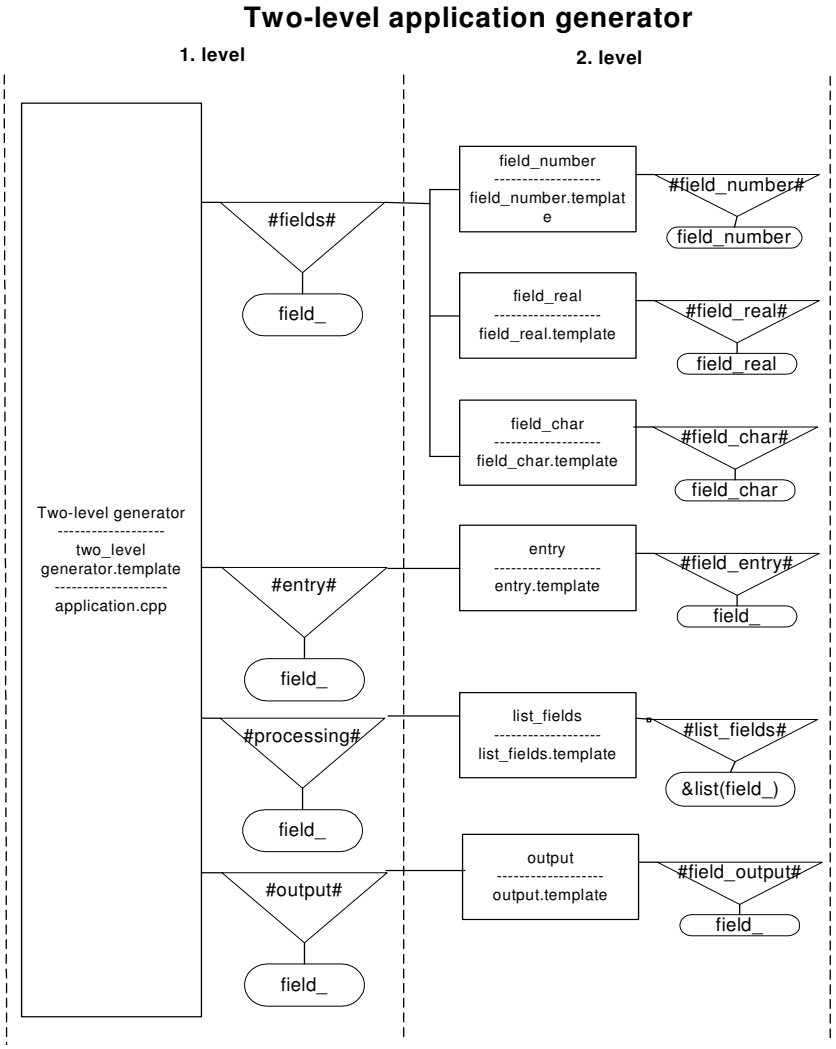

Fig. 10. The metascripts diagram of two-level application generator from the example

Diagram (Fig. 11) shows distribution of tags from diagram of the application specification parameters (element *source*) on appropriate metascripts. The rationalization at second level with '_' is used at metascripts 'entry', 'list_fields' and 'output', so the sources are defined with 'field_' (means all that started with *field*). The possibility of preprocessing is used to form a list of fields because all elements of the list ends by comma, except the last one (in source defined by *&list(field_)*). Furthermore, diagram shows the hierarchy of metascripts that defines the hierarchy of

generated parts of application. The main metascript at first level defines output file for generated program code ('application.cpp').

The example uses following metascripts:

Two-level generator:

```
#include <iostream.h>
#fields#
void main(){
//entry of values
#entry#
//processing -forming the list of fields
#processing#
//console output of values
cout << "-------------------" << endl;
#output#
}
```

field_number:

```
int #field_number#;
```

field_real:

```
float #field_real#;
```

field_char:

```
char #field_char#[40];
```

entry:

```
cout << "#field_entry# = ";
cin >> #field_entry#;
```

list_fields:

```
cout << "List of fields:#list_fields#";
```

output:

```
cout << "#field_output# = ";
cout << #field_output# << endl;
```

The generated program code in C++ for the example is following:

```
#include <iostream.h>
int first;
float second;                                   data declarations
char third[40];
void main(){
//entry of values
cout << "first = ";
cin >> first;
cout << "second = ";                            data entry
cin >> second;
cout << "third = ";
cin >> third;
// processing -forming the list of fields
cout << "Lista polja:first,second,third";       list of fields
// console output of values
cout << endl << "--------------------" << endl;
cout << "first = ";
cout << first << endl;
cout << "second = ";                            data output
cout << second << endl;
cout << "third = ";
cout << third << endl;
}
```

So, the fields 'first', 'second' and 'third' are distributed on different parts of generated application according to the metascripts diagram, which (here) defines two-level generator.

5. Fields of use and up-today developed generators

The problem of maintenance

There are several generators made according to the offered concept of generative programming. Those generators are used for generating of web applications in scripting languages, so they produce code in Perl, ASP and HTML. One of the features of those applications is that they consist of many scripts. That complicates maintenance because each change must be updated inside many program modules. Typical example is application for distant administration of database through web interface:

- it's necessary to maintain forms and scripts for each table in database, which perform following operations:

- output of table content,
- entry of new record into a table,
- update of table record and
- deleting record in table.

- except that, some master-detail and look-up connections among tables should be implemented

It was shown in the example of application generator for distant database administration that one single data, like *Surname*, which is specified only once in the application specification, ends on 26 different positions in different scripts of generated application.

Modifications of generators

The main benefit of using scripting languages in development of generators should be easier modifications of generators, so these should be easier adapted to generation of specific types of applications. The example from the praxis is adaptation of the web questionnaire generator to generate web tests. The generator specification of web questionnaire application includes questions, question types (one possible answer, more possible answers, scale of answers etc.) and variants of showing results. The needed modification was consisted of evaluating answers. So the application specification was modified by defining number of points for every possible answer. Metascripts were modified by adding code for showing points in the results.

Frequent modifications on different levels, as shown on Fig. 3 are the point of generative application development according to the offered concept of generative programming, based on scripting languages. The existing generators are, according to the needs, adapting for use in new projects. Developed generators become the knowledge base for developing of new applications (Fig. 4).

6. Future work

The future work on improving the generative application development could be divided into these main areas:
- application rengineering. Making generators starts with rengineering existing prototype applications. Some features which should be part of program specification have to be extracted, as well as code templates. Some methods od machine learning, like Decision Tree Learning, which is appropriate for *attribute-value* parts (Mitchell, 1997), could be used because program specification consist of *attribute-value* parts. Except these, some methods from automatic document classification, like *bag of words* representation could be used in extracting key words from prototype applications.
- connecting scripting model to existing object model given by UML diagrams. Scripting model of generator is, as well as UML, the graphic model. Some concepts

from scripting model can be compared to the UML, e.g. metascripts correspond to classes, links among metascripts correspond (partly) to inheritage.
- developing new program platforms for making generators. Some experiment were made in development of C++ libraries for making generators. In that case structure of generator could be given in UML diagrams, while the process of generating apllications is still given by scripting model diagrams.

7. Conclusion

This paper offers the scripting model of application generators and appropriate generative application development, based on scripting languages. The accent was put on modeling the generator as a whole, and its architecture, in a difference to some other investigations that put the accent to the development of specialized scripting languages for making application generators, and within that, on implementation of different kinds of character strings (Lemaire, 2003)(Štuikys et al., 2001). The offered scripting model leaves the possibility of generating specific character strings through preprocessing of sources, by appropriate user defined functions. In that sense, this investigation has some complementarities with mentioned specialized scripting languages.

In relation to existing object model, designated by UML diagrams, scripting model is easier, because it consists of only two diagrams, but there some other differences:

- scripting model is a model of aspects, while the object model is a model of data and functionalities
- scripting model defines application generator, not an individual application
- scripting model is independent to target programming language of application, because that is specified on lower level, within metascripts.

Scripting model is a base for generative application development, as a cyclic process that includes development of applications, but also the generators, as a knowledge base for future applications. The basic feature of that process must be the flexibility, which is achieved by using scripting languages, because of frequent needs for modifications of generators; to adapt them to the new/modified problem domains.

Bibliography

Czarnecki, K., Eisenecker, U. W. (2000).*Generative programming: methods, tools and applications, Addison-Wesley*, ISBN 0-201-30977-7

Gray J., Lin Y., Zhang J.(2003). Levels of Independence in Aspect-Oriented Modeling, *Middleware 2003: Workshop on Model-driven Approaches to Middleware Applications Development*, Rio de Janeiro, Brazil

Kinczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J.M., Irwin, J. (1997). Aspect-Oriented Programming, *Proceedings of the European*

Conference on Object-Oriented Programming (ECOOP), Springer-Verlag LNCS 1241., Jyväskylä, Finland

Kühl, D. (2000).: STL and OO Don't Easily Mix, *Proceedings of the GSCE, Workshop on C++ Template Programming*, Erfurt

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

Lemaire, C.(2003). *CODEWORKER Parsing tool and Code generator - User's guide & Reference manual*, CodeWorker.free.fr, Available from:http://codeworker.free.fr/CodeWorker.pdf, Accessed 2006-03-31

Mitchell, T.M. (1997). *Machine Learning*, WB/McGraw-Hill, ISBN 0-07-042807-7

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

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, Available from: http://soften.ktu.lt/~damarobe/publications/Vytautas_Stuikys.pdf, Accessed 2006-03-31

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, Božidar Kliček, Jasminka Dobša**
2. **Title of Manuscript:** Generative Application Development Using Scripting Model of Application Generators
3. **Key words:** generative programming, scripting model, generative application development, aspects, generator
4. **Abstract:** The paper offers graphic and aspect oriented model of application generators based on scripting languages. Generative programming based on scripting languages is an alternative to recently predominant object oriented approach. Scripting model is a model of application generators, while object model defined by UML diagrams is only an application model and has considerable problems with aspect modeling. Except that, scripting model is simpler than the object and enables more flexibility in development of generators. The paper also suggests the cyclic generative model of application development based on scripting model, which includes parallel development of generators and applications. Such application development enables shortening of application development cycle, performance optimization and simplifying of maintenance. Scripting model is tested in development of application generators in Perl and development of dif-ferent web applications. It's shown that offered scripting model enables rapid de-velopment and simple adaptation of generators to the problem domain modifications.
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.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: 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

1. Digital Photo:
2. First / Middle / Family Name: Jasminka Dobša

3. Titles: Msc
4. Position / Since: Assistant /1997
5. Institution/Firm:Faculty of organization and informatics, University of Zagreb
6. Place and Date of Birth (yyyyy-mm-dd): Čakovec, Croatia, 1971-08-28
7. Nationality / Citizenship: Croatian/ Čakovec
8. Field of interests (key words): machine learning, text mining
9. Hobbies: -
10. E-mail address: jasminka.dobsa@foi.hr
11. Home Page: http://www.foi.hr/nastavnici/dobsa.jasminka/index.html
12. Postal address: Pavlinska 2, Varaždin, Croatia
13. Phone & Fax #:385 042 390 800, 385 042 213413