Generating Web Applications Using CodeWorker

Josip Mlakar, Danijel Radošević, Ivan Magdalenić
Faculty of Organization and Informatics
University of Zagreb
Pavlinska 2, 42000 Varaždin, Croatia
{josip.mlakar, danijel.radosevic, ivan.magdalenic}@foi.hr

Abstract. *Generative programming as a discipline of Automatic programming uses different approaches and tools in building of generators and associated applications. CodeWorker enables possibility to define program specification by using Extended BNF notation. This approach was described in the paper and compared to some others, including Open Promol, XVCL, GenVoca and SCT. An example of Web application was given showing the advantages and drawbacks of this approach in building of Web applications.*

Keywords. CodeWorker, generative programming, code generators

1 Introduction

Generative programming [1] is a programming approach focused on automation of application development process. Key for the automation of a software development is a generative domain model which consists of the problem space, the solution space and the configuration knowledge which maps between them [1]. The solution space consists of the implementation components and their combinations. The implementation components should be designed in a way so they maximize their combination capacity and reusability while minimizing redundancy of a code. The problem space consists of the concepts and domain specifics features. The problem space defines Domain-Specific Language (DSL) [12], while configuration knowledge defines feature combinations, default settings, default dependencies and construction rules [12].

Most of the methods for an object oriented analysis and design are focused on a development of software individually [2]. Biggest flaws of those methods are that they don't recognize the difference between the development for reuse and the development with reuse [2].

Focus of this paper will be on one of the tools devoted to generative programming, CodeWorker [7]. Using the example of a web application generator it will demonstrated the process of developing generators using CodeWorker and the benefits of the generative approach in general. Also there will be given a quick overview of some other tools devoted to generative programming and their comparison to CodeWorker.

2 Background of the research

Besides generative programming there are other popular approaches to modern day software development. Some of them will be described in the following sections.

Object-oriented programming (OOP) is currently the most dominating programming approach. OOP reflects the structure of the application domain with the programming structure. The two core terms of OOP are object and class. While class presents the type which describes data and operations for processing data, object presents an instance of the class. OOP contains some characteristics important for creating reusable components like polymorphism, encapsulation and class inheritance [6]. While OOP provides a way to create reusable components in a sense of a frameworks it does not provide a way to automate creation of meaningful application or system from those components. Wanted components must be connected manually by a programmer. But before that, programmer has to learn how to use the framework which can be very costly and time consuming.

Some of the programming problems can't be adequately described with OOP techniques. Aspect-oriented programming (AOP) extends OOP paradigm by adding the aspects. Aspects are described as properties of the system that spans trough different parts of the source code [14]. AOP enables splitting of each aspect and describing aspects in their natural form. Most of the currently popular programming languages provide a way to describe some of the aspects. Aspect oriented languages are mostly the extensions of already existing languages so there are AspectC++ which is the extension for C++ and AspectJ which is the extension for Java programming language.

Metaprogramming is a process of specifying generic software templates which can be used for an automated generation of new software components [13]. Those software templates are called *metaprograms*. Metaprogramming usually involves generative programming techniques but it doesn't always have to be the case. In example metaprogramming can be used for programs that are modified during runtime. In that case there is no code generation and there is no need for generative programming techniques.

Generic programming [4] focuses on finding similarities between similar implementations of the same algorithm. Main process of generic programming is called lifting [5]. Algorithms are lifted until they reach suitable level of abstraction, which maximize the algorithm reusability. When lifting is done it's easy to find patterns between requests. It's a common thing to find the same set of requests between different algorithms. Those sets are then grouped into concepts. Concepts describe a set of abstractions in which every abstraction meets all the requests of the certain concept. Difference between generative and generic programming is that generative programming focuses on creating generators that can provide solution for every distinct problem inside certain domain while generic programming focuses on implementing algorithms that can work in every distinct problem inside domain. Generative programming results with tailored solution for each domain problem, while generic programming results with a generic solution that solves all possible domain problems.

Domain engineering is a process of active colleting, organizing and storing past experiences in system or component development of a certain domain in a way that provides reusable means for future projects [12]. There are three phases of domain engineering [3][12]. Domain analysis is the first phase. In this phase domain focus is determined and all important domain information are collected and integrated into coherent domain model. Second phase is domain design in which architecture family of the system is developed and development plan of a system is determined. Architecture family [3][12] represents the systems that have common properties but there are still some properties that different one system from another. Third and the final phase is domain implementation in which architecture, components and plan of production are implemented.

This paper focuses on creating application generators in a CodeWorker, but to get a wider picture about tools devoted to generative programming there will be briefly described some other tools that serve the same purpose. Those tools are Open PROMOL, XVCL, GenVoca and SCT.

2.1 Open Promol

Open PROgram MOdification Language (Open PROMOL) is a script language for specifying modifications of application written in arbitrary programming language [11]. It allows application modifications using the process of gluing. Gluing is a process of inserting data or components in wanted part of a source code. There are two types of gluing. First type is outside gluing which is used for creating composition structure of more different and mutually independent components. Second type is inside gluing. It's is used for development of monolith components.

2.2 XVCL

XML-based Variant Configuration Language (XVCL) is script language for configuring and managing variants in programs and other kinds of documents [15]. Variants can be described as differences between system requests that belong to the same domain. Automating variant management reduces the risk of errors in applications which makes development of domain specific applications cheaper and less time consuming. Software Product Line (SPL) is achieved with that process of automation. Software Product Line (SPL) or Product Family (PF) is a group of software intensive systems that share a common set of features that meet particular stakeholders' specific needs [10].

To accomplish that XVCL separates generic and adjustable fragments into x-frames. Those frames are organized into hierarchy which makes architecture of the software product line. XVCL processor is going through x-frame architecture executing commands defined in each frame. In that process every x-frame adjusts his sub-frame creating unique system that meets all domain and system specific requests.

2.3 GenVoca

GenVoca is a composition model that uses component to define scalable hierarchy system families [1]. GenVoca speaks in favor of layer decomposition of implementation. Layer decomposition implies decomposition of the system on components so that every component is made only of primitive domain features. Since those components are very small their reusability capacity is relatively big. Software system hierarchy is defined with the series of progressively abstract virtual machines. Component is an implementation of a virtual machine while realm is components cluster. Realms and their components define a grammar whose sentences are software system. As set of sentences define language so set of compositions defines system family [1].

2.4 SCT

Specification-Configuration-Templates (SCT) [9] is generator model that is based on dynamic frames. Every frame consists of specification, configuration and template.

Specification defines features of a generated application. Generator generates source code by merging features with template in designated spots. Besides defining features of generated applications, specification also defines place where generated source code should be generated. Specification consists of attribute-value pairs which are organized in hierarchy order. A value of the attributes presents features of a generated application.

Configuration consists of configuration rules that define the connection between specification and template. Configuration rule is made of connection, source and code template. Connections mark spots where content from specification should be put. Every connection has a source that is defined in a specification. Source of a connection is a value of an attribute that has the same name as the name of a connection. Code template is the fragment of a code that contains connections.

The process of source code generation starts with the initial SCT frame that contains the complete Specification and Configuration and only one template from the set of all Templates. Other SCT frames are produced dynamically for each connection in the template, forming a generation tree [8] (Fig. 1).

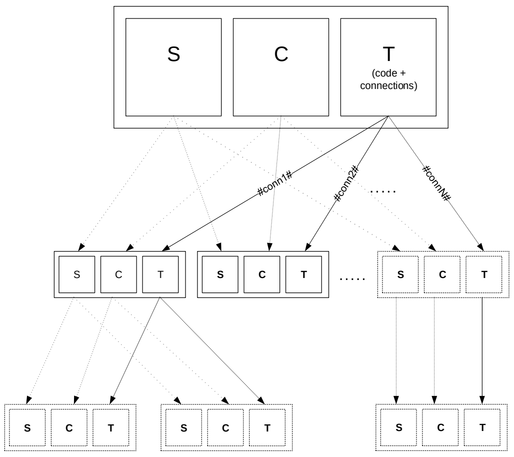


Fig. 1. Generation tree

3 CodeWorker

CodeWorker [7] is a parsing and code generation tool. It supports three ways of code generation which can all be combined into the same application generator.

First way is called generation mode. Generation mode consists of four steps (Fig. 2). First step is to create a specification, second to write parsing script which parse specification and populates parsing tree, third to write templates and fourth step which is optional is to write leader scripts which call all the commands for parsing and code generation.

Second way is called expansion mode. Expansion mode is used for expanding already generated or by hand written file. It allows inserting small portions of a code into existing file. Points where those portions should be inserted are marked with a ##markup## and wanted identifier name.

Third and the last way is called translation mode. Translation mode is used when file must be rewritten in different syntax (source-to-source translation) or when source file has to change for optimizing, refactoring, rewriting some portions.

All data that wants to be inserted into templates, thus creating specific features of an application, should be fed into data structure called parse tree (root of a parse tree is called *project*). Specification is an abstraction of the tree structure so it can be said that specification explicitly describes features of a generated file.

Since specification can be written in arbitrary syntax, it's required to write scripts to parse specification and populate parse tree with specified data. Those scripts are called parse scripts and CodeWorker provides two methods to write them, declarative and procedural. Declarative method implies on using Extended Backus-Naur Form (EBNF) to write the parse scripts while procedural method implies using CodeWorker script language. Procedural method is the older way to write scripts and it's faster than declarative method but it's not so flexible. Because of that CodeWorker authors finds that method obsolete. It's also worth mentioning that CodeWorker script language can be used in EBNF scripts if there is need but it should be announced with characters "=>" and end with character ";".

Templates are skeletons of generated code. They are written with combination of a targeted programming language and CodeWorker script language. CodeWorker script language in a template should be marked with character "@" at the start and the end of an instructions or with characters "<%" at the start and "%>" at the end of instructions. Script language is used for specifying where and what data from the parse tree should be inserted into output file.

Leader scripts are used to execute all the commands for parsing and generating applications. They are not necessary in development of application generators but they automate the process of code generation even more.

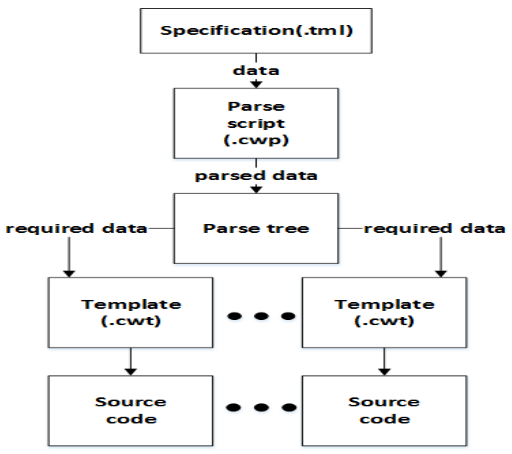


Fig. 2. CodeWorker generation process

4 An example

Simple web application generator[1] was created to show the capabilities and benefits of using CodeWorker and generative approach in general.

Even the smallest change in database can cause big problems in adopting the software to the new structure. It often requires changes in more than one script or class which can, depending on a size of a project, require a lot of time and concentration to do it right without creating more problems and errors.

In this section it is described how to address this issue with a generative approach. First step is to define the problem space. Problem space for this specific example is a cash registers application domain. To make the focus primarily on usage of generative approach and process of developing application generators using CodeWorker example used in this paper is fairly simple. Data model of all generated applications consists of four tables, table with data about users (workers in a store), table with data about products, table with data about orders and table with data about order items. Every generated application should use that database model, but names of tables including names, types and number of attributes are completely arbitrary.

Content and structure inside the tables, including some user interface properties like title, organization and number of navigation items are variants of distinct applications inside chosen domain. Those variants are all defined in the specification and depending on specification content all scripts (PHP is used in this example) are generated according to that content. With this approach whenever change is made in a database, only adjustments that have to made are in a specification, generator does all other work (i.e. generates all the scripts according to those changes).

Configuration knowledge of a generator consists of a leader script, specification, parse script and parts of a template written in a CodeWorker script language, while solution space consists of templates as a whole. Parts of a configuration knowledge and solution space will be further explained in the following passages.

Leader script (*leader_script.cws*) calls set of commands required for a generation process on a CodeWorker processor. This script is called directly from a CodeWorker command line with a command -*script*. Command *parseAsBNF* reads the specification (*specification.tml*) and using parse script (*parse.cwp*) extracts data and populates the parse tree. Command *generate* generates source code (*index.php, getOrder.php, etc.*) by combining templates (*index.tml, getOrder.tml, etc.*) with an appropriate data from the parse tree (Fig 3.).

[1] Example of generated web application is available at http://gpml.foi.hr/php_codeworker

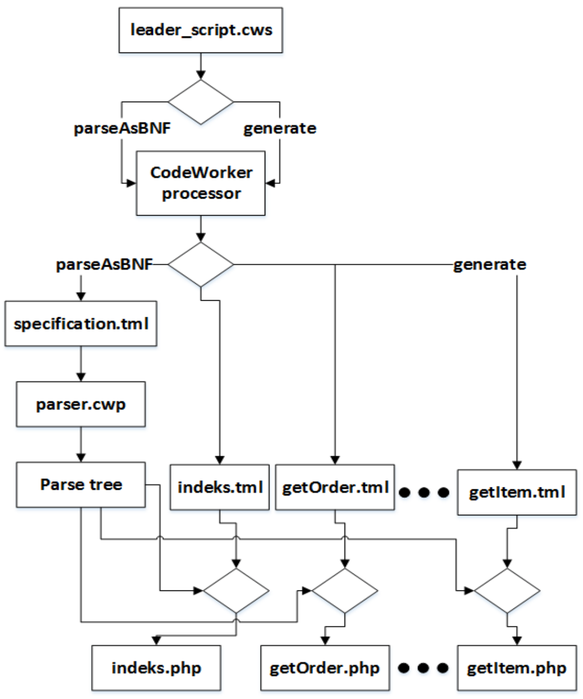


Fig. 3. Web application generator

In example specification is written in a new syntax, created specifically for the purpose of an example. Character "D" defines the data about database. First block contains attributes for database connection (server, database name, user and password).

```
D
{
    server localhost
    database shopDB
    username root
    password pass
}
```

Second block is actually an array of blocks containing the data about structure of each table. Each table block starts with a character "T" and follows up with a name of a table in a database. In each table block there is a special attribute "P" which stands for a purpose of that specific table (whether is a table that contains data for users, products, order or order items). That attribute should not be changed, it tells the CodeWorker processor for what is each block precisely meant. Other attributes in table block present each of the attribute in a database table. Every attribute can have some of the special properties. Those properties are defined with special characters: "A" property can have values "pk" (attribute is primary key), "*table*_vk" (attribute is a foreign key for an attribute of a same name in a table which name is written before "_vk")," username" (this attribute is for username), "password" (it is for password); "N" property should be followed with "yes" and it tells processor that this attribute presents name of a product, employee...; "R" property should be followed with "yes" and it tells the processor that this attribute value should be shown in receipt; "H" property should be followed with "yes" and it tells the processor that this attribute value should be shown on purchase history; "-" property should be followed with "yes" and it tells the processor that this attribute represents number of products in stock; "+" property should be followed with "yes" and it tells the processor that this attribute value should be accounted into receipt.

```
[
        T products
        {
                P products
                product_id A pk R yes
                product_name N yes R yes
                price R yes + yes
                warehouse_state - yes
        }
... ]
```

Character "S" defines the block of data about user interface. "N" is followed with the title of the generated app. Other attributes are for defining navigation bar. Every element is followed with character "L" and the link to the page it should lead. If element consists of more than one word those words should be separated with underscore.

```
S
{
        N Custom_shop
        Cashier L index.php
        Order_history L order_history.php
        Sign_out L signout.php
}
```

Character "G" defines block with data where each script should be generated and which branch of a parse tree should be used in process.

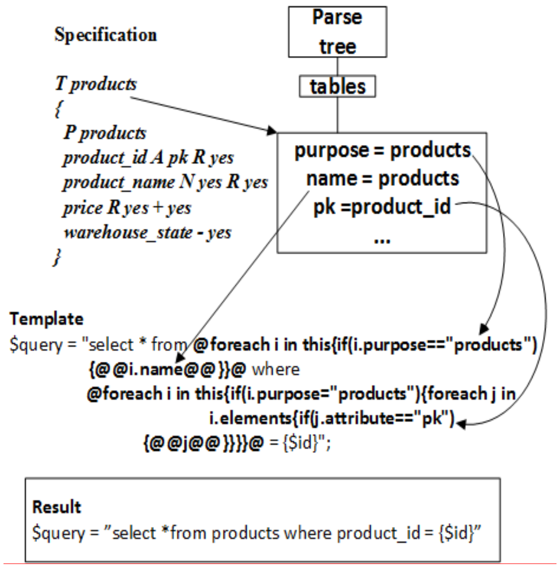
```
D
{
        index.cwt project.table path/index.php
        ...
        getItem.cwt project.table path/getItem.php
}
```

Declarative method for creating parsing scripts is used for parsing specification. Using the EBNF, parsing script goes trough specification while extracting data. Those extracted data is inserted into parse tree using CodeWorker scripting language.

Leader script makes sure all the necessary actions are executed. It firstly parses the specification into parse tree and then generates final scripts from templates.

```
parseAsBNF("parser.cwp",project,"skripte/specification.tml");
foreach i in project.files
{
        if(i.cnode=="project.db")
        {
                generate (i.cwt,project.db,i.out);
        }
        else if (i.node=="project.table")
        {
                generate(i.cwt,project.table,i.out);
        }
        else
        {
                generate (i.cwt, project, i.out);
        }
}
```

Templates consists of PHP code and CodeWorker scripting language which make sure right data from the parse tree is inserted into right place. In a generated code scripting language is replaced with those data (Fig. 3).



```
T products
{
  P products
  product_id A pk R yes
  product_name N yes R yes
  price R yes + yes
  warehouse_state - yes
}
```

```
$query = "select * from @foreach i in this{if(i.purpose=="products")
        {@@i.name@@}}@ where
        @foreach i in this{if(i.purpose="products"){foreach j in
                i.elements{if(j.attribute=="pk")
            {@@j@@}}}}@ = {$id}";
```

```
$query = "select *from products where product_id = {$id}"
```

Fig. 3. Generation process

5 Conclusions

This paper describes development process of application generator using CodeWorker and generative approach. It was also given a preview of other tools devoted to generative programming.

CodeWorker enables writing specification of a generator in the custom syntax which adds to flexibility compared to other tools described in this paper. Writing specification in custom syntax means that syntax can be fully adjusted to domain that generator is developed for. But this flexibility also has its drawbacks since allowing so much freedom opens more space for errors. It also makes process of development longer since this approach requires writing parsing scripts for specifications.

Generative approach in software engineering focuses on creating generator that satisfies variants inside problem domain. Creating application generators takes more time then to make application in a classic way so generative approach is not cost effective in a short term. Nevertheless application created just for one system can be used only in that one system, while application generator (once developed) can generate applications that fit the requests of each system inside problem domain. So with each generated application, cost effectiveness rises and overcomes the classical approach.

Software systems are growing larger and more complex so there is a great need for automation of as much software development process as possible. Usage of generators can be seen in most modern integrated development environment as they generate class skeletons and other generic parts of an application.

In our further research we will focus on studying other advanced programming techniques which are expected to dominate in the future where most of software development process will be automated.

References

[1] Blair, J., Batory, D. A Comparison of Generative Approaches: XVCL and GenVoca. Technical report, The University of Texas at Austin, Department of Computer Sciences, December 2004.

[2] Czarnecki, K., Eisenecker, U. Components and Generative Programming. ACM SIGSOFT Software Engineering Notes, vol. 24, no. 6, pp. 2-19, 1999.

[3] Czarnecki K,. Eisenecker, U.W. Generative Programming: Methods, Techniques, and Applications. Addison-Wesley, 2000.

[4] Garcia R., Jarvi J., Lumsdaine A., Siek J. and Willcock J. "An extended comparative study of language support for generic programming", Journal of Functional Programming, 17, pp 145-205.,Cambridge University Press, 2007.

[5] Gregor D., Järvi J., Siek J., Stroustrup B., Dos Reis G., Lumsdaine A. "Concepts: Linguistic Support for Generic Programming in C++", Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, OOPSLA 2006, Pg. 291-310, Portland, USA

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

[7] Lemaire, C. "CODEWORKER Parsing tool and Code generator - User's guide & Reference manual," http://codeworker.free.fr/CodeWorker.pdf, 2010.

[8] Magdalenić, D. Radošević, and T. Orehovački, "Autogenerator: generation and execution of programming code on demand" Expert Systems with Applications, vol. 40, no. 8, pp. 2845–2857, 2013.

[9] Radošević D., and Magdalenić I. Source Code Generator Based on Dynamic Frames, Journal of Information and Organizational Sciences, vol. 35, no. 2, pp. 73–91, 2011.

[10] Roško, Z. "Predicting the Changeability of Software Product Lines for Business Application", 23rd International Conference on Information Systems (ISD 2014), Varaždin, Croatia, 2014.

[11] 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.,

[12] Tolvanen J.P., Rossi M. Metaedit+: Defining and using domain-specific modeling languages and code generators. In OOPSLA 2003 demonstration, 2003.

[13] Trujillo S., Azanza M., Diaz O. Generative metaprogramming. GPCE '07: Proceedings of the 6th international conference on Generative programming and component engineering, October 2007.

[14] Yi S., He C., "A Comparison of Approaches Toward Reusable Aspects", International Conference on Computer Science and Intelligent Communication (CSIC 2015), Zhengzhou, China, 2015.

[15] Zhang H., Jarzabek S. XVCL: a mechanism for handling variants in software product lines, Science of Computer Programming, Volume 53, Issue 3 (December 2004) Pages: 381 – 407