ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)

https://doi.org/10.17559/TV-20200603143209
Preliminary communication

Usage of Autogenerator in Handling of Dynamic Specification

Danijel RADOŠEVIĆ*, Ivan MAGDALENIĆ, Andrija BERNIK

Abstract: Generative Programming (GP) is a discipline of Automatic programming introduced in the late nineties with the intention of introducing more flexibility into the development process of generators. But, despite the recent advances in this area, most of GP approaches still use the static form of specification for defining target application features. The dynamic form of specification changes during the application execution and thus requires a dynamic generation system to generate the code in real time. This paper presents the usage of Dynamic Specification that is based on our previously introduced generation system named Autogenerator. The model is tested on a case study in the form of an online language dictionary.

Keywords: autogenerator; dynamic specification; generative programming

1 INTRODUCTION

Generative programming (GP) is a discipline of Automatic programming [1]. Its aim is to produce code generators for specific problem domains. Some goals of GP include achieving properties like better reusability of software artifacts and configuration elements, improvement of generators development process and easier adaptation to the problem domain. For that purpose, GP enables concepts like code generation using multiple generators, usage of advanced object-oriented programming concepts and usage of scripting languages. There are different models developed within GP and some of them are supported by development tools. The form of software specification is one of the main differences among these GP models. While the common role of the specification is to define software features on higher abstraction level than the target programming language of the application, there are different approaches to achieve that goal. Some of the approaches are based on the Domain Specific Languages (DSLs) that are the programming languages oriented to a specific problem domain, unlike the general purpose programming languages [2]. The others use some kind of frames, as the building blocks of the application to be generated, which were introduced by Basset's frames [3]. While the form of the specification varies, the general principle is to fully specify the application before its execution.

This paper considers the possibility of using database data in a form of generator specification. In this case, the records could have the heterogeneous form, governing the generation system to use them in the appropriate way. The main benefit of such approach is in reducing the complexity of data structures, by reducing the number of necessary data tables and their interconnections. For this purpose, our previously introduced generation system, named Autogenerator [4], was adapted in order to handle Dynamic Specification.

2 HANDLING OF APPLICATION SPECIFICATION IN DIFFERENT APPROACHES OF GENERATIVE PROGRAMMING

This section provides a brief overview of GP approaches and their ways of handling the application specification.

Most frameworks, mechanisms and languages within GP use inlining as a technic for definition and control of source code generation. For example, C++ provides a solution based on template metaprogramming [1], where generated programs are expressed as parameterized types and code is produced by a compiler through inlining [5]. Java uses annotations for number of use cases and one of them is a code generation during the compile-time and deployment-time processing [6]. Although inlining is a powerful way to extend software functionality by generating additional source code later on, it ties itself to one specific programming language. Avoiding of inlining enables the generation of code in any programming language depending on the code Templates [7, 8].

X - frames have been shown by authors of [9] as building blocks for code to be generated. These x - frames are organized in a tree structure, where specification of x - frames contains software specification [10]. Other x - frames combine the code with break sections that define insertion of variable parts (defined by other x - frames). Configuration elements are specified implicitly, in break sections, defining different kinds of insertion and adaptation [11]. XVCL uses XML for definition of both x-frames and application specification, and application specification is dispersed across all x - frames [12, 13].

A template engine is a software component that takes previously prepared text and data as its input, integrates them by following certain processing rules and results in a text document containing the data. Velocity is an example of such a template engine. It is Java-based template engine created by the Apache Software Foundation [14]. Java template engines Apache Velocity was used instead of Freemarkeror MVEL2 because it is widely used and the most powerful. Velocity uses four components in its transformation process: data model, templates, references and properties. Data model contains information about specific transformation and it is unique for each application. Velocity is specific to its different definition and organization of data model, template engine and templates. The data model can be specified by a set of key-value pairs written in Java programming language or defined with a set directive in Templates. Velocity uses Backus normal form (BNF) as a format of application specification.

Stringtemplate is another template engine which defines rules in code that deals with user defined markup in Templates. In most cases, user defined markup is replaced with some text [15].

Rascal [16] is a meta-programming language for source code analysis and transformation. It uses its own data types, control structures, functions and other elements of standard programming languages on a meta-level. Those elements are not used only for code generation, but also for code analysis, including syntax definition and parsing. Rascal uses Syntax Definition Formalism (SDF) [17], as kind of a meta-syntax to specify grammars. SDF modules can be imported in a Rascal module and these grammar rules can be applied in writing patterns of source code [18, 19].

CodeWorker [20] is an open source parsing tool and a source code generator. It supports code generation by parsing the existing or by creating the new language. The parsing and source code generation processes are driven by the Codeworker's scripting language that is analogous to the C programming language whereas syntax of templates is similar to JSP, ASP, and Velocity [21].

Drawing on Service-Oriented Architecture (SOA) [22, 23] combines the principles of distributed, component-based computing, Model-Driven Architecture, service and quality of service guarantees, and generative techniques. Its main purpose is the design of unified framework that would facilitate the interoperation of heterogeneous distributed components as well as the construction of high-quality computing systems based on them. UniFrame [24] consists of a standards-based meta-model for indicating the contracts and constraints of the components, an automatic generation system to achieve interoperability, guidelines for specifying and verifying the quality of individual components, a mechanism for automatic discovery of appropriate components on a network, a methodology for developing distributed SOA based architectures, and mechanisms for evaluating the quality of the resulting software products [25]. Application specifications in Uniframe are textual, in separated files and use Two-Level Grammar (TLG) based on OOP concepts [26].

Business Process Model and Notation (BPMN) [33] is a graphical representation for specifying business processes in a business process model. Supported by tools like 'bizagi' (www.bizagi.com) it is used for generation of some part of applications e.g. user interfaces [29] and GUI Prototypes [30]. BPMN can also be used for generation of executable Web services in Enterprise resource planning (ERP) domain [31]. BPMN models can be used as a source for transformation to target model for further verification and deployment as presented in [32]. JSON can be used as format to store specification of BPM objects which can be later used for application generation. If a JSON file is runtime loaded and if it contains definitions of BPM objects which are then used for application generation then it can be considered as dynamic specification as well. Libraries are developed and available to convert IBM BPM objects (bpmObj) to JSON strings and vice versa e.g. IBMBPM_JSON-js (https://github.com/NithinBiliya/IBMBPM_JSON-js).

SCT Autogenerator model (Specification, Configuration, Templates) introduces the definition of application to be generated in a separated file called Specification [11]. That model was upgraded with the ability to change Specification in runtime [27]. The next step is to produce the whole specification prior to the source code generation process. We named this kind of specification Dynamic Specification. The main benefit of such an approach is the customization of software features according to the user request in real time. In this paper, an approach for automatic handling of Dynamic Specification is presented trough an online dictionary case study [28].

3 DYNAMIC SPECIFICATION IN AUTOGENERATOR

Autogenerator is a model for generation of source code and its execution on demand [4]. It is an upgrade of SCT generator model whose original purpose was to generate the source code of entire applications. The model of Autogenerator is presented in Fig. 1. User's request is sent to Autogenerator's component called Request handler. Request handler parses the user request and sends a command to SCT generator to generate the appropriate source code. SCT generator generates only the source code needed to fulfil the user request and forwards it to the Execution unit. When the source code is ready, the Request handler invokes the Execution unit to execute the source code together with parameters extracted from Application context.

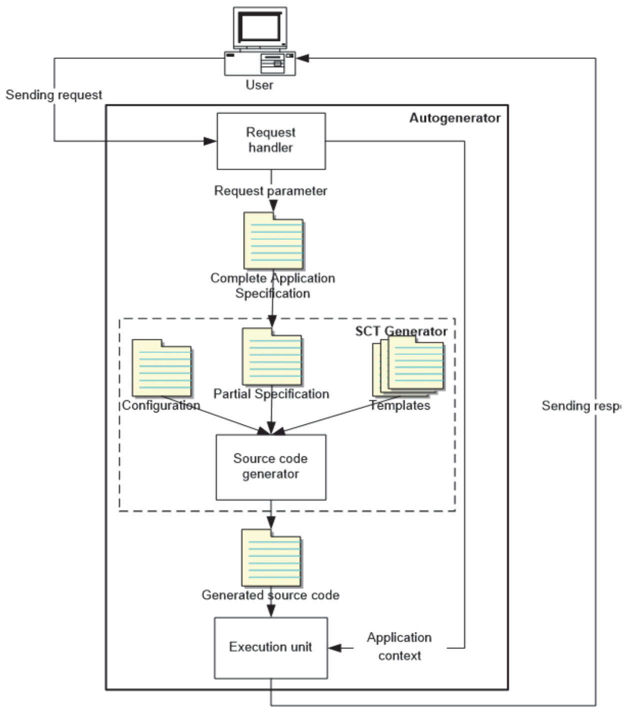

Figure 1 Model of Autogenerator [19]

The original model of SCT generator is aimed to generate source code from three building elements: Specification, Configuration and a set of Templates [11]. Specification contains information about application features in the form of attribute-value pairs. SCT generator is used to define the application family in the target domain. Specification of the SCT generator contains information for generation of one possible application from that family. Configuration is a set of rules that define how code Templates are assembled together, depending on the definition from Specification. Templates are sets of code artifacts that are developed for the target domain and that are used as building blocks of target applications. In the original SCT generator model, Specification is static and it is used to generate the entire application.

Autogenerator uses Specification in a different way. It is used to generate source code for a part of the application depending on the user request. There is no need to send entire Specification to SCT source code generator, but only a part of it. Original Specification is filtered depending on the request parameter and only a part of Specification is sent to the source code generator. In the original Autogenerator model, Specification was static and only a part of it was used to fulfill user request.

In this case study, we are introducing Dynamic Specification. This kind of specification is formed prior to the source code generation process. The general structure of the Specification is defined during the application modeling process, as well as its building blocks. The final Specification is produced on demand, depending on the user request. Fig. 2. presents difference among static and dynamic specifications.

As it can be seen in Fig. 2, there are two main differences among static and dynamic specifications. The first difference applies to specification creation time. Static specification is created during a development of application, unlike dynamic specification that is created in application runtime. The second difference is in specification content. Static specification is just subset of complete specification prepared for all possible applications. Dynamic specification can contain variants unknown during a development process and its creation is triggered by a user request. Dynamic specification represents a substructure in the main data structure (database table) that enables automatic creation of program code and its immediate execution using autogenerator. For example, in our case study we store custom-structured parts of specification in relation database column and we use them as a specification building blocks during runtime.

The process of forming Specification is as follows: The Request handler from the original model of Autogenerator is upgraded with functionality to dynamically form Specification. The database contains user data and prepared parts of the Specification, which define the application structure. Request handler executes database queries, collects results and assembles Specification according to the structure defined during application modeling process. Finally, the Request handler sends Specification to Autogenerator and invokes source code generation process. Once the source code has been generated, it is sent to the Execution unit together with the Application context. The Application context is usually made of information about the user, user session, request parameters, etc.

4 CASE STUDY-ONLINE DICTIONARY

Relational database columns were originally designed to store only homogeneous data structures. Those imply data structures that can contain only similar types of data. The data structures that contain a variety or dissimilar types of data make heterogeneous data structures. A heterogeneous data structure could contain data of different types. Nowadays we can store heterogeneous data in a database using Binary Large Object (BLOB) columns which can contain eXtensible Markup Language (XML), JavaScript Object Notation (JSON), encoded files or any other custom-structured data.

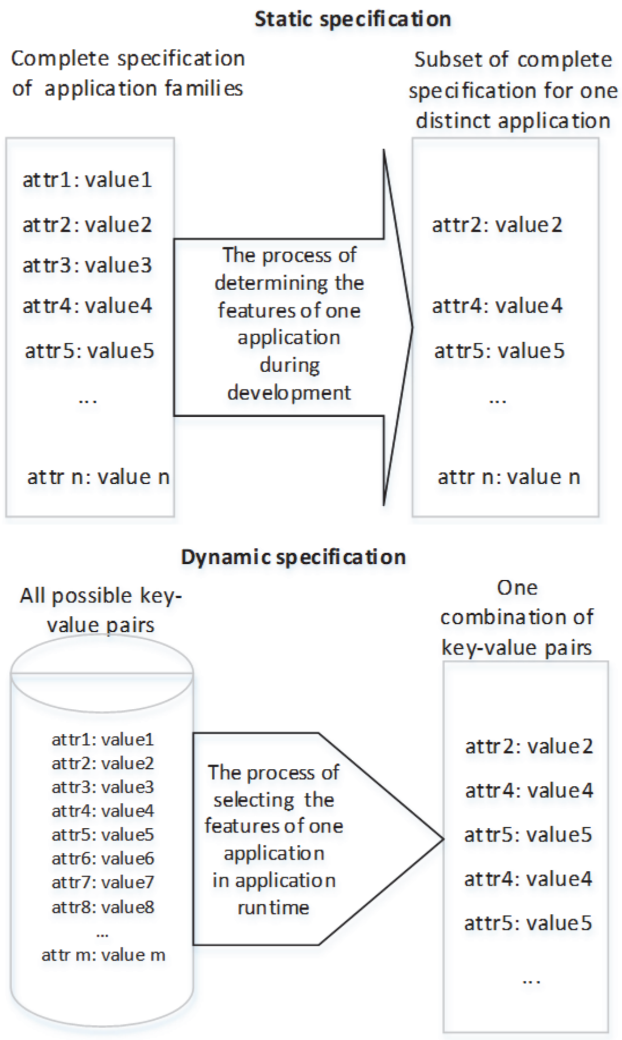

Figure 2 Difference among static and dynamic specifications

XML and JSON data structures resemble the structure of the object data and thus they are commonly used to serialize data in object-oriented application development [34]. However, when saving heterogeneous data in a relational database, we lose some of its core features like relational integrity, quick access, efficient search and usage of complex Structured Query Language (SQL) functions. In today's modern business everything is fast: data is created fast and data changes come faster than ever. As relational databases require doing the modeling all up front, they provide poor support for differing schemas and schemas that need to change. Relational model is complex and sensitive, so one small change can result in cascading impacts across the database. On the other hand, most of the data in everyday use is unstructured which presents a problem for relational databases.

In this paper, we explore a GP approach in handling of heterogeneous data stored in a relational database. Our case study in a form of an online dictionary (Croatian-French and French-Croatian) is based on Autogenerator, a dynamic frames generator based on SCT generator model that stores custom-structured parts of its Specification in a relational database column. It is a mechanism that produces and immediately executes the code. The mechanism is based on the scripting languages ability to evaluate their own code from variables. This approach brings several benefits including high reusability, easier modification and maintenance.

4.1 Online Dictionary Based on Dynamic Specification

The SCT application used for this case study is Croatian-French and French-Croatian on-line dictionary (https://gpml.foi.hr/dictionnaire/), that is realized in a form of Autogenerator. An Administration System is used to manage the database specification of words in both languages and users of the application. It uses a standard, static, predefined Specification. The User Interface of the application enables searching for translations, adding/correcting words and expressions, forums, comments and similar. It uses the dynamic form of Specification, which is defined based on the user request. The key process of this case study is the forming of Dynamic Specification. The process starts with the user request that is transformed into database query by the Request Handler. Each word expression in the dictionary database is represented by the appropriate record in Specification. The Specification of a word expression represents a custom data structure that defines the identification number of the expression, different meanings of some expressions in both languages, word pronunciation, word types, usage examples, forum entries, category and editor comments (visible only to the author of the expression). The result of the database query is an array of Specifications that are used by the Request Handler for assembly, together with a fixed part of the Specification. This produces a complete Specification used by Autogenerator to form the results view in different polymorphic generated forms, like table view or a pop-up window.

4.1.1 Database Tables

Only two database tables are used in this case study application: words and users. The table words contains specifications of particular words in Croatian and French, together with forum entries, comments, pronunciation, and possibly other features that are used in generation of different kinds of result views (display record, edit record, forum/comment entry, new record entry, delete entry, etc.). The table words also contains the first meanings of words (separately from other meanings) and the owner of the entry (one of the editors who has entered the word). On the other hand, the table *users* contains the data of registered application users, together with their roles (role 1 = editor and role 2 = user). Users have the right to read forums and to use the bookmarks, while editors can enter new words and expressions, put comments on other editor's entries and use the forum.

The tables can be reached through the searching interface and a Content Management System (CMS), where both sides use Autogenerator as a mechanism that produces and immediately executes the code. Autogenerator uses the feature of scripting languages (Python in this case study) to evaluate the code from variables, not only from source files.

The whole application is performed by Autogenerator script which produces/executes code from SCT model elements (Specification, Configuration and a set of Templates).

4.1.2 Administration System

Two Specifications are used in the example application. The first one is static and it defines the structure of the two database tables (words and users) and manages the data administration part of the application. The second one is dynamic and it describes the user interface with display of selected words and expressions in both languages, i.e. each expression has its own piece of Specification that contains meanings in both languages and, optionally, pronunciation, comments and forum entries.

The Configuration defines the connections between the application Specification and Templates [11]. In the case study application, there are two Configurations: The Configuration of the data management and the Specification of the user interface.

4.1.3 Data Management Specification

Specification of database tables defines output types used in production of virtual files (Python scripts and HTML code), names of database tables together with their attributes, some features for displaying data (like arrangement of fields on web form) and similar.

Output types are connected to the highest-level code Templates, as defined in Specification, and they define types of code to be generated. In the example, the output types are defined as follows:

```
OUTPUT:entry_html// entry web page
OUTPUT:py_script// python script
OUTPUT:html_form// html form
```

The virtual files to be generated are specified by using output types:

```
entry_html:output/index.html  // entry web page
py_script:output/words.cgi    // python script
html_form:output/words_form.html// html form
```

The Specification of all features defining the output files follows the usage of output types ('+' signs define subordination of fields):

```
table:dic_words//database table name
title:words    // identifies the group
+title_display:Words   //name to be displayed
primary_key:word_id    // primary key
field_number:word_id   // numeric field
+field_display:ID      // name to be displayed
field_text:word_h      // textual field
+field_display:Croatian
field_text:word_f      // textual field
+field_display:French
field_memo:description // textual field
(textarea)
+field_display:Description
field_text:owner       // textual field
+field_display:Owner
```

The field description is used to store the Specification of particular word or expression. This Specification is being repeated for each database table, so the fields that are specified for the table users are name, own_name, surname, password, role, temporary_password and enrollment_date. The Specification diagram [11] of data administration is given in Fig. 3.

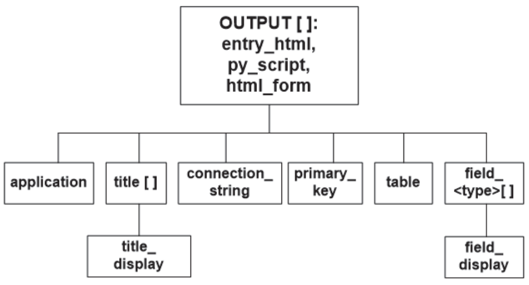

Figure 3 Specification diagram of data administration

Three output types are specified. The type entry_html is used for the entry page (index.html), py_script defines features of Python script that deals with database and html_form is used for data editing form. The square brackets define groups (here: all output types and field types, like field_int and field_float). Some attributes have their subordinated attributes (title and members of field_ group) which is defined by the '+' sign in the Specification.

4.1.4 Data Management Configuration

Configuration of the data management deals with the different data structures that are defined in the Specification. In its first part, types of outputs from Specification of the data structures are attached to their highest-level templates (Tab. 1).

The number between the '#' signs defines the ordinal number of the output type. The rest of the Configuration defines three element groups where: the first element is a connection (physically present in Templates), the second element is an attribute name from Specification and the third element is the attached Template (omitted if there is no need for a Template).

Table 1 Types of outputs with their highest-level templates

Configuration	Specification	Description
#1#,,index.template	OUTPUT:entry_html	Entry page (html)
#2#,,script.template	OUTPUT:py_script	Database management (Python scripts)
#3#,,form.template	OUTPUT:html_form	Data editing form (html)

For example, the line:

```
#table#,table
```

defines that the connection #table# should be replaced by the value of the attribute table from Specification in all their occurrences in the appropriate Template. At the same time,

```
"#links#,title,links.template
```

defines that connection #links# should be replaced by the whole Templatelinks.template for each occurrence of the attribute title, that is used for generating links on the index page. Groups of attributes are specified as follows:

```
#fields_on_form#,field_*,field_form_*.template
```

which determines that the connection #fields_on_form# should be replaced by the appropriate Template for each occurrence of attribute with a name starting with field_ (e.g. field_integer or field_text). The Template name is given by replacing the asterisk by field type (e.g. field_form_text.template).

In case of Autogenerator, the Configuration can also include definition of some Imperative statements [4]. Imperative statements are used to perform certain instructions only once, which are usually connected with some software dependencies like databases. This example uses such statements for changing the database table structure along with the change of code, e.g.:

```
#imperatives#,ADD_field_*,ADD_field_*.template
#ADD_field#,ADD_field_*
```

The link #imperatives# is used in the appropriate code Template for the code that performs ALTER TABLE operation on a database table to add an additional column, which is defined in Specification (e.g. ADD_field_text:newfield). If there are no imperatives in Specification, the link #imperatives# will be replaced with blank. The prefix ADD is deleted from the appropriate Specification line after performing the imperative statement.

4.1.5 User Interface (UI) Specification and Configuration

User interface of the online dictionary example is defined by its main Template (index.template) containing HTML, JavaScript and CSS parts and the variable part that is defined by the other code Templates and the Dynamic Specification that is formed by the Request Handler. Once the Dynamic Specification is produced, the Autogenerator generates the display of search results in some polymorphic form (table view or pop-up window).

User interface is produced by the Autogenerator using the Dynamic Specification, Configuration and Templates. According to the user request, the Dynamic Specification of selected expressions is formed from three parts: the common part which is the same for all expressions (contains output types and output files), the dynamically generated part (information about user status and id number of the next database record to be entered) and the Specifications of all particular words (from the description field of the table words). The common part of the Specification defines output types and output files:

Table 2 Specification of output types and output files

Output type	Output (virtual) files
OUTPUT:out1	out1:output/index.html
OUTPUT:out2	out2:output/select.cgi
OUTPUT:out3	out3:output/create_specif.cgi
OUTPUT:out4	out4:output/table_results.html
OUTPUT:out5	out5:output/login.cgi
OUTPUT:out6	out6:output/comment.cgi
OUTPUT:out7	out7:output/create_comments.cgi
OUTPUT:out8	out8:output/table_comments.html
OUTPUT:out9	out9:output/update.cgi
OUTPUT:out10	out10:output/entry.cgi

These generated virtual files represent different polymorphic forms of the features specified in the following lines. The next two lines are created dynamically and specify the user status and the next database record to be entered:

```
logged_in:no   // not logged in, user or editor
next_id:6101   // id for adding the next database
record
```

The rest of the Specification contains features of all selected words and expressions. For example, the Specification of one particular word could look like this:

```
id:14          // id number of the expression
+word_H:preživjeli     // meaning in croatian
+word_H:preživjela
+word_F:rescapé nm
+word_F:rescapéenf     // meaning in french
+word_F:[ʀeskape]      // french pronunciation
+comment:Dodatiprimjer.// comment to author
+forum:<b>editor1</b>05.02.2020.20:19:00<br>
// forum entry: Marek est le seul rescapé - Marek
je jedini preživjeli
```

The user interface Configuration manages the process of automatic code generation which is performed on demand by the Autogenerator. There are ten top-level Templates that are connected with the output types in Specification, as previously described:

```
#1#,,index.template    // entry html page
#2#,,select.template   // combo-box for words
#3#,,create_specification.template
// assembles the Specification
#4#,,table_results.template
// search results (html table)
#5#,,login.template    // login script
#6#,,comment.template  // shows comments
#7#,,create_comments.template
// Specification for comments
#8#,,table_comments.template
// html table with comments
#9#,,update.template   // update database record
#10#,,entry.template   // insert into database
```

The rest of the Configuration defines Specification attributes and their connection with code Templates:

```
#translation#,id,translation.template
#word_H#,word_H,word_H.template#word_F#,word_F,w
ord_F.template
#word_HH#,word_H       // all croatian exp.
#word_FF#,word_F       // all french exp.
#id#,id // id number
#logged_in#,logged_in  // user status
#comments#,id,comments.template//comments
#all_comments#,comment,all_comments.template
#comment#,comment      // comment text
#owner#,owner// author of the record
#next_id#,next_id      // next id number
#forum#,forum  // forum text
#forum_save#,forum,forum_save.template
#forum_show#,forum,forum_show.template
```

The template table_results is the top-level Template connected to the output type out3, which is used in generation of create_specification.cgi (Python script that selects records in the database according to the user request and forms the Specification used in display of the search results). The template translation defines the display of a particular query result and it is used for each occurrence of the id attribute in the Specification.

4.2 Achieved Features and Dispersion of Connections

The aim of the SCT based generator is to generate an application with all the required features from a relatively small Specification. This is also the case with the Dynamic Specification, which is created based on the user request, containing search results from the database. The discussed Autogenerator case study includes three output types in the data administration part (Python scripts, HTML and JavaScript code) and ten output types in a creation of application user interface (Python scripts and HTML code). The generator inputs and outputs can be compared by the number of files and number of lines.

Table 3 Dispersion of connections through *Templates* in example generator

Number of connections (data administration)	Occurrences in templates	Average connections per config. line	Average connections per template
49	402	8,20	5,91
Number of connections (user interface)			
16	241	15,06	12,68

Table 4 Usage of particular *Specification* attribute values in generated example application

Attribute (data administration)	Total occurences in generated code	Number of virtual files where att. value occurs
application	2	1
title	56	4
field_memo	101	2
field_number	41	2
field_display	7	2
Attribute (UI)		
next_id	21	3
id	324	2
word_H	6	2
word_F	6	2
forum	2	2
comment	2	2

Connections are included in Templates, defining variable parts of the code to be generated. A single connection from Configuration may be contained in more than one Template, more than once in a particular Template, as shown in Tab. 3.

Multiplying connections across Templates enable the dispersion of Specification values across the generated application, so a feature that is defined once can be later used repeatedly. The usage of some particular Specification attribute value in the generated example application is shown in Tab. 4.

5 CONCLUSION

The majority of GP approaches use static Specification as the way of defining features for target application. Such Specification has to be defined in full prior to the process of source code generation.

Such a large Specification is opposite to basic principles of GP, where a relatively small Specification results in a large quantity of generated code. This paper introduces a Dynamic Specification as an approach to customize the application features according to the user request in real time. The automated mechanism for building and handling of Dynamic Specification is presented. This mechanism is applied in the Croatian-French online dictionary case study. In the case study words and language expressions are represented by pieces of Specification and the full Specification is being assembled dynamically on demand. Autogenerator uses Specification to produce user interface containing the search results and interaction elements. This approach has resulted with features and benefits such as usage of Specification defined on a high level and easier application updating.

The multi-dispersion of connections and attribute values could be used for application updating. Updating can be performed through changing of Specification, by adding/deleting of features, which changes the application features within the scope (problem domain) proposed by Configuration. Changes in Templates has the impact on the way how Specification attribute values are used, including the used programming language (or any other piece of code i.e. HTML). The updating of Configuration changes the way how generator builds the code, so introducing a new line in Configuration can enable the usage of a new Specification attribute and a new code Template. The aim is to make any later modifications of the generated code unnecessary which represents the major benefit for application updating.

6 REFERENCES

[1] Czarnecki, K. & Eisenecker, U. W. (2000). Generative programming.

[2] Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, *37*(4), 316-344. https://doi.org/10.1145/1118890.1118892

[3] Basset, P. (1997). Framing Software Reuse-Lessons from Real World.

[4] Magdalenić, I., Radošević, D., & Orehovački, T. (2013). Autogenerator: Generation and execution of programming code on demand. *Expert Systems with Applications*, *40*(8), 2845-2857. https://doi.org/10.1016/j.eswa.2012.12.003

[5] Rosenmüller, M., Siegmund, N., Saake, G., & Apel, S. (2008). Code generation to support static and dynamic composition of software product lines. *In Proceedings of the 7th international conference on Generative programming and component engineering*, 3-12. https://doi.org/10.1145/1449913.1449917

[6] JAVA Annotations, The Java Tutorials, Oracle Java Documentation, https://docs.oracle.com/javase/tutorial/java/annotations/ (last accessed May, 2020)

[7] Klonatos, Y., Koch, C., Rompf, T., & Chafi, H. (2014). Building efficient query engines in a high-level language. *Proceedings of the VLDB Endowment*, *7*(10), 853-864. https://doi.org/10.14778/2732951.2732959

[8] Esterie, P., Falcou, J., Gaunard, M., Lapresté, J. T., & Lacassagne, L. (2014). The numerical template toolbox: A modern c++ design for scientific computing. *Journal of Parallel and Distributed Computing*, *74*(12), 3240-3253. https://doi.org/10.1016/j.jpdc.2014.07.002

[9] Jarzabek, S., Bassett, P., Zhang, H., & Zhang, W. (2003). XVCL: XML-based variant configuration language. *In 25th International Conference on Software Engineering, 2003. Proceedings*, 810-811. https://doi.org/10.1109/ICSE.2003.1201298

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

[11] Radošević, D. & Magdalenić, I. (2011). Source code generator based on dynamic frames. *Journal of Information and Organizational Sciences*, *35*(1), 73-91.

[12] Jarzabek, S. & Kumar, K. (2016). On Interplay between Separation of Concerns and Genericity Principles: beyond code weaving. *Computer Science and Information Systems*, *13*(3), 731-758. https://doi.org/10.2298/CSIS160129028J

[13] Vázquez-Ingelmo, A., García-Peñalvo, F. J., & Therón, R. (2019). Addressing fine-grained variability in user-centered software product lines: a case study on dashboards. *In World Conference on Information Systems and Technologies*, 855-864. https://doi.org/10.1007/978-3-030-16181-1_80

[14] Apache Velocity Project, http://velocity.apache.org/engine/devel/, (last accessed May, 2020)

[15] Stringtemplate, http://www.stringtemplate.org/, (last accessed May, 2020)

[16] Klint, P., Van Der Storm, T., & Vinju, J. (2009, July). EASY Meta-programming with Rascal. *In International Summer School on Generative and Transformational Techniques in Software Engineering*, 222-289. https://doi.org/10.1007/978-3-642-18023-1_6

[17] Heering, J., Hendriks, P. R. H., Klint, P., & Rekers, J. (1989). The syntax definition formalism sdf-reference manual. *ACM Sigplan Notices*, *24*(11), 43-75. https://doi.org/10.1145/71605.71607

[18] Kolovos, D., Medhat, F., Paige, R., Di Ruscio, D., Van Der Storm, T., Scholze, S., & Zolotas, A. (2019). Domain-specific languages for the design, deployment and manipulation of heterogeneous databases. *In 2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering (MiSE)*, 89-92. https://doi.org/10.1109/MiSE.2019.00021

[19] Basten, B., Van den Bos, J., Hills, M., Klint, P., Lankamp, A., Lisser, B., & Vinju, J. (2015). Modular language implementation in Rascal-experience report. *Science of Computer Programming*, *114*, 7-19. https://doi.org/10.1016/j.scico.2015.11.003

[20] Lemaire, C. (2008). CODEWORKER Parsing tool and Code generator-User's guide & Reference manual. Worker. pdf. Available at: (http://codeworker.free.fr/CodeWorker.pdf).

[21] Mlakar, J., Radošević, D., & Magdalenić, I. (2015). Generating Web Applications Using CodeWorker. *In 26th Central European Conference on Information and Intelligent Systems (Ceciis 2015).*

[22] Chaves, J. T. F. & de Freitas, S. A. A. (2019). A Systematic Literature Review for Service-Oriented Architecture and Agile Development. *In International Conference on Computational Science and Its Applications*, 120-135. https://doi.org/10.1007/978-3-030-24308-1_11

[23] Huang, Z. (2003). *The UniFrame system-level generative programming framework*. Indiana Univ-Purdue Univ at Indianapolis Dept of Computer and Information Sciences.

[24] Olson, A. M., Raje, R. R., Bryant, B. R., Burt, C. C., & Auguston, M. (2005). UniFrame: A Unified Framework for developing Service-Oriented, Component-Based Distributed Software Systems. *In Service-oriented software system engineering: Challenges and practices*, 68-87. https://doi.org/10.4018/978-1-59140-426-2.ch004

[25] Derhamy, H., Eliasson, J., & Delsing, J. (2019). System of System Composition Based on Decentralized Service-Oriented Architecture. *IEEE Systems Journal*, *13*(4), 3675-3686. https://doi.org/10.1109/JSYST.2019.2894649

[26] Chen, X., Zheng, Z., Yu, Q., & Lyu, M. R. (2013). Web service recommendation via exploiting location and QoS

information. *IEEE Transactions on Parallel and distributed systems*, *25*(7), 1913-1924. https://doi.org/10.1109/TPDS.2013.308

[27] Fabac, R., Radošević, D., & Magdalenić, I. (2014). Autogenerator-based modelling framework for development of strategic games simulations: rational pigs game extended. *The Scientific World Journal*, 2014. https://doi.org/10.1155/2014/158679

[28] Strmečki, D., Magdalenić, I., & Radosević, D. (2018). A systematic literature review on the application of ontologies in automatic programming. *International Journal of Software Engineering and Knowledge Engineering*, *28*(05), 559-591. https://doi.org/10.1142/S0218194018300014

[29] Diaz, E. & Rueda, S. (2019, June). Generation of user interfaces from business process model notation (BPMN). *In Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 1-5. https://doi.org/10.1145/3319499.3328242

[30] Diaz, E., Panach, J. I., Rueda, S., & Pastor, O. (2018). Towards a method to generate gui prototypes from bpmn. *In 2018 12th International Conference on Research Challenges in Information Science (RCIS)*, 1-12. https://doi.org/10.1109/RCIS.2018.8406675

[31] Zafar, I., Azam, F., Anwar, M. W., Maqbool, B., Butt, W. H., & Nazir, A. (2019). A Novel Framework to Automatically Generate Executable Web Services from BPMN Models. *IEEE Access*, *7*. https://doi.org/10.1109/ACCESS.2019.2927785

[32] Zafar, I., Azam, F., Anwar, M. W., Butt, W. H., Maqbool, B., & Nazir, A. K. (2018, November). Business process models to Web services generation: A systematic literature review. *In 2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 789-794. https://doi.org/10.1109/IEMCON.2018.8615096

[33] BPMN.org, Object Management Group, Business Process Model and Notation,http://www.bpmn.org/,(last accessed December, 2020)

[34] Schaffner, B. (2001). Storing object data as XML, Tech Republic, http://www.techrepublic.com/article/storing-object-data-as-xml/ (last accessed May, 2020)

Contact information:

Danijel RADOŠEVIĆ, PhD, Full Professor
(Corresponding author)
University of Zagreb, Faculty of Organization and Informatics,
Pavlinska 2, 42000 Varazdin, Croatia
E-mail: danijel.radosevic@foi.hr

Ivan MAGDALENIĆ, PhD, Full Professor
University of Zagreb, Faculty of Organization and Informatics,
Pavlinska 2, 42000 Varazdin, Croatia
E-mail: Ivan.magdalenic@foi.hr

Andrija BERNIK, PhD, Assistant Professor
University North,
104. brigade 1, 42000 Varazdin, Croatia
E-mail: andrija.bernik@unin.hr