ATLANTIS PRESS
Advances in Computer Science Research (ACSR), volume 72
IV International Research Conference "Information Technologies in Science, Management, Social Sphere and Medicine" (ITSMSSM 2017)

Semantic modeling of the program code generators for distributed automated systems

Dmitry Zhevnerchuk
Computer systems and technologies department
Nizhny Novgorod State Technical University n. a. R.E. Alexeev
Nizhny Novgorod, Russian Federation
zhevnerchuk@yandex.ru

Alexander Zakharov
Computer systems and technologies department
Nizhny Novgorod State Technical University n. a. R.E. Alexeev
Nizhny Novgorod, Russian Federation
lukalex.nnov@gmail.com

Lubov Lomakina
Computer systems and technologies department
Nizhny Novgorod State Technical University n. a. R.E. Alexeev
Nizhny Novgorod, Russian Federation
llomakina@list.ru

Anna Surkova
Computer systems and technologies department
Nizhny Novgorod State Technical University n. a. R.E. Alexeev
Nizhny Novgorod, Russian Federation
ansurkova@yandex.ru

***Abstract*— The article presents a semantic model and a technique for application in the development of software and based on synthesis of open information systems a program code generator needs to comply with the following requirements: extensibility, scalability, cross-platform and interoperability. A fragment of concept scheme of distributed automated systems program code generator is presented. In the course of the work we used the methodology of the open information systems synthesis, which allowed us to create scalable, interoperable and cross-platform program code generators.**

***Keywords*— *semantic modeling, interface, code generation, configuration*.**

I. INTRODUCTION

One of the properties of computability theory (the theory of recursive functions) is Turing completeness. This property means that there is an element that can calculate the function [1]. In other words, giving this definition in the area of generating program code, we get the following fact: if a program can be written, then there is an element that can write it. Such an element will be called the code generator (CG).

Currently, it is becoming more and more urgent to create a software product that automates the work of the application software developer. The "developer" in the sentence means a team member who participates in the creation of a software product at any stage. When developers design software they try to make the structure of the application as simple and standardized as possible which lead to easy programming and easy maintenance. To start a new project, the developer often has to perform the same type of actions to create a basic template.

With the correct input data and the correct development of the generator, most of the code can be generated automatically. The developer will still have to make minor changes to the final code of the program.

Code generators are the best used in those areas where small in size and essentially identical operations are performed [2]. Potentially, generators can be used in any areas (tasks) where you can automate the process of working with data. The closest to this concept are the tasks of generating a project code that interacts with a database with similar classes and structures for objects or applications for testing code or writing documentation for an existing program.

II. ANALYTICAL PART

In order to reduce the time for the development of the initial part of the project, we use automated systems (AS) for generating code.

Generation of code using the method described in this article is carried out in several stages: template, configuration, rendering. To understand these steps, we introduce the following definitions:

Definition 1**. The Template** is a structured text that defines a block of code. Each block contains fields and unchanged constructs that are typical for a programming language. Fields are replaced by data types and the names of the variables, scopes, etc.

Definition 2. **The Configuration** is a process of forming pairs. The first element of a pair is a template, and the second element is a data vector.

Definition 3. **The Rendering** is a process of replacing template fields with configuration fields and then generating a block of code.

The code generation scheme includes two steps. The first step is to create a template and configuration. At the input of the template engine we pass the language constructs and mutable fields to get a template as an output. For configuration process we submit the domain models as an input to get the final configuration as the output. Domain models formalize structural and functional aspects, and models are a source of names, types, and constraints used in the system.

In the development of an automated system in addition to CG developers use generators that create a basic prototype of a software system, including a required folder structure, configuration files and modules with the described interface part, various add-ons and their settings (i.e. modules with ready functionality, resources, plug-ins). There are various tools for this problem to solve based on different technologies, programming languages and platforms we use [3].

To create automated systems, oriented to a certain range of tasks, a framework approach is being used. This approach involves building an initial prototype based on a software platform (framework), which consists of a permanent part called the core, and plug-ins or extension points.

The most famous tool for a fast start-up project creation is the Yeoman scaffolder. Scaffolding is a method of meta programming based on the project requirements analysis [4]. The main function of metaprogramming is a creation of programs using other programs. Yeoman is based on three main components: the package dependency manager for downloading, updating and removing additional packages (bower); a tool for assembling java-script projects using pre-written tasks (grunt); a basic application to generate a database for a new application (yo). The whole project is configured with one command to start one of the more than 5,500 code generators described on the site [4].

For developers of dotNet platform, Microsoft has developed a Text Template Transformation Toolkit or T4 [5] template-based code generator. To use this generator, you write a template file in * .tt format. There are many ready-made templates for code generation, however, due to the lack of a single repository, the search for these templates is extremely difficult. In addition, T4 exclusively support only generating of C# code.

In Java technology stack, multiple generators are used to create a repeating code. Generators are also used to solve such typical tasks as creating code for class designers with a large number of fields, creating and editing JavaDocs, getters / setters, managing code collections and compositors, code management for testing, etc. Some of them are embedded in the development environment and others are independent systems supplied as jar files: FreeBuilder, Proto. Integration of external generators often requires writing their own specific routine code, counted in dozens of lines.

For a Java application (Spring), you need to include a large number of different classes – beans. You also need to find additional libraries and also include them in an existing project.

To generate a startup project, you can use Grails [6] platform, which generates a project using one command: "grails create-app %app-name%".

In Web development, there are add-ins called "syntactic sugar" that help reduce the number of lines of source code by generating it. For example SugarJS and CoffeeScript [7]. "Syntactic sugar" translates code written in new programming languages into code written in Java-script. Independent templating tools such as Underscore templates or HandleBars allow storing parametrically customizable templates of html pages and their fragments on the server and, if necessary, even generate html code and forward it to the client.

To summarize this approaches and code generation programs are not effective in a distributed automated systems development (AS), which solves the tasks of corporate and strategic level, because:

-It is difficult to train a program to create code for two or more interoperable modules which written with a variety of programming languages and presentation formats,

-In most cases, the systems are oriented to implement a low-level generation based on known patterns which requires to study unique architectures, object models, notations, etc.

-Generally, code generation systems are not suitable for the code creation with algorithmic and language constraints or for subsequent merging with existing code,

Generally, it is not possible to make changes to the code generation system without recompiling it.

III. FORMULATION OF THE PROBLEM

The task of this project is to develop a component representation of a hybrid (i.e. multi-module, multilingual) code generator with its semantic model. Each component of the generator requires interfaces to connect: a) with other generator components, b) with the configuration units to render the code templates. In addition, each generator can be used both separately and in a chain of generators.

IV. CONCEPTUAL MODELING OF THE CODE GENERATOR

Based on synthesis of open information systems [8,9] a generator needs to comply with the following requirements: extensibility, scalability, cross-platform and interoperability [10] and representing the collection of components with dedicated interfaces. There are several definitions to be made:

Definition 4 **The generator component** is an abstract structural element of the code generator. It has an input / output and basic properties. Basic properties essentially are mutable fields that define input configuration interfaces and present program code as a result of rendering.

Let's introduce two types of generator components.

Definition 5 **The program code render unit** is a structural element of the code generator. It accepts the configuration and returns the program code based on the template.

Definition 6 **The program code configuration unit** receives configuration from the client systems and transmits it to the renderer.

Fig. 1 represents a concept diagram of the component generator organization.

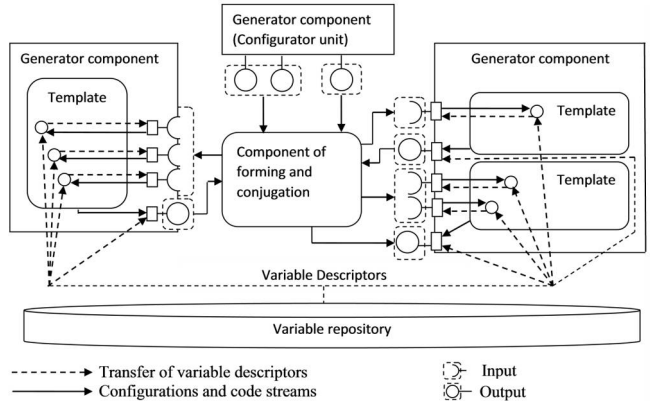

Fig. 1. Conceptual diagram of the generator

Elements of render and configuration units are basic templates. The render templates include immutable code patterns and unique field descriptors to identify them. The configurator templates include only a list of descriptors for the fields to be changed. It is required to prepare the configuration for each of the fields by the client system.

In addition to descriptors of mutable fields, the template can have other basic properties which allows you to build interfaces. Using these interfaces, the generator unit can connect with logical and datum level models, classifiers or blocks.

The method of synthesis of open information systems gives us ability to present the code generator and its external environment as standard components. These components have unified interfaces, synthesized based on basic properties.

Dashed arrows represent streams of variable descriptors. The stream is generated when templates and interfaces are being created.

Solid lines represents the streams of configurations and template rendering results. These streams occur program code is being generated.

The generator component (c_r) contains a template. The program code is generated based on this template. The template is defined by the properties $T_{c_r} = (t_{c_{r_1}}, t_{c_{r_2}}, ..., t_{c_{r_n}})$. The components of the generator $C = (c_1, c_2, ..., c_j, ..., c_s)$ are potential providers of configurations which being determined by the properties of their templates $T_{c_j} = (t_{c_{j_1}}, t_{c_{j_2}}, ..., t_{c_{j_n}})$. Let's consider the conjugation singularities c_r and c_j, where $j = 1..s$.

There are three main cases:

1) $|T_{c_r}| = 0$, the component template (c_r) does not contain a mutable descriptor of the field. The template has an output interface associated with the variable descriptor. This interface lets us transfer immutable line of code to the external environment with the final program code. The template code (c_r) is a ready-made program code.

2) $|T_{c_r}| = 1$, the component template (c_r) contains one mutable field. One component of the generator is sufficient for rendering $(c_0 \in C)$. The following conditions must be met:

   2.1) $\exists q \in T_{c_r}\, p \in T_{c_0} : Id_q = Id_p$ (c_r and c_0 components have basic properties defining variables with the same descriptors)

   2.2) q is the descriptor of the variable c_r. Defined on the set of entry points of block c_r, and p is the descriptor of the rendering result which means that it is defined on the set of exit points c_0

3) $|T_{c_r}| > 1$, the component template c_r contains more than one variable descriptor. To render the template several components of the generator $C_0 \subseteq C_j$ is needed and the following conditions must be met:

   3.1) $\forall q \in T_{c_r}\, p \in T_{c_0} : Id_q = Id_p$ (for any mutable field of the template (c_r) there is an existing basic component property from the set C_0 that defines a variable field with the same descriptor)

   3.2) all properties of q are mutable fields descriptors. These are defined on the set of entry block points c_r, p are descriptors of the rendering result.

Based on the concept scheme a set of concepts and roles for generator of distributed automated systems construction is proposed.

V. SEMANTIC MODELING OF THE CODE GENERATOR

A fragment of concept scheme of distributed automated systems generator is presented below in Fig. 2 which defines the namespace of concepts and roles. The model contains basic concepts that formalizes the render unit and the program code configuration unit, as subclasses of the generator component. Input/output streams associated with the generator.

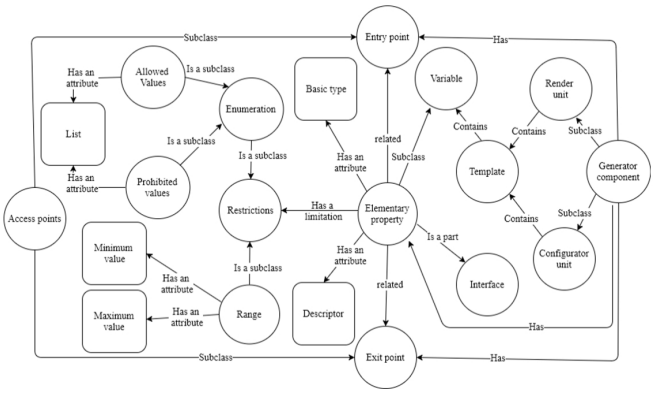

Fig. 2. Semantic model of the generator (fragment)

A certain set of basic properties can be associated with each point. These properties define mutable fields and have a unique set of a descriptor, a base type and constraints. The descriptor and base type are not related to other concepts, so they can be implemented with simple attributes. The constraint is implemented by a class that has subclasses that specify valid lists, forbidden lists, and values range.

In addition, the render and the configuration units contain templates, implemented by simple string based on attributes containing program code for software build and substitution parameters that have field descriptor values. Mutable fields can be combined using the concept that defines the interface.

Fig. 3 provides a description of the methodology for synthesizing the program code.

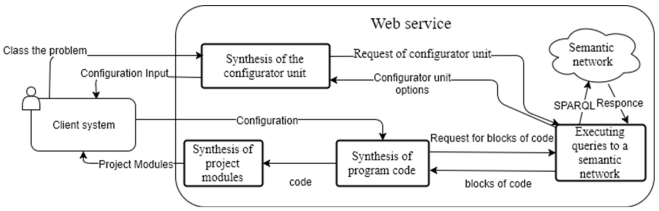

Fig. 3. Scheme for the description of the synthesis procedure

The first step is to create a query to define a class of the task to be solved as well as requirements, programming language, architecture features, application features (in terms of security or real-time application). The second step is to create queries to the semantic network based on the data we collected on the first step. The queries return configuration parameters. Based on the queries, the client application is generated for configuration.

The third step is to build a configuration from client system for generating a program code when a sequence of queries to the semantic network is formed. These queries define the generator parameters used to configure the code synthesis objects. At the last step, the final code is placed in the target module files of the automated system. These files are structured in a certain way and arranged in folders, and arrangement rules depend on the programming technologies used. After such arrangement modules are ready for transferring to the client.

VI. THE EXPERIMENTAL PART

Fig. 4 shows a fragment of the POJO generator semantic network

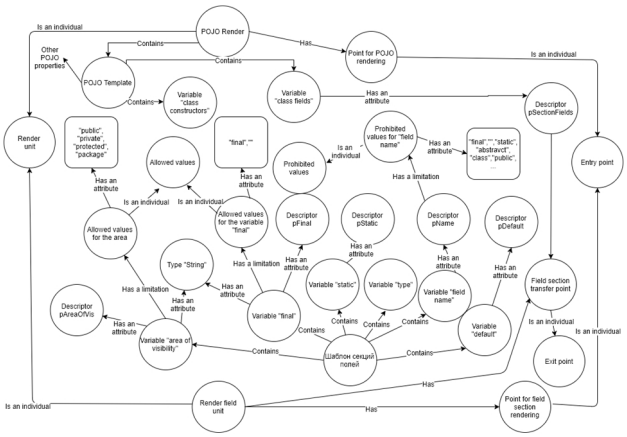

Fig. 4. Fragment of semantic network (POJO)

To confirm the theoretical and methodological aspects, a series of experiments with a prototype generator was conducted.

These models of specific code generators including POJO, JPA Entity, HTML form were constructed. Based on the constructed semantic models, the program code was generated. The code was successfully compiled and passed a series of tests.

However a human factor error rate in the process of constructing semantic models was 29%. Primarily it happens due to a large amount of input data and the lack of collision control algorithms. Collisions occur during the input and editing of data.

VII. SUMMARY AND CONCLUSION

In the course of the work, the methodology of the synthesis of open information systems was adapted to a program code generators which allowed us to create scalable, interoperable and cross-platform generators. We have constructed a semantic models of the generator, which can be used as an knowledge base of the ontological type. This semantic models successfully passed the tests in experiment we carried out to test its abilities. The ontology type will let us to describe the scripts of code

generation, independent of the language tools and template technologies.

REFERENCES

[1] G. B. Evgeniev, "Intelligent Design Systems," Bauman Moscow State Technical University Press, Moscow, 2009.

[2] M. N Kustov, "Increase the efficiency of algorithms for generating code based on SADT methodology. Information Intelligence Systems," in Proceedings of the XVI International Youth Forum "Radioelectronics and Youth in the 21st Century", *Kharkiv, KNURE*, pp. 228-229, 2012.

[3] Y. I. Klykov, "Situational management of large systems," *Moscow: Energy*, 1974

[4] S. Begaudeau, "Modern Web Application Development Workflow," in Proceedings of the Eclipse CON (France), 2014, Access mode https://www.eclipsecon.org/europe2014/sites/default/files/slides/Web%20Applications%20Development%20Workflow%20-%20EclipseCon%20Europe.pdf

[5] D. Ross, "Structured Analysis (SA): A Language for Communicating Ideas, IEEE Transactions on Software Engineering," in Proceedings of the IEEE Transactions on Software Engineering, vol. SE-3. no. 1, pp. 16-34, 1977

[6] A. Basu, "Metagraphs and Their Applications," *Springer Computer Society Press*, 174 P, 2007.

[7] S. Kelly, "Domain-Specific Modeling: enabling full code generation, Hoboken, New Jersey, USA: Wiley-IEEE Computer Society Press," 444 P, 2008.

[8] D.V. Zhevnerchuk and V.V. Kondrat'ev, "Application of methods of self-organization theory to problems of profiling and configuring computational systems," *Doklady Mathematics,* vol. 90, no. 3, pp. 788-790, 2014.

[9] L.S. Lomakina and D.V. Zhevnerchuk, "Synthesis of open information systems using algebraic structures as models," Fundamentalnie Issledovaniya, no. 10, pp. 29-33, 2017.

[10] M. Fowler, "Patterns of Enterprise Application Architecture," *Addison-Wesley*, 2014.