УДК 004.435
DOI: 10.25559/SITITO.16.202003.653-663

Оригинальная статья

Язык описания проблемы и исследование его возможностей

Е. А. Доренская[1*], А. А. Куликовская[1,2], Ю. А. Семенов[1,3]

[1] ФГБУ «Институт теоретической и экспериментальной физики имени А.И.Алиханова Национального исследовательского центра «Курчатовский институт», г. Москва, Российская Федерация
117218, Российская Федерация, г. Москва, ул. Большая Черёмушкинская, д. 25
* dorenskaya@itep.ru

[2] ФГБУ «Национальный исследовательский центр «Курчатовский институт», г. Москва, Российская Федерация
123182, Российская Федерация, г. Москва, пл. Академика Курчатова, д. 1

[3] ФГАОУ ВО «Московский физико-технический институт (национальный исследовательский университет)», г. Долгопрудный, Российская Федерация
141701, Российская Федерация, Московская область, г. Долгопрудный, Институтский пер., д. 9

Аннотация

Разработан язык описания проблем Problem Description Language (PDL). Рассмотрены базовые архитектурные особенности языка PDL. Представлены преимущества использования языка описания проблемы. Поясняются особенности составления описаний на языке PDL и его синтаксиса. Рассмотрен универсальный конфигуратор ПО для пакета программ защиты Web-сервера, описаны основные его особенности. Приводится пример апробации PDL с помощью транслятора, конвертирующего описание проблемы в код языка Perl. Приводится структура транслятора описания проблемы на PDL в Perl (фаза анализа и фаза синтеза). Описана каждая из этих фаз. Работа на фазе синтеза в данном трансляторе существенно отличается от аналогичной фазы других трансляторов. Она может быть 3-х видов: 1. Если в описании проблемы не указан ни один из операторов действия (особый класс операторов в PDL). 2. Если в описании присутствует хотя бы один оператор действия и нужно проводить поиск в базе данных алгоритмов. 3. Если при тех же условиях что в пункте 2 в банке алгоритмов не найдено нужного модуля. В первом случае используются примитивы. Во втором - проводится поиск модулей в банке алгоритмов. В 3-м случае необходимый модуль придётся частично писать самому программисту. Рассматривается процедура генерации пакета программ защиты Web-сервера на Perl с помощью описания задачи на PDL. Приводятся примеры и рассказывается о причинах уменьшения количества программных ошибок при использовании транслятора PDL в Perl.

Ключевые слова: язык описания проблемы, problem description language, PDL, неалгоритмический язык, описание проблемы, транслятор, анализ, синтез, Perl, примитив.

Авторы заявляют об отсутствии конфликта интересов.

Для цитирования: Доренская, Е. А. Язык описания проблемы и исследование его возможностей / Е. А. Доренская, А. А. Куликовская, Ю. А. Семенов. – DOI 10.25559/SITITO.16.202003.653-663 // Современные информационные технологии и ИТ-образование. – 2020. – Т. 16, № 3. – С. 653-663.

Original article

Exploring Possibilities of Language for Describing the Problem

E. A. Dorenskaya[a*], A. A. Kulikovskaya[a,b], Y. A. Semenov[a,c]

[a] Institute for Theoretical and Experimental Physics named by A.I. Alikhanov of National Research Centre "Kurchatov Institute", Moscow, Russian Federation
25 Bolshaya Cheremushkinskaya St., Moscow 117218, Russian Federation
* dorenskaya@itep.ru

[b] National Research Center "Kurchatov Institute", Moscow, Russian Federation
1 Akademika Kurchatova pl., Moscow 123182, Russian Federation

[c] Moscow Institute of Physics and Technology (National Research University), Dolgoprudny, Russian Federation
9 Institutskiy per., Moscow Region, Dolgoprudny 141701, Russian Federation

Abstract

The problem description language (PDL) has been developed. The main architectural features of the PDL language are considered. The advantages of using PDL are discussed. The rules for creating descriptions in PDL are described. A universal software configurator for the Web server security software package is considered. Here is an example of testing PDL using a translator that converts the task description to the executable Perl code. The structure of the PDL task description translator in Perl (analysis phase and synthesis phase) is given. The features of this translator at the synthesis stage are described. There are 3 modes of operation at the synthesis phase: 1. If the task description does not specify any of the action operators (a special class of operators in the PDL). 2. If the description contains at least one action operator and you need to search the algorithm in the database. 3. If the required module is not found in the algorithm bank under the same conditions as in point 2. In the first case, primitives are used. The second one searches for modules in the algorithm bank. In the 3rd case, the required module must be partially written by the programmer himself. The procedure for generating a web server security software package in Perl using the PDL task description is considered. Examples are given and the reasons for reducing the number of program errors when using the PDL translator in Perl are described.

Keywords: problem description language, PDL, non-algorithmic language, problem description, translator, analysis, synthesis, Perl, primitive.

The authors declare no conflict of interest.

For citation: Dorenskaya E.A., Kulikovskaya A.A., Semenov Y.A. Exploring Possibilities of Language for Describing the Problem. *Sovremennye informacionnye tehnologii i IT-obrazovanie* = Modern Information Technologies and IT-Education. 2020; 16(3):653-663. DOI: https://doi.org/10.25559/SITITO.16.202003.653-663

Введение

Алгоритм — это последовательность операций, исполнение которых, реализует решение проблемы. Как правило, описание проблемы многозначно. Например, даже задание “вычислить sin(X)” может быть реализовано разными способами: через библиотечную функцию, с помощью разложения в ряд или посредством таблицы значений и интерполяции, возможен вариант, исключающий ошибки при больших значениях X. Какой способ будет выбран? Это можно фиксировать, задав уточняющие вопросы пользователю или реализовав вариант “по-умолчанию” (например, библиотечная функция).

Стоит заметить, что вряд ли удастся создать универсальный язык описания любых проблем. Могут быть созданы и существовать языки описания для конкретных предметных областей (по аналогии с алгоритмическими языками).

Проблема наличия в коде программ большого количества ошибок [9-12, 21] на данный момент является очень серьёзной и актуальной из-за стремительного прогресса в области информационных технологий, а также всё большего их внедрения в нашу жизнь. Для уменьшения количества ошибок был разработан неалгоритмический язык описания проблем (Problem Description Language или PDL). Основным его преимуществом по сравнению с алгоритмическими языками являются минимизация участия программиста в составлении программы [3, 16, 20, 22, 23, 25].

Цель исследования

Целью нашего исследования является разработка синтаксиса языка описания проблемы PDL и транслятора PDL-Perl. Язык PDL служит для минимизации участия программиста в написании программы. Для апробации Problem Description Language был создан транслятор описания проблемы на PDL в код языка Perl. С помощью него была проведена генерация пакета программ защиты Web-сервера на основе набора PDL-описаний.

Основная часть

При описании алгоритма каждой строке кода соответствует определенная команда или последовательность команд процессора. При описании проблемы генерируемая программа может зависеть не только от текущего фрагмента текста, но также от предыдущего и, возможно, последующего текста. По этой причине синтаксис языка PDL должен минимизировать такие возможности и сократить многозначность интерпретации его фрагментов.

Тем не менее описание задания, даже выполненное строго по правилам, не гарантирует того, что существует решение данной проблемы. Возможно, что одному описанию будет соответствовать несколько программ реализации. Синтаксис языка PDL должен минимизировать такую вероятность.

Внутри описания задания может содержаться описание субпроблемы. Начало описания задачи отмечается знаком <#, а завершение — #>. Начальный и завершающий разделители отделяются от описания проблемы пробелом. Например, <# name=»filter.pl»; .... #>. Имя описанной программы задается оператором name, здесь «filter.pl» — имя будущей программы.

Язык PDL состоит из операторов, атрибутов, объектов, параметров, описаний, переменных и массивов (табл. 1). Пример PDL-описания при наличии субпроблем:

```
<# name=»graf1»; convert.file.name=»/home/strigiforme/programs/pril1.txt».to.picture.name=
«/var/www/html/secur/html/images/pril1.png»; #>                    ## subproblem 1
<# name=”graf2”; convert.file.name= “/home/strigiforme/programs/pril2.txt”.to.picture.name=
“/var/www/html/secur/html/images/pril2.png”; #>                    ## subproblem 2
<# name=”appro2” approximate{ requests by time, requests with “////” or more
create.file.name=/var/www/html/secur/txt/pril1.txt;
create.file.name=/var/www/html/secur/txt/pril2.txt; graf1 graf2 #>
```

Операторы в языке PDL, как и в обычном языке программирования обозначают задание, которое необходимо выполнить [28]. Операторы могут быть простыми и композитными. Простые подразделяются на операторы описания и операторы действия. Операторы действия являются глаголами, указывающими на конкретные задачи. Операторы описания — это существительные, обозначающие объект, который надо добавить в программу. Например, оператор librs задаёт список библиотек, которые должны быть добавлены.

Композитные операторы создаются по следующей схеме: Оператор.объект.атрибут=значение;. Пример: create.picture. name=var/www/mrtg/hostname_bytes-day.png; Возможны разные вариации, например такая: оператора и атрибута нет, но есть подряд 2 объекта (например chart.name=gnuplot1.png). Обязательно должна соблюдаться последовательность оператор=>объект=>атрибут. При этом объектов может быть более одного, а оператор или атрибут отсутствовать.

Объекты (второй ряд рис. 1) позволяют уточнить задание операции. Объект определяет то, над чем будет произведена та или иная операция, заданная оператором action.

Атрибуты предназначены для дальнейшего уточнения смысла. С помощью атрибутов описывают как нужно что-то сделать, например, с какой частотой проводить измерения, с какой точностью. Атрибуты в PDL бывают 2-х видов: атрибуты объектов и атрибуты операторов (см. рис. 1). Отличаются они между собой тем, что первые уточняют смысл объекта, а вторые — оператора.

Параметры — это какие-либо значения переменных, функ-

ций и т.д., задающиеся для конкретного описания. Например, dt=300; (временной интервал в секундах).

Описания — поясняют то, что программа должна сделать. Например, System_control (управление системой).

Переменные в PDL бывают разных видов: входные, выходные, внутренние, системные. Входная (IN) и выходная переменная (OUT) могут встречаться в тексте только один раз. Остальные виды переменных могут встречаться сколько угодно раз.

Входная переменная — это строка описания входных данных. Она может содержать имена файлов, переменных, строк, массивов, имена баз данных и таблиц, терминальный ввод, а также URL в том числе и удаленных. Здесь могут присутствовать описания форматов и атрибуты (например, имя-пароль-сертификат). Источником входных данных могут быть отклики на запросы к поисковым системам, а также почтовые сообщения, SMS, голосовые и графические данные.

Выходная переменная — это строка описания выходных данных. Она может содержать имена файлов, включая графические и медийные, переменных, строк, массивов, URL, mail-адреса, телефонные номера для SMS, и атрибуты типа имя-пароль. Здесь могут фигурировать также описания форматов и типов терминального оборудования, в том числе управляющих сигналов. Имеются в виду описания, предназначенные и анализируемые программой, а не человеком.

Внутренняя переменная используется в тексте описания PDL. Задаются такие переменные, как и обычные переменные в языке Perl. Например, $x=5;

Системная переменная может быть целочисленной или символьной (строковой). Она может изменять функцию оператора, например, action, модифицировать значения переменных, изменять работу функций или подпрограмм. Изменение происходит простым присвоением, например, context=filter. Для каждого значения может быть несколько переменных, функций или подпрограмм с одинаковыми именами, но с разной функциональностью или значением. Например, когда значение переменной context=программа, допускается уточнение значения контекста [2,4] в виде программа[код|вычисление] или программа[обучения] или программа[обучения|университет] или программа[внедрения|сетевой график].

Таблица 1. **Примеры элементов языка PDL**

Table 1. **Examples of PDL elements**

Операторы	Атрибуты	Объекты	Параметры	Описания	Переменные	Массивы
Develop; Create; Write; Read; Analyze;	Fast; double-precision; By polinomial,	Program; File; files DB; Table;	Label; parameters Accuracy; dt;	Что программа должна делать: Compute System_control;	Задаются IN= OUT= $=	Задаются @имя

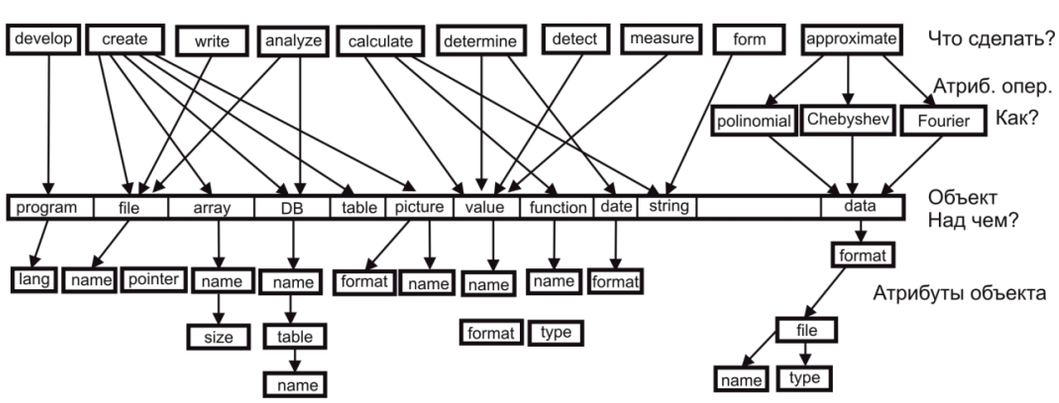


Рис. 1. Семантическое дерево операций

Fig. 1. Semantic operations tree

Массивы в PDL представляются аналогами массивов в Perl и задаются аналогично. Например @core_f=(«1»,»2»,»3»);
Внутри описаний PDL могут использоваться циклы, которые аналогичны тем, что применяются в языке Perl. Комментарий в PDL начинается с помощью последовательности ## и продолжается до конца строки.

На рис. 1 показано, как создается словарь дополнений (имен) объектов, над которыми выполняются действия (операции). Например:
program, routine, file, array, list, web-cite, value, constant, date, string, symbol, DB, table, pointer, limit, approximate, convert(from/to),

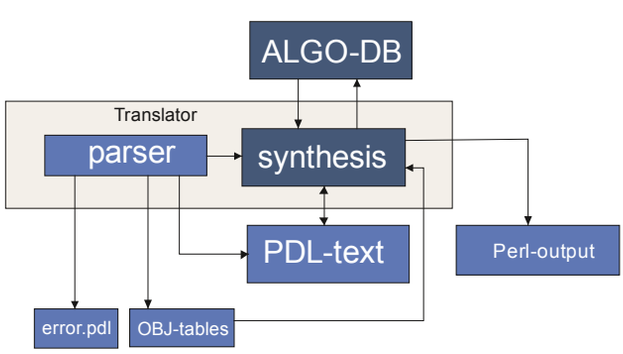
Каждая операция из верхнего ряда рис. 1 может подразумевать бесконечное число вариантов описаний.

Мощность множества операций (actions) $M_A=\infty$. Если мы выбрали какое-то конкретное значение action, мы выделили некоторый достаточно большой (возможно бесконечный) объем в этом многомерном пространстве. Объекты (второй ряд рис. 1) позволяют уточнить задание операции и сократить этот объем. Объект определяет то, над чем будет произведена та или иная операция, заданная action. Атрибуты объектов производят дальнейшее уточнение. Конечной целью создания описания на PDL является сведение мощности оконечного множества к 1 ($M_A=1$).

Т а б л и ц а 2. **Примеры семантических связей**

T a b l e 2. **Examples of semantic links**

Название операции (action)	**Объект**	**Параметры объекта (obj_par)**	**Задание**	**Назначение оператора**
develop	**Program**	**name**	Что программа должна делать: Calculate; System control; Measure, ...	Разработать программу
create	**file**	**name**		Создать файл
	DB	**name**		Создать базу данных
	table	**name**		Создать таблицу DB
	array	**name**		Создать массив
	picture	**format**		Создать рисунок
write	**file** **array**			Произвести запись в файл, базу данных
read	**file** **record**			Прочитать строку файла или запись BD
analyze	**file** **array** **value**			Анализ текста, массива данных или результатов измерения с целью, например, оптимизации



Р и с. 2. Блок схема транслятора для PDL

PDL-text — текст описания проблемы на PDL;

ALGO-DB — банк описаний алгоритмов;

Perl-output — программа, формируемая транслятором (результат);

error.pdl — журнал ошибок, выявленных парсером;

OBJ-tables — таблицы результатов анализа текста описания.

F i g. 2. Translator PDL block diagram

PDL-text — text describing the problem on the PDL;

ALGO-DB — algorithm description bank;

Perl-output — program generated by the translator (the result);

error.pdl — the log of errors detected by the parser;

OBJ-tables — tables of the analysis results of the description text.

Для апробации языка PDL нами был создан транслятор (см. рис. 2), который читает описание проблемы (расширение имени файла .pdl) строка за строкой и формирует таблицы для: descriptions, variables, arrays, operators, primitives и т.д. и формирует на выходе файл программы на perl — name.pl (name представляет в данном случае имя формируемой программы). Синтаксис записей в строках проверяется по мере их чтения. При выявлении ошибок производится запись в журнальный файл ошибок (errors.pdl). Заносится номер строки, сама строка с ошибкой и указывается слово, которое содержит ошибку. Каждому типу ошибки будет присвоен определенный код, который будет также занесен в рекорд файла errors.pdl.

Каждое из слов проверяется на наличие его в таблицах, в частности, в таблице operators. Каждое слово в таблицах характеризуется его принадлежностью к определенному классу: actions, objects, attributes, parameters, operators, composite operators и т. д. В транслятор встроены разные таблицы для разных классов.

Работа транслятора имеет 2 фазы: фазу анализа (парсер) и фазу синтеза.

На фазе анализа (parser) описания проблемы сначала формируется каталог с именем <имя> (имя каталога совпадает с именем файла описания проблемы, заданным name). Исходный текст файла <имя>.pdl читается построчно и формируют-

ся таблицы: operators (операторов), descriptions (описаний), variables (переменных), arrays (массивов), objects (объектов), attributes (атрибутов) и parameters (параметров) и т. д. (см. рис. 3). Все перечисленные файлы укладываются в каталог <имя>. Если в описании проблемы содержится несколько описаний субпроблем, то для них создаются субкаталоги с именами, совпадающими с именем субпроблемы. В этих субкаталогах формируются соответствующие таблицы-файлы. Транслятор может распознавать вставки на языке Perl, добавленные в описание проблемы. Они должны быть выделены специальными тегами (<perl> текст программы на Perl </perl>). При выявлении ошибок формируется файл errors.pdl. Этот файл размещается в корневом каталоге описания проблемы и является общим для всех субпроблем. Все слова описания проблемы должны быть распознаны. Если какое-то слово не распознано, производится запись в файл ошибок errors.pdl. Если в результате работы транслятора на фазе анализа обнаружена хотя бы одна ошибка в описании, фаза синтеза транслятора не запускается. Большинство файлов в таблице 3 формируются при анализе описания проблемы. Исключение составляет только файл с текстом описания проблемы.

Таблица 3. **Файлы, создаваемые и читаемые парсером**
Table 3. **Files created and read by the parser**

Имя файла	Назначение
<имя>.pdl	Текст описания проблемы *(формируется пользователем)*
operators.pdl	Таблица описания системных операторов
errors.pdl	Таблица ошибок, найденных при чтении и анализе файла описания проблемы
descriptions.pdl	Таблица описанных проблем и субпроблем
variables.pdl	Таблица описанных переменных
arrays.pdl	Таблица описанных массивов
objects.pdl	Таблица описанных объектов
attributes.pdl	Таблица описанных атрибутов
parameters.pdl	Таблица описанных параметров
composite.pdl	Таблица описанных композитных операторов
cycles.pdl	Таблица описанных циклов
internal.pdl	Таблица внутренних и системных переменных
primitives.pdl	Таблица использованных примитивов
perl.pdl	Таблица вставок на перле, добавленных в описание

На фазе синтеза последовательно рассматриваются описания проблем и с использованием имеющихся таблиц для каждого описания формируется текст программы на языке Perl.

Фаза синтеза может работать 3-мя способами:

1. Если в описании проблемы не упоминается ни один из операторов действия, но имеются имена примитивов.
2. Если в описании присутствует хотя бы один оператор действия и нужно проводить поиск в банке алгоритмов (хранилище описаний алгоритмов), [5,7].
3. Если при тех же условиях что в пункте 2 в банке алгоритмов не найдено нужного модуля. В этом случае необходимый модуль придётся частично или полностью писать самому программисту.

В первом случае используются примитивы. Примитив — программный модуль, имеющий одну и только одну функцию, например, вычисление факториала или анализ рекордов журнального файла по какому-то параметру. Примитивы образуют библиотеку, каждая из них снабжается описанием (например, libr1.pm). Может существовать несколько тематических библиотек примитивов.

Программа, сгенерированная с помощью синтеза 1-го типа, может представлять собой список имен примитивов, между которыми могут вставляться короткие вставки кода на Perl. Желательно, чтобы примитивы и любые модули в банке алгоритмов были выполнены с соблюдением правил Хольцмана [1]. Это упростит выявление и исправление программных ошибок.

Далее могут следовать описания входных параметров, а также характеристики и форматы выходных данных. Примитив обязательно должен содержаться в банке алгоритмов. Результирующая программа будет формироваться из суперпозиции примитивов.

Среди примитивов могут быть модули подпрограмм, например, для вычисления полинома любого порядка, или факториала, обращения матрицы, вычисление определителя матрицы произвольного порядка, упорядочивания списка по возрастанию или убыванию, вычисления среднего значения или дисперсии (для заданного списка результатов измерения) и т. д..

Имена примитивов надо отличать от программных модулей, записанных в банках. Поэтому у нас они начинаются с p_(например **p_ip_to_num)**. Мощность множества примитивов будет существенно меньше мощности множества программных модулей, хранящихся в банке алгоритмов.

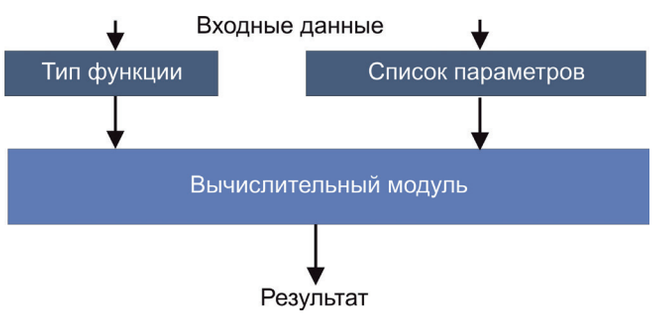
Во втором случае, как уже было сказано, необходимо провести поиск в банке алгоритмов.

Ячейка таблицы базы данных алгоритмов должна содержать**:**

1. Имя программы
2. Цель (назначение)

3. Что реализует (функция)
4. Перечень входных параметров и их характеристики
5. Перечень выходных объектов (вычисленные значения, файлы, списки, массивы и пр.) и их характеристики.
6. Перечень модификаторов и их функции
7. Список используемых библиотек

В банке алгоритмов находятся модули, меняя входные параметры которых, можно варьировать их функциональность. Это расширяет их область применения и уменьшает размер генерируемого кода (см. рис. 3). В настоящий момент в банке содержится 54 программных модуля.

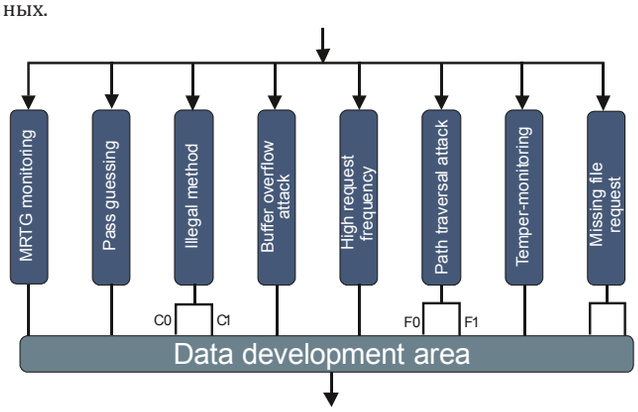


Р и с. 3. Структура модуля с входными параметрами
F i g. 3. Module structure with input parameters

В блоке “Список параметров” может производиться подготовка (преобразование) данных. Помимо этого, входные параметры можно задавать на языке описания проблемы. Подобный способ использования программных модулей был нами опробован на примере некоторых программ, на которые были получены авторские свидетельства.
Поиск в банке алгоритмов проводится диалоговым методом, описанным в статье [5]. Если для решения проблемы нужны HTML-страницы, то они могут быть извлечены из банка алгоритмов, или написаны самим программистом.

Полученные результаты

Был разработан синтаксис языка описания проблемы PDL. Сокращенное PDL-описание скрипта filter.pl в качестве примера предоставлено ниже. Скрипт filter.pl читает журнальный файл Apache access_log и удаляет из него вспомогательные запросы, например, графических файлов, запросы со стороны поисковых серверов и т.д., что в разы ускоряет последующую работу программного пакета.
<# name=”filter.pl”; librs “IO::File, Fcntl, Socket, Time::Local, Time::localtime”; IN=”access_log”; OUT = “all records from IN, not_containing elements from @list_of_exceptions”; #>;
Создан транслятор, который генерирует из описаний проблемы на PDL исполняемый код алгоритмического языка высокого уровня (в нашем случае это Perl).
Был создан универсальный конфигуратор ПО, для пакета программ защиты Web-сервера, который предназначен для операционной системы Scientific Linux версии 6.10. Главной функцией конфигуратора является настройка операционной системы для использования пакета программ защиты Web-сервера. С помощью данной программы можно проверить физическую конфигурацию компьютера, наличие необходимого ПО и библиотек. Особенность конфигуратора заключается в том, что он устанавливает необходимое ПО автоматически, уменьшая вероятность ошибок из-за человеческого фактора [17-19]. Минимальное участие пользователя заключается только в том, что он вводит свой адрес почты, если ему надо настроить модуль контроля перегрева процессора proc.pl.
Была проведена апробация языка PDL путём создания транслятора, генерирующего пакет программ защиты Web-сервера из описаний, составленных на PDL. Схема работы пакета программ безопасности WEB-узла предоставлена на рисунке 4. Этот пакет программ мониторирует входящий и исходящий информационный трафик, детектирует атаки переполнения буфера, попытки DoS-атак и path traversal, а также случаи попыток использования нелегальных HTTP-методов (put, delete и пр.) и некоторых других атак. Предусмотрен контроль температуры ядер процессора и автоматическое отключения компьютера при перегреве. Если с какого-то IP-адреса атаки происходят слишком часто, программа редактирует конфигурационный файл Apache (вводит в httpd.conf строки «deny from IP») и перезапускает это приложение, что блокирует атаки со стороны выявленных IP-адресов. Результаты мониторинга при необходимости записываются в специальный банк данных.



Р и с. 4. Структура конфигурации программы безопасности WEB-сервера
Обозначения: C0 — использование методов put или delete. C1 — использование методов post и т.д.. F0/1 — детектирование в журнальном файле access_log случаев «../» , «/../../../../», %2e%2e%2f, ..%c0%af или %2e%2e/ отдельно.
F i g. 4. Configuration structure of the WEB-server security program
Notation: C0-using the put or delete methods. C1-using post methods, etc. F0 / 1-detecting cases in the access_log log file"../" , "/../../../../", %2e%2e%2f, ..%c0%af or %2e%2e / separately.

MRTG-monitoring осуществляется с помощью специальной программы net-snmp-utils, которая устанавливается конфигуратором. Выявление случаев попыток подбора пароля (pass guessing) проводится путём анализа системного файла **secure**. Случаи illegal HTTP-method, buffer overflow, high request frequency и path traversal attack регистрируются при анализе файла Apache access_log. Temper-monitoring — мониторинг температуры ядер процессора выполняется скриптом proc.pl.

Процесс “Missing file request” проводит анализ файла Apache error_log и выявляет попытки хакеров найти установленные на компьютере уязвимые программы.
В нашем трансляторе скрипты error.pl, pars_access_2.pl и secur_cur.pl синтезируются 1-м способом, а скрипты filter.pl и proc.pl — 2-м способом.

Заключение

Исследование применения языка PDL показало, что полностью устранить программные ошибки не получится, хоть их количество и сокращается. Язык позволяет упростить процесс программирования. Данный способ генерации кода может улучшить качество программ. Предлагаемая технология станет более эффективной при появлении компьютеров с искусственным интеллектом. Но данный подход может быть применен не только в программировании. Можно с помощью языка типа PDL описывать самые разные проблемы, надо будет только создать банки описания соответствующих технологий.

Список использованных источников

[1] Доренская, Е. А. О технологии программирования, ориентированной на минимизацию программных ошибок / Е. А. Доренская, Ю. А. Семенов. — DOI 10.25559/SITITO.2017.2.226 // Современные информационные технологии и ИТ-образование. — 2017. — Т. 13, № 2. — С. 50-56. – URL: https://elibrary.ru/item.asp?id=30258630 (дата обращения: 14.08.2020). — Рез. англ.

[2] Доренская, Е. А. Метод определения контекстных значений слов и документов / Е. А. Доренская, Ю. А. Семенов. — DOI 10.25559/SITITO.14.201804.896-902 // Современные информационные технологии и ИТ-образование. — 2018. — Т. 14, № 4. — С. 896-902. – URL: https://elibrary.ru/item.asp?id=37267550 (дата обращения: 14.08.2020). — Рез. англ.

[3] Доренская, Е. А. Новые методы минимизации ошибок в программном обеспечении / Е. А. Доренская, Ю. А. Семенов // CEUR Workshop Proceedings: Материалы 8-й Международной конференции «Распределенные вычисления и GRID-технологии в науке и образовании». (10-14 Сентября 2018 г., Дубна). — 2018. — Т. 2267. — С. 150-154. – URL: http://ceur-ws.org/Vol-2267/150-154-paper-27.pdf (дата обращения: 14.08.2020).

[4] Доренская, Е. А. Улучшенный алгоритм вычисления контекстного значения слов в тексте / Е. А. Доренская, Ю. А. Семенов. — DOI 10.25559/SITITO.15.201904.954-960 // Современные информационные технологии и ИТ-образование. — 2019. — Т. 15, №4. — 2019. — С. 954-960. – URL: https://elibrary.ru/item.asp?id=43575617 (дата обращения: 14.08.2020). — Рез. англ.

[5] Куликовская, А. А. Разработка банка алгоритмов и метода поиска программ в соответствии с требованиями пользователей / А. А. Куликовская, Е. А. Доренская, Ю. А. Семенов. — DOI 10.25559/SITITO.16.202001.81-89 // Современные информационные технологии и ИТ-образование. — 2020. — Т. 16, № 1. — С. 81-89. – URL: https://elibrary.ru/item.asp?id=44674777 (дата обращения: 14.08.2020). — Рез. англ.

[6] Семенов, Ю. А. Разработка банка алгоритмов и основ языка описания проблемы с целью минимизации числа программных ошибок / Ю. А. Семенов, А. П. Овсянников, Т. В. Овсянникова // Труды научно-исследовательского института системных исследований Российской академии наук. — 2016. — Т. 6, № 2. — С. 96-100. — URL: https://elibrary.ru/item.asp?id=29798446 (дата обращения: 14.08.2020). — Рез. англ.

[7] Котов, Э. М. Исследование моделей информационного поиска / Э. М. Котов, А. Н. Целых // Известия ЮФУ. Технические науки. — 2009. — № 4(93). — С. 163-168. — URL: https://elibrary.ru/item.asp?id=12834639 (дата обращения: 14.08.2020). — Рез. англ.

[8] Eidlin, A. A. Analyzing Weak Semantic Map of Word Senses / A. A. Eidlin, M. A. Eidlina, A. V. Samsonovich. — DOI 10.1016/j.procs.2018.01.023 // Procedia Computer Science. — 2018. — Vol. 123. — Pp. 140-148. — URL: https://www.sciencedirect.com/science/article/pii/S1877050918300243 (дата обращения: 14.08.2020).

[9] Afric, P. REPD: Source Code Defect Prediction As Anomaly Detection / P. Afric, L. Sikic, A. S. Kurdija, G. Delac, M. Silic. — DOI 10.1109/QRS-C.2019.00052 // 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C). — Sofia, Bulgaria, 2019. — Pp. 227-234. — URL: https://ieeexplore.ieee.org/document/8859503 (дата обращения: 14.08.2020).

[10] Noller, Y. Badger: complexity analysis with fuzzing and symbolic execution / Y. Noller, R. Kersten, C. S. Păsăreanu. — DOI 10.1145/3213846.3213868 // Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2018). — Association for Computing Machinery, New York, NY, USA, 2018. — Pp. 322-332. — URL: https://dl.acm.org/doi/10.1145/3213846.3213868 (дата обращения: 14.08.2020).

[11] Ye, M. EvoIsolator: Evolving Program Slices for Hardware Isolation Based Security / M. Ye M, M. B. Cohen, W. Srisa-an, S. Wei. — DOI 10.1007/978-3-319-99241-9_24 // Search-Based Software Engineering. SSBSE 2018. Lecture Notes in Computer Science; T. Colanzi, P. McMinn (ed.). Springer, Cham. — 2018. — Vol. 11036. — Pp. 377-382. — URL: https://link.springer.com/chapter/10.1007%2F978-3-319-99241-9_24 (дата обращения: 14.08.2020).

[12] Kádár, I. The optimization of a symbolic execution engine for detecting runtime errors / I. Kádár. — DOI 10.14232/actacyb.23.2.2017.9 // Acta Cybernetica. — 2017. — Vol. 23, no. 2. — Pp. 573-597. — URL: https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3942 (дата обращения: 14.08.2020).

[13] Wang, J. Dynamic Translation Optimization Method Based on Static Pre-Translation / J. Wang, J. Pang, X. Liu, F. Yue, J. Tan, L. Fu. — DOI 10.1109/ACCESS.2019.2897611 // IEEE Access. — 2019. — Vol. 7. — Pp. 21491-21501. — URL: https://ieeexplore.ieee.org/document/8635523 (дата обращения: 14.08.2020).

[14] Ferrara, P. From CIL to Java bytecode: Semantics-based translation for static analysis leveraging / P. Ferrara, A. Cortesi, F. Spoto. — DOI 10.1016/j.scico.2020.102392

// Science of Computer Programming. — 2020. — Vol. 191. — Article 102392. — URL: https://www.sciencedirect.com/science/article/pii/S0167642320300034 (дата обращения: 14.08.2020).

[15] Zaytsev, V. Modelling of language syntax and semantics: The case of the assembler compiler / V. Zaytsev. — DOI 10.5381/jot.2020.19.2.a5 // Journal of Object Technology. — 2020. — Vol. 19, no. 2. — Pp. 1-22. — URL: http://www.jot.fm/contents/issue_2020_02/article5.html (дата обращения: 14.08.2020).

[16] Rahman, M. M. Source Code Assessment and Classification Based on Estimated Error Probability Using Attentive LSTM Language Model and Its Application in Programming Education / M. M. Rahman, Y. Watanobe, K. Nakamura. — DOI 10.3390/app10082973 // Applied Sciences. — 2020. — Vol. 10, issue 8. — Article 2973. — URL: https://www.mdpi.com/2076-3417/10/8/2973 (дата обращения: 14.08.2020).

[17] Система порождения программ / В. Д. Ильин. — М.: Наука, 1989.

[18] Концептуальное программирование / Э. Х. Тыугу. — М.: Наука, 1984.

[19] Ильин, А. В. Систематизация знаний о программируемых задачах / А. В. Ильин, В. Д. Ильин. — DOI 10.14357/08696527140314 // Системы и средства информатики. — 2014. — Т. 24, № 3. — С. 192-203. — URL: https://www.elibrary.ru/item.asp?id=22482082 (дата обращения: 14.08.2020). — Рез. англ.

[20] Казакова, А. Е. Особенности семантики языков программирования / А. Е. Казакова // Вестник Московского университета. Серия 7: Философия. — 2007. — № 6. — С. 69-75. — URL: https://www.elibrary.ru/item.asp?id=11725960 (дата обращения: 14.08.2020). — Рез. англ.

[21] Егоров, М. А. Методика оценки безопасности программного кода корпоративных приложений / М. А. Егоров // Вестник МГТУ им. Н.Э. Баумана. Серия «Приборостроение». — 2011. — № S1. — С. 67-78. — URL: https://www.elibrary.ru/item.asp?id=17681815 (дата обращения: 14.08.2020). — Рез. англ.

[22] Наумов, Р. В. Актуальные языки программирования / Р. В. Наумов // Academy. — 2016. — № 1. — С. 49-50. — URL: https://www.elibrary.ru/item.asp?id=25345843 (дата обращения: 14.08.2020).

[23] Левушкин, А. В. Основные современные языки программирования / А. В. Левушкин, М. К. Турчанинов, А. А. Жиганов, В. В. Ермолаева // Молодой ученый. — 2018. — № 25(211). — С. 96-97. — URL: https://www.elibrary.ru/item.asp?id=35161929 (дата обращения: 14.08.2020).

[24] Стась, А. Н. Методика обучения разработке трансляторов / А. Н Стась // Вестник Томского государственного педагогического университета. — 2015. — № 8(161). — С. 76-81. — URL: https://www.elibrary.ru/item.asp?id=24075526 (дата обращения: 14.08.2020). — Рез. англ.

[25] Пахунов, А. В. Языки программирования: классификация, особенности, критерии выбора / А. В. Пахунов // Современная наука. — 2015. — № 4. — С. 89-91. — URL: https://www.elibrary.ru/item.asp?id=25430775 (дата обращения: 14.08.2020). — Рез. англ.

[26] Хохлов, Д. Г. Интерпретатор языка С для электронного обучения программированию / Д. Г. Хохлов, А. П. Кирпичников, Г. Х. Халид // Вестник технологического университета. — 2017. — Т. 20, № 14. — С. 109-111. — URL: https://www.elibrary.ru/item.asp?id=29880255 (дата обращения: 14.08.2020).

[27] Новичкова, М. И. Разработка транслятора языка программирования высокого уровня / М. И. Новичкова, Ю. А. Трофимов // Образование и наука в современных условиях. — 2015. — № 3. — С. 213-214. — URL: https://www.elibrary.ru/item.asp?id=23892014 (дата обращения: 14.08.2020).

[28] Лебедева, Т. Н. Формализация данных в языке программирования 1С / Т. Н. Лебедева, Л. С. Носова // Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика. — 2015. — № 3. — С. 113-121. — URL: https://www.elibrary.ru/item.asp?id=23826932 (дата обращения: 14.08.2020). — Рез. англ.

Поступила 14.08.2020;
одобрена после рецензирования 28.10.2020;
принята к публикации 17.11.2020.

Об авторах:

Доренская Елизавета Александровна, инженер-программист, ФГБУ «Институт теоретической и экспериментальной физики имени А.И.Алиханова Национального исследовательского центра «Курчатовский институт» (117218, Российская Федерация, г. Москва, ул. Большая Черёмушкинская, д. 25), ORCID: http://orcid.org/0000-0002-4249-5131, dorenskaya@itep.ru

Куликовская Анна Алексеевна, младший научный сотрудник, ФГБУ «Институт теоретической и экспериментальной физики имени А.И.Алиханова Национального исследовательского центра «Курчатовский институт» (117218, Российская Федерация, г. Москва, ул. Большая Черёмушкинская, д. 25); аспирант, ФГБУ «Национальный исследовательский центр «Курчатовский институт» (123182, Российская Федерация, г. Москва, пл. Академика Курчатова, д. 1), ORCID: https://orcid.org/0000-0002-0214-1697, kulikovskaya@phystech.edu

Семенов Юрий Алексеевич, ведущий научный сотрудник, ФГБУ «Институт теоретической и экспериментальной физики имени А.И.Алиханова Национального исследовательского центра «Курчатовский институт» (117218, Российская Федерация, г. Москва, ул. Большая Черёмушкинская, д. 25); заместитель заведующего кафедрой информатики и вычислительных сетей, Институт нано-, био-, информационных, когнитивных и социогуманитарных наук и технологий, ФГАОУ ВО «Московский физико-технический институт (национальный исследовательский университет)» (141701, Российская Федерация, Московская область, г. Долгопрудный, Институтский пер., д. 9), кандидат физико-математических наук, ORCID: http://orcid.org/0000-0002-3855-3650, semenov@itep.ru

Все авторы прочитали и одобрили окончательный вариант рукописи.

References

[1] Dorenskaya E.A., Semenov Yu.A. About the programming techniques, oriented to minimize errors. *Sovremennye informacionnye tehnologii i IT-obrazovanie* = Modern Information Technologies and IT-Education. 2017; 13(2):50-56. (In Russ., abstract in Eng.) DOI: https://doi.org/10.25559/SITITO.2017.2.226

[2] Dorenskaya E.A., Semenov Yu.A. The determination method for contextual meanings of words and documents. *Sovremennye informacionnye tehnologii i IT-obrazovanie* = Modern Information Technologies and IT-Education. 2018; 14(4):896-902. (In Russ., abstract in Eng.) DOI: https://doi.org/10.25559/SITITO.14.201804.896-902

[3] Dorenskaya E.A., Semenov Yu.A. New methods of minimizing the errors in the software. *CEUR Workshop Proceedings.* 2018; 2267:150-154. Available at: http://ceur-ws.org/Vol-2267/150-154-paper-27.pdf (accessed 14.08.2020). (In Eng.)

[4] Dorenskaya E.A., Semenov Yu.A. The improved algorithm for calculation of the contextual words meaning in the text. *Sovremennye informacionnye tehnologii i IT-obrazovanie* = Modern Information Technologies and IT-Education. 2019; 15(4):954-960. (In Eng.) DOI: https://doi.org/10.25559/SITITO.15.201904.954-960

[5] Kulikovskaya A.A., Dorenskaya E.A., Semenov Yu.A. Development of an Algorithm's Bank and Method for Searching Programs in Accordance with User Requirements. *Sovremennye informacionnye tehnologii i IT-obrazovanie* = Modern Information Technologies and IT-Education. 2020; 16(1):81-89. (In Russ., abstract in Eng.) DOI: https://doi.org/10.25559/SITITO.16.202001.81-89

[6] Semenov Yu.A., Ovsyannikov A.P., Ovsyannikova T.V. Development of the algorithm bank and basics of the language for problem description to minimize a number of program errors. *Trudy NIISI RAN* = Proceedings of SRISA RAS. 2016; 6(2):96-100. Available at: https://elibrary.ru/item.asp?id=29798446 (accessed 14.08.2020). (In Russ., abstract in Eng.)

[7] Kotov E.M., Tselykh A.N. Research of models for information retrieval. *Izvestiya SFedU. Engineering Sciences*. 2009; (4):163-168. Available at: https://elibrary.ru/item.asp?id=12834639 (accessed 14.08.2020). (In Russ., abstract in Eng.)

[8] Eidlin A.A., Eidlina M.A., Samsonovich A.V. Weak Semantic Map of Word Senses. *Procedia Computer Science*. 2018; 123:140-148. (In Eng.) DOI: https://doi.org/10.1016/j.procs.2018.01.023

[9] Afric P., Sikic L., Kurdija A.S., Delac G., Silic M. REPD: Source Code Defect Prediction As Anomaly Detection. In: *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. Sofia, Bulgaria; 2019. p. 227-234. (In Eng.) DOI: https://doi.org/10.1109/QRS-C.2019.00052

[10] Noller Y., Kersten R., Păsăreanu C.S. Badger: complexity analysis with fuzzing and symbolic execution. In: *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2018).* Association for Computing Machinery, New York, NY, USA; 2018. p. 322-332. (In Eng.) DOI: https://doi.org/10.1145/3213846.3213868

[11] Ye M., Cohen M.B., Srisa-an W., Wei S. EvoIsolator: Evolving Program Slices for Hardware Isolation Based Security. In: Colanzi T., McMinn P. (ed.) Search-Based Software Engineering. SSBSE 2018. *Lecture Notes in Computer Science*. 2018; 11036:377-382. Springer, Cham. (In Eng.) DOI: https://doi.org/10.1007/978-3-319-99241-9_24

[12] Kádár I. The optimization of a symbolic execution engine for detecting runtime errors. *Acta Cybernetica*. 2017; 23(2):573-597. (In Eng.) DOI: https://doi.org/10.14232/actacyb.23.2.2017.9

[13] Wang J., Pang J., Liu X., Yue F., Tan J., Fu L. Dynamic Translation Optimization Method Based on Static Pre-Translation. *IEEE Access*. 2019; 7:21491-21501. (In Eng.) DOI: https://doi.org/10.1109/ACCESS.2019.2897611

[14] Ferrara P., Cortesi A., Spoto F. From CIL to Java bytecode: Semantics-based translation for static analysis leveraging. *Science of Computer Programming*. 2020; 191:102392. (In Eng.) DOI: https://doi.org/10.1016/j.scico.2020.102392

[15] Zaytsev V. Modelling of Language Syntax and Semantics: The Case of the Assembler Compiler. *Journal of Object Technology*. 2020; 19(2):1-22. (In Eng.) DOI: https://doi.org/10.5381/jot.2020.19.2.a5

[16] Rahman M.M., Watanobe Y., Nakamura K. Source Code Assessment and Classification Based on Estimated Error Probability Using Attentive LSTM Language Model and Its Application in Programming Education. *Applied Sciences*. 2020; 10(8):2973. (In Eng.) DOI: https://doi.org/10.3390/app10082973

[17] Ilyin V.D. *Sistema porozhdeniya programm* [Program Generating System]. Nauka, Moscow; 1989. (In Russ.)

[18] Tyugu E.H. *Konceptual'noe programmirovanie* [Conceptual Programming]. Nauka, Moscow; 1984. (In Russ.)

[19] Ilyin A.V., IlyinV.D. Systematization of Knowledge about Programmable Tasks. *Systems and Means of Informatics*. 2014; 24(3):192-203. (In Russ., abstract in Eng.) DOI: https://doi.org/10.14357/08696527140314

[20] Kazakova A.E. Peculiarities of semantics of programming languages. *Moscow University Bulletin. Series 7. Philosophy*. 2007; (6):69-75. Available at: https://www.elibrary.ru/item.asp?id=11725960 (accessed 14.08.2020). (In Russ., abstract in Eng.)

[21] Egorov M.A. *Metodika ocenki bezopasnosti programmnogo koda korporativnyh prilozhenij* [Methodology for Evaluating the Security of Corporate Application Software Code]. *Herald of the Bauman Moscow State Technical University. Series Instrument Engineering*. 2011; (S1):67-78. Available at: https://www.elibrary.ru/item.asp?id=17681815 (accessed 14.08.2020). (In Russ., abstract in Eng.)

[22] Naumov R.V. *Aktual'nye jazyki programmirovanija* [Current Programming Languages]. Academy. 2016; (1):49-50. Available at: https://www.elibrary.ru/item.asp?id=25345843 (accessed 14.08.2020). (In Russ.)

[23] Levushkin A.V., Turchaninov M.K., Zhiganov A.A., Yermolaeva V.V. *Osnovnye sovremennye jazyki programmirovanija* [The Main Modern Programming Languages]. *Young Scientist*. 2018; (25):96-97. Available at: https://www.elibrary.ru/item.asp?id=35161929 (accessed 14.08.2020). (In Russ.)

[24] Stas A.N. Methods of Teaching the Designing of Translators. *Tomsk State Pedagogical University Bulletin*. 2015; (8):76-81. Available at: https://www.elibrary.ru/item.asp?id=24075526 (accessed 14.08.2020). (In Russ., abstract in Eng.)

[25] Pakhunov A.V. Programming Languages: Classification, Features, Criteria of Choice. Modern Science. 2015; (4):89-91. Available at: https://www.elibrary.ru/item.asp?id=25430775 (accessed 14.08.2020). (In Russ., abstract in Eng.)

[26] Khokhlov D.G., Kirpichnikov A.P., Khalid G.H. *Interpretator jazyka C dlja jelektronnogo obuchenija programmirovaniju* [C interpreter for e-learning programming]. *Bulletin of the Technological University*. 2017; 20(14):109-111. Available at: https://www.elibrary.ru/item.asp?id=29880255 (accessed 14.08.2020). (In Russ.)

[27] Novichkova M.I., Trofimov I.A. *Razrabotka transljatora jazyka programmirovanija vysokogo urovnja* [Development of a high-level programming language translator]. *Obrazovanie i nauka v sovremennyh uslovijah* = Education and Science in the Modern Context. 2015; (3):213-214. Available at: https://www.elibrary.ru/item.asp?id=23892014 (accessed 14.08.2020). (In Russ.)

[28] Lebedeva T.N., Nosova L.S. Formalization of Data in the Programming Language 1C. *Vestnik of Astrakhan State Technical University. Series: Management, Computer Sciences and Informatics*. 2015; (3):113-121. Available at: https://www.elibrary.ru/item.asp?id=23826932 (accessed 14.08.2020). (In Russ., abstract in Eng.)

Submitted 14.08.2020; approved after reviewing 28.10.2020; accepted for publication 17.11.2020.

About the authors:

Elizaveta A. Dorenskaya, Software Engineer, Institute for Theoretical and Experimental Physics named by A.I. Alikhanov of National Research Centre "Kurchatov Institute" (25 Bolshaya Cheremushkinskaya St., Moscow 117218, Russian Federation), ORCID: http://orcid.org/0000-0002-4249-5131, dorenskaya@itep.ru

Anna A. Kulikovskayaa, Junior Researcher, Institute for Theoretical and Experimental Physics named by A.I. Alikhanov of National Research Centre "Kurchatov Institute" (25 Bolshaya Cheremushkinskaya St., Moscow 117218, Russian Federation); Postgraduate Student, National Research Center "Kurchatov Institute" (1 Akademika Kurchatova pl., Moscow 123182, Russian Federation), ORCID: https://orcid.org/0000-0002-0214-1697, kulikovskaya@phystech.edu

Yuri A. Semenov, Lead Researcher, Institute for Theoretical and Experimental Physics named by A.I. Alikhanov of National Research Centre "Kurchatov Institute" (25 Bolshaya Cheremushkinskaya St., Moscow 117218, Russian Federation); Deputy Head of the Chair for Computer Science, Institute of Nano-, Bio-, Information, Cognitive and Socio-humanistic Sciences and Technologies, Moscow Institute of Physics and Technology (National Research University) (9 Institutskiy per., Moscow Region, Dolgoprudny 141701, Russian Federation), Ph.D. (Phys.-Math.), ORCID: http://orcid.org/0000-0002-3855-3650, semenov@itep.ru

All authors have read and approved the final manuscript.