ТРУДЫ
ИНСТИТУТА СИСТЕМНОГО ПРОГРАММИРОВАНИЯ РАН

PROCEEDINGS OF THE INSTITUTE FOR SYSTEM PROGRAMMING OF THE RAS

ISSN Print 2079-8156
Том 33 Выпуск 3

ISSN Online 2220-6426
Volume 33 Issue 3

Институт системного
программирования
им. В.П. Иванникова РАН

Москва, 2021

ИСП РАН

ISSN 2079-8156 (Print), ISSN 2220-6426 (Online)

Труды Института системного программирования РАН
Proceedings of the Institute for System Programming of the RAS

Труды ИСП РАН – это издание с двойной анонимной системой рецензирования, публикующее научные статьи, относящиеся ко всем областях системного программирования, технологий программирования и вычислительной техники. Целью издания является формирование научно-информационной среды в этих областях путем публикации высококачественных статей в открытом доступе.
Издание предназначено для исследователей, студентов и аспирантов, а также практиков. Оно охватывает широкий спектр тем, включая, в частности, следующие:

- операционные системы;
- компиляторные технологии;
- базы данных и информационные системы;
- параллельные и распределенные системы;
- автоматизированная разработка программ;
- верификация, валидация и тестирование;
- статический и динамический анализ;
- защита и обеспечение безопасности ПО;
- компьютерные алгоритмы;
- искусственный интеллект.

Журнал издается по одному тому в год, шесть выпусков в каждом томе.
Поддерживается открытый доступ к содержанию издания, обеспечивая доступность результатов исследований для общественности и поддерживая глобальный обмен знаниями.
Труды ИСП РАН реферируются и/или индексируются в:

Proceedings of ISP RAS are a double-blind peer-reviewed journal publishing scientific articles in the areas of system programming, software engineering, and computer science. The journal's goal is to develop a respected network of knowledge in the mentioned above areas by publishing high quality articles on open access.
The journal is intended for researchers, students, and practitioners. It covers a wide variety of topics including (but not limited to):

- Operating Systems.
- Compiler Technology.
- Databases and Information Systems.
- Parallel and Distributed Systems.
- Software Engineering.
- Software Modeling and Design Tools.
- Verification, Validation, and Testing.
- Static and Dynamic Analysis.
- Software Safety and Security.
- Computer Algorithms.
- Artificial Intelligence.

The journal is published one volume per year, six issues in each volume.
Open access to the journal content allows to provide public access to the research results and to support global exchange of knowledge. **Proceedings of ISP RAS** is abstracted and/or indexed in:

УДК004.45

Адрес: 109004, г. Москва, ул. А. Солженицына, дом 25.
Телефон: +7(495) 912-44-25
E-mail: proceedings@ispras.ru
Сайт: https://ispranproceedings.elpub.ru/

Address: 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
Tel: +7(495) 912-44-25
E-mail: proceedings@ispras.ru
Web: https://ispranproceedings.elpub.ru/

Содержание

Table of Contents

DOI: 10.15514/ISPRAS-2021-33(3)-1

What Software Architecture Styles are Popular?

A.A Mitsyuk, ORCID: 0000-0003-2352-3384<amitsyuk@hse.ru>
N.A. Jamgaryan, ORCID: 0000-0001-9964-5850 <nazhamgaryan@edu.hse.ru>
HSE University,
11, Pokrovsky boulevard, Moscow, 109028, Russia

Abstract. One can meet the software architecture style's notion in the software engineering literature. This notion is considered important in books on software architecture and university sources. However, many software developers are not so optimistic about it. It is not clear, whether this notion is just an academic concept, or is actually used in the software industry. In this paper, we measured industrial software developers' attitudes towards the concept of software architecture style. We also investigated the popularity of eleven concrete architecture styles. We applied two methods. A developers’ survey was applied to estimate developers' overall attitude and define what the community thinks about the automatic recognition of software architecture styles. Automatic crawlers were applied to mine the open-source code from the GitHub platform. These crawlers identified style smells in repositories using the features we proposed for the architecture styles. We found that the notion of software architecture style is not just a concept of academics in universities. Many software developers apply this concept in their work. We formulated features for the eleven concrete software architecture styles and developed crawlers based on these features. The results of repository mining using the features showed which styles are popular among developers of open-source projects from commercial companies and non-commercial communities. Automatic mining results were additionally validated by the Github developers survey.

Keywords: software architecture style; software design; code smells; software repository mining; survey

For citation: Mitsyuk A.A., Jamgaryan N.A. What Software Architecture Styles are Popular? Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 7-26. DOI: 10.15514/ISPRAS-2021-33(3)-1

Acknowledgments. This work is an output of a research project implemented as part of the Basic Research Program at the Natinal Research University Higher School of Economics (HSE University). We thank our colleagues from PAIS Lab (HSE University) whose advice was very helpful in doing our developer surveys better. In particular, Sergey A. Shershakov proposed useful improvements.

Какие стили архитектуры программного обеспечения популярны?

А.А. Мицюк, ORCID: 0000-0003-2352-3384<amitsyuk@hse.ru>
Н.А. Жамгарян, ORCID: 0000-0001-9964-5850 <nazhamgaryan@edu.hse.ru>
Национальный исследовательский университет Высшая школа экономики,
109028, Россия, Москва, Покровский бульвар 11

Аннотация. В литературе по программной инженерии можно встретить понятие архитектурного стиля программного обеспечения (ПО). Во многих книгах по архитектуре ПО и академических лекциях это понятие рассматривается как одно из важных. Однако, многие разработчики-практики пессимистично настроены в отношении понятия архитектурного стиля. Таким образом, не вполне понятно, является ли данное понятие чисто академической концепцией или действительно используется разработчиками прикладного программного обеспечения. В этой статье делается попытка оценить отношение разработчиков-практиков к концепции архитектурного стиля ПО. Также оценивается популярность одиннадцати конкретных архитектурных стилей. Применяются два метода. Опрос разработчиков был применен для оценки отношения разработчиков и определения того, считает ли сообщество разработчиков возможным автоматическое распознавание архитектурных стилей. Для интеллектуального анализа открытого исходного кода с платформы GitHub применялись автоматические скрипты. Эти скрипты позволяют выявлять факт использования стилей в конкретных репозиториях. Скрипты работают на основе самостоятельно разработанных наборов свойств для выбранных стилей. Было обнаружено, что понятие стиля архитектуры программного обеспечения – это не только «университетская» концепция. Многие разработчики ПО применяют это понятие и соответствующую концепцию в своей работе. В работе сформулированы свойства для одиннадцати архитектурных стилей ПО и описаны разработанные на основе этих свойств автоматические скрипты. Результаты интеллектуального анализа репозиториев с использованием предложенных свойств показали, какие стили популярны среди разработчиков проектов с открытым исходным кодом, опубликованных коммерческими компаниями и некоммерческими сообществами. Результаты интеллектуального анализа репозиториев дополнительно валидируются опросом GitHub-разработчиков.

Ключевые слова: стиль архитектуры программного обеспечения; проектирование программного обеспечения; запахи кода; интеллектуальный анализ репозиториев с исходным кодом; опрос

Для цитирования: Мицюк А.А., Жамгарян Н.А. Какие стили архитектуры программного обеспечения популярны? Труды ИСП РАН, том 33, вып. 3, 2021 г., стр. 7-26 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(3)–1.

Благодарности. Работа выполнена в рамках Программы фундаментальных исследований НИУ ВШЭ. Мы благодарим наших коллег из PAIS Lab (НИУ ВШЭ), чьи советы помогли нам улучшить наши исследования разработчиков. В частности, полезные улучшения предложил С.А. Шершаков.

1. Introduction

Software architecture [1] is a discipline within software engineering dealing with software systems' structural and behavioral design. Software architects and designers define how the system is organized, its components, how these components communicate, etc. Software engineering literature (see, for example, foundational works by Shaw and Garlan [2], Taylor et al. [3], Richards and Neal [4]) applies a notion of *architecture style* or *pattern*. Shaw and Garlan [2] define it as follows: «*An architectural style defines: a family of systems in terms of a pattern of structural organization; a vocabulary of components and connectors, with constraints on how they can be combined*.» Taylor et al. [3] proposed another definition: «*An architectural style is a named collection of architectural design decisions that (1) are applicable in a given development context, (2) constrain architectural design decisions that are specific to a particular system within that context, and (3) elicit beneficial qualities in each resulting system*.» These definitions are general and relatively abstract as well as most other definitions from software engineering books and papers. Usually, no clues on how these styles can be identified and implemented in a concrete software source code are given.

This work summarizes our team's first results to understand better the concept of *software architecture style* and make it more tangible.

To do so, we first tried to find out the developers' attitude towards the concept of software architectural style. Are real non-academic developers familiar with this concept in general and with different particular styles? Do they consider this concept useful in their everyday professional activities? Secondly, we tried to identify empirical features of software architecture styles, which can be used in practice to recognize the usage of software architecture styles in Java and Python programs.

For our research, certain architecture styles were chosen. Then, we chose a small sample of software repositories, which were investigated to get empirical features of the architecture styles in source code. Afterward, the crawlers were written and applied to parse the bigger sample of open source repositories on GitHub[1] service.

Besides, we conducted two developers' surveys. The first survey aimed to find out the developers' attitude towards the concept of software architecture styles. We held the survey to understand better whether this topic is worth researching. The second survey aimed to validate the results of the crawlers' parsing.

The aim of our research project – to identify empirical features of architecture styles – is new to software engineering, while the applied methods are well known. Surveys and repository mining were applied in many other research projects on code smells detection and design patterns identification (see Section 6). The methods we used had shown themselves as feasible in exploratory research projects.

Due to the first survey results, developers have *positive* attitude towards architecture styles. Many of them apply this concept in their projects, and even more of them think it is beneficial to be acquainted with the concept. In data provided by the automatic crawlers we found, how much each of the chosen architecture styles is used in practice. We validated the results of crawlers mining using the second survey.

2. Research Questions and Paper Structure

In this paper, we consider the three following research questions.

RQ1: *What is the community attitude towards the concept of software architecture style?* Software architecture is taught in universities. Technical experts and master coaches promote advanced styles.

However, what does a typical software engineer think about this concept? We try to answer this question in Section 3 using a developers' survey.

RQ2: *How can we detect a software architecture style in code?* Results of the RQ1 survey encouraged us to try to construct a procedure for detecting software architecture styles in an actual source code. To do so, we first needed to select features related to particular styles using which we can automatically detect them. Section 4 answers the second research question and presents style features and automated scripts which help us to detect styles in code.

RQ3: *What software architecture styles are popular in open-source projects?* Finally, it is of interest to investigate the source code of existing software to decide what styles are popular. Fortunately, much open-source software is available for researchers in the modern world. Thus, we can mine open-source repositories and apply our architecture style detection tool to them.

This procedure is presented and discussed in Section 5.

Section 6 describes some works related to our research project, while Section 7 concludes this paper and proposes the ideas for further work.

3. Software Architectural Styles (RQ1)

Our first questions were as follows. Whether the concept of software architecture style is familiar to developers? Is this concept considered applicable? What particular styles are familiar to developers and are worth considering in the following steps of our research?

3.1 Architecture Styles Survey

To answer these questions, we provided a developers' survey described in this section.

For our research, we have created a survey[2] using Google Forms[3]. This survey consisted of 3 categories of questions.

[1] GitHub web-page: https://github.com/
[2] It can be found at the web-page of our project: https://pais.hse.ru/en/research/projects/softarchstyles
[3] Google Forms: https://docs.google.com/forms

Demographical questions: These are questions about programming experience, job area, preferences in technologies, and a respondent's frameworks.

General questions about software architecture styles: Whether participant had or had not heard and used the concept of architecture styles in their professional life?

Questions about the set of particular architecture styles we selected for our research: We asked whether the participant knew the name of the style and how he or she thought it is possible to identify that certain style in code.

These questions aimed to find out what community of developers thinks and knows about architecture styles usage and architecture styles identification.

3.2 What Styles did We Select?

We have selected the following eleven software architecture styles for this research:

- Model-View-Controller (MVC) architecture;
- Main and sub-programs;
- Machine-learning-based software;
- Event-driven software architecture;
- Reflection-using software;
- Data-centric software architecture;
- Expert system;
- Cloud-service-based software;
- Software with containerization;
- Aspect-oriented software architecture;
- Reactive-based software architecture.

These particular styles were chosen based on software architecture pattern and style catalogs from foundational literature of the field [3-7]. Usually, software architecture books are large and contain profound discussions on each of the styles considered important by book authors. The list of software architecture styles is a massive one. We had to limit this list somehow for it to be treatable within a single research project's borders. To select the particular set of styles, we consulted with literature of the field [3-7] as well as Wikipedia.org information[4]. Some of these styles (for example, MVC and Event-driven architectures) are popular and frequently used among software developers. Others (for example, aspect-based software and expert systems) are not famous in modern software engineering. Besides, we selected styles for which we can define features based on which the style smells can be detected in source code. Thus, we consider it worth investigating this particular set of styles. However, we do not state that this is an exhaustive set.

3.3 Survey Data

The survey was held from September till December 2020. As it has been mentioned, Google Forms were used for the survey. The survey form was spread in different developer communities connected with various areas of development: game development, back-end development, front-end development, data science, etc. We hoped to achieve randomness and broader coverage by doing so. In total, 111 developers participated in the survey.

Participants of the survey have different experiences in software programming. Fig. 1 shows participant programming experience in years. From this figure we can conclude that about half of all participants were in the middle of the experience range: slightly less than one quarter have experience from 1 to 2 years, a little bit more than one-quarter of the total have experience from 3

[4] See page https://en.wikipedia.org/wiki/List_of_software_architecture_styles_and_patterns which itself refers to the paper of Sharma et al. [8].

to 5 years. Experienced developers make one-third of the total number: about one-fifth have experience from 6 to 10, and slightly more than 15% have experience from 11 to 20 years. At the ends of the distribution, we can observe 5% of developers with experience less than 1 year and about 3% of very experienced developers who are in the field for more than 35 years.

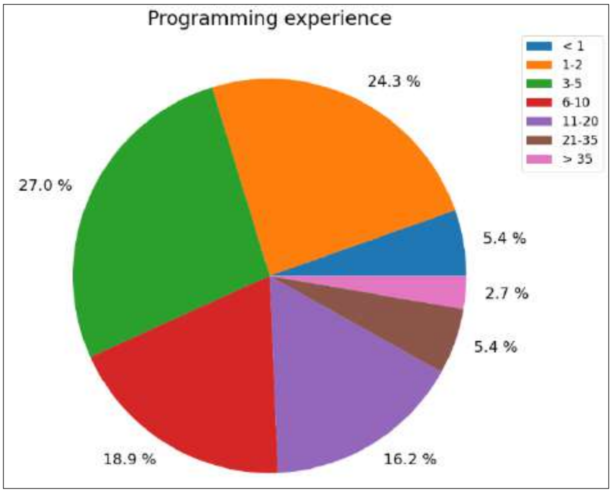


Fig. 1. Participant programming experience in years

Fig. 2 represents fields of software engineering which participants selected as their primary occupation. Note that a participant could select several fields as their primary occupations. We can conclude that survey participants in different areas, with most of them, are back-end developers. The top 7 categories of participant job areas were: back-end development (65.8%), front-end development (34.2%), mobile development (22.5%), data analytics (19.8%), machine learning (18.9%), research (18%), and game development (9.9%).

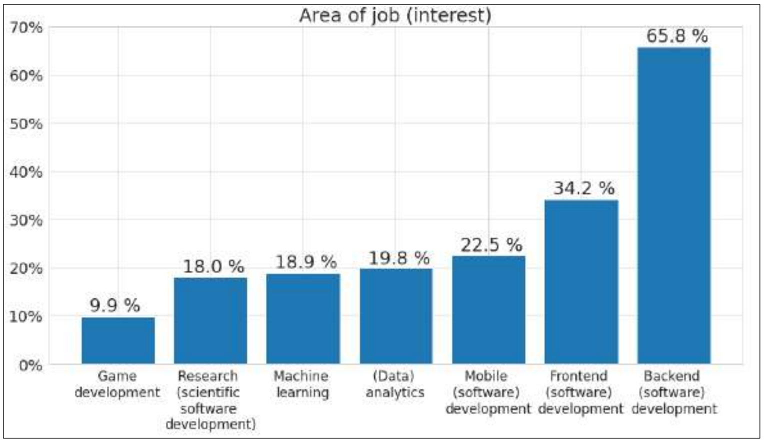


Fig. 2. Participant occupations

Finally, we asked participants about the programming languages they used in their work. Fig. 3 show how they answered. It can be seen that the survey participants are using different languages in their practice. The top three most popular languages in our survey are Python (45.9%), Java (33.3%), and SQL (32.4%). Partially because of these results, we decided to continue our research based on Java and Python source code.

Demographic data showed that our survey participants were similar to the typical software developers. For example, the participants' set of main languages is very similar to the well-known TIOBE Index[5]. Our selection is somehow shifted to object-oriented languages for back-end development. However, of the top 10 languages in TIOBE Mar 2021 (C, Java, Python, C++, C\#, Visual Basic, JavaScript, PHP, Assembly language, SQL) 7 are also presented in the top 10 languages used by survey participants. Developers came from different fields, which are popular in modern software engineering.

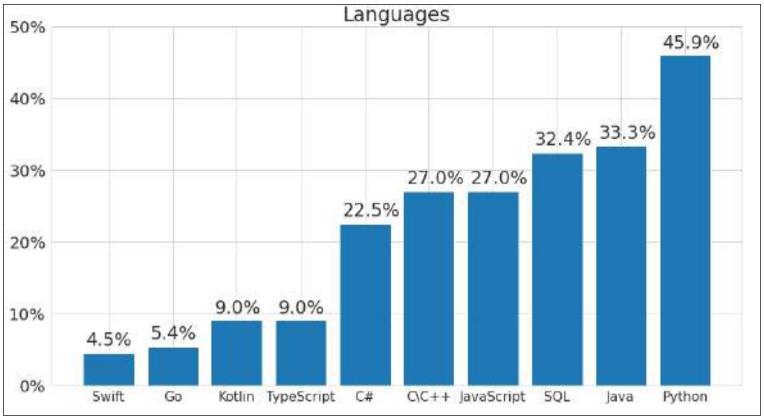


Fig. 3. Participant main programming languages

3.4 Survey Results

Our survey asked whether participants used the concept of software architecture style in their daily work practice. Fig. 4 shows how they answered this question. In this figure, we can see that almost 40\% frequently use the concept of architecture styles in development. 36% of all participants use them from time to time, and one quarter does not use architecture styles at all.

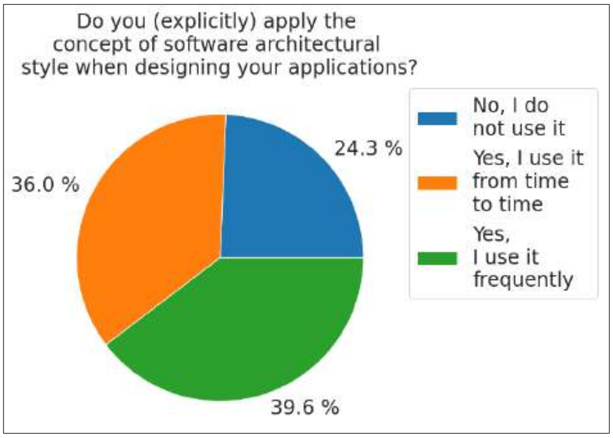


Fig. 4. Do participants apply the concept of software architecture style in their work?

The next question brought us surprising results. We decided to find out what participants thought about the developers' community in general. In particular, we asked what participants thought about how their colleagues applied the concept of software architecture style in their work? Fig. 5 shows that only 14.4% think that developers from their community do not use the concept of architecture

[5] TIOBE Index: https://www.tiobe.com/tiobe-index/

styles. Interestingly, developers tend to think their colleagues are significantly more familiar with the concept of software architecture style than themselves.

Finally, we were interested in what participants think about the feasibility of detecting architecture styles. The developers were asked whether they thought it is possible to identify the usage of a software architecture style in the source code automatically or manually.

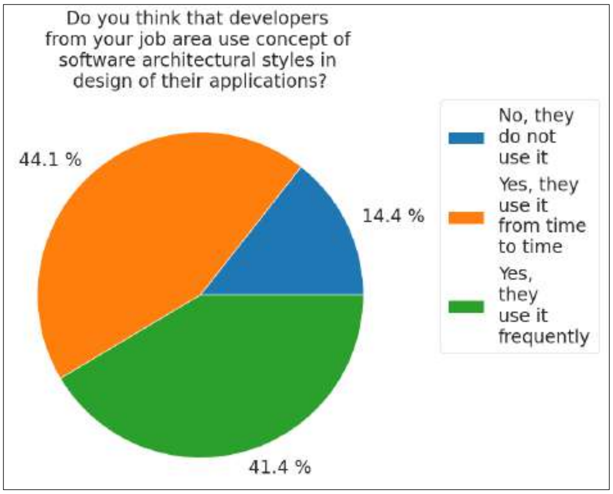


Fig. 5. What participants think about how their colleagues apply the concept of software architecture style in their work?

For each of the styles there were five possible answers as follows:

- Yes, by looking at language constructions manually;
- Yes, by looking at language constructions automatically;
- Yes, by looking at frameworks (you can list frameworks in ``other'' section);
- haven't used this style;
- Other (open answer).

A participant was able to select several answers simultaneously.

Fig. 6 summarizes the answers. To make the figure more illustrative we merged all the answers into the three categories:

- Manually: variant 1) and some of variant 5);
- Auto: variant 2), variant 3), and some of variant 5);
- Have not used: variant 4).

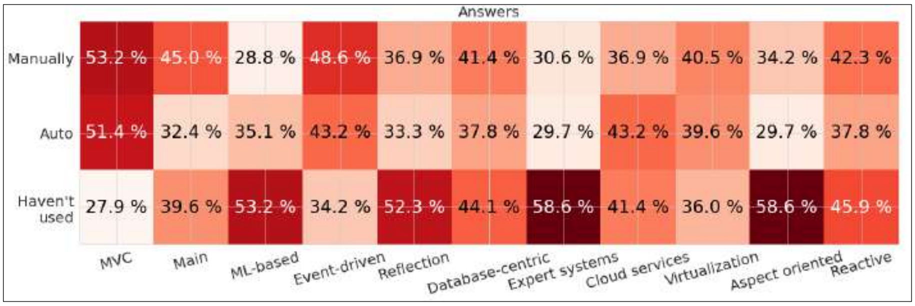


Fig. 6. What participants think about whether particular software architecture styles can be detected, or not?

Numbers in fig. 6 show how many developers selected this particular answer category for a particular architecture style. According to our data, developers are not very optimistic regarding software architecture style detection. However, for most architectural styles, at least one-third of all survey developers believe they can be identified either automatically (30% – 50%) or manually. Developers tend to think that familiar styles are more likely to be identified. For example, MVC is the most known style among the others. Most developers think it can be identified manually (53.2%) and automatically (51.4%). Expert systems and aspect-oriented software are unfamiliar to more than half of the developers from our selection. Not so many participants believe these styles can be identified by investigating the software source code.

3.5 Conclusions

The results of the developers' community survey are interesting events separately. However, we analyze them in the context of a larger project.

The survey shows that 3 of 4 typical developers apply the concept of software architecture styles in their practice. Moreover, developers believe their colleagues use this concept even more often. This means that developers consider the concept important and valuable in the software engineering process.

From 3.5 to 4 out of 10 typical developers believe that software architecture style can be identified by investigating the software source code. In general, slightly more developers think that a style can be identified manually. Besides, the more familiar a particular style is to the developer, the more likely it would be considered identifiable by this developer.

Thus, it would be interesting to investigate if software architecture styles can be identified using an automated procedure, how this can be done, and what styles are more prevalent in open source.

Note that we can somehow estimate styles' popularity by comparing developers' numbers unfamiliar with different styles. However, we believe that such research based on survey data only would be insufficient.

4. Architecture Style Identification (RQ2)

4.1 Detection Methods and Data Sources

In this research open-source software repositories were used as data. We chose 10 technological communities and companies with extensive lists of open-source repositories on GitHub, which is the largest resource with open software sources. Repositories related to the following companies' Github accounts are considered in this paper:

1) Adobe – https://github.com/adobe,
2) Amazon – https://github.com/amzn,
3) Amazon Web Services – https://github.com/aws,
4) AWS Labs – https://github.com/awslabs,
5) Apache Foundation – https://github.com/apache,
6) Apple – https://github.com/apple,
7) Google – https://github.com/google,
8) IBM – https://github.com/IBM,
9) Microsoft – https://github.com/microsoft,
10) 18F – https://github.com/18F.

We decided to consider only repositories with the code written in Python and Java as these two programming languages are among the most popular according to both well-known indices[6] and to our preliminary developer survey.

[6] For example, see TIOBE Index here: \ https://www.tiobe.com/tiobe-index/

The crawler was written in Python 3. We used the library, called PyGitHub[7]. Every crawler gets access to the companies' repositories by tokens previously generated by us manually on Github.

Our crawlers got access to GitHub repositories by using the token mechanism. Every token allows making ten thousand requests to Github per hour. For the mining process to continue flawlessly, several tokens have been used. The tool iterates through the token list and requests the source code from every repository taken for the research. For every software architecture style, we created a separate specific crawler. Their code is accessible at the project web page.

4.2 Features of Software Architecture Styles

We have created features of different origins for eleven styles from our research. These features can be grouped into two main categories.

The first group of features contains *framework-based* features. We firstly identified frameworks that propose implementations of particular architectural styles. After that, we identified usage of the style by finding usage of these frameworks in source code. Such features were used when identifying Model View Controller (4 python frameworks, 4 java frameworks), Machine Learning based style (24 python frameworks, 11 java frameworks), Event-driven (10 python frameworks, 8 java frameworks), Data-centric (25 python frameworks, 22 java frameworks), Expert systems (7 python frameworks, 3 java frameworks), Cloud systems (10 python frameworks, 7 java frameworks), Aspect-based applications (3 python frameworks, 1 java frameworks).

The second group of features contains *language-based* features. This means that we first identified how certain styles are implemented in specific languages (Java, Python). After that, we identified usage of the style by finding particular language constructs. These features were used when identifying Main and Sub-programs, Reflection architecture styles.

Table 1 provides a short description of every architecture style we have chosen for our research. It is assumed to hint about how they are presented in books and online resources. Besides, we give examples of features that we have used to identify the architectural styles. The complete list of features we used is available on the project web page.

Table 1. Empirical features of eleven software architecture styles

Architecture style	**Short description**	**Examples of empirical features**
Model-View-Controller	Architecture style includes: model (a dynamic data structure), view (a component to represent the information), and controller (this component accepts user input and converts it to commands).	***Python, Spring***: @Controller, import org.springframework.stereotype.Controller ***Java, Django***: from django.db import models, from models import
Main and sub-programs	Architecture style assumes an absence of classes. It means that application use only functions/ procedures and may use classes only as storage for functions/procedures without creating instances of classes.	***Python features:*** def main, fwithout defg main() ***Python anti-feature:*** def __init__ ***Java feature:*** public static void main(String[] args), ***Java anti-features:*** class fClassNameg, new fClassName)
Machine-learning based	Architecture style assumes usage of any data science-related frameworks and libraries.	***Python, Scikit learn:*** from sklearn, import sklearn ***Java, Apache Spark ML-lib:*** org.apache.spark.mllib

[7] PyGitHub web page: {https://pygithub.readthedocs.io

Event-driven software	Architecture style implies production, detection, consumption and reaction to events. Usually, implemented based on special frameworks	***Python, Apache Kafka:*** from kafka, import kafka ***Java, Apache Kafka:*** org.apache.kafka
Reflection-using software	Architecture style assumes that application’s processes can and do examine, introspect and modify their own structure and behavior	***Python features:*** type(obj), isinstance(obj, obj) ***Java features:*** java.lang.reflect, .getClass()
Data-centric software	Architecture style implies that database is a crucial (central) part of application	***Python, MySQL:*** from mysql, import mysql ***Java, MySQL:*** import java.sql, com.mysql.jdbc.Driver
Expert system	Architecture style assumes usage of any expert system frameworks and libraries as a part of the considered software.	*Python, Experta:* from experta, import experta ***Java, Apache Jena:*** import org.apache.jena
Cloud-service-based	Architecture style implies usage of frameworks and libraries, which let usage of cloud based delivery and inter-cloud network.	***Python, Apache Libcloud:*** from libcloud, import libcloud ***Java, Google Cloud:*** com.google.cloud
Software with containerization	Architecture style assumes usage of frameworks and libraries which let usage of virtual machines.	***Python, VMWare:*** from vmware, import vmware ***Java, VMWare:*** com.vmware
Aspect-oriented software	Architecture style aims to increase modularity by allowing separation of cross-cutting concerns.	***Python, AspectLib***: import aspectlib, from aspectlib ***Java, AspectJ:*** @Aspect, import org.aspectj
Reactive-based software	Architecture style pays attention to data streams and propagation of change.	***Python, ReactiveX:*** from rx, import rx ***Java, ReactiveX***: import io.reactivex

Investigating Popularity of Particular Styles in Open-source Software (RQ3)}

5. Investigating Popularity of Particular Styles in Open-source Software (RQ3)

5.1 Dataset Description

Our web crawlers gathered a dataset that we used to answer RQ3. This dataset consists of JSON files. Each file in the dataset is related to a triple: (`programming\_language; company\_name; software\_architecture\_style`). In total, the 3057 repositories were processed. 1682 of them are repositories with source code in Java, whereas 1375 contain Python source code. Each repository can contain code in other programming languages as well. The results of mining contain 172 JSON files. These files contain data on features identified for a particular triple. Each file includes a set of pairs `(repository : [found_features])`, where `[found_features]` is a list of features of the particular architecture style which were found in the `repository`.

Here is the example of such a pair: … `"EmbeddedSocial-Android-SDK": ["NONE", "getName\_feature", "getClass\_feature"],` …

Every string includes a constant indicating if the related repository's processing was finished. It was used for repository processing and did not have any special meaning. Some of the lines may contain constant indicating that the mining process was stopped. This happened when a repository weighted too much to be processed by 10 000 requests of the crawler. In these cases, a GitHub API token reaches its' limit. This case was not frequent. In particular, 2162 pairs out of a total 27107 contain these stops.

The full dataset is available on the project web page.

5.2 Data Analysis and Discussion

Summarized results derived from the dataset are presented in this section. The following tables show these results. Let us consider and discuss the popularity of particular styles. Note that open repositories of Apple company contain no source code in Java.

5.2.1 Model-View-Controller (MVC) architecture

In Table 2 one can see that MVC is used in approximately 25% of Microsoft, IBM, and Apache Java repositories. In Java repositories, the MVC style is mostly represented by the usage of the Spring framework that is very popular, especially in Apache Foundation projects.

Table 2. MVC style usage frequency (Java repositories)

	Microsoft	**IBM**	**Google**	**Awslabs**	**Aws**	**Apache**	**Amzn**	**Adobe**	**18F**
Processed	118	135	205	75	28	1044	18	53	6
MVC	29	28	4	7	6	274	0	4	0
Spring	29	27	4	1	4	237	0	4	0
Free Marker	0	1	0	0	4	48	0	0	0
Apache Struts	0	0	1	1	0	29	0	0	0

MVC is used in slightly less than 10% of Microsoft, IBM, Google, AWSlabs Python repositories (see Table 3). In Python repositories, MVC style is mostly represented by Django framework. Thus, web development is responsible for a significant fraction of usage cases in Python community. It is also interesting that Python is relatively more popular in open projects of commercial companies, whereas Apache Foundation is the leader in the development of Java projects.

Table 2. MVC style usage frequency (Pithon repositories)

	Microsoft	**IBM**	**Google**	**Awslabs**	**Aws**	**Apple**	**Apache**	**Amzn**	**Adobe**	**18F**
Processed	295	279	337	159	51	20	68	15	26	127
MVC	29	28	4	7	6		274	0	4	0
Django	19	10	20	4	2	1	5	1	1	42
Giotto	7	4	12	9	6	1	4	1	2	8
CherryPy	2	1	2	1	0	0	2	0	0	1
Turbo Gears	0	0	0	0	1	0	3	0	1	0

This style is the second most popular} from all styles in our style set. It is a significantly more popular style than others. This conclusion agrees with the survey results shown in fig. 6.

5.2.2 Main and sub-programs

Let us consider fig. 7 and 8. Each of these two figures shows two intersecting disks. The left one shows the number of repositories with «main» function. The right one shows a number of repositories without the usage of constructors. Thus, repositories that satisfy our criteria lie in the intersection.

It is easy to see no more than 1 repository of such type in Java. It is not unexpected because Java is a pure object-oriented language. So, any Java program contains objects or classes.

On the other hand, there are about 7% of all Python repositories (59) in which this procedural style was applied.

In general, we can conclude that *this style is not very* popular among open-source repositories from our dataset.

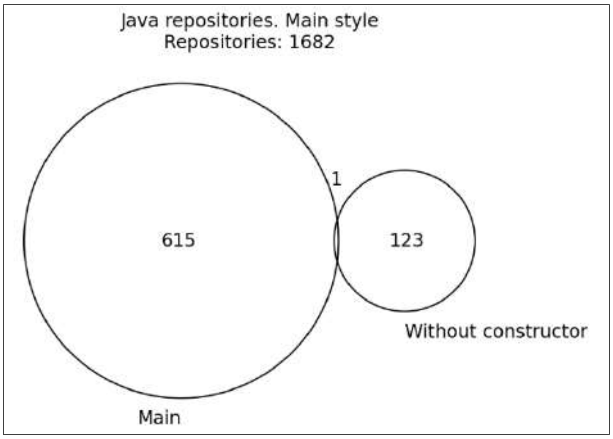


Fig. 7. Main and sub-programs style (Java repositories)

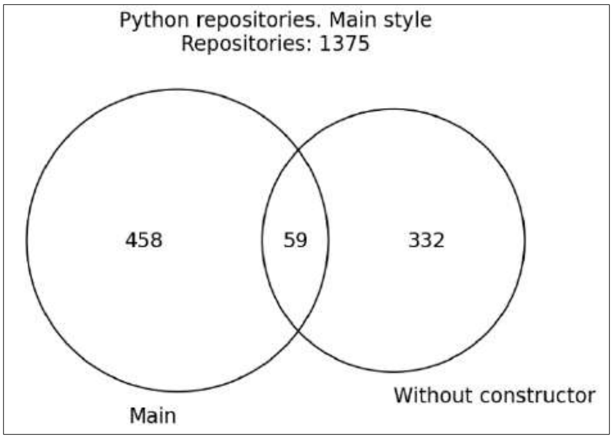


Fig. 7. Main and sub-programs style (Python repositories)

5.2.3 Event-driven software architecture

According to Tables 4 and 5 Event-driven architecture style is used in approximately 9% of IBM and Apache Java repositories. Curiously, event-driven style is applied in more than 50% of AWS Python repositories and approximately 20% of Apache Python repositories. Such applications are related to distributed and asynchronous software for web applications. Other companies tend to apply event-driven style in less than 1% of their Java and Python repositories. Event-driven architectures are mostly represented by the usage of Kafka framework and Amazon Active MQ framework in both Java and Python repositories.

Table 4. Event-driven style usage frequency (Java repositories)

	Microsoft	**IBM**	**Google**	**Awslabs**	**Aws**	**Apache**	**Amzn**	**Adobe**	**18F**

Processed	118	135	205	75	28	1044	18	53	6
Event-driven	3	10	0	0	0	88	0	2	0
Kafka	3	9	0	0	0	41	0	2	0
Apache Qpid	1	0	0	0	0	10	0	0	0
RabbitMQ	0	1	0	0	0	6	0	0	0
Amazon ActiveMQ	0	0	0	0	0	41	0	0	0
Apache RocketMQ	0	0	0	0	0	7	0	0	0
Zero MQ	0	0	0	0	0	2	0	0	0

Table 5. Event-driven style usage frequency (Python repositories)

	Microsoft	**IBM**	**Google**	**Awslabs**	**Aws**	**Apple**	**Apache**	**Amzn**	**Adobe**	**18F**
Processed	295	279	337	159	51	20	68	15	26	127
Event-driven	9	11	5	0	32	9	13	1	3	6
Kafka	0	5	0	0	0	0	3	0	0	0
Apache Qpid	0	0	0	0	0	0	3	0	0	0
RabbitMQ	0	1	0	0	0	0	3	0	0	0
Amazon ActiveMQ	8	5	5	0	32	0	6	1	3	6
Apache RocketMQ	0	0	0	0	0	0	1	0	0	0
Zero MQ	1	0	0	0	0	0	1	0	0	0

We can conclude that usage of *event-driven architectures hugely variates* from company to company and *relatively popular in projects* of AWS and Apache whose business is mostly *web-based and large-scale oriented*. Thus, these companies invest in scalable web applications and infrastructure code. Other companies concentrate more on desktop, mobile, and web applications without such need in scaling and asynchronous code.

5.2.4 Machine-learning-based software

The first general finding is that Java is not used commonly to develop machine-learning-based software. According to Table 6 we found smells of machine-learning-based style only in 16 Java repositories. On the other hand (see Table 7), this style is often used in Python repositories by various companies: Microsoft (61%), IBM (52%), Google (38%). ML source code is mostly represented by the usage of Numpy, Pandas, Matplotlib, and libraries for neural networks.

Table 6. Other architecture styles usage frequency (Java repositories)

	Microsoft	**IBM**	**Google**	**Awslabs**	**Aws**	**Apache**	**Amzn**	**Adobe**	**18F**
Processed	118	135	205	75	28	1044	18	53	6
ML-based	1	1	0	3	0	11	0	0	0
Data-centric	21	20	34	15	9	228	1	3	1
Cloud-based	0	0	0	0	25	0	0	0	0
Container	0	0	0	0	0	0	0	0	0
Aspect-oriented	0	0	0	0	0	0	0	0	0

Reactive-based	0	0	0	0	1	0	0	0	0
Expert system	0	0	0	0	1	13	0	0	0

We can conclude that *machine-learning applications are very popular in Python ecosystem*. Most Python repositories of companies contain smells of ML. Moreover, we can conclude that *machine-learning software is the most popular* software style (with respect to a total number of repositories with this style) according to our data.

Table 7. Other architecture styles usage frequency (Python repositories)

	Microsoft	**IBM**	**Google**	**Awslabs**	**Aws**	**Apple**	**Apache**	**Amzn**	**Adobe**	**18F**
Processed	295	279	337	159	51	20	68	15	26	127
ML-based	180	146	129	45	24	13	13	8	12	25
Data-centric	47	25	29	10	3	2	22	0	2	24
Cloud-based	3	1	42	1	0	0	2	0	1	2
Container	1	0	0	0	0	1	1	0	0	0
Aspect-oriented	0	0	0	0	0	0	0	0	0	0
Reactive-based	0	0	0	0	0	0	0	0	0	0
Expert system	0	0	0	0	0	0	0	0	0	0

5.2.5 Data-centric software architecture

We found smells of data-centric style in 15%–25% of Java repositories (Microsoft, IBM, Google, Apache, see Table 6) and in 8%–20% of Python repositories (Microsoft, IBM, Google, 18F, see Table 7). In Java repositories, data-centric software style is mostly represented by PostgreSQL and MySQL libraries' usage. This style is represented by the usage of the SQLAlchemy library in Python repositories.

We can conclude that *this style is the third most popular* of all styles thanks to Apache Foundation with more than two hundred such projects. Other companies apply the style as well.

5.2.6 Reflection-using software

This style was detected using several language features that indicate reflection appliances in a source code. Many repositories contain at least one of the features of a reflective code. However, we believe that the code with such an ephemeral smell can be called reflection-using. However, what should be the number of reflective features in code to call it reflection-using software. It is not that easy to define the concrete number. Thus, we decided to show the summarized data in this paper. A better definition of this architecture style will be a subject for future work.

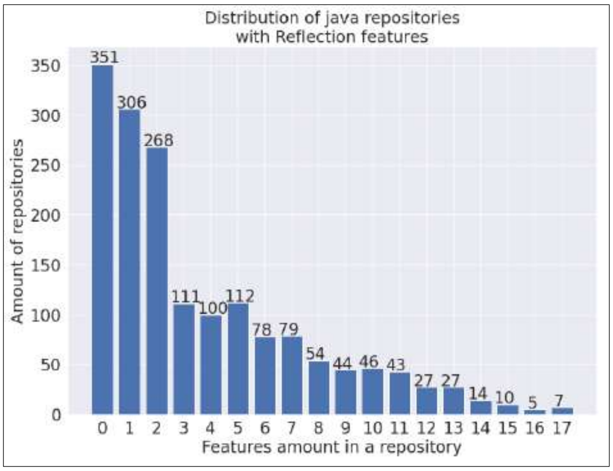


Fig. 9. Reflective code features in Java repositories

Fig. 9 shows the results for Java repositories, whereas fig. 10 considers Python repositories. In both cases, one can see that about 20% of repositories contain no reflective code smells. So, we can conclude that about 80% of Java and Python repositories have at least one feature of Reflection-using software.

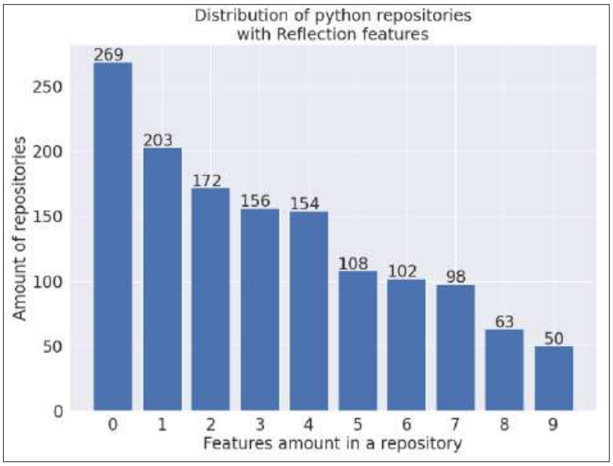


Fig. 9. Reflective code features in Python repositories

Our conclusion is that *most of the open-source software in our dataset contains some reflective code features*. Our definition for this style is too vague and has to be refined.

5.2.7 All other styles: cloud-service-based, aspect-oriented, reactive-based software, expert systems, software with containerization

It is clearly seen in tables 6 and 7 that smells of all other software architecture styles are very uncommon in our dataset.

Cloud-service-based style tends to appear in AWS Java repositories. This can be explained by the usage of AWS's own library for cloud development. Also, the Cloud-service-based style was found in Google Python repositories. It can be explained by the usage of Google's library for cloud development. These two cases are outliers, and overall we did not find out Cloud-service-based style as a popular one.

The same is true for other styles. We did not find almost any usage of these styles with the proposed features. Thus, we can conclude *all these software architecture styles are unpopular* in open-source repositories of our dataset.

This can be due to at least two reasons: either these styles are uncommon in open-source software, or we use flawed features. Both reasons are possible. The future work will be to elaborate on this issue.

5.3 Additional Results Validation

To verify the proposed feature model, we decided to ask developers of repositories, which we have processed, about the usage of the 11 architecture styles in their repositories. We extracted developers' emails from each repository we have processed. There were about 10 thousand repositories. Then we used Python code to send every one of them a letter with a link to the particular survey based on Google Forms. This form asked developers to specify what of our 11 software architecture styles they used in their repository. We got 69 replies to our Google Form. Out of these replies we extracted information on 78 repositories that we have previously processed.

One can easily see that we have no significant number of answers here. Thus, the following can not be considered as an extensive validation. However, we believe these results still can be of interest to the reader.

On every architecture style out of 11 we counted 4 metrics: accuracy, precision, recall, and F1-score. We considered developers' answers from the form as correct data and our answers as predictions. Among the repositories that authors answered our survey, there was no that used the following styles: Main and sub-programs, Expert system, or software with containerization.

Table 8 shows the results. According to the table, the best F1-score was reached for Reflection-using software (0.58) and Data-centric software (0.45). Recall overall was less than 30\% with such exceptions as Reflection-using software (0.64), Model-View-Controller (0.39), and Data-centric software (0.39). The highest precision was achieved for Event-driven software (0.75).

Table 8. Additional results validation

	MVC	**Main**	**ML-based**	**Event -driv.**	**Reflect.**	**DB-centr.**
Accuracy	0.65	0.92	0.79	0.51	0.58	0.72
Precision	0.41	—	0.44	0.75	0.53	0.53
Recall	0.39	0	0.27	0.14	0.64	0.39
F1	0.4	—	0.33	0.24	0.58	0.45
	Expert Sys.	**Cloud-based**	**Container**	**Aspec t**	**React.-based**	
Accuracy	0.88	0.56	0.62	0.76	0.68	
Precision	—	0.5	—	1	0.67	
Recall	0	0.09	0	0.14	0.08	
F1	—	0.15	—	0.25	0.14	

The results are different for various architectural styles. We can conclude the following.

Features for Main and sub-programs style and Expert systems could not be validated because among the repositories from the validation survey, there was no use of these two styles. Features for software with containerization style are not full. Using our features, we did not find it in any of the repositories in which the style was used according to their developers. Features for Reflection-using software and Data-centric software styles have not been enough for perfect identification, but they showed appropriate F1-score results. Features for Model-View-Controller, Machine-learning-based, Event-driven, Cloud-service-based, Aspect-based, and Reactive-based software styles show bad performance, mostly because of low recall. This means that our features have not fully covered the usage of these styles, and further investigation is needed.

5.4 Conclusions

Most of the results obtained by our automatic crawlers agree with the survey results, which are shown in fig. 6. Less-known styles are less common in open-source repositories; well-known styles can be found in many more repositories.

An outlier here is *software with containerization* style. Feature for this style seems ill-designed because many people are accounted for it, whereas we can not detect it in source code.

Both an automated analysis and a survey indicate aspect-based software and expert systems as the *least popular* architecture styles.

However, the additional validating survey (with a small number of answers) indicated that our features for some architecture styles show lousy performance. Thus, additional work is needed to improve the style and feature sets.

6. Related Work

We consider two large fields as related to our research. These fields are *software architecture research* and *software repository mining*. Whereas the former field is relatively old in terms of software engineering time scale, the latter is relatively young and fast-growing. We will try to observe both fields in this section.

Sharma, Kumar, and Agarwal [8] listed 23 software architecture styles in 6 categories due to the application type. This paper can be considered as a starting point to discuss architecture styles. The authors have chosen some styles (what styles?) out of all mentioned and gave short descriptions to them. However, there cannot be observed any code features of any style which can be used to identify it in a real project. The paper also leaves without attention statistical aspects of architecture style popularity in practice.

Automated software architecture recovery is related as well. Researchers in this field aim at constructing models of architecture decisions of existing software using data analysis and other automated techniques [9].

In software repository mining papers on code smell detection are close related to our project. Fontana et al. [10] concentrated on code smells and a machine learning-based approach to code smells detection. The authors collected a dataset of heterogeneous systems and a set of tools for detecting code smells and trained different machine learning algorithms with default parameters. Boussaa et al. [11] introduced code smell detection based on genetic algorithms that are called the competitive-co-evolution-based method. The method's idea is to generate two data samples: a sample of code smells and a sample of solutions. The aim of code smells generation is to escape from search methods, and the solution aims to cover more code smells. These works do not pay attention to software architecture styles, but their general approach seems attractive.

A repository mining method has been applied to reveal how software architecture evolves with time [12]. Code mining can help to evaluate software architecture as well [13]. Kouroshfar et al. \[14] applied automated architecture recovery techniques to show how the erosion of software architecture decisions influences software evolution.

There is a massive corpus of literature on software architectural smells and their automated detection. Architectural smells are signs of bad practices in the software design process, similar to code smells. The difference is that architectural smells are related to the level of general design decisions, whereas code smells are related to anti-patterns and bad practices on the level of software code. Fontana et al. [15] investigated how these two types of smell are interrelated. Previously, many automated tools have been developed to detect or predict architectural smells [16-20]. Azadi et al. [21] even proposed a catalog of such smells which different tools can detect. Features of software architecture styles that we consider in this paper are similar to smells. However, our features do not sign *bad* practices or anti-patterns. Contrariwise, our features indicate the presence of an architectural style.

Note that surveys are considered a good research tool in empirical software engineering. For example, Palomba et al. [22] used surveying to understand how developers feel a relationship between code and community smells. In our research, we apply surveys as well.

Recently, repository mining has been used to explore software in an empirical study on what software project artifacts are [23]. Not surprisingly, software projects consist of code but also of documentation, data, and many more different artifacts. Our research is similar in the sense of intentions. We seek for better understanding of the current field of software development.

6. Conclusions and Further work

In this paper, we measured industrial software developers' attitudes to the concept of software architecture style. We also investigated the popularity of eleven concrete architecture styles.

We found that the notion of software architecture style is not just a concept of academics at universities. Programmers apply this concept in their work. Moreover, industrial software developers consider the concept as improving their professional skill-set.

We formulated features for eleven concrete software architecture styles and developed crawlers based on these features. The results of repository mining using the features show that the most popular styles among developers of open-source projects are machine-learning-based software, Model-View-Controller architecture, and Data-centric software architecture.

We additionally validated the results obtained by crawlers using a special developer survey.

This validation shows that features for some architecture styles are ill-defined and have to be improved.

This paper presents up-to-date results of our research project. We plan to continue the project to understand the concept of software architecture style better. Updates can be found at the project web page: https://pais.hse.ru/en/research/projects/softarchstyles.

The set of software architecture styles we used in the paper is not comprehensive. It is possible to modify and extend it based on this work's results. This will be one of the directions of our future work.

Besides, the dataset gathered by our crawlers is related to a limited set of open-source repositories related to large software communities and companies. It is possible that our results are somehow biased and overfitted to this particular dataset. So, additional research is needed based on wider datasets.

Particular software architecture styles are still not sufficiently well-defined. Some of them – like reflection-using software --- need better and clearer definitions to deal with them in a less vague manner. We believe it is possible to construct concise and rigorous definitions based on more profound empirical research results.

References

[1] P. C. Clements and M. Shaw. “The Golden Age of Software Architecture” revisited. IEEE Software, vol. 26, no. 4, 2009, pp. 70-72.

[2] M. Shaw and D. Garlan. Software architecture - perspectives on an emerging discipline. Prentice Hall, 1996, 264 p.

[3] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. Software Architecture - Foundations, Theory, and Practice. Wiley, 2010, 750 p.

[4] M. Richards and N. Ford. Fundamentals of Software Architecture: An Engineering Approach. O’Reilly, 2020, 432 p.

[5] M. Richards. Software architecture patterns. O’Reilly Media, 2015, 47 p.

[6] M. Kleppmann. Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. O'Reilly Media, 2017, 616 p.
[7] L. Atchison. Architecting for Scale: High Availability for Your Growing Applications. O'Reilly Media, 2016, 230 p.
[8] A. Sharma, M. Kumar, and S. Agarwal. A complete survey on software architectural styles and patterns. Procedia Computer Science, vol. 70, 2015, pp. 16-28.
[9] A. Shahbazian, Y. K. Lee et al. Recovering Architectural Design Decisions. In Proc. of the 2018 IEEE International Conference on Software Architecture (ICSA), 2018, pp. 95-104.
[10] F. A. Fontana, M. Zanoni et al. Code smell detection: Towards a machine learning-based approach. In Proc. of the 2013 IEEE International Conference on Software Maintenance, 2013, pp. 396-399.
[11] M. Boussaa, W. Kessentini et al. Competitive coevolutionary code-smells detection. Lecture Notes in Computer Science, vol. 8084, 2013, pp. 50–65.
[12] D. M. Le, P. Behnamghader et al. An empirical study of architectural change in open source software systems. In Proc. of the 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, 2015, pp. 235–245.
[13] L. Zhu, M. A. Babar, and D. R. Jeffery. Mining patterns to support software architecture evaluation. in WICSA. In Proc. of the Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004), 2004, pp. 25-36.
[14] E. Kouroshfar, M. Mirakhorliet al. A study on the role of software architecture in the evolution and quality of software. In Proc. of the 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, 2015, pp. 246-257.
[15] F.A. Fontana, V. Lenarduzzi et al. Are architectural smells independent from code smells? An empirical study. Journal of Systems and Software, vol. 154, 2019, pp. 139-156.
[16] F.A. Fontana, I. Pigazzini et al. Automatic detection of instability architectural smells. In Proc. of the 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2016, pp. 433–437.
[17] F.A. Fontana, I. Pigazzini et al. Arcan: A tool for architectural smells detection. In Proc. of the 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), 2017, pp. 282–285.
[18] A. Biaggi, F. A. Fontana, and R. Roveda. An architectural smells detection tool for C and C++ projects. In Proc. of the 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2018, pp. 417–420.
[19] U. Azadi, F. A. Fontana, and M. Zanoni. Machine learning based code smell detection through WekaNose. In Proc. of the 40th International Conference on Software Engineering: Companion Proceeedings, 2018, pp. 288–289.
[20] F. A. Fontana, P. Avgeriou et al. A study on architectural smells prediction. In Proc. of the 2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2019, pp. 333–337.
[21] U. Azadi, F. A. Fontana, and D. Taibi. Architectural smells detected by tools: a catalogue proposal. In Proc. of the 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), 2019, pp. 88–97.
[22] F. Palomba, D. A. Tamburri et al. How do community smells influence code smells? In Proc. of the 40th International Conference on Software Engineering: Companion Proceeedings, 2018, pp. 240–241.
[23] R. Pfeiffer. What constitutes software? An empirical, descriptive study of artifacts. In Proc. of the 17th International Conference on Mining Software Repositories, 2020, pp. 481–491.

Информация об авторах / Information about authors

Алексей Александрович МИЦЮК, кандидат компьютерных наук, доцент, старший научный сотрудник. Научные интересы: извлечение и анализ процессов, информационные системы, архитектура программного обеспечения, сети Петри

Alexey Alexandrovich MITSYUK, PhD in Computer Science, Associate Professor, Senior Research Fellow. Research interests: process mining, information systems, software architecture, Petri nets.

Николай Арсенович ЖАМГАРЯН, бакалавр программной инженерии, НИУ ВШЭ, студент магистратуры, университет Мичигана, ИТ аудитор, КПМГ СНГ. Научные интересы: наука о данных, обработка естественного языка, машинное зрение, машинное обучение, глубокое обучение.

Nikolay Arsenovich JAMGARYAN, Bachelor of Software Engineering, HSE, Master's student, University of Michigan, IT auditor, KPMG CIS. Research interests: data science, natural language processing, computer vision, machine learning, deep learning.

DOI: 10.15514/ISPRAS-2021-33(3)-2

Review of Static Analyzer Service Models

M.A. Menshikov, ORCID: 0000-0002-7169-7402 <info@menshikov.org>
Saint Petersburg State University,
7/9 Universitetskaya Emb., St Petersburg, 199034, Russia

Abstract. The static program analysis is gradually adopting advanced use cases, and integration with programming tools becomes more necessary than ever. However, each integration requires a different kind of functionality implemented within an analyzer. For example, continuous integration tools typically analyze projects from scratch, while doing the same for code querying is not efficient performance-wise. The code behind such use cases makes «service models», and it tends to differ significantly between them. In this paper, we analyze the models which might be used by the static analyzer to provide its services based on aspects of security, performance, long-term storage. All models are assigned to one of the groups: logical presence (where the actual computation is performed), resource acquisition, input/output, change accounting and historic data tracking. The usage recommendations, advantages and disadvantages are listed for each reviewed model. Input/output models are tested for actual network throughput. We also describe the model which might aggregate all these use cases. The model is partially evaluated within the work-in-progress static analyzer Equid, and the observations are presented.

Keywords: static analysis; integration; service model; review; classification

For citation: Menshikov M.A. Review of Static Analyzer Service Models. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 27-40. DOI: 10.15514/ISPRAS-2021-33(3)-2

Обзор моделей работы статических анализаторов

М.А. Меньшиков, ORCID: 0000-0002-7169-7402 <info@menshikov.org>
Санкт-Петербургский государственный университет,
Россия, 199034, Санкт-Петербург, Университетская наб., д. 7–9

Аннотация. Статический анализ программ постепенно осваивает продвинутые случаи использования, и плотная интеграция с инструментами программирования становится все более необходимой. Однако, каждая интеграция требует реализации особенной архитектуры или определенной функциональности в анализаторе. Например, инструменты для Continuous Integration обычно анализируют проекты с нуля, в то время, как тот же самый анализ с нуля малоэффективен для выполнения запросов по коду. Код, который реализует архитектуру для разных интеграций, составляет различные модели работы. В данной статье анализируются модели, которые могут использоваться статическими анализаторами, с точки зрения безопасности, производительности, долговременного хранения данных. Все модели отнесены к одной из групп на основе данных о логическом расположении вычислителя, способах получения ресурсов, методах организации ввода-вывода, а также возможностей по учету изменений и исторических данных. Описаны преимущества и недостатки моделей, приведены рекомендации по их использованию. Для моделей ввода-вывода также протестирована пропускная способность сети. Приводится модель, объединяющая все данные случаи использования. Она протестирована в разрабатываемом статическом анализаторе Equid, и в статье приведены наблюдения об особенностях её работы и реализации.

Ключевые слова: статический анализ; интеграция; модель работы; обзор; классификация

Для цитирования: Меньшиков М.А. Обзор моделей работы статических анализаторов. Труды ИСП РАН, том 33, вып. 3, 2021 г., стр. 27-40 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(3)–2.

1. Introduction

Static analyzers are widely used in the industry for different purposes: defect search, verification, linting, quality assurance, code refactoring [1]. Most of these use cases can be implemented via a standard sequential model. The more projects are created the more efforts are put into developing lifelong support tools. One example is clangd [2], the tool acting as a language server [3] providing syntax highlighting, code inspections and refactoring. We believe that analysis tools have the potential to be used by a larger audience comprising not only engineers but also architects, technical management, quality assurance staff. Partially this extended audience uses analyzers nowadays, but mostly to understand code quality, while analyzers may provide more kinds of information. Currently, static analyzers are either isolated or are running locally. That limits the possibilities of the analyzer. To become agnostic to the way the analyzer is called, tools have to adopt more user scenarios and service models.

One way to approach this issue is to research how are analyzers used and in which circumstances. Combined with the technical review, classification of these *service models* would show the positive and negative aspects of each model. The paradox is that each model is so interconnected with the underlying architecture that it is hard to judge which entity is primary and which is secondary. By reviewing service models, we review the analysis architectures as well. Working out a way to support all models contributes to developing a more unified analyzer structure, improving user experience [4], and, ultimately, may lead to wider adoption of static analysis tools.

The *goal* of this paper is to classify service models that can be used by static analyzers and analyze their positive and negative networking, performance and other aspects. The *novelty* is that these models are analyzed towards application to analysis tools concerning an extended set of parameters and are combined in one model.

This paper is organized as follows. In section 2, the literature is examined. In section [3], we review all models, including logical presence models (subsection 3.1), resource acquisition (subsection 3.2), input/output (subsection 3.3), change accounting (subsection 3.4) and historic data tracking models (subsection 3.5). The most widespread models are wrapped in section 4. Then, in section 5, we define what's required for service model agnostic static analyzers. Our model-agnostic static analyzer, as well as some of the models, are tested and discussed in section 6.

2. Related work

Most works in the static analysis field explore improvements that can be applied to the analysis algorithms. The effects of service models are not typically reviewed. Common software architectural patterns [5] and patterns for data-intensive applications [6] still apply to static analyzers.

As for classification, [7] bases taxonomy on rules, technology, supported languages, configurability, etc. This separation is developer-centric, while our research is focused on the technical effects of implementation. A different approach is explored in [8], in which authors introduce a notion of *development context* comprising *local programming*, *continuous integration* and *code review* contexts. We expand further on it by exploring the service model from an analyzer's point of view, such as when handling incremental input, performing time-limited operations for IDE, etc.

The research [4] focuses on finding an answer to the question why static analysis tools are not widely adopted. One of the concerns presented by authors is that tools don't integrate into existing development processes, which intersects with our implicit thesis that industry needs more sophisticated service models. The mentioned research [8] also confirms that developers tend to avoid using the same tools for different development contexts, which means that a single analysis tool might benefit from employing more service models.

3. Models

Any software may be used via different service models. In this research, we review models based on the influence on software cooperation. Namely, the physical location influences the distance between the analysis requester and the analysis executor. In modern networking [9], such a distance is logical rather than physical since server and client might reside in the same host, so we define such models as *logical presence* models.

The second question is how are resources needed for analysis, such as input sources, headers and libraries, are retrieved. These models form a group of *resource acquisition models*.

The third problem is the propagation of input parameters from the requester to the server and the delivery of results back. This is about *input/output* models.

The fourth question is the attitude of the model to incremental analysis: *change accounting* models.

The fifth issue is similar to incremental analysis: the handling of *historic data*, such as revisions in version control systems.

In the next subsections, all these model groups are reviewed.

3.1 Logical presence models

The first theoretical model is based on where the actual computation is done. As mentioned, the location of the analysis executor is mostly logical rather than physical in presence of network namespaces (containers) and virtual machines. All reviewed models are summarized in Table 1.

Table 1. Logical presence models & their properties

Model	**Security**	**Data leak risk**	**Stable connection**	**Network load**	**Environment**	**Performance**	**Score**
Local computation	Low (1)	High (1)	Unneeded (3)	None (3)	Preserved (3)	Low (1)	12
Isolated computation	High (3)	Low (3)	Unneeded (3)	High (1)	Not preserved (1)	High (3)	14
Remote computation	High (3)	Medium (2)	Required (1)	Medium (2)	Manageable (2)	High (3)	13

3.1.1 Local computation

The model is widely used in static analyzer projects. In that case, the static analyzer is located on the machine requesting the analysis. The examples are LLVM and Clang [10, Svace [11], cppcheck [12] and other tools.

- *Security*: by default, the analyzer has access to all the sources and has an access to the Internet, which lowers the security in general. Moreover, access to the most data located on the host is possible. Research like [13] also stresses that the employees of companies fail to comply with security regulations. In security-critical cases, it is important to limit available file system locations by tools such as AppArmor and SELinux [14], disable internet access for the application.
- *Networking*: unused except for loopback communication or inter-process communication, which imply no use of networking hardware.
- *Performance*: developer work stations tend to have limited resources, so performance & concurrent work is limited. The solution involving the use of server-grade performant work stations is not economically effective.
- *Long-term storage*: storing artifacts for a long time is not feasible on developer work stations, except for the case when network file systems, such as NFS [15] or SSHFS [16], are involved.

3.1.2 Isolated computation

The schema is used by modern Continuous Integration (CI) tools, such as Jenkins [17], SonarQube [18], Coverity [19]. The computation is moved to a designated server that has access to committed input sources.

- *Security*: CI has all the data required to constrain allowed file system locations, for example, via SELinux, AppArmor [14]. This schema can be achieved using containerization platforms like LXC [20[, Docker [21]. Even though containers have several weak points [22, 23] and setting them up correctly requires an understanding of parameters and a modern kernel, exploiting such errors is not easy. Going forward, a designated virtual machine without direct Internet access, built solely for the static analysis of one project, is the most secure solution.
- *Networking*: such systems typically create workspaces by downloading repositories from scratch, causing significant traffic flows. However, this operation mode is usually network hardware-friendly since, as a rule of thumb, such servers have good network adapters and are connected to central switches by wire, so they are close to the repository server.
- *Performance*: the raw power needed for computations is offloaded to a server, reducing the load on developer stations to zero. Incremental operation is usually impossible due to the way workspaces are prepared and discarded.
- *Long-term storage*: storing analysis artifacts is mandatory because users might need to check results later. This shouldn't have a significant influence on disk space (since such servers have designated storage, in general) and analysis runs sporadically.

3.1.3 Remote computation with resource acquisition

The model implies that the computation is done on a separate server, but resources are acquired from developer machines via various communication channels. Clangd [2] and other language servers [3] present tools that are not technically recognizable from static analyzers but provide a similar set of services. We present the model in [24], but in this research the model is evaluated from a non-architectural perspective.

The following characteristics are seen in this model:

- *Security*: derived from isolated computation model, but data leaks are possible on the way from a local machine to a server [13]. This can be solved by using secure communication with certificate pinning.
- *Networking*: the model in which the workspace is obtained from the user directly is inefficient in the case of large projects. For example, Linux 5.10.26[1] is 1GB (174MB in tar.gz format), which would take 80 seconds (14 seconds for compressed format) on a perfect 100 Mbps link. In the case of compressed format, it takes 6 seconds to unpack on Intel Core i7-7700HQ based laptop with Samsung 980 Pro SSD, Ubuntu 20.04. Compressing to this format takes 30 seconds on the same host. That means that, if the workspace is obtained from the user, the complete transmission time is 80 seconds or 30 + 14 = 44 seconds (considering the receiver a more advanced host with higher unpack performance). The link is, however, usually not perfect: for example, WiFi links are ailing from network congestion [25], decreasing available bandwidth even further. The viable option is collecting changes from the revision known to the static analysis host (this option is discussed in subsection 3.4).
- *Performance*: the computation is offloaded to the high performant server, with no load to developer stations. The incremental operation is possible in case snapshots of the internal state are stored by the static analysis host.
- *Long-term storage*: storing analysis artifacts is also mandatory, the influence on sparse runs is the same as for isolated computation, however, significant disk space might be consumed by

[1] https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.10.26.tar.gz

per-developer incremental runs. As a result, the recommendation is to prepare a mechanism for discarding old per-developer analysis results.

3.2 Resource acquisition models

All resource acquisition models are reviewed in Table 2.

Table 2. Resource acquisition models & their properties

Model	**Input length**	**R&D efforts**	**Preparatory work**	**Stable connection**	**Compiler compatibility**	**Score**
Local resources	Optimal (3)	Low (3)	None (3)	Unneeded (3)	Full (3)	15
Shared repository	Moderate (2)	Low (3)	None (3)	Unneeded (3)	Full (3)	14
Preprocessing	Large (1)	Low (3)	High (1)	Unneeded (3)	Absent (1)	9
Pre-tracing	Moderate (2)	Moderate (2)	High (1)	Unneeded (3)	Full (3)	11
Virtual file system	Optimal (3)	High (1)	Low (3)	Required (1)	Full (3)	11

3.2.1 Local resources. Shared repository

These two models just define the typical schemas used in software engineering. The local resource model is used in all tools running locally, such as compilers, static analyzers. The shared repository model is enforced by continuous integration environments.

3.2.2 Preprocessing

In this schema, the input is preprocessed locally and the analyzer gets a preprocessed version for further analysis.

- *Input size*: preprocessed definitions are very large. The file with only \path{<iostream>} header included and an empty main() is 49 bytes long, while the preprocessed version is 751954 bytes long (GCC 9.3.0 on Ubuntu 20.04).
- *Analysis*: the problem might be the preprocessor's output is not compatible between the requester and analyzer hosts. This is better seen if source and target hosts have different operating systems and toolchains.

If it is clear that the compiler used on both localhost and analysis host matches or at least is compatible, and the analysis runs on one input file at most, then this schema might be a simple and cost-effective solution for the implementation of remote analysis.

3.2.3 Pre-tracing of dependencies

The core idea is to perform tracing of all needed files before sending an analysis request. This process can be not straightforward. Tools like Build EAR [26] intercept commands passed to compilers, but don't provide lists of all needed files. This tool can be used in conjunction with utilities tracing system calls to get this information (such as strace[2]).

- *Input size*: reasonable since it includes only needed files.
- *Analysis*: requires integration of virtual file system with pre-downloaded files into parsing stage.

This model is similar to preprocessing, however, files are packed into request individually. This schema is less problematic than preprocessing because files are not present in the request twice or more times, reducing the cost of networking transfer.

[2] https://github.com/strace/strace

3.2.4 Virtual network file system

A virtual network file system is a technique that can be used to acquire resources from a source host on-demand. It can be used through the well-known implementation such as NFS [15] and SSHFS [16], or via a custom protocol. This schema has the following properties:

- *Input size*: optimal because taken on demand (in case files are cached on a server).
- *Analysis*: requires integration with the parsing stage. This model reduces the cost of analysis in case of early termination which may occur if an input has obvious syntax defects.
- *Networking*: needs a stable connection between a local host and an analysis host. It can be problematic considering that a significant part of hosts is behind Network Address Translation (NAT) [9] gateways and thus doesn't have a fixed IP. In such systems, the hosts need to use keep-alive techniques to avoid early preemption of entries in gateway NAT tables. Also, the use of well-known implementation may exhibit the problem of passing traffic through in case the static analysis client is behind NAT or a firewall and the implementation uses the pipe in the direction from server to client.

3.3 Input/output models

All input/output models are reviewed in Table 3.

Table 3. Input/output models & their properties

Model	**R&D efforts**	**Stable connection**	**Network load**	**Notifications**	**Score**
CLI	Trivial (3)	Unneeded (3)	None (3)	Unneeded (3)	12
Stateless client/server	Trivial (3)	Unneeded (3)	High (1)	Impossible (1)	8
Stateful client/server	Moderate (2)	Required (1)	Low (3)	Possible (3)	9
Streaming model	High (1)	Required (1)	Low (3)	Possible (3)	7

3.3.1 Command line interface model

This model is widespread in the industry. The input is provided with input arguments and input stream, the output – with the result code and stdout/stderr stream.

3.3.2 A stateless client/server model

The input is the request to the server, the output is a response to the request.

- *Networking*: this model implies that after the request is sent, the response must follow after analysis is done, not necessarily to the same request (might be a status request).

  The problem with this model is that notifications need a side-by-side implementation (i.e. a communication channel directed towards the client). Without notifications, the status polling is redundant, but not harmful due to small absolute packet sizes.

  A significant performance issue in real conditions may occur if a large amount of input data is sent over short-living TCP sessions. The reason is that most home-grade gateways accelerate network traffic only if the session reaches a specific number of packets (e.g., 5). Shorter sessions may appear unaccelerated and may be processed via CPU, not reaching a maximum practically performance (in the author's experiments with 1gbit links, the accelerated performance tops at 940 Mbps, while unaccelerated traffic reaches 50 Mbps, at most).
- *Practical aspects*: the approach can be implemented within the REST paradigm, which has many available implementations for any platform.

Practically, this limits the usage of the model to short requests. That's the reason the model is used within Continuous Integration systems, data management cases (such as the configuration of services like Jenkins, GitLab; manipulation of objects in bug trackers, etc). In other cases (e.g. compiler support case), this paradigm is not efficient.

3.3.3 A stateful client/server model with or without notifications

The input is a series of requests to the server, the output is a series of responses from the server.

- *Networking*: this model is efficient regarding networking hardware in the case of long-term TCP sessions. Most traffic will be accelerated, so the maximum performance will be demonstrated.
- *Practical aspects*: the model requires a custom state machine, notification system. The development cost is higher.

3.3.4 A streaming model

This is a variation of the client/server model, so the throughput is nearly the same. The input seen on the server is dynamically formed by requests, the computation is performed for currently known data.

3.4 Change accounting models

3.4.1 Fixed revision

The analyzer pulls the specific version of a source. If it is needed to re-analyze some part of the code, the complete analysis is performed.

- *Time*: complete execution every time.
- *Analysis*: requires no special handling from the analyzer's side.
- *Networking*: download of the complete repository might take significant time, however, this process is unconditionally networking hardware-friendly.

This schema is suitable for Continuous Integration processes, but long analysis time blocks the interactive use cases.

3.4.2 Incremental updates

The analyzer builds the model of a program on the first run. If the user decides to reanalyze a file or two, changes are obtained incrementally.

- *Time*: slow once, fast on incremental updates. However, in the case of global changes, the analysis time might increase dramatically, reaching the complete time or even overcoming it due to preliminary dependency graph analysis.
- *Analysis*: puts additional requirements, such as discardable state that is trivial to invalidate when a part of dependency graph changes. Dependent parts of the state should be rewritable.
- *Networking*: the difference between projects typically has negligible size compared to complete repository, so the process of obtaining differences is networking hardware-friendly, especially in the case of one TCP session or the same UDP source/destination addresses and ports.

3.4.3 Daily updated global revision with incremental user-defined changes

A typical use case would be that the analyzer runs every night on the latest revision, but if the user requests the analysis of a diverging source, the «latest» revision is forked and only differences are reanalyzed.

- *Time*: this schema improves analysis performance for developers running the analysis on a large codebase with minor differences.

- *Analysis*: the incremental schema requirements plus scheduling of daily updates, temporary storage of analysis artifacts.
- *Networking*: developers typically don't change the large codebase significantly. Because of that, the difference is ought to be minor, and the network load is the same as for the incremental schema.

If the analysis state is transferrable, the developers might cache the state and run the analysis locally. This is possible for some analysis kinds, such as code queries, dependency analysis.

3.5 Historic data tracking models

Some analyzers might take advantage of historic data. In addition to the usual code metrics changing over time, the practically useful case would be to narrow down a revision with a specific defect not tracked by analyzers (i.e. logical mistake)

3.5.1 A model without tracking of historic data

- *Analysis*: trivial to implement compared to a model with tracking.
- *Data storage*: only needs one specific revision, no extra data is needed.

3.5.2 A model with complete snapshots of historic analysis data

- *Analysis*: requires meta run of analysis over two or more revisions, which complicates the structure of analysis.
- *Data storage*: the analysis data for all revisions in question should be collected.

3.5.3 A model with differential snapshots of historic data

- *Analysis*: more complicated compared to the model with complete snapshots, additional invalidation of data is needed. That also requires maintenance of algorithms for propagating analysis data differences, which may make the complete task difficult.
- *Data storage*: analysis can be done once, and then only analysis database differences can be stored.

Model without tracking is trivial to implement. Models with snapshots may support use cases in which historic data is important, but it comes with a cost of extra time, data storage (high in the model with complete snapshots) and development complexity (high in the model with differential snapshots).

4. Combination shortcuts

After review of basic models, it is obvious that their combinations are already used worldwide:

1) **Local (incremental) model** – local computation, local resources, command line or server model with a fixed revision (incremental updates) and no tracking of historic data.
2) **Continuous Integration model** – remote computation, source repository, stateless client/server model with custom notifications, fixed revision, no tracking of historic data.

5. Considerations for service model agnostic static analyzers

Considering suggested use cases, it is possible to form suggestions on what should be done in a static analyzer to support more these models (fig. 1)

Logical presence models and input/output models are tightly coupled. A service model agnostic analyzer should have an abstraction layer for the complete execution – the job subsystem.

Resource acquisition methods imply that there must a separate abstraction layer for retrieving file data from different hosts.

Incremental change support implies that objects must be addressable in a unified and interchangeable manner, so that older object versions might be discarded, while new versions added as-is. This should be done right after retrieving data and remote resources.

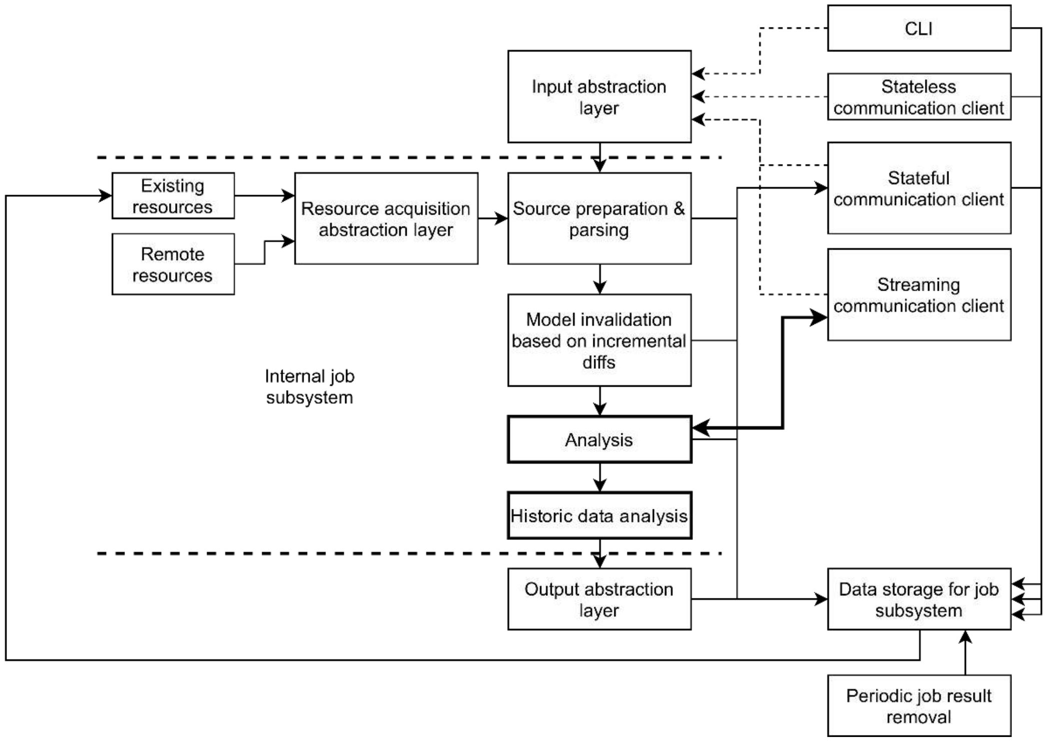

CLI
Input abstraction layer
Stateless communication client
Stateful communication client
Existing resources
Resource acquisition abstraction layer
Source preparation & parsing
Remote resources
Streaming communication client
Model invalidation based on incremental diffs
Internal job subsystem
Analysis
Historic data analysis
Output abstraction layer
Data storage for job subsystem
Periodic job result removal

Fig. 1. Possible schema for model agnostic static analyzer

To facilitate status polling, incremental change handling and historic data tracking, the output should be saved to data storage, accessible for extended periods. Historic data tracking also implies having a subsystem of meta-analysis, which allows reviewing deltas between revisions.

6. Testing and discussion

6.1 Characteristic-based evaluation

The characteristic-based evaluation of models was performed in tables 1, 2 and 3. For each characteristic, a numeric value ranging from 1 (worst) to 3 (best) had been chosen. The total score for each model is written in the column «Score». This evaluation is partially subjective but had been discussed with a few experts in relevant domains.

The results are as follows. The best model among logical presence models is an isolated computation, which is confirmed by its popularity in the software engineering industry. The second model is a remote computation with resource acquisition. It combines the high performance of isolated computation with manageable customization to comply with the environment and use cases. The third model is a local computation. The problems of this model lie in practically low performance and high data leak risk (the developer machine is likely to be insecure). However, if this risk is diminished by using a secure operating system and working firewall rules, this model would share the score with remote computation.

Among resource acquisition models, classical local and shared repository models are the best. When considering models for non-classical use cases, pre-tracing of dependencies and virtual file system are better choices than preprocessing method. But, in practice, preparatory work for tracing might be time-consuming, making the provision of thin clients hard. So, in the author's opinion, the virtual file system is the preferred choice for non-classical use cases.

For input/output models, the choice is related to the use case even more than for previous models. However, when choosing among networking models, stateful client/server communication is preferred as it reduces network load, provides notifications (reducing polling) while keeping R\&D efforts moderate.

The preference between change accounting models is unambiguous. The incremental model supports more use cases, and at the same time, the daily updated revision enhances it with much better performance in common developer routines.

Historic data tracking stays a little apart from this comparison. The more data is processed, the more time is taken and the more useful data is carved, therefore it is hard to name the preferred model.

6.2 Evaluation in static analyzer project

The considerations for service model agnostic analyzers were used as a basis for our project – Equid static analyzer [27]. We emphasize that the project is not following the schema in all ways since there is a lag between design and actual implementation. Our implementation includes a frontend library – the part that manages jobs for a specific workspace. The frontend library is used by the command line interface and server binary, both of them construct the workspace and fill it with job types, paths and environment information. The job types define the semantic visitors that are invoked at the end of analysis stages, during meta run, and have an impact on the selection of verification rules. The frontend library starts the analysis and provides an interface to get the current status or stop execution if needed. After finishing all jobs related to a specific run, the user might obtain the result of the analysis in all requested forms. The supported forms are defects, dependency analysis, call graphs, language identification.

The incremental analysis model lags behind the design. The support of incremental analysis is built into an object database, and it is possible to discard old objects and then drop new objects in. There is a saved dependency graph that can be used to invalidate parts of the analysis run. However, the incremental analysis support is not finished yet and we can only experiment with it. In our testing, if the incrementally changed file makes 10\% of input size, then the time to recompute it will match 20.07% (on average) of time taken for the whole input due to the need to invalidate the map. In case of excessive dependencies between updated and untouched files, the computation might take up to 40% of the original analysis time, although it is possible to design a case that will invalidate the complete program model.

The supported mediums are JSONRPC[3] and binary streaming over TCP with TLS enabled. These mechanisms are implemented in a straightforward manner and are adequate considering networking and security requirements.

During the evaluation, we have found that the optimal model effectively falls back to trivial software architecture if some functionality is not needed. When they are needed, extra stages get enabled and start adding expected diagnostic data to reports. That is the reason why it is possible to experiment with unfinished functionality in Equid's architecture. This is an advantage of the model.

The other advantage is a clear decomposition between *the core* and *the service*. The analysis functionality is a black box for the service. The service part provides input arguments, takes notifications provided by the core, passes streaming data to the analyzer and reuses the output as many times as needed, however, those are only extension points available. As seen in the schema, the main part remains sequential, therefore, still simple for development.

There are certain problems. While the *simple* design matches the *complex* architecture, imminent conditional jumps still make performance penalties. Also, it is harder to maintain the support of these service models, though this issue may be neglected by keeping the core as minimal as possible.

[3] https://www.jsonrpc.org

6.3 Network performance

As for network performance numbers, we performed testing of:

- Stateless polling versus notification model. In the case of using exponential backoff variation (5, 10, 20, 40, 80, 160 seconds at most), there are around 294 bytes per request and 210 bytes per response (Table 4). In the case of notifications (Table 5), a response is around 140 bytes and keepalive packets are around 70 bytes (Analysis start/destroy is not considered for the case of polling, TCP session instantiation/finalization is not considered for notifications). The time difference is large between polling and keepalive models, but in absolute numbers, these differences don't impact allocated bandwidth significantly and thus might be ignored.

Table 4. Data transfers with polling

Total time (sec)	**Start (sec)**	**End (sec)**	**Steps**	**Delta (sec)**	**Data transmitted (bytes)**	**Data received (bytes)**
630	5	160	8	5	2352	1680
95	5	160	5	60	1460	1046
13080	5	160	86	35	25542	18232

Table 5. Continuous data transfers

Total time (sec)	**Keepalive packets (pkts)**	**Data transmitted (bytes)**	**Data received (bytes)**
634	10	700	140
92	1	70	140
13189	219	15330	140

- Data transfers over WiFi (Table 6). A dual-band home gateway based on MediaTek platform with IEEE802.11n and IEEE802.11ac bands was used for testing. The test server is connected to the gateway over the 2.4GHz band (actual frequency is 2.412GHz), the client is connected to the gateway over the 5GHz band (actual frequency is 5.3GHz). For single-thread TCP performance, the data has been sent in the biggest possible packets according to MTU/MRU in the network. For SSHFS, the data has been sent file by file. The actual performance numbers demonstrate that the preprocessing schema is, indeed, slower due to higher input size. The difference between single-thread TCP with raw input is around 18.57 Mbps (21.6% of raw TCP performance), however, this difference may be either judged by the simpler implementation of SSHFS. On the other hand, a possible reduction of input based on the existence of files on the server not only in one user's sandbox might have a positive impact on the performance of custom protocols based on TCP. At the same time, the local model has zero penalties on file transfers and this result cannot be surpassed.

Table 6. Source code transfers

Approach	**Total time (sec)**	**Input length (MB)**	**Links**	**Throughput (Mbps)**
Single-thread TCP (raw input)	9.85	101	5.3GHz → Gateway → 2.412GHz	82
Single-thread TCP (preprocessed input)	48.75	470.66	5.3GHz → Gateway → 2.412GHz	77.2
SSHFS (raw input)	12.054	101	5.3GHz → Gateway → 2.412GHz	67.03

6.4 Limitations of the approaches and further development

The proposed schema of the service model agnostic analyzer aggregates models in a straightforward manner. The problem with it is that it is not optimized as there was no research on the most optimal model. In our view, an improvement can be achieved if some numerical quality measure for service model combinations is proposed.

The problem with the comparison of models is that it is biased towards implementation. The most widespread cases were carefully chosen, such as source code transfer evaluation or polling versus continuous data transfer testing, however, actual implementation may work around negative aspects shown in the paper. That may happen since analyzers cannot be seen as pure implementations of these discrete models. Combining models for reaching the best quality of output model is encouraged, even if complete aggregation is not in question.

Also, as the research's goal is to study common models and their generalizations, it is often the case that a widespread example of the specific model does not exist, and we have no resources to implement all of them in the analyzer with sufficient detail level. That limits model reviewing possibilities. A further improvement would be achieved after developing such examples (toy analyzers) and verifying them on many samples.

6.5 Suggested use cases

These models may work on different occasions. Based on the review of models, we propose the following mapping from use cases:

- **Complete project and inter-project analysis**: based on the advantages of isolation, the continuous integration model seems a better choice.
- **Basic reference search, refactoring**: since these use cases don't imply deep project inspection [28], a local (incremental) model should be optimal.
- **Code queries** [29]. Depends on the size of a project: small projects might be analyzed locally in a separate instance of the analyzer. Big projects with a distributed team mostly sharing the same source may take advantage of remote computation with a virtual file system, a stateful client/server model, a daily updated global revision with incremental changes model and historic data tracking.
- **Project import & dependency analysis**. Depending on the requirements such as the location of the project and its size, the preferred model might range from a simple *local model* to a remote computation (with or without a virtual file system), source repository and a fixed revision model.
- **Debugger support** – analyzer supports debugger with code insights (e.g., similar model is seen in [29]). The local model is sufficient for small projects, but large projects should be analyzed within the remote computation, virtual file system, daily updated global revision and incremental updates model.
- **Compiler supporting model**. In that case, the compiler does code generation, but the analyzer supports it with additional inferred contract checks, the information about clearly unsatisfied assertions, et cetera. Local computation, local resources, streaming model, fixed revision.
- Static/dynamic analysis cooperation. Such cooperation is suggested by FSTEC [30] «Protection against unauthorized access to information» certification. For example, a dynamic analyzer might trace the execution to let the static analyzer verify that all traces are valid. It might be done in a remote execution model with a virtual file system, daily updated global revision with incremental updates.
- **Technical documentation preparation**. Also a part of FSTEC [30]. Usually, the process is done once at the end of a release cycle. Considering the importance of precision, Continuous Integration is the most efficient model.

6. Conclusion

The service models that can be used by static analyzers were described. This list includes logical presence, resource acquisition, input/output, change accounting and historic data handling models. An aggregate model enabling significantly diverging use cases is presented. It was tested in a real-world static analyzer and demonstrated technical advantages and disadvantages. Part of the models was compared directly by characteristics, and recommendations for model selection were provided.

Список литературы / References

[1] D. Binkley. Source code analysis: A road map. In Proc. of the Symposium on Future of Software Engineering (FOSE '07), 2007, pp. 104-119.

[2] What is clangd? Available at https://clangd:llvm:org, accessed 15.03.2021.

[3] Langserver.org - A community-driven source of knowledge for Language Server Protocol implementations. Available at: https://langserver:org, accessed 15.03.2021.

[4] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge. Why don't software developers use static analysis tools to find bugs? In Proc. of the 2013 International Conference on Software Engineering, , 2013, p. 672-681.

[5] M. Richards. Software architecture patterns. O'Reilly Media, 2015, 47 p.

[6] M. Kleppmann. Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. O'Reilly Media, 2017, 616 p.

[7] J. Novak, A. Krajnc et al. Taxonomy of static code analysis tools. In Proc. of the 33rd International Convention MIPRO, 2010, pp. 418-422.

[8] C. Vassallo, S. Panichella et al. How developers engage with static analysis tools in different contexts. Empirical Software Engineering, vol. 25, no. 2, 2020, pp. 1419-1457.

[9] A.S Tanenbaum and D.J Wetherall. Computer networks. Pearson, 5th edition, 2010, 960 p.

[10] C. Lattner and V. Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In Proc. of the International Symposium on Code Generation and Optimization, 2004, pp. 75-86.

[11] В.П. Иванников, А.А. Белеванцев и др. Статический анализатор Svace для поиска дефектов в исходном коде программ. Труды ИСП РАН, том 26, вып. 1, 2014 г, стр. 231-250. DOI: 10.15514/ISPRAS-2014-26(1)-7 / V.P. Ivannikov, A.A. Belevantsev et al. Static analyzer Svace for finding defects in a source program code. Programming and Computer Software, vol. 40, no. 5, 2014, pp. 265-275.

[12] Cppcheck - a tool for static C/C++ code analysis. Available at http://cppcheck:sourceforge:net, accessed 15.03.2021.

[13] F. Bélanger, S. Collignon at al. Determinants of early conformance with information security policies. Information & Management, vol. 54, no. 7, 2017, pp. 887-901.

[14] Z.C. Schreuders, T. McGill, and C. Payne. Empowering end users to confine their own applications: The results of a usability study comparing SELinux, AppArmor, and FBAC-LSM. ACM Transactions on Information and System Security, vol. 14, no. 2, 2011, pp. 1-28.

[15] S. Shepler, B. Callaghan et al. Rfc3530: Network file system (nfs) version 4 protocol, 2003. Available at https://www.rfc-editor.org/info/rfc3530, accessed 15.03.2021.

[16] M.E. Hoskins. SSHFS: super easy file access over SSH. Linux Journal, no. 146, 2006, pp. 1-4.

[17] J.F. Smart. Jenkins: The Definitive Guide: Continuous Integration for the Masses. O'Reilly Media, 2011, 404 p.

[18] G.A. Campbell and P.P. Papapetrou. SonarQube in action. Manning Publications, 2013, 392 p.

[19] A. Bessey, K. Block et al. A few billion lines of code later: using static analysis to find bugs in the real world. Communications of the ACM, vol. 53, no. 2, 2010, pp. 66–75.

[20] K. Ivanov. Containerization with LXC. Packt Publishing, 2017, 352 p.

[21] D. Merkel. Docker: lightweight Linux containers for consistent development and deployment. Linux journal, no. 239, 2014, pp. 1-2.

[22] J. Wenhao and L. Zheng. Vulnerability analysis and security research of Docker container. In Proc. of the IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE), 2020, pp. 354-357.

[23] T. Combe, A. Martin, and R. Di Pietro. To Docker or not to Docker: A security perspective. IEEE Cloud Computing, vol. 3, no. 5, pp. 54-62, 2016.

[24] M. Menshikov. Towards a resident static analysis. Lecture Notes in Computer Science, vol. 11620, 2019, pp. 62-71.

[25] Z. Hays, G. Richter et al. Alleviating airport WiFi congestion: An comparison of 2.4 ghz and 5 ghz wifi usage and capabilities. In Proc. of the Texas Symposium on Wireless and Microwave Circuits and Systems, 2014, pp. 1-–4.

[26] rizsotto/bear: Bear is a tool that generates a compilation database for clang tooling. Available at https://github:com/rizsotto/Bear, accessed 15.03.2021.

[27] M. Menshikov. Equid – a static analysis framework for industrial applications. Lecture Notes in Computer Science, vol. 11620, 2019, pp. 677-692.
[28] M. Fowler. Refactoring: improving the design of existing code. Addison-Wesley Professional, 2nd edition, 2018, 448 p.
[29] M. Martin, B. Livshits, and M. S. Lam. Finding application errors and security flaws using PQL: A program query language. in Proc. of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, 2005, pp. 365-383.
[30] Federal Service for Technical and Export Control. Available at https://fstec:ru, accessed 15.03.2021.

Информация об авторе / Information about the author

Максим Александрович МЕНЬШИКОВ, аспирант кафедры системного программирования. Научные интересы: статический анализ, обратная разработка, инструменты разработки, высокопроизводительные вычисления, виртуализация.

Maxim Aleksandrovich MENSHIKOV, PhD student. Research interests: static analysis, reverse engineering, development tools, high-performance computing, virtualization.

DOI: 10.15514/ISPRAS-2021-33(3)-3

An Automated Framework for Testing Source Code Static Analysis Tools

[1,2] *D.M. Gimatdinov, ORCID: 0000-0002-1329-4541 <damir.gimatdinov@huawei.com>*
[2] *A.Y. Gerasimov, ORCID: 0000-0001-9964-5850 <gerasimov.alexander@huawei.com>*
[2] *P.A. Privalov, ORCID: 0000-0002-8939-5824 <petr.privalov@huawei.com>*
[2] *V.N. Butkevich, ORCID: 0000-0001-9376-9051 <butkevich.veronika.nikolaevna@huawei.com>*
[2] *N.A. Chernova, ORCID: 0000-0001-8678-9193 <chernova.natalya@huawei.com>*
[2] *A.A. Gorelova, ORCID: 0000-0001-7974-7913 <gorelova. anna @huawei.com>*

[1] *Higher School of Economics,*
11, Pokrovsky boulevard, Moscow, 109028, Russia
[2] *Huawei Technologies Co., Ltd.,*
7b9, Derbenevskaya naberezhnaya, Moscow, 115114, Russia

Abstract. Automated testing frameworks are widely used for assuring quality of modern software in secure software development lifecycle. Sometimes it is needed to assure quality of specific software and, hence specific approach should be applied. In this paper, we present an approach and implementation details of automated testing framework suitable for acceptance testing of static source code analysis tools. The presented framework is used for continuous testing of static source code analyzers for C, C++ and Python programs.

Keywords: automated testing; quality assurance; source code static analysis

For citation: Gimatdinov D.M., Gerasimov A.Y., Privalov P.A., Butkevich V.N., Chernova N.A., Gorelova A.A. An Automated Framework for Testing Source Code Static Analysis Tools. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 41-50. DOI: 10.15514/ISPRAS-2021-33(3)-3

Автоматизированная система тестирования инструментов статического анализа кода

[1,2] *Д.М. Гиматдинов, ORCID: 0000-0002-1329-4541 <damir.gimatdinov@huawei.com>*
[2] *А.Ю. Герасимов, ORCID: 0000-0001-9964-5850 <gerasimov.alexander@huawei.com>*
[2] *П.А. Привалов, ORCID: 0000-0002-8939-5824 <petr.privalov@huawei.com>*
[2] *В.Н. Буткевич, ORCID: 0000-0001-9376-9051 <butkevich.veronika.nikolaevna@huawei.com>*
[2] *Н.А. Чернова, ORCID: 0000-0001-8678-9193 <chernova.natalya@huawei.com>*
[2] *А.А. Горелова, ORCID: 0000-0001-7974-7913 <gorelova. anna @huawei.com>*

[1] *Национальный исследовательский университет Высшая школа экономики,*
109028, Россия, Москва, Покровский бульвар 11
[2] *Техкомпания Хуавэй,*
115114, Россия, Москва, Дербеневская набережная, 7с9

Аннотация. Среды автоматизированного тестирования широко используются для обеспечения качества современного программного обеспечения в жизненном цикле разработки безопасного программного обеспечения. Иногда требуется проверка качества специфического программного обеспечения и поэтому требуется применение специфического подхода для решения этой задачи. В этой статье мы представляем подход и детали реализации среды автоматического тестирования, предназначенной для приёмочного тестирования инструментов статического анализа исходного кода программ. Представленная среда используется для непрерывного тестирования инструментов статического анализа исходного кода программ на языках C, C++ и Python.

Ключевые слова: автоматизированное тестирование; обеспечение качества; статический анализ исходного кода.

Для цитирования: Гиматдинов Д.М., Герасимов А.Ю., Привалов П.А., Буткевич В.Н., Чернова Н.А., Горелова А.А. Автоматизированная система тестирования инструментов статического анализа кода. Труды ИСП РАН, том 33, вып. 3, 2021 г., стр. 41-50 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(3)–3.

1. Introduction

Acceptance testing is a very common approach to make sure required software functionality is satisfying needs of end user in an automatic way. Wide usage of continuous integration systems with automatic tests run allows to automate testing process to make sure the functionality is not broken by separate change in a program code. That is why it is important to build suitable testing framework to satisfy needs in continuous testing of specific software.

A source code static analysis tools are become an industrial standard for software quality assurance at early stages in secure software development lifecycle. They are commonly used for detection of program issues and logical errors. Being quality assurance tools by nature they need to satisfy specific requirements such as an analysis precision, completeness and performance. A possibility to introduce bug warnings of a safe code, also known as false positive warnings, set a target for a testing framework to control as true positive warning, as false positive warnings. An acceptance testing of such tools controls behavior of a tool on specific code snippets, which represent as buggy code, as code which has no bugs and issues.

At the same time, such tools are very complex in implementation details, because consist of general analysis framework, frequently called engine, which propose general analysis techniques such as reaching definitions, live variables, taint analysis and others, and a number of specific wrong program behavior checkers build on top of an engine. Any small change to the engine can broke checkers behavior. That's why it is important to have testing framework which can check and state sanity of the tool during development lifecycle.

In our previous talk[1], we have described a generalized approach for testing static source code analysis tools, which includes Acceptance Testing Framework and Regression Testing System called Report History Server.

In this paper, we introduce requirements, implementation details, evaluation and limitations of Acceptance Testing Framework for static source code analysis tools based on our experience of development and daily usage of such a framework in industrial development of static source code analysis tools. This paper is organized as follows. Section 2 describes in detail requirements to such kind of framework, Section 3 provides overview of existing approaches, Section 4 provides an overview of proposed approach. Section 5 describes in detail implementation of proposed approach, Section 6 contains evaluation results of proposed approach, Section 7 concludes proposed approach and future directions of development.

2. Requirements to acceptance testing framework

Source code static analysis tools have to check conditions of source code of programs from the point of view of very different rules, which can be applied as industrial or companywide coding standard. Despite of focus for modern static source code analyzers on code security, lack of logical errors and

[1] Alexander Gerasimov, Petr Privalov, Sergey Vladimirov, Veronica Butkevich, Natalya Chernova, Anna Gorelova. An approach to assuring quality of automatic program analysis tools. Ivannikov Ispras Open Conference (ISPRAS), 2020

performance, some kind of coding rules applied in companies or industry can contain such requirements to the code as style of indentation, naming conventions, etc. For example, if we take a look to Python programs then source code can contain commentaries of the specific look, such as Shebang [1], encoding of the file [2], company code ownership statement and version or license notes. That's why trying to satisfy needs of testing industrial static source code analyzers such a framework cannot rely on specific comments and code formatting, such as used in most known test cases database Juliet of National Institute for Standardization and Technology of USA [3].

Instead of that we have to have a database of error code snippet describers. Such kind of describers provide all necessary information on test case in a file or set of files with directories structure, separated and independent of language for a source code of target analyzer and target language of analyzed programs. We use specific JSON [4] formatted descriptions of test cases which describe every test case as for erroneous examples, as for clean code examples.

On the other hand, we have set a goal to compare tested static source code analyzer with competing ones. That's why we put as a requirement ability to run competing static source code analyzers in one bundle to compare precision, completeness and performance of such tools. That is second requirement.

Next, we need to have solution for different environments such as operating systems and hardware platforms. That's why we set it as one of requirements to the framework.

And, last, but not least, we want to make out Acceptance Testing Framework independent of target language of analyzed programs. It should be suitable for testing analyzers for programming languages C, C++, Java, C#, Python and other languages.

To summarize:

- Independence of target environment, such as hardware and operating system.
- Independence of analyzed programming languages.
- Possibility to check source code snippets without modification of original code even in comments part.
- Possibility to check as erroneous, as clean code examples (true positive and false positive warnings checks).
- Support pretty unlimited number of checkers for coding rules, including, but not limited to formatting and comment styles.
- Possibility to compare different static source code analysis tools.
- Possibility to represent results of analysis in different formats: machine readable (JSON, XML and others), output formatted to represent result on the screen, HTML format, etc. with possibility to extend list of reporting formats on demand.

3. Existing approaches

There are a lot of research papers dedicated to evaluation of static code analysis tools [5, 6, 7]. These works observe behavior of static code analysis tools on selected subset of NIST SAMATE test cases for selected OWASP [8] Top 10 vulnerabilities. But these papers a dedicated to manual evaluation of static code analysis tools and does not solve the problem of automated frameworks implementation.

The work [9] attempts to solve the problem of creating automated test suite to evaluate static analysis tools by designing test cases as small code snippets, which automatically in-lined into template program to specific placeholder.

The work [10] describes an approach of detecting minimal original test cases from real-world found errors and tries to add code to the original test code snippet to check sensitivity of analysis to paths and call context. The difference of our approach is in common automation of acceptance testing and evaluation system for static source code analysis tools. In this paper, we describe technical details and evaluation of proposed approach.

4. Overview

Acceptance Testing Framework solves problem of evaluating the quality of automatic program analysis tools. The quality is measured by parameters such as: performance, scalability, precision, completeness.

Performance – how fast an analysis tool can provide an analysis result and how much resources it consumes.

Scalability – how analysis time reduces if we providing additional computational resources.

Precision – how precise an analysis result is (small number of false positive warnings or noise).

Completeness – how many true positive warnings issued by a tool in comparison to errors exist in the test suite (number of false negatives – errors has been missed).

To compute such parameters Acceptance Testing Framework allows to run program analysis tool against a limited, manually crafted set of test cases combined in one test suite. Test suite represents behavior of defective and similar to defective programs. The defective one gives rate of true positive warnings should be found and similar to defective gives rate of false positive warnings, which absence is expected. So far the resulting precision and completeness are calculated and evaluated.

As far as precision and completeness are evaluated by Acceptance Testing Framework for program analysis tool, decision about quality could be made. In theory perfect tool has 100% completeness of test suite (all defects detected) and 100% precision (no noise and no defect detected on similar to defective code snippets), but such values cannot be achieved at current stage of engineering and have the theoretical limitation of Rice’s theorem [11].

There are no strict generally accepted values for performance and scalability as far as these parameters depend on depth, complexity and target of analysis and vary greatly among analysis tools. Moreover, the exact conclusion about the quality of analysis tools directly depends on the test suite. Acceptance Testing Framework doesn't contain built-in features to get performance and scalability on its own for now. Despite this Acceptance Testing Framework could be used in the computation process of these parameters by running program analysis tool against set of different complexity (from low to high) test suites and observe how performance dynamic depends on complexity of test suite or scalability dynamic in the case of additional computational resources involved in computation process.

Test suite could follow company or industrial standards, contain code snippets with security vulnerabilities, code style or leading to crash errors. In our case test suite follows company standard and together with Acceptance Testing Framework has deployed in continuous integration processes of static analysis tool development in Huawei Russian Research Institute.

5. Design and implementation

In this section, we describe the design and implementation of our framework. We describe it from requirements perspective.

5.1 Independence of target environment

To satisfy requirement of an independence of target environment such as hardware and operating system we managed to implement our framework in Python programming language as far as it has Python source code interpreters for most of industrial operating systems and for most popular hardware platforms.

5.2 Independence of analyzed programming language

The framework does not rely somehow on code snippets content by using JSON formatted test case annotations.

5.3 Possibility to check code snippets without modification of original code, even in comments. Possibility to check as erroneous, as clean code snippets without modification

We use test case annotation files in JSON format. Test case for Acceptance Testing Framework is a tuple of annotation file and source code snippet. JSON annotation file contains following information:

- Kind of a snippet: does it contains a defect (True Positive) or it is not expected in this code snippet (True Negative).
- Kind of a defect expected to be reported or not reported.
- Description of a test case.
- Skip flag for marking test cases which are not supported, but planned to be supported in future.
- Defect location: filename, line and offset in the line for expected defect.
- Additional service information. For example, if test case designed for specific version of language, to configure analyzer appropriately, or additional field describing the goal of test case to QA engineer or developer.

Such decision allows to keep all this information independent of test cases and needed by Acceptance Testing Framework to configure analysis tools appropriately, and do not rely somehow on number of test cases, because it is enough to just point the location of file system directory with test suite formatted to be used with Acceptance Testing Framework while running framework and all work related to running analysis tools on the test suite handled by framework itself via traversing directories structure.

5.4 Possibility to compare different analysis tools

Acceptance Testing Framework satisfy this requirement by introducing abstract interface *Tool* to run external analysis tool as executable program and get results of analysis in Acceptance Testing Framework internal representation. Having such kind of interface to support of new analysis tool ones need to implement interface *Tool* to convert test case settings from test case annotations to expected arguments of analysis tool and run this tool as external process. We have developed a number of interface implementations for tools, such as PyLint [12], JetBrains PyCharm [13] and eight more tools, which have different paradigm of analysis. For example, PyLint accepts analysis of single file and can be run on every test case separately. PyCharm expects a file system directory and treats it as one project to analyze.

On the other hand, analysis results representation of different tools can vary significantly. An implementation of *Tool* interface also responsible for interpretation of external analysis tool results and converting it to Acceptance Testing Framework internal representation. This representation is a kind of map for every test case to analysis result in term of *Passed* or *Failed* state.

Thus all logic of working with analysis tool is encapsulated inside of *Tool* interface implementation.

5.5 Possibility to represent results of analysis in different formats

Acceptance Testing Framework provides universal interface *Reporter* which provides one public method *report* accepting internal representation of analysis tool run results. A responsibility of implementation of interface is to issue report in specific format. We have implemented three reporters supported out of the box:

- Output reporter. Represents test suite run results in human readable text format.
- JUnit reporter. Represents test suite run results in JUnit format.
- HTML reporter. Represents test suite run results in format of static web-site with possibility to represent result in different view up to source code snippet of test case.

Architecture diagram of Acceptance Testing Framework is shown on fig. 1. It consists of following blocks (classes):

- *Driver*. It is entry point of framework. It allows to configure test suite, reporter and tools accordingly to parameters passed to framework on the run.
- *TestSuite* is a collection of *TestCases* which constructed using provided path to test suite directory, where every test case has its annotation in JSON format and test case source code files directory structure.

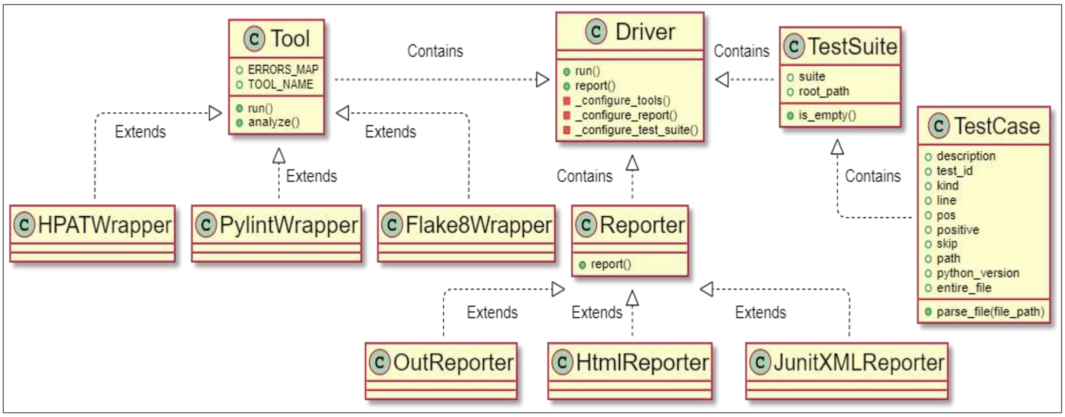


Fig. 1. Acceptance Testing Framework architecture diagram

- *Tool*. It is an interface representing a tool runner. Instantiations of this interface depends on settings of the framework passed as command line arguments.
- *Reporter*. It is an interface allowing to represent analysis results using unified internal test suite run results representation.

In general, Acceptance Testing Framework is a *Driver*, which responsible for:

- Instantiation of supported analysis tool wrappers, which are implementations of *Tool* interface, accordingly to parameters passed to the *Driver* by user.
- Instantiation of the *Reporter* which will be used to output result of analysis by every tool.
- Running the analysis process to collect analysis result in internal representation form and pass received result to *Reporter*.

6. Results & evaluation

This section aims to obtain a classification of tools according to the metrics applied to the results obtained from the execution of the tools against our test suite.

Tested static analysis tools:

- Huawei Python Analysis Tool (HPAT) is a PyCharm plugin with the set of inspections requested by Huawei Python Code Style Guide and Huawei Secure Coding Style Guide.
- Flake8 [14] is an open source tool that glues together pep8 [15], pyflakes [16], mccabe [17],

and third-party plugins to check the style and quality of some python code.

- PyLint is an open source tool that checks for errors in Python code, tries to enforce a coding standard and looks for code smells.

The summary of metrics used is:

- True positives rate – TP (correct detections).
- False positive – FP (reporting false error warning).
- Number of vulnerability categories for which the tool was tested.
- Precision (1). Proportion of the total TP detections:

$$TP\ /\ (TP\ +\ FP) \tag{1}$$

- Recall (2). Ratio of detected vulnerabilities to the number that really exists in the code. Recall is also referred to as the True Positive Rate:

$$TP\ /\ (TP\ +\ FN) \tag{2}$$

Table 1. Number of vulnerability categories

Tool Metric	*HPAT*	*Pylint*	*Flake8*
NVC	68	32	15

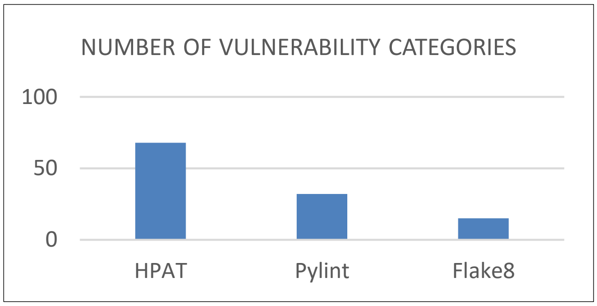


Fig. 2. Number of checked defect types

Table 2. Vulnerabilities detection. Numbers of true/false positive, true/false negative test case detection

Tool Metric	*HPAT*	*Pylint*	*Flake8*
TP	695	91	102
FN	0	324	368
FP	0	0	0
TN	591	121	184
Total	1286	536	654

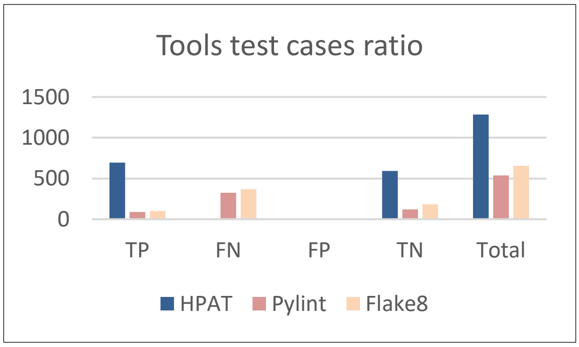


Fig 3. Test cases ratio obtained by the tools comparison

Table 1 and fig. 2 show a number of vulnerability categories (NVC) for which the tool is tested. HPAT has the biggest value because test suite is developed exactly for satisfying needs of Huawei coding standards.

Table 2 and fig. 3 show a result of running tools on test suite in terms of true/false positive, true/false negative.

Tab. 3 and Fig. 4 show metrics results of all tools included in this analysis.

Table 3. Assessment results computing and ranking the selected metrics by TP ratio

Metric Tool	*TP ratio*	*FP ratio*	*Precision*	*Recall*
HPAT	1	0	1	1
Pylint	0.219	0	1	0.219
Flake8	0.217	0	1	0.217

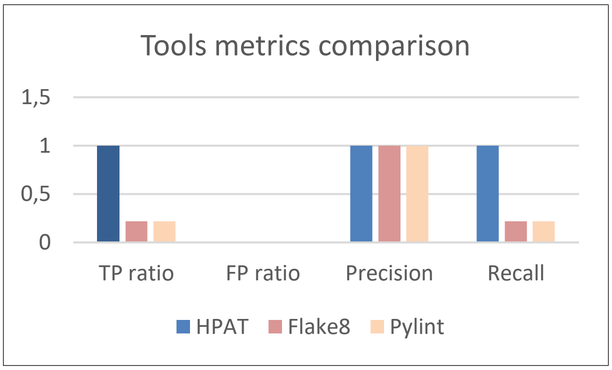


Fig 4. Metrics obtained by the tools comparison

Implemented framework allows to assess tools on the same testing code base and present relative results

7. Conclusion

In this paper, we focused on checking quality of static source code analysis tools with help of an automated framework for running such tools against a number of test cases combined in one suite. This approach allows us to control quality of the tool in terms of created erroneous and error free test cases as code snippets on target for analysis programming language. The framework allows to use any kind of test suites if configured well within a profile or manifest in expected format.

This approach to testing static source code analysis tools has applied in development process of static source code analysis tools for Python and C/C++ in Huawei Russian Research institute. In future we plan to extend functionality of Acceptance Testing Framework to check non-functional requirements for tools such as time of running, memory consumption and CPU utilization.

References

[1] M. Cooper. Advanced Bash Scripting Guide – Volume 1: An in-depth exploration of the art of shell scripting. (Revision 10). Independently published, 2019, 589 p.

[2] M.-A. Lemburg, M. von Löwis. PEP-263 – Defining Python Source Code Encodings. 2001. URL: https://www.python.org/dev/peps/pep-0263/.

[3] NIST SAMATE Juliet Test Suite. URL: https://samate.nist.gov/SRD/testsuite.php.

[4] RFC-8259. The JavaScript Object Notation (JSON) Data Interchange Format, 2017. URL: https://datatracker.ietf.org/doc/html/rfc8259.

[5] H.H. AlBreiki, Q.H. Mahmoud. Evaluation of static analysis tools for software security. In Proc. of the IEEE 2014 10th International Conference on Innovations in Information Technology, 2014, pp. 93-98,

[6] R. Mamood, Q.H. Mahmoud. Evaluation of static analysis tools for finding vulnerabilitites in Java and C/C++ source code. arXiv:1805.09040, 2018, 7 p.
[7] T. Hofer. Evaluating static source code analysis tools. Master’s thesis. École Polytechnique Fédérale de Lausanne, 2010, pp. 1-74.
[8] OWASP – Open web application security project. URL: https://owasp.org
[9] M. Johns, M. Jodeit. Scanstud: a methodology for systematic, fine-grained, evaluation of static analysis tools. 4th International conference on software testing, verification and validation workshops. In Proc. of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, 2011, pp. 523-530.
[10] G. Hao, F. Li et al. Constructing benchmarks for supporting explainable evaluations of static application security testing tools. In Proc. of the 2019 International symposium on Theoretical Aspects of Software Engineering, 2019, pp. 66-72.
[11] H.G. Rice. Classes of Recursively Enumerable Sets and Their Decision Problems. Transactions of the American Mathematical Society, vol. 74, no. 2, 1953, pp. 358-366.
[12] Pylint. URL: https://pypi.org/project/pylint/.
[13] JetBrains PyCharm. URL: https://www.jetbrains.com/pycharm/.
[14] Flake8. URL: https://pypi.org/project/flake8/.
[15] Pep8 – Python style guide checker. URL: https://pypi.org/project/pep8/.
[16] Pyflakes. URL: https://github.com/PyCQA/pyflakes.
[17] McCabe complexity checker. URL: https://github.com/PyCQA/mccabe.

Информация об авторах / Information about authors

Дамир Маратович ГИМАТДИНОВ, выпускник ВШЭ, магистр, младший инженер в Техкомпании Хуавэй. Научные интересы: статический анализ исходного кода программ.

Damir Maratovich GIMATDINOV, HSE graduate, master, Junior engineer in Huawei Technologies. Research interests: Source code static analysis.

Александр Юрьевич ГЕРАСИМОВ, кандидат физико-математических наук, старший эксперт в области автоматического и автоматизированного анализа программ электронных вычислительных машин. Научные интересы: статический анализ программ, динамический анализ программ, обеспечение качества программ, обнаружение ошибок в программах.

Alexander Yurievich GERASIMOV, Doctor of Philosophy in Computer Sciences, Senior Expert in the field of automatic and automated analysis of electronic computer programs in Huawei Technologies. Research interests: static program analysis, dynamic program analysis, quality assurance, program defects detection.

Пётр Алексеевич ПРИВАЛОВ, магистр, ведущий инженер-программист. Научные интересы: статический и динамический анализ программ, фаззинг.

Petr Alekseevich PRIVALOV, master, Senior software engineer. Research interests: static and dynamic program analysis, fuzzing.

Вероника Николаевна БУТКЕВИЧ, магистр, старший инженер. Научные интересы: статический анализ исходного кода программ, обнаружение уязвимостей в программном коде.

Veronika Nikolaevna BUTKEVICH, master, developer. Research interests: static analysis, security vulnerabilities in software

Наталья Андреевна ЧЕРНОВА, магистр, младший инженер. Научные интересы: статический анализ программ, анализ потока данных.

Natalya Andreevna CHERNOVA, master, junior developer. Research interests: static analysis of programs, data-flow analysis.

Анна Антоновна ГОРЕЛОВА, младший инженер. научные интересы: искусственный интеллект, машинное обучение.

Anna Antonovna GORELOVA, Junior Developer. Research interests: artificial intelligence, machine learning.

DOI: 10.15514/ISPRAS-2021-33(3)-4

Data Layout Optimization for the LCC Compiler

[1,2] *V.E. Shamparov, ORCID: 0000-0002-0938-3824 <Victor.E.Shamparov@mcst.ru>*
[2] M.*I. Neiman-zade, ORCID: 0000-0002-4250-9724 <Murad.I.Neiman-zade@mcst.ru>*
[1] *MCST,*
24 Vavilova str., Moscow, 119334, Russia
[2] *Moscow Institute of Physics and Technology,*
9 Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russia

Abstract. In this research-in-progress report, we propose a novel approach to unified cache usage analysis for implementing data layout optimizations in the LCC compiler for the Elbrus and SPARC architectures. The approach consists of three parts. The first part is generalizing two methods of estimating cache miss amount and choosing more applicable one in the compiler. The second part is finding an applicable solution for the problem of cache miss amount minimization. The third part is implementing this analysis in the compiler and using analysis results for data layout transformations.

Keywords: Compilers; Compiler Optimization; Cache Analysis; Data Layout Transformation

For citation: Shamparov V.E., Neiman-zade M.I. Data Layout Optimization for the LCC Compiler. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 51-60. DOI: 10.15514/ISPRAS-2021-33(3)-4.

Оптимизации расположения данных для компилятора LCC для архитектуры Эльбрус

[1,2] *В.Е. Шампаров, ORCID: 0000-0002-0938-3824 <Victor.E.Shamparov@mcst.ru>*
[2] *М.И. Нейман-заде, ORCID: 0000-0002-4250-9724 <Murad.I.Neiman-zade@mcst.ru>*
[1] *АО «МЦСТ»*
Россия, 119701, Москва, ул. Вавилова, д. 24
[2] *Московский физико-технический институт,*
Россия, 141701, Московская область, г. Долгопрудный, Институтский пер., 9

Аннотация. В данной статье о проводимом исследовании мы предлагаем новый подход к единому анализу использования кэш-памяти для разработки оптимизаций расположения данных в составе компилятора LCC для архитектур Эльбрус и SPARC. Подход состоит из трёх частей. Первая часть - обобщение двух методов оценки количества кэш-промахов и выбор из них более подходящего для реализации в компиляторе метода. Вторая часть - поиск применимого в компиляторе решения задачи минимизации количества промахов кэша. Третья часть - реализация выбранного метода анализа в компиляторе и использование результатов анализа для оптимизаций расположения данных.

Ключевые слова: компиляторы; оптимизации компилятора; анализ кэша; оптимизации расположения данных.

Для цитирования: Шампаров В.Е., Нейман-заде М.И. Оптимизации расположения данных для компилятора LCC для архитектуры Эльбрус. Труды ИСП РАН, том 33, вып. 3, 2021 г., стр. 51-60 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(3)–4.

1. Introduction

Improving computer resources usage efficiency by a program is one of the main tasks for optimizing compilers. Particularly, improving memory usage is especially important because hardware developers have introduced multi-level intermediate memory, called cache memory, due to the growing performance difference between memory and CPU. Cache memory capabilities must be used efficiently.

Cache memory is structured for using the following program properties effectively *temporal locality* and *spatial locality*. *Temporal locality* means that the program often works with the same data in memory. *Spatial locality* means that the program is likely to work with adjacent data. Thus, to make compiled program use cache memory efficiently, the compiler must improve these two programs' properties.

Nowadays, compilers optimize the programs' temporal locality well by loop optimizations, but optimizing spatial locality is more complicated since it requires choosing the correct data structures for the program. Therefore, optimizing spatial locality is often entrusted to the programmer, although data location optimizations are implemented for some relatively simple cases.

In this article, we describe the ongoing research on cache memory usage for the further development of a high-quality automatic cache usage analysis in the compiler for applying an optimal set of data layout optimizations.

The article is organized as follows. In section 2, we substantiate the potential effect of optimizing data layout. In section 3, we state the problem. In section 4, we analyze papers on this topic and related ones. In section~5, we propose further research approach. In section 6, we describe current progress. Finally, in section 7, we provide a conclusion.

2. Motivation

It is known that part of program execution time is spent waiting for data from memory. This is especially evident for processors with in-order execution. They have fewer opportunities to mask this wasted time by executing other instructions than processors with out-of-order execution.

To illustrate this problem and determine the potential effect of optimization, we measured the percentage of test execution time from SPEC~CPU benchmark packages that the processor spends waiting for data from memory. This data is shown in Table 1. We used a computer with an Elbrus-4C processor for measurement. It has VLIW ISA, in-order execution and two-level cache memory. Benchmarks were compiled with peak options.

Table 1. Number of benchmark launches from SPEC~CPU packages that use more than 10% of time to wait for data

Set	**Part of time**	**Number of launches**	**Set**	**Part of time**	**Number of launches**
1995	10...15%	12	**2000**	10...15%	6
	15...20%	4		15...20%	6
	20...25%	0		20...25%	6
	25...30%	1		25...30%	1
	≥ 30%	0		≥ 30%	6
	Total in set	37		Total in set	44
f2006	10...15%	4	**i2006**	10...15%	3
	15...20%	1		15...20%	1
	20...25%	1		20...25%	3
	25...30%	1		25...30%	3
	≥ 30%	2		≥ 30%	12

	Total in set	20		Total in set	35
f2017	10...15%	1	**i2017**	10...15%	1
	15...20%	2		15...20%	2
	20...25%	3		20...25%	0
	25...30%	0		25...30%	3
	≥ 30%	1		≥ 30%	6
	Total in set	16		Total in set	20
All	10...15%	27			
	15...20%	16			
	20...25%	13			
	25...30%	9			
	≥ 30%	27			
	Total in set	172			

The table shows that more than 10% of the execution time is spent waiting for data from memory in 92 from 172 launches, which is more than a half.

Some of this spent time is due to inefficient use of cache memory. Mainly, these inefficiencies are:

1) loading unnecessary for further work data into the cache, which fact is a violation of spatial locality;
2) conflicts between different data chunks due to hitting the same cache set.

For example, it was found during our previous work that it is possible to reduce the number of cache misses with the help of optimization called Structure Splitting [1]. This optimization improves the spatial locality of the program in some cases. Such CPU pipeline stalls number decrease and consequent execution speeding up are shown in the Table 2.

Table 2. CPU pipeline stalls number decrease and following program execution speeding up

Benchmark	**SPEC CPU package**	**CPU pipeline stalls number decrease**	**Speed-up**	
181.mcf	2000	27%	26%	
429.mcf	2006	19%	13%	

From this example, it can be seen that at least some of the losses due to waiting for data can be removed by data layout transformations improving spatial locality. These transformations require unified analysis for an effective combination.

3. Problem statement

Thus, we need to:

1) Theoretically analyze cache memory usage by programs and develop a method of solving the problem of minimizing time losses based on this theoretical analysis.
2) Based on theoretical results, make applicable automatic analysis in the LCC compiler for the Elbrus and SPARC ISA.
3) Implement in the same compiler a set of data layout transformations, which transform data layout of a program based on the analysis results.

In this case, it is necessary to take into account some restrictions arising from the fact that the implementation is planned in the form of compiler optimizations:

1) Various data structures need to be handled correctly. Particularly, they are:
    a) Arrays, structures and their combinations.

   b) Various data structures that use pointers to other elements internally and allocate memory for new elements via *malloc* and similar memory allocation functions. For example, lists and trees.

4) We need to handle data structures altogether, as their transformations may conflict with each other. Therefore, it is necessary to analytically process not only regular access to memory but also random access.
5) Analysis and transformations must be static (in the compiler) but can be supported with runtime libraries and special profiling, but not memory access trace.
6) Developed analysis and transformations must correctly work in modular build mode.

4. Related work

Several works on related topics have already been written, but each of them does not solve assigned tasks entirely due to different reasons.

Chris Lattner proposed automatic Data Structure Analysis to detect data structures whose elements are allocated on the heap in his thesis «Macroscopic Data Structure Analysis and Optimization» [2]. Using the results of this analysis, he proposed a compiler optimization called Automatic Pool Allocation with runtime support, designed to group the elements of such data structures in specific regions of the heap, which improves the spatial and, in some cases, temporal locality of the program. In addition, he offered several optimizations for code already optimized in this way.

Unfortunately, there is no explicit cache memory usage analysis in Lattner's work.

Christopher Haine in his thesis «Kernel optimization by layout restructuring» [3] offered an analyzer, which detects accessing memory regularly simple data structures like structures and arrays and proposes layout transformations using heuristics data. This analysis is separated from the compiler. In addition, this analyzer provides user with information about the complexities of code vectorization. For our purposes, this work is not suitable since there is no explicit cache memory usage analysis.

Mostafa Hagog and Caroline Tice in their article «Cache Aware Data Layout Reorganization Optimization in GCC» [4] proposed several improving spatial locality optimizations of structures and arrays of structures: Structure Peeling, Structure Splitting, and Field Reordering. These optimizations were later implemented in the GCC compiler. Although the authors limited themselves to working with structures, they implemented an analysis handling every structure access, not just regular access. During optimization, particular Field Reference Graphs are built for each analyzed structure for each procedure. Field Reference Graph (FRG) is an analogue of a control-flow graph, where nodes contain operations accessing fields of the analyzed structure and arcs contain information about the amount of data loaded into the cache between nodes. In fact, this is an implicit analysis of cache memory usage. Further, after processing, this information is used in heuristics to apply the specified optimizations and reduce the computational complexity of further algorithms.

This approach can potentially be used for explicit cache memory usage analysis, provided it is generalized for working on all program data in all procedures.

Ghosh et al. [5] and Fraguela et al. [6] suggested more explicit techniques for cache memory usage analysis for regular access cases.

Ghosh et al. [5] proposed to compose and solve systems of linear Diophantine equations to estimate the number of cache misses for each cycle. They implemented this algorithm in the SUIF compiler and implemented the choice of padding size in the Array Padding optimization as an example. However, they did not implement an automatic solution of systems in parametric form - only a particular solution for Array Padding. In addition, this approach was created only for regular memory access.

An alternative approach was suggested by Fraguela et al. [6] for regular memory access. It was improved by Andrade in [7] thesis for some cases of irregular memory access: regular access under condition and access to an array, where the indices are read from another array. This approach is based on estimating the probability of cache misses in each analyzable cycle using Probabilistic Miss Equations (PME) generated from regular access characteristics and cache memory characteristics. To do this, for each processed access in the loop, a partial Probabilistic Miss Equation is built, and then they are combined into a complete equation for the loop or loop nest. This complete equation gives an estimation of cache misses amount. In addition, they did not offer any solution to the problem of minimizing cache misses amount and did not handle random memory access. Thus, the PME approach can potentially be applied for explicit cache memory usage analysis, provided the analysis is generalized for working for all irregular memory access.

Data layout transformations were described in many papers. Particularly, a small catalogue of such transformations was created in the article [8]. Following transformations are listed in this article:

1) Array Padding – adding padding between arrays to reduce number of conflicts between arrays;
2) Array Merging – element-wise arrays merging;
3) Array Transpose – changing dimensions' order of an array by analogy with transposing a matrix.

In addition to these, in the above-mentioned article [4] and thesis [2] some other transformations were described:

1) Structure Peeling – splitting an array of structures element by element into several arrays;
2) Structure Splitting – splitting an array of structures element by element into several arrays and addition of links between the elements corresponding to the initial element;
3) Field Reordering – changing order of fields inside the structure;
4) Automatic Pool Allocation – replacing memory allocation for data structure elements in the heap with memory allocation in a specific pool.

5. Proposal

Firstly, it is proposed to investigate and compare following methods for cache memory usage analysis:

1) the method described in [4] using FRG graphs, generalized for working with all program data in all procedures;
2) the method described in [6, 7] using the Probabilistic Miss Equations, generalized for the case of random access.

We propose to choose one method for cache memory usage analysis that is more suitable for implementation in the compiler. The selection criterion is the accuracy of the estimation of cache misses amount. Another selection criterion is analysis time.

Further, we propose to develop an analytical or another compiler-applicable method for solving the problem of minimizing the obtained estimation of the cache misses amount using data layout transformations. This problem is a discrete optimization problem, in which the objective function is the dependence of the cache misses amount on the applied data layout transformations, and a countable set of feasible solutions is the data layout transformations.

Finally, based on the developed analysis method and the method for solving the problem of minimizing the cache misses amount, it is proposed to implement automatic analysis in the compiler that controls a set of data layout transformations. Also, we will need to implement missing transformations.

5.1 Generalizing FRG analysis

This method should be generalized for working on all program data in all procedures and provide an estimation of cache misses amount. To do this, based on the FRG graph for structures, we need to make a generalized graph for structures, arrays, their combinations and other data structures. Such graphs need to be created for each program object. Let us call such graphs Object Reference Graph – ORG. In addition, we need to build a general RGP (Reference Graph in Procedure) graph consisting of all memory accesses in the procedure and including profile information. So any ORG graph in a procedure contains a subset of RGP nodes; therefore, using RGP, one can estimate the probabilities of transitions through various ORG arcs and cache memory usage characteristics between ORG nodes. In addition, RGP is required to analyze conflicts between different data structures.

It is required to determine the probability of a particular cache line being evicted from the cache memory to estimate the probability of a cache miss in each ORG node. Since the probability of preempting a particular cache line depends on the amount of memory loaded into the cache in the general case in a complex way, it is better to store on the arcs of ORG graphs, not the amount of memory loaded into the cache, but the probability of preempting a particular cache line.

To estimate the probabilities, one must know in which memory regions the memory addressed by each pointer is located and the size of these memory regions. To obtain this information, we need to use pointer analysis and a particular version of the profile, which collects data on the size of the allocated memory.

5.2 Generalizing PME analysis

To use this method, we need to generalize it for processing irregular memory access.

For this, we need to:

1) Create a way to calculate cache misses’ probability for random access.
2) Generalize PME to those cases of near-regular access where it is possible to estimate cache misses amount more accurately than using a random access model.
3) Combine PME for regular access and ones for random access.
4) Use the developed techniques for estimating cache misses amount for the entire code, not just for loops.

To estimate the probabilities, one must know in which memory regions the memory addressed by each pointer is located and the size of these memory regions. To obtain this information, we need to use pointer analysis and a particular version of the profile, which collects data on the size of the allocated memory.

6. Current progress

In the work [1] we described the particular version of data layout transformation called Structure Splitting, which we had implemented in the LCC compiler for the Elbrus and SPARC architectures. In addition, in this compiler Structure Peeling, Array Transpose, Array Linearization, and Array Padding have already been implemented.

6.1 Cache miss probability for random access

To generalize the PME-based analysis, a method was created for calculating the cache misses probability for random access. It is supposed that the memory region is known for this access, but the address of the region beginning is unknown. PME will be merged with this method.

The method is based on determining cache state transformations for each memory access operation. For this, the operations are traversed sequentially in the basic blocks of the procedure, and the transformations on the code blocks are combined according to the probabilities in the profiled control-flow graph. Any operation of the procedure is traversed once for random access case. For any other case number of single operation traversals must be $O(1)$ due to the analysis applicability requirement.

The cache state notation for the general case of regular and random access has not been determined yet, but the following notation has been chosen for the random access model: matrix **P** composed of N vectors $\mathbf{P}_{\boldsymbol{i}}$ corresponding to N memory regions. Each vector has $S + 1$ size, where S is the number of cache lines in the cache. The element of the matrix $\mathbf{P}_{\boldsymbol{ij}}$ is the probability that exactly j lines corresponding to the area i are stored in the cache memory at the moment.

An example of the chosen cache state notation for three memory regions called a_i, where $i = 1..3$, is shown in Table 3. In the shown state it is implied that region a_1 has no lines in cache with 100% probability. Also, probability of a_2 taking all lines of cache is 90% and probability of a_3 taking one line and a_2 taking all other lines is 10%.

Table 3. Chosen cache state notation example $\boldsymbol{P_{ij}}$ *for three memory regions called* a_i, $i = 1..3$

$\boldsymbol{j}$	$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$
$\boldsymbol{S}$	0%	90%	0%
$\boldsymbol{S-1}$	0%	10%	0%
…	…	…	…
1	0%	0%	10%
0	0%	100%	90%

Let us introduce for each operation or code section c an operator for changing the state T^{c}. If there was state P^b before executing c, then state P^a after executing c is: $\mathrm{P}^a = \mathrm{T}^{\mathrm{c}}\mathrm{P}^b$. We require the following properties for the operator:

1) For a code section c, consisting of K consecutive code sections or operations $\mathrm{c}_1, \dots, \mathrm{c}_K$, the operator is a composition of operators for parts of the section: $\mathrm{T}^{\mathrm{c}} = \mathrm{T}^{\mathrm{c}_K} \dots \mathrm{T}^{\mathrm{c}_1}$.
2) For a code section consisting of K alternative code sections or operations $c_1, \dots, c_K$ with probabilities of passing through them $p_1, \dots, p_K$ (for example, `if` block and `else` block), with $\sum_{j=1}^{K} p_j = 1$, the operator is a linear combination of operators for parts of the section: $\mathrm{T}^{\mathrm{c}} = \sum_{j=1}^{K} p_j \mathrm{T}^{\mathrm{c}_j}$.
3) Similarly, if during the execution of one operation op of the K different state changes $\mathrm{T}_1^{\mathrm{op}} \dots \mathrm{T}_K^{\mathrm{op}}$ may occur with probabilities $p_1, \dots, p_K$, and $\sum_{j=1}^{K} p_j = 1$, the operator is a linear combination of their operators: $\mathrm{T}^{op} = \sum_{j=1}^{K} p_j \mathrm{T}_j^{\mathrm{op}}$.

For the chosen matrix cache state notation, we also introduce an element-wise product $\circ$ of the operator and coefficients.

Let us consider one memory access operation. It can cause three different outcomes:

1) Cache hit. In this case, cache state in the selected notation is not changed.
2) Cache miss with a conflict in the memory region. In this case, cache state in the selected notation does not change since it only stores the probabilities of having a certain amount.
3) Cache miss with a conflict with another memory region. A new line is loaded into the cache for the memory region the operation is working with. For one of the other memory regions, the line is evicted from the cache.

Thus, change in cache state for a single operation for a specific memory region can consist only in loading a new cache line for memory region, deleting cache line from the cache for memory region, or no changes for memory region. For such changes we introduce operators for the movement of cache state in selected notation:

1) M^+ – moves the matrix values up by 1: if $\mathrm{P}^a = \mathrm{M}^+\mathrm{P}^b$, then

$$\forall i \in 1 \dots N \mapsto \begin{cases} \mathrm{P}_{ij}^a = \mathrm{P}_{i(j-1)}^b, j = 0, S-1 \\ \mathrm{P}_{iS}^a = \mathrm{P}_{iS}^b + \mathrm{P}_{i(S-1)}^b \\ \mathrm{P}_{i0}^a = 0 \end{cases}$$

2) M^- – moves the matrix values down by 1: if $\mathrm{P}^a = \mathrm{M}^-\mathrm{P}^b$, then

$$\forall i \in 1 \dots N \mapsto \begin{cases} \mathrm{P}_{ij}^{a} = \mathrm{P}_{i(j+1)}^{b}, j = 0, S-1 \\ \mathrm{P}_{i0}^{a} = \mathrm{P}_{i0}^{b} + \mathrm{P}_{i1}^{b} \\ \mathrm{P}_{iS}^{a} = 0 \end{cases}$$

3) M^0 – does not move matrix values.

Writing down cache state change operator T^{op} for operation, working with the memory region i, we get:

$$\mathrm{T}^{\mathrm{op}} = \rho_{i+} \circ \mathrm{M}^{+} + \rho_{i0} \circ \mathrm{M}^{0} + \rho_{i-} \circ \mathrm{M}^{-}$$

where:

1) ρ_{i+} – matrix of coefficients for loading a new line of i into the cache; this matrix consists of a nonzero column for the i -th vector, other coefficients are equal to zero;
2) ρ_{i0} – matrix of coefficients for saving cache state as it is;
3) ρ_{i-} – matrix of coefficients for evicting a line from the cache when loading a new line of the i area into the cache; this matrix consists of nonzero columns for all vectors except the i -th.

An example of applying operator T^{op} to cache state example above is shown in Table 4. Operation op accesses memory region a_1 so one line of a_1 is loaded into cache and one line of a_2 or a_3 is evicted from the cache.

Table 3. Result of applying operator T^{op} to cache state from Table 3 when op works with memory region a_1 ($i = 1$)

j	$\mathbf{a_1}$	$\mathbf{a_2}$	$\mathbf{a_3}$
S	0%	0%	0%
$S-1$	0%	90%+10%·$\frac{1}{S}$	0%
$S-2$	0%	10%·$\frac{S-1}{S}$	0%
…	…	…	…
1	100%	0%	10%·$\frac{S-1}{S}$
0	0%	0%	90%+10%·$\frac{1}{S}$

7. Conclusion

Publications analysis showed that there is no unified solution to the problem of improving cache usage of compiled programs. In this paper, we propose a research approach, which can lead to a solution to this problem in compilers.

Список литературы / References

[1] В.Е. Шампаров, А.Л. Маркин. Механизм оптимизации Structure Splitting в составе компилятора для микропроцессоров Эльбрус. Программная инженерия, том 12, no. 2, 2021 г., стр. 82-88 / V. E. Shamparov and A. L. Markin. Structure splitting for elbrus processor compiler. Software Engineering, vol. 12, no. 2, 2021, pp. 82-88 (in Russian).
[2] C. Lattner. Macroscopic Data Structure Analysis and Optimization. Ph.D. dissertation, Computer Science Dept., University of Illinois at Urbana-Champaign, 2005, 225 p.
[3] C. Haine. Estimation d'efficacité et restructuration automatisées de noyaux de calcul. (Kernel optimization by layout restructuring). Ph.D. dissertation, University of Bordeaux, France, 2017, 114 p.
[4] M. Hagog and C. Tice. Cache aware data layout reorganization optimization in gcc. In Proc. of the GCC Developers' Summit, 2005, pp. 69-92.
[5] S. Ghosh, M. Martonosi, and S. Malik. Cache miss equations: A compiler framework for analyzing and tuning memory behavior. ACM Transactions on Programming Languages and Systems, vol. 21, no. 4, 1999, pp. 703-746.
[6] B.B. Fraguela, R. Doallo, and E.L. Zapata. Probabilistic miss equations: Evaluating memory hierarchy performance. IEEE Transactions on Computers, vol. 52, no. 3, 2003, pp. 321-336

[7] D. Andrade. Systematic analysis of the cache behavior of irregular codes. Ph.D. dissertation, Department of Electronics and Systems, University of A Coruña, Spain, 2007, 165 p.

[8] M. Kowarschik and C. Weiß. An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms. Lecture Notes in Computer Science, vol, 2625, 2003, pp. 213-232.

Информация об авторах / Information about authors

Виктор Евгеньевич ШАМПАРОВ, аспирант МФТИ, программист АО «МЦСТ». Научные интересы: компиляторы, оптимизация кода, VLIW-архитектура.

Viktor SHAMPAROV, PhD student at MIPT, software engineer at MCST. Research interests: compilers, code optimization, VLIW architecture.

Мурад Искендер-оглы НЕЙМАН-ЗАДЕ, к.ф.-м.н., доцент. Научные интересы: компиляторы, оптимизация кода, VLIW-архитектура.

Murad NEIMAN-ZADE, PhD in mathematics, associated professor. Research interests: compilers, code optimization, VLIW architecture.

DOI: 10.15514/ISPRAS-2021-33(3)-5

Localized Lama Gradual Typing

V.S. Kryshtapovich, ORCID: 0000-0002-3941-6201 <kry127@yandex.ru>
ITMO University,
Kronverksky Pr. 49, bldg. A, St. Petersburg, 197101, Russia

Abstract. Gradual typing is a modern approach for combining benefits of static typing and dynamic typing. Although scientific research aim for soundness of type systems, many of languages intentionally make their type system unsound for speeding up performance. This paper describes an implementation of a dialect for Lama programming language that supports gradual typing with explicit annotation of dangerous parts of code. The target of current implementation is to grant type safety to programs while keeping their power of untyped expressiveness. This paper covers implementation issues and properties of created type system. Finally, some perspectives on improving precision and soundness of type system are discussed.

Keywords: programming languages; gradual typing; type safety; cast calculus

For citation: Kryshtapovich V.S. Localized Lama Gradual Typing. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 61-76. DOI: 10.15514/ISPRAS-2021-33(3)-5.

Локализованное применение частичной типизации

В.С. Крыштапович, ORCID: 0000-0002-3941-6201 <kry127@yandex.ru>
Университет ИТМО
197101, г. Санкт-Петербург, Кронверкский проспект, д.49

Аннотация. Частичная типизация – это современный подход для сочетания преимуществ статической и динамической типизации. Но несмотря на то, что научные исследования направлены на корректность систем типов, многие языки намеренно делают систему некорректной для ускорения производительности. Данная работа посвящена реализации диалекта языка Лама, который поддерживает частичную типизацию для явно указанных участков кода. Целью реализации является сочетание двух подходов: обеспечение типобезопасности в одних участках кода и производительность языка в других участках кода. Статья раскрывает детали реализации и свойства полученной системы типов. Также рассматриваются способы улучшения полноты и корректности полученной системы типов.

Ключевые слова: языки программирования; частичная типизация; системы типов; исчисление преобразований

Для цитирования: Крыштапович В.С. Локализованное применение частичной типизации. Труды ИСП РАН, том 33, вып. 3, 2021 г., стр. 61-76 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(3)–5.

1. Introduction

There are different approaches of type system implementation. Static type systems are well-known for preventing many undesired behaviors of the program at compile time by reasoning about possible values that expression may or may not take (e.g., Java, Haskell, ...). On the opposite side, dynamic type systems are well-known to be the most flexible type systems – low compilation prerequisites and delegation type safety to runtime allows rapid development and prototyping (e.g., Python, Racket, ...).

There is a combination of both mentioned approaches named «Gradual Typing». This technique of program typing drained a lot of attention since the article of Siek and Taha [1] was published. Article presents sound type system for Lisp dialect which represents partially typed functional language. The presence of sound system for this model language gave rise to lots of research in this field.

But practical application of sound gradual systems is still questionable because of the performance issues [2]. The key purpose of this article is to see how gradual typing and explicit unsafe code annotations can be integrated with each other as native language syntax. The desired result is to acquire language that allows programmer to control trade-off between performance and type safety. The Lama [3] version 1.00 will be used as our target language of research.

Let us imagine typical Python code, and most probably it would be some untyped piece of code. Surprisingly or not, only $\mathbf{3.8}\%$ of repositories have type annotations by 2020 year [4]. But the idea of gradual typing is powerful: let programmers add static type information expression by expression in the code. Thus, we can step-by-step convert untyped code into fully statically typed code with corresponding static guarantees.

This is so called *gradual* typing: on the one hand we have power of static annotations preventing us from misusing functions, modules and preserving contracts. On the other hand, we shut down static type system whenever we choke down with abyss of static type errors.

The most important result of original article [1] was soundness of gradual type system. This was reached by exploiting *cast calculus* and rewriting original program with casts. The cast can be imagined as the bridge that value surpass during runtime from untyped part of code to typed part of code. This kind of "bridge" is annotated with static type and value should conform to it while moving from less typed part of code to more typed part of code. So, the main idea is to correctly insert casts and yield a program with *soundness* property.

1) If program does not typecheck, the program execution path may stuck with static type error emerged at runtime. (If there is a possibility to launch untyped programms at all)
2) If program typechecks, it can produce only dynamic type error or cast errors. *No errors involving incompatibility of static types may occur at runtime.*

In other words, if program is accepted by sound typechecker it can never fail contracts that was given to expression by the programmers in the form of types. For instance, you cannot acquire string value in variable statically typed as integer.

Gradual typing has been presented in several languages and in various forms, such as:

1) Python [5, 6] (MyPy [7] and PyType [8] projects);
2) Typed Racket [9];
3) JavaScript: TypeScript;
4) C#4.0 with dynamic keyword.

Although they are all have gradual typing property (in the sense, that not all objects have known type at compile time), their implementation of gradual type system has strong differences. Some of them are compiled into dynamic target language, such as TypeScript program is converted to pure JavaScript after compilation. Some of them are static by the nature as C# and then bring up a «dynamic» keyword which marks that object has unknown type until runtime. Some of them incorporate optional typing annotations and leave them alone for documentation and external tools (linters, typecheckers, IDE) as Python do.

The most noticeable state-of-the-art of gradual typing: every industrial-level language doesn't care much about soundness of the type system. This is because of the performance issues. Some real programs exhibit slowdown over $\mathbf{20}\times$, likely rendering them unusable for their actual purpose. To increase performance many of them reduce number of dynamic casts or remove them at all. This leads to trade-off between soundness and performance of gradually typed language.

To sum up, gradual typing provides mechanism to check program correctness having this pros and cons:

- Types can be added ad hoc by the programmers.

- Gradual type system can be sound in certain languages (more frequently academic ones).
- Dynamic typechecks is giving significant overhead at runtime.

No doubt: looking at the diversity of implementation and approaches it is interesting to look at the result of implementation of gradual typing in the language with different model of computation and semantics. We will test some new syntax conceptions experimenting with Lama programming language.

$\boldsymbol{\lambda^{\alpha}\ \mathcal{M}^{\alpha}}$ is a programming language developed by JetBrains Research for educational purposes as an exemplary language to introduce the domain of programming languages, compilers and tools [3]. The most noticeable property of this language that it is fundamentally untyped. The reference manual says that the lack of a type system is an intentional decision which allows to show the unchained diversity of runtime behaviors. But at the same time manual says that the language can be used in future as a raw substrate to apply various ways of software verification (including type systems) on [10]. So why wouldn't we try to implement some kind of type system upon it?

In our work we will test new approach of combining parts of code where different rules of static verification are applied: some parts of code will be gradually typed, and some parts of code will be left untyped. The expected result is programming language that can mix two types of code:

- with semantics that respects type safety in necessary parts of the code (e.g., sound);
- with original semantics without overheads.

This should allow programmer to choose what parts of program should be gradually typed, and what parts of program should not be typed.

Another expected result is producing a program with decreasing speed of execution of gradually typed code. The slowdown may be arbitrary, but we will try to reproduce results from article (at least $\times\ \mathbf{2}$ slowdown).

2. Examples

To give reader a proof of concept we should consider concrete syntax and pragmatics of the pieces of code written in Lama and describe how to introduce types into our language and what they expected to do. Normally, code in Lama looks as follows. No types, just anarchy of undefined behaviors:

```
fun closure(x) {
   fun (y) {
      2*x*y
   }
}
```

In this example we see function that takes `x` as an argument and returns function that multiplies input argument by `2 * x`. One expects it to be used upon integers, but Lama won't restrict to call function like `closure ("Hello, ") ("world!")` and pray for runtime not to fall. We can use type annotations to designate our intentions about the code like so:

```
fun closure(x :: Int) :: Int -> Int {
   fun (y :: Int) :: Int {
      2*x*y
   }
}
```

What do we expect from introduced type annotations?

- Backward compatibility with existing untyped source code;
- Static compile-time checks;
- Dynamic runtime checks.

Moreover, we would like something like type inference.

```
fun closure(x :: Int) {
   fun (y :: Int) {
        2*x*y
    }
}
```

If `x` and `y` have known at compile time types, then type of the functions can be inferred: inner function has type `Int -> Int`, and outer function has type `Int -> Int -> Int`.

Moreover, Lama nowadays supports operations only with integer constants (`Int`). If we take a closer look to the untyped example, it can be inferred that x should have type `Int`, `y` should have type `Int`, because they are used in expression like `2 * x * y`, and further infer function types, which makes this concrete piece of code fully typed.

At first glance type inference seems to be contradictory with backward compatibility. That is because some of the untyped expressions become implicitly typed, as first example do. Thus, runtime typechecks are inserted in parts of code that were initially untyped, which affects their semantics. Thankfully, the developers of Lama left regression tests that check backward compatibility. So we can bring up type inference features with awareness on backward compatibility.

Another example of typing Lama programs is pattern matching

```
fun processA(a) {
    case a of
        A (0) -> "1"
    | A (x) -> "2"
    esac
}
```

The `A(0)` notation is so called S-expression [11]. *Quick Lama-specific introduction*: you can consider S-expression as labeled array of arbitrary values. Name should be capitalized, number of values is not bounded. Two S-expression labels are considered equal in Lama if their five first letters are the same, so `Branch(Leaf, Leaf, 3)` and `Branc(Leaf, Leaf, 3)` are equal S-expressions. By the way, Leaf is nested S-expression with zero values in it, so brackets are optional for zero-arity S-expressions.

Side note: S-exprs like `Int` and `Str` has type `Int :: Int()` and `Str :: Str()` to distinguish them from integers `(3 :: Int)` and strings `("smoothie" :: Str)` type.

Back to our `processA` function, we can see, that if a matches `A(0)`, then “1” produced, for other value `A(smth)` where smth is not 0 we would get “2” produced by the function. If we call `processA(B(0))` we would get runtime error from pattern matching. So, other things that we would like from our type system are:

- Check that all branches cover matching expressions. E.g. no runtime error would occur in pattern matching.
- Check branches that would never succeed: either covered by previous branch or just don’t conform to matching expression.

For example, type system should reject this Lama program:

```
local foo = fun (x :: A(Int)) {
case x of
        A (0) -> "1"
    | A (x, y) -> "3" anything
esac
};
```

Here type system can check two things. First of all, `x = A(1)` won’t meet any branch, so not whole possible values of x are covered. And the second: `A(x, y)` would never match values with type `x :: A(Int)`.

Also note, that functions in Lama has beautiful sugar that combines pattern matching, that can be used to check input arguments:

```
public fun id2 (Abc (x, y)) :: ? {
    x
}
write(id2(Abc(6, 8)));
write(id2(Xyz(6, 8))); -- static fail
```

The last example that we should consider relates to runtime checks. Let's look at this simple piece of code:

```
fun intStringer(x :: Int) {
  x.string
}
local dyn :: ? = "Can be anything";
dyn := intStringer; -- forget type
dyn("input") -- should it fail?
```

At first glance it is unclear, where is the problem, because `dyn("input")` would reduce to `"input".string` and then to `"input"`. Do we actually care about function, that originally takes Int and store it at runtime? The answer is yes:

```
fun intStringer(x :: Int) {
  (x + 1).string
}
```

Of course, if we try to reduce `dyn("input")` we get `"input" + 1`, and then we will now end up with runtime error of casting `"input"` to `Int`. But what is the real cause of this error, whom to blame [12] [13] [14] for this mess – a plus operator, or input to the `intStringer`? That is why we should check function arguments wrapping them with appropriate dynamic casts. So, if follow blame ideology in both implementations `dyn("input")` would fail with the same reason: function expected `Int`, but given `Str`. But this solution could lead to extra checks and execution speed decrease.

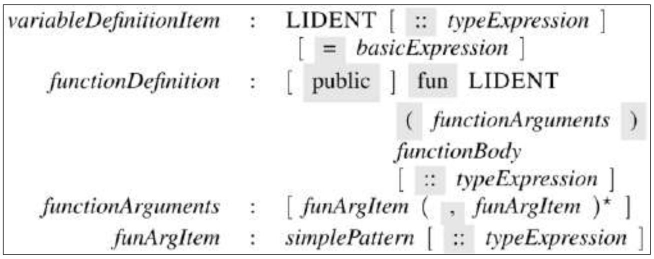
After seeing quite a bit of examples we conclude that these features would be handful in untyped Lama language. Typechecker would decrease number of errors in code made by programmers and runtime casts would inform programmer when untyped code does not conform contracts of the typed code. In next section we will define syntax of gradual types and their semantics.

3. Type Annotations Definition and Semantics

Gradual typing assumes that user annotates parts of the program with certain type. So, we should provide this feature in Lama compiler. Syntax rules have been described in Lama specification. We will fix them a little bit, because we only change variable definition (global and scope), function definition and their input parameters, look at p. 10 [10] for more detailed language syntax specification.

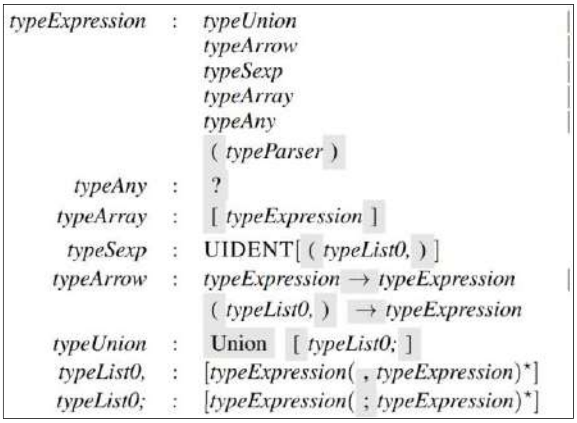
We slightly modified this nonterminals on the fig. 1: just put static type annotations to variable definition and function definition. Also, nonterminal *functionArguments* was slightly changed in comparison to specification to respect pattern matching sugar. This sugar is not included in concrete syntax definition for some reason. Other nonterminals assumed taken from section "Concrete syntax and semantics" of specification [10].

The definition of type annotations *typeExpression* is presented on the fig. 2. It semantic (see τ in fig. 3) is almost straightforward: syntax rule *typeAny* corresponds to dynamic type `TAny`, which can hold arbitrary value. Syntax rule *typeArray* corresponds to the array `TArr` of certain type. Syntax rule *typeSexp* corresponds to `TSexp` with parsed `UIDENT` as the name of S-expression and list of types forming type of S-expression. Syntax rule *typeArrow* corresponds to arrow `TLambda`. Note that input arguments can vary from zero to arbitrary amount. Syntax rule *typeUnion* corresponds to `TUnion` and lists all types that value can conform.



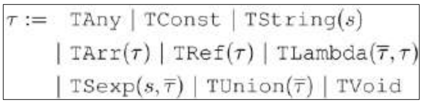
```
variableDefinitionItem  :  LIDENT [ :: typeExpression ]
                             [ = basicExpression ]
    functionDefinition  :  [ public ] fun LIDENT
                                     ( functionArguments )
                                     functionBody
                                     [ :: typeExpression ]
     functionArguments  :  [ funArgItem ( , funArgItem )* ]
            funArgItem  :  simplePattern [ :: typeExpression ]
```

Fig. 1. Syntax extension: scope expressions with type annotations



```
typeExpression  :  typeUnion                                  |
                   typeArrow                                  |
                   typeSexp                                   |
                   typeArray                                  |
                   typeAny                                    |
                   ( typeParser )
       typeAny  :  ?
     typeArray  :  [ typeExpression ]
      typeSexp  :  UIDENT[ ( typeList0, ) ]
     typeArrow  :  typeExpression → typeExpression            |
                   ( typeList0, ) → typeExpression
     typeUnion  :  Union [ typeList0; ]
    typeList0,  :  [typeExpression( , typeExpression)*]
    typeList0;  :  [typeExpression( ; typeExpression)*]
```

Fig. 2. Typing expression syntax



$$
\begin{aligned}
\tau := \; & \texttt{TAny} \mid \texttt{TConst} \mid \texttt{TString}(s) \\
& \mid \texttt{TArr}(\tau) \mid \texttt{TRef}(\tau) \mid \texttt{TLambda}(\overline{\tau}, \tau) \\
& \mid \texttt{TSexp}(s, \overline{\tau}) \mid \texttt{TUnion}(\overline{\tau}) \mid \texttt{TVoid}
\end{aligned}
$$

Fig. 3. Typing expression semantics

Only *typeSexp* rule with zero arity has non straightforward semantics. If type parameters of S-expression type are not presented, and `UIDENT` is one of the

- `Int` – corresponds to integers `τ = TInteger;`
- `Str` – corresponds to strings `τ = TString;`
- `Void` – corresponds to empty set of values `τ = TVoid;`
- otherwise, it corresponds to S-expression with specified name and no arguments.

If *typeSexp* is specified with brackets, it has straightforward semantics of S-expression. So, for example, `Cons` and `Cons()` has the same semantics of `TSexp("Cons")`, but semantics of `Int` and `Int()` are different as integer and S-expression types: `TConst` and `TSexp("Int")` correspondingly.

4. Typechecking Rules

The typechecking is inserted in the compilation pipeline directly after AST (Abstract Syntax Tree) representation of the program has been built (see “src/Language.ml” and “src/Driver.ml” in Lama source code [3]). The typechecking simultaneously performs the following procedures with AST: type checking, type inference and cast insertion.

For detailed description of this three type system problems we need to describe such classes as *expressions*, *values*, *patterns* and *types* of the language.

- ***τ*** is class of type expressions (see fig. 3);
- ***e*** is class of expressions (see fig. 4);

- $\boldsymbol{v}$ is class of values (see fig. 5);
- $\boldsymbol{p}$ is class of patterns (see fig. 6).

$$
\begin{aligned}
e :=\;& \\
& \texttt{Const}(i) \mid \texttt{Arr}(\overline{e}) \mid \texttt{String}(s) \mid \texttt{Sexp}(s, \overline{e}) \\
& \mid \texttt{Var}(s) \mid \texttt{Ref}(s) \mid \texttt{Cast}(e, \tau) \mid \texttt{Binop}(s, e, e) \\
& \mid \texttt{Elem}(e, e) \mid \texttt{ElemRef}(e, e) \mid \texttt{Length(e)} \\
& \mid \texttt{StringVal}(e) \mid \texttt{Call}(e, \overline{e}) \mid \texttt{Assign}(e, e) \\
& \mid \texttt{Seq}(e, e) \mid \texttt{Skip} \mid \texttt{If}(e, e, e) \mid \texttt{While}(e, e) \\
& \mid \texttt{Repeat}(e, e) \mid \texttt{Case}(e, \overline{(p, e)}) \mid \texttt{Return}(e) \\
& \mid \texttt{Ignore}(e) \mid \texttt{Scope}(\overline{(s, e)}, e) \mid \texttt{Lambda}(\overline{(s, \tau)}, e, \tau)
\end{aligned}
$$

Fig. 4. Lama expression class

$$
\begin{aligned}
v :=\;& \\
& \texttt{VVar}(s) \mid \texttt{VElem}(v, i) \mid \texttt{VInt}(i) \mid \texttt{VString}(s) \\
& \mid \texttt{VArray}(\overline{v}) \mid \texttt{VSexp}(s, \overline{v}) \mid \texttt{VClosure}(\overline{s}, e, \sigma) \\
& \mid \texttt{VFunRef}(s, \overline{s}, e, i) \mid \texttt{VBuiltin}(s) \mid \texttt{VCast}(v, \tau)
\end{aligned}
$$

Fig. 5. Lama value class

$$
\begin{aligned}
p :=\;& \texttt{PWildcard} \mid \texttt{PConst}(i) \mid \texttt{PString}(s) \\
& \mid \texttt{PArray}(\overline{p}) \mid \texttt{PSexp}(s, \overline{p}) \mid \texttt{Named}(s, p) \\
& \mid \texttt{PBoxed} \mid \texttt{PUnBoxed} \mid \texttt{PStringTag} \\
& \mid \texttt{PSexpTag} \mid \texttt{PArrayTag} \mid \texttt{PClosureTag}
\end{aligned}
$$

Fig. 6. Lama pattern class

There is also additional classes that are built-in of implementation language (OCaml). They can be considered as value class:

- $\boldsymbol{i}$ – integer;
- $\boldsymbol{s}$ – string.

Let us denote set of variables by $\mathbb{V}$, which represented by OCaml string $\boldsymbol{s}$, and set of types $\mathbb{T}$. We should think about $\mathbb{T}$ wider, that types induced by type constructors of fig. 3. In other words, some type $\boldsymbol{\gamma} \in \mathbb{T}$ may not be expressed with type constructors.

If we simplify process of compilation a little bit and ignore external symbol resolvance, Lama parser generates expression of $\boldsymbol{e}$ class without `Cast` constructors, i.e. pure untyped Lama expression. Notice, that expression can also contain patterns $\boldsymbol{p}$ due to pattern matching in `Case` expression.

Then, we have some options how to deal with generated AST. The trivial option is to left expression untouched and get the semantics of classic Lama language. The first option is trying to statically typecheck expression. If we succeed to acquire static type of program represented as whole expression, we can conclude that there is no static misuse of typed expressions. The second option is to transform AST to insert casts where values are passing from untyped parts of code to typed one. We will build up an algorithm that makes static typechecking and dynamic cast insertion simultaneously. For type checking we need to answer a question: does some type $\boldsymbol{\tau\_1} \in \mathbb{T}$ conforms to other type $\boldsymbol{\tau\_2} \in \mathbb{T}$? That answer is given by $\sim$ relationship named "conforms" which is constructed by axioms presented at fig. 7.

We should put additional attention to `TUnion` type and its rules. It denotes type that holds all possible values which can hold its constituent types. It is naturally coming from such language expressions as `If`, `Case` and `Return`. We have chosen set-theoretic approach on typing such expressions. Although there is an algorithm for union contraction, set-theoretic approach for type combination may lead to certain drawback in correctness and decreased performance during compile time.

Speaking about correctness: rules `ConfTUnion1` and `ConfTUnion2` generally cannot proof that two type representation conform to each other if they really do. Thus, the lack of completeness is reflected in false positives generated by static typechecker. That means correct type-annotated Lama expressions can be rejected by typechecker with such relationship definition $\sim$. This is a common illness of every static typechecker because we would like to check nontrivial property of the code: to be statically correct [15].

$$\frac{}{\tau \sim \texttt{TAny}}\ \texttt{[ConfTAny]}$$

$$\frac{}{\texttt{TAny} \sim \tau}\ \texttt{[ConfTAny2]}$$

$$\frac{\tau \sim \tau'}{\texttt{TArr}(\tau) \sim \texttt{TArr}(\tau')}\ \texttt{[ConfTArr]}$$

$$\frac{\tau \sim \tau'}{\texttt{TRef}(\tau) \sim \texttt{TRef}(\tau')}\ \texttt{[ConfTRef]}$$

$$\frac{s = s' \wedge \left[\bigwedge_{i=1}^{n} \tau_i \sim \tau_i'\right]}{\texttt{TSexp}(s, \tau_1 \ldots \tau_n) \sim \texttt{TSexp}(s', \tau_1' \ldots \tau_n')}\ \texttt{[ConfTSexp]}$$

$$\texttt{[ConfTLambda]}$$
$$\frac{\tau_r \sim \tau_r' \wedge \left[\bigwedge_{i=1}^{n} \tau_i' \sim \tau_i\right]}{\texttt{TLambda}(\tau_1 \ldots \tau_n, \tau_r) \sim \texttt{TLambda}(\tau_1' \ldots \tau_n', \tau_r')}$$

$$\frac{\bigwedge_{i=1}^{n} \tau_i \sim \tau'}{\texttt{TUnion}(\tau_1 \ldots \tau_n) \sim \tau'}\ \texttt{[ConfTUnion1]}$$

$$\frac{\bigvee_{i=1}^{n} \tau \sim \tau_i'}{\tau \sim \texttt{TUnion}(\tau_1' \ldots \tau_n')}\ \texttt{[ConfTUnion2]}$$

$$\frac{\tau = \tau'}{\tau \sim \tau'}\ \texttt{[ConfTGround]}$$

Fig. 7. Rules of conformance to the other type

But the good news is that no type intersections `TIntersection` or type subtractions `TSubstraction` are coming – we try to avoid them when building type system for Lama.

Now we can make an analogy of $\sim$ relation for expression $\boldsymbol{e}$ and type $\boldsymbol{\tau}$. But instead we will be inferring type of expression. To start with something simple let's define type inference for patterns (see fig. 8).

Notice, that we infer both lower and upper bound for pattern type. This interval style inference of patterns is crucial for analyzing case expressions. Let's denote $\boldsymbol{\tau_l(p)} \in \mathbb{T}$ for lower bound inferred type for pattern and $\boldsymbol{\tau_r(p)} \in \mathbb{T}$ for upper bound inferred type for pattern. Notation $\boldsymbol{\tau(p)}$ means theoretic set of all possible values that are captured by pattern $\boldsymbol{p}$. With the chosen type constructors and their semantics we can conclude:

- $\boldsymbol{\tau_r}$ is representing type that covers all possible values captured by pattern (upper bound);
- $\boldsymbol{\tau_l}$ is representing type that is covered by all possible values captured by pattern (lower bound).

For example, value `Suc(1)` has type `TSexp("Suc", TConst)`, but this value alone covers almost nothing, so `TVoid ⊏ {Suc(1)} ⊏ TSexp("Suc", TConst).`

Now we are ready to describe our main part of algorithm: type inference and cast insertion for Lama expressions. We will use such notation: $\boldsymbol{e} \mapsto \boldsymbol{e}'\colon \boldsymbol{\tau}$. That means that expression $\boldsymbol{e}$ has type $\boldsymbol{\tau}$, and cast insertion into that expression produces expression $\boldsymbol{e}'$, which has the same type $\boldsymbol{\tau}$. In addition, we have two types of contexts: $\boldsymbol{\Gamma}\colon \mathbb{V} \to \mathbb{T}$ for typing context of variables (which assigns types to variable typenames) and set of types $\boldsymbol{\Delta} \subset \mathbb{T}$ for collecting information about function return type. Then, typechecker by given context and collected return types produce another collection of return types (probably, bigger than the original), expression rewritten with casts and it's type. So, the full notation of this algorithm should be:

$$\boldsymbol{\Gamma}, \boldsymbol{\Delta} \vdash \boldsymbol{e} \mapsto \boldsymbol{\Delta}' \vdash \boldsymbol{e}'\colon \boldsymbol{\tau}.$$

$$\frac{}{\mathtt{TAny} \sqsubset \tau(\mathtt{PWildcard}) \sqsubset \mathtt{TAny}}\ \mathtt{[InferPWildcard]}$$

$$\frac{\tau_l \sqsubset \tau(p) \sqsubset \tau_r}{\tau_l \sqsubset \tau(\mathtt{PNamed(s,\ p)}) \sqsubset \tau_r}\ \mathtt{[InferPNamed]}$$

$$\frac{}{\mathtt{TVoid} \sqsubset \tau(\mathtt{PConst}(i)) \sqsubset \mathtt{TConst}}\ \mathtt{[InferPConst]}$$

$$\frac{}{\mathtt{TVoid} \sqsubset \tau(\mathtt{PString}(s)) \sqsubset \mathtt{TString}}\ \mathtt{[InferPString]}$$

$$\frac{}{\mathtt{TConst} \sqsubset \tau(\mathtt{PUnboxed}) \sqsubset \mathtt{TConst}}\ \mathtt{[InferPUnboxed]}$$

$$\frac{}{\mathtt{TString} \sqsubset \tau(\mathtt{PStringTag}) \sqsubset \mathtt{TString}}\ \mathtt{[InferPStrTag]}$$

$$\frac{}{\mathtt{TVoid} \sqsubset \tau(\mathtt{PSexpTag}) \sqsubset \mathtt{TAny}}\ \mathtt{[InferPSexpTag]}$$

$$\frac{}{\mathtt{TVoid} \sqsubset \tau(\mathtt{PClosureTag}) \sqsubset \mathtt{TAny}}\ \mathtt{[InferPClosureTag]}$$

$$\mathtt{[InferPBoxed]}$$

$$\frac{}{\mathtt{TUnion(TString,\ TArr(TAny))} \sqsubset \tau(\mathtt{PBoxed}) \sqsubset \mathtt{TAny}}$$

$$\mathtt{[InferPArrTag]}$$

$$\frac{}{\mathtt{TArr(TAny)} \sqsubset \tau(\mathtt{PStringTag}) \sqsubset \mathtt{TArr(TAny)}}$$

$$\mathtt{[InferPSexp]}$$

$$\frac{\tau_i \sqsubset \tau(p_i) \sqsubset \tau'_i}{\begin{array}{c}\mathtt{TSexp}(s, \tau_1 \dots \tau_n) \sqsubset \\ \tau(\mathtt{PSexp}(s, p_1 \dots p_n)) \sqsubset \mathtt{TSexp}(s, \tau'_1 \dots \tau'_n)\end{array}}$$

$$\mathtt{[InferPArray]}$$

$$\frac{\tau_i \sqsubset \tau(p_i) \sqsubset \tau'_i}{\begin{array}{c}\mathtt{TArr}(\mathtt{TUnion}(\tau_1 \dots \tau_n)) \sqsubset \\ \tau(\mathtt{PArr}(p_1 \dots p_n)) \sqsubset \mathtt{TArr}(\mathtt{TUnion}(\tau'_1 \dots \tau'_n))\end{array}}$$

Fig. 8. Rules of lower and upper bound type inference for patterns

Fig. 9 and 10 presenting all set of rules for type inference of Lama expression with $\mathbf{\Gamma}, \mathbf{\Delta} \vdash \boldsymbol{e} \mapsto \mathbf{\Delta}' \vdash \boldsymbol{e}' \colon \boldsymbol{\tau}$ notation used. Let us highlight some features about presented algorithm.

The set of return types for expression $\mathbf{\Delta}$ is initialized with $\varnothing$. Note, that initial context $\boldsymbol{\Gamma}$ maps every variable occurrence to type `TAny`. The typechecker does not check, is symbol is defined in upper scopes or correctly imported, but context is called to provide correct surrounding type information for expressions.

Notation $\boldsymbol{\tau} \in \langle \mathtt{TSexp}, \mathtt{TString}, \dots \rangle$ in rule `[InferLength]` means that $\boldsymbol{\tau}$'s top level constructor should be one of the listed in angle brackets.

In rule `InferCall` cast to `TAny` is optional. It is used in inference rules to be consistent with `InferCall3` rule which process call of the union type object.

Many of the rules can be simplified by removing $\mathbf{\Delta}$ because they do not change it, such as `InferArr` and `InferSexp`, et cetera. That is because they recompute $\mathbf{\Delta}$ for expressions that never change $\mathbf{\Delta}$ in correct Lama expressions. There are a few places where $\mathbf{\Delta}$ is useful: it is `InferLambda`, `InferReturn1` and `InferReturn2` rules. Notice, that we are inferring return type of the function just to acknowledge that it fits type declared by the user, the declared interface is not changing. But if the type is not specified by user, the inferred type for variable will be used implicitly.

Also notice rules in `InferCase`. First, we collect return types from the branches while dragging $\mathbf{\Delta}$ through the computation pipeline. The second one, look at notation $\boldsymbol{\Gamma} \cup \boldsymbol{\tau_\Gamma}\ (\boldsymbol{p\_i}\ )$ – it fulfills typing context with mapping of `PNamed` named pattern to its types. The $\boldsymbol{\tau_\Gamma}$ can be defined via $\boldsymbol{\tau_r}$ as follows:

$$\boldsymbol{\tau_\Gamma}(\boldsymbol{p}) = \begin{cases} \boldsymbol{\tau_\Gamma}(\boldsymbol{p}') \cup \{\boldsymbol{s}\colon \boldsymbol{\tau_r}(\boldsymbol{p})\} & \boldsymbol{p} = \texttt{PNamed}(\boldsymbol{s}, \boldsymbol{p}') \\ \bigcup_{\boldsymbol{i}=\boldsymbol{1}}^{\boldsymbol{n}} \boldsymbol{\tau_\Gamma}\,(\boldsymbol{p_i}) & \boldsymbol{p} = \texttt{PArr}(\boldsymbol{p_1}, \dots, \boldsymbol{p_n}) \\ \bigcup_{\boldsymbol{i}=\boldsymbol{1}}^{\boldsymbol{n}} \boldsymbol{\tau_\Gamma}\,(\boldsymbol{p_i}) & \boldsymbol{p} = \texttt{PSexp}(\boldsymbol{s}, \boldsymbol{p_1}, \dots, \boldsymbol{p_n}) \\ \varnothing & \textbf{otherwise} \end{cases}$$

$$\frac{}{\Gamma, \Delta \vdash \texttt{Const}(i) \mapsto \Delta \vdash \texttt{Const}(i) : \texttt{TConst}}\;[\texttt{InferConst}]$$

$$\frac{}{\Gamma, \Delta \vdash \texttt{String}(s) \mapsto \Delta \vdash \texttt{String}(s) : \texttt{TString}}\;[\texttt{InferString}]$$

$$\frac{\Delta_0 := \Delta \qquad \Gamma, \Delta_{i-1} \vdash e_i \mapsto \Delta_i \vdash e_i' : \tau_i}{\Gamma, \Delta \vdash \texttt{Arr}(e_1 \dots e_n) \mapsto \Delta_n \vdash \texttt{Arr}(\overline{e_i'}) : \texttt{TArr}(\texttt{TUnion}(\overline{\tau_i}))}\;[\texttt{InferArr}]$$

$$\frac{\Delta_0 := \Delta \qquad \Gamma, \Delta_{i-1} \vdash e_i \mapsto \Delta_i \vdash e_i' : \tau_i}{\Gamma, \Delta \vdash \texttt{Sexp}(s, e_1 \dots e_n) \mapsto \Delta_n \vdash \texttt{Sexp}(s, \overline{e_i'}) : \texttt{TSexp}(s, \overline{\tau_i})}\;[\texttt{InferSexp}]$$

$$\frac{\Gamma(s) = \tau}{\Gamma, \Delta \vdash \texttt{Var}(s) \mapsto \Delta \vdash \texttt{Var}(s) : \tau}\;[\texttt{InferVar}]$$

$$\frac{\Gamma(s) = \tau}{\Gamma, \Delta \vdash \texttt{Ref}(s) \mapsto \Delta \vdash \texttt{Ref}(s) : \texttt{TRef}(\tau)}\;[\texttt{InferRef}]$$

$$\frac{\Gamma, \Delta \vdash e_1 \mapsto \Delta_1 \vdash e_1' : \tau_1 \qquad \tau_1 \sim \texttt{TConst} \qquad \Gamma, \Delta_1 \vdash e_2 \mapsto \Delta_2 \vdash e_2' : \tau_2 \qquad \tau_2 \sim \texttt{TConst}}{\Gamma, \Delta \vdash \texttt{Binop}(s, e_1, e_2) \mapsto \Delta_2 \vdash \texttt{Binop}(s, \texttt{Cast}(e_1', \texttt{TConst}), \texttt{Cast}(e_2', \texttt{TConst})) : \texttt{TConst}}\;[\texttt{InferBinop}]$$

$$\frac{\Gamma, \Delta \vdash e_1 \mapsto \Delta_1 \vdash e_1' : \texttt{TAny} \qquad \Gamma, \Delta_1 \vdash e_2 \mapsto \Delta_2 \vdash e_2' : \tau_2 \qquad \tau_2 \sim \texttt{TConst}}{\Gamma, \Delta \vdash \texttt{Elem}(e_1, e_2) \mapsto \Delta_2 \vdash \texttt{Elem}(e_1', \texttt{Cast}(e_2', \texttt{TConst})) : \texttt{TAny}}\;[\texttt{InferElem}]$$

$$\frac{\Gamma, \Delta \vdash e_1 \mapsto \Delta_1 \vdash e_1' : \texttt{TStr} \qquad \Gamma, \Delta_1 \vdash e_2 \mapsto \Delta_2 \vdash e_2' : \tau_2 \qquad \tau_2 \sim \texttt{TConst}}{\Gamma, \Delta \vdash \texttt{Elem}(e_1, e_2) \mapsto \Delta_2 \vdash \texttt{Elem}(e_1', \texttt{Cast}(e_2', \texttt{TConst})) : \texttt{TConst}}\;[\texttt{InferElemOfStr}]$$

$$\frac{\Gamma, \Delta \vdash e_1 \mapsto \Delta_1 \vdash e_1' : \texttt{TArr}(\tau_1) \qquad \Gamma, \Delta_1 \vdash e_2 \mapsto \Delta_2 \vdash e_2' : \tau_2 \qquad \tau_2 \sim \texttt{TConst}}{\Gamma, \Delta \vdash \texttt{Elem}(e_1, e_2) \mapsto \Delta_2 \vdash \texttt{Elem}(e_1', \texttt{Cast}(e_2', \texttt{TConst})) : \tau_1}\;[\texttt{InferElemOfArr}]$$

$$\frac{\Gamma, \Delta \vdash e \mapsto \Delta_1 \vdash e' : \tau \qquad \tau \in \langle \texttt{TAny}, \texttt{TArr}, \texttt{TString}, \texttt{TSexp} \rangle}{\Gamma, \Delta \vdash \texttt{Length}(e) \mapsto \Delta_1 \vdash \texttt{Length}(e') : \texttt{TConst}}\;[\texttt{InferLength}]$$

$$\frac{\Gamma, \Delta \vdash e \mapsto \Delta_1 \vdash e' : \tau}{\Gamma, \Delta \vdash \texttt{StringVal}(e) \mapsto \Delta_1 \vdash \texttt{StringVal}(e') : \texttt{TString}}\;[\texttt{InferStringVal}]$$

$$\frac{\Gamma, \Delta \vdash f \mapsto \Delta_0 \vdash f' : \texttt{TAny} \qquad \Gamma, \Delta_{i-1} \vdash e_i \mapsto \Delta_i, e_i' : \tau_i}{\Gamma, \Delta \vdash \texttt{Call}(f, e_1 \dots e_n) \mapsto \Delta_n \vdash \texttt{Cast}(\texttt{Call}(f', \overline{e_i'}), \texttt{TAny}) : \texttt{TAny}}\;[\texttt{InferCall}]$$

$$\frac{\Gamma, \Delta \vdash f \mapsto \Delta_0 \vdash f' : \texttt{TLambda}(\gamma_1 \dots \gamma_m, \tau) \qquad m = n \qquad \Gamma, \Delta_{i-1} \vdash e_i \mapsto \Delta_i, e_i' : \tau_i \qquad \tau_i \sim \gamma_i}{\Gamma, \Delta \vdash \texttt{Call}(f, e_1 \dots e_n) \mapsto \Delta_n \vdash \texttt{Cast}(\texttt{Call}(f', \overline{\texttt{Cast}(e_i', \gamma_i)}), \tau) : \tau}\;[\texttt{InferCall2}]$$

$$\frac{\Gamma, \Delta \vdash f \mapsto \Delta_0 \vdash f' : \texttt{TUnion}(\gamma_1 \dots \gamma_m) \qquad \Gamma \cup \{x : \gamma_i\}, \Delta_{i-1} \vdash \texttt{Call}(x, \overline{e_j}^{i-1}) \mapsto \Delta_i, \texttt{Cast}(\texttt{Call}(x', \overline{e_j}^{i}), \tau_i) : \tau_i}{\Gamma, \Delta \vdash \texttt{Call}(f, e_1^0 \dots e_n^0) \mapsto \Delta_m \vdash \texttt{Cast}(\texttt{Call}(f', \overline{e_j}^{m}), \texttt{TUnion}(\overline{\tau_i})) : \texttt{TUnion}(\overline{\tau_i})}\;[\texttt{InferCall3}$$

$$\frac{\Gamma, \Delta \vdash r \mapsto \Delta_1 \vdash r' : \texttt{TAny} \qquad \Gamma, \Delta_1 \vdash e \mapsto \Delta_2 \vdash e' : \tau}{\Gamma, \Delta \vdash \texttt{Assign}(r, e) \mapsto \Delta_1 \vdash \texttt{Assign}(r', e') : \texttt{TAny}}\;[\texttt{InferAssign1}]$$

$$\frac{\Gamma, \Delta \vdash r \mapsto \Delta_1 \vdash r' : \texttt{TRef}(\rho) \qquad \Gamma, \Delta_1 \vdash e \mapsto \Delta_2 \vdash e' : \tau \qquad \tau \sim \rho}{\Gamma, \Delta \vdash \texttt{Assign}(r, e) \mapsto \Delta_1 \vdash \texttt{Assign}(r', e') : \rho}\;[\texttt{InferAssign2}]$$

Fig. 9. Rules of type inference and cast insertion

The third one about `InferCase` is that there is a check that all branches cover target type: $\boldsymbol{\omega} \sim \texttt{TUnion}(\overline{\boldsymbol{\tau_l}(\boldsymbol{p_i})})$. And the fourth: notice that each pattern is checked for code execution availability $\boldsymbol{\tau_r}(\boldsymbol{p_i}) \sim \boldsymbol{\omega}$, and at the same time we check that branch is not hidden by earlier branch $\boldsymbol{\tau_r}(\boldsymbol{p_i}) \nsim \boldsymbol{\tau_l}(\boldsymbol{p_j})$. According to inequalities

$$\boldsymbol{\tau_r}(\boldsymbol{p_i}) \sim \boldsymbol{\tau_l}(\boldsymbol{p_j}) \Rightarrow \boldsymbol{\tau}(\boldsymbol{p_i}) \sqsubset \boldsymbol{\tau_r}(\boldsymbol{p_i}) \sqsubset \boldsymbol{\tau_l}(\boldsymbol{p_j}) \sqsubset (\boldsymbol{p_j}).$$

$$\frac{\Gamma,\Delta \vdash e_1 \mapsto \Delta_1 \vdash e_1' : \sigma \qquad \Gamma,\Delta_1 \vdash e_2 \mapsto \Delta_2 \vdash e_2' : \tau}{\Gamma,\Delta \vdash \texttt{Seq}(e_1,e_2) \mapsto \Delta_2 \vdash \texttt{Seq}(e_1',e_2') : \tau}\ \texttt{[InferSeq]}$$

$$\frac{}{\Gamma,\Delta \vdash \texttt{Skip} \mapsto \Delta \vdash \texttt{Skip} : \texttt{TVoid}}\ \texttt{[InferSkip]}$$

$$\frac{\Gamma,\Delta \vdash c \mapsto \Delta_1 \vdash c' : \sigma \sim \texttt{TConst} \qquad \Gamma,\Delta_1 \vdash e_t \mapsto \Delta_2 \vdash e_t' : \tau_t \qquad \Gamma,\Delta_2 \vdash e_f \mapsto \Delta_3 \vdash e_f' : \tau_f}{\Gamma,\Delta \vdash \texttt{If}(c,e_t,e_f) \mapsto \Delta_3 \vdash \texttt{If}(\texttt{Cast}(c',\texttt{TConst}),e_t',e_f') : \texttt{TUnion}(\tau_t,\tau_f)}\ \texttt{[InferIf]}$$

$$\frac{\Gamma,\Delta \vdash c \mapsto \Delta_1 \vdash c' : \sigma \sim \texttt{TConst} \qquad \Gamma,\Delta_1 \vdash e \mapsto \Delta_2 \vdash e' : \tau}{\Gamma,\Delta \vdash \texttt{While}(c,e) \mapsto \Delta_2 \vdash \texttt{While}(\texttt{Cast}(c',\texttt{TConst}),e') : \texttt{TVoid}}\ \texttt{[InferWhile]}$$

$$\frac{\Gamma,\Delta \vdash c \mapsto \Delta_1 \vdash c' : \sigma \sim \texttt{TConst} \qquad \Gamma,\Delta_1 \vdash e \mapsto \Delta_2 \vdash e' : \tau}{\Gamma,\Delta \vdash \texttt{Repeat}(e,c) \mapsto \Delta_2 \vdash \texttt{Repeat}(e',\texttt{Cast}(c',\texttt{TConst})) : \texttt{TVoid}}\ \texttt{[InferRepeat]}$$

$$\frac{\begin{array}{l}\Gamma,\Delta \vdash m \mapsto \Delta_0 \vdash m' : \omega \\ \Gamma \cup \tau_\Gamma(p_i),\Delta_{i-1} \vdash e_i \mapsto \Delta_i \vdash e_i' : \tau_i \qquad \tau_r(p_i) \sim \omega \qquad \omega \sim \texttt{TUnion}(\overline{\tau_l(p_i)}) \qquad \forall j < i\ .\ \tau_r(p_i) \nsim \tau_l(p_j)\end{array}}{\Gamma,\Delta \vdash \texttt{Case}(m,(p_1,e_1)\ldots(p_n,e_n)) \mapsto \Delta_n \vdash \texttt{Case}(m',(p_1,e_1')\ldots(p_n,e_n')) : \texttt{TUnion}(\overline{\tau_i})}\ \texttt{[InferCase]}$$

$$\frac{}{\Gamma,\Delta \vdash \texttt{Return} \mapsto \Delta \cup \{\texttt{TVoid}\} \vdash \texttt{Return} : \texttt{TVoid}}\ \texttt{[InferReturn1]}$$

$$\frac{\Gamma,\Delta \vdash e \mapsto \Delta' \vdash e' : \tau}{\Gamma,\Delta \vdash \texttt{Return}(e) \mapsto \Delta' \cup \{\tau\} \vdash \texttt{Return}(e') : \texttt{TVoid}}\ \texttt{[InferReturn2]}$$

$$\frac{\Gamma,\Delta \vdash e \mapsto \Delta' \vdash e' : \tau}{\Gamma,\Delta \vdash \texttt{Ignore}(e) \mapsto \Delta' \vdash \texttt{Ignore}(e') : \texttt{TVoid}}\ \texttt{[InferIgnore]}$$

$$\frac{\Delta_0 := \Delta \qquad \Gamma \cup \{\overline{(s_i,\tau_i)}_{k=1}^{i-1}\},\Delta_{i-1} \vdash e_i \mapsto \Delta_i \vdash e_i' : \tau_i \qquad \Gamma \cup \{\overline{(s_i,\tau_i)}_{k=1}^{n}\},\Delta_n \vdash e \mapsto \Delta' \vdash e' : \omega}{\Gamma,\Delta \vdash \texttt{Scope}(\overline{(s_i,e_i)}_{i=1}^{n},e) \mapsto \Delta' \vdash \texttt{Scope}(\overline{(s_i,e_i')}_{i=1}^{n},e') : \omega}\ \texttt{[InferScope]}$$

$$\frac{\Gamma \cup \{\overline{(s_i,\sigma_i)}_{i=1}^{n}\},\varnothing \vdash e \mapsto \Delta',e' : \delta \qquad \texttt{TUnion}(\Delta' \cup \{\delta\}) \sim \tau}{\Gamma,\Delta \vdash \texttt{Lambda}(\overline{(s_i,\sigma_i)}_{i=1}^{n},e,\tau) \mapsto \Delta, \texttt{Lambda}(\overline{(s_i,\sigma_i)}_{i=1}^{n},e',\tau) : \texttt{TLambda}(\overline{\sigma_i},\tau)}\ \texttt{[InferLambda]}$$

Fig. 10. Rules of type inference and cast insertion (part II).

In other words, when expression holds, it is certain that pattern $\boldsymbol{p_i}$ was covered by more recent cover $\boldsymbol{p_j}$. In that way we eliminated the need of introduction of intersection or difference types in our type system. But it doesn't mean we cannot deal with intersection and difference types, see [17] or [18] for example of polymorphic type system that handles that.

The most complex is `[InferScope]` rule. It is intentionally simplified, because it's implementations is more subtle. Here it simply overwrites variable or function definition and updates context $\boldsymbol{\Gamma}$. But implementation also checks that previous usage is corresponding with current typing when no expression is provided to variable. But to describe that strictly we would need to introduce a class for declarations and this rule would get even more complex.

So, this rule lead to new language feature – type usage of expression inside the scope:

```
{
  f :: Int -> Str;
  g :: Int -> Int;
  f(g(0)); -- ok
  f(g(D(0))) -- error
};
{
  f :: D(Int) -> Str;
  g :: D(Int) -> D(Int);
  f(g(0)); -- error
```

```
      f(g(D(0))); -- ok
      {
        f :: [Int] -> Int;
        g :: Str -> [Int];
        f(g("hello, world"))
      }
    };
```

Other type checking rules either trivial or common in corresponding field of study [16], so we wouldn’t dive too deep into them. In next chapter we will discuss performance issues of our typechecking algorithm.

5. Cast Performance Analyzing

It is obvious that rules presented at fig.9 introduce new kind of expression `Cast`$(\boldsymbol{e}, \boldsymbol{\tau})$. It’s runtime semantics is simple: when expression $\boldsymbol{e}$ evaluates to value $\boldsymbol{v}$, we should check that value $\boldsymbol{v}$ corresponds to type $\boldsymbol{\tau}$. If $\boldsymbol{v}$ conforms to $\boldsymbol{e}$, the result of evaluation of `Cast`$(\boldsymbol{e}, \boldsymbol{\tau})$ is $\boldsymbol{v}$, otherwise cast error $\perp$ produced as the result.

Runtime check that value corresponds to some type may be time consumptive, especially when type and expression are complex and have big nestings. Thus, we can introduce and explicit syntax for parts of code where we wish not to insert casts like this:

```
    fun mod(x :: ?, m :: ?) :: ? {
      #NoTypecheck {
        (if x < 0 then 0-x else x fi) % m
      }
    }
```

Typechecker will see this annotation and completely ignore annotated part of code. The implementation of gradual typing for Lama offers us three options to maintain typechecking procedure:

- `#NoTypecheck` – drops AST from typechecking at all;
- `#StaticTypecheck` –disables cast insertion into AST, but static checks are still enabled;
- `#GradualTyping` –enables cast insertion into AST.

You can nest `#StaticTypecheck` and `#GradualTyping` annotations in order to enable or disable cast insertion while typechecking. But there is no point to nest type related information into `#NoTypecheck` annotation, because they would be completely ignored by typechecker.

Having all power of gradual types and unchained diversity of undefined behaviour, let’s user interpretation mode of Lama compiler to see the slowdown in the code execution. We will use sample code:

```
    fun fibonacci(k) {
      if   k == 0 then return  0
      elif k == 1 then return  1
      elif k <  0 then return -1
      else return fibonacci(k-1)
                + fibonacci(k-2) fi
    }
    write(fibonacci(read()))
```

It is not obvious where are the casts in this example, but in section 2 we have noticed that + operator coerces both its arguments to `Const` at runtime, so appropriate casts to `TConst` types from unknown type are inserted. Hence, this code is modeling situation of frequent value passage from untyped part of code to typed part of code.

We will compare this code wrapped in `#GradualTyping` which is the default, and `#NoTypecheck` annotations. The time measurement is performed with Unix time utility, thus compile time included in both measures.

```
+----+-----------+-----------+
| n  | Untyped   | Typed     |
+----+-----------+-----------+
| 10 | 0m 0,119s | 0m 0,092s |
| 11 | 0m 0,097s | 0m 0,079s |
| 12 | 0m 0,088s | 0m 0,087s |
| 13 | 0m 0,094s | 0m 0,093s |
| 14 | 0m 0,091s | 0m 0,095s |
| 15 | 0m 0,086s | 0m 0,090s |
| 16 | 0m 0,092s | 0m 0,094s |
| 17 | 0m 0,095s | 0m 0,088s |
| 18 | 0m 0,093s | 0m 0,100s |
| 19 | 0m 0,102s | 0m 0,105s |
| 20 | 0m 0,106s | 0m 0,125s |
| 21 | 0m 0,124s | 0m 0,124s |
| 22 | 0m 0,132s | 0m 0,154s |
| 23 | 0m 0,162s | 0m 0,192s |
| 24 | 0m 0,208s | 0m 0,279s |
| 25 | 0m 0,284s | 0m 0,389s |
| 26 | 0m 0,416s | 0m 0,581s |
| 27 | 0m 0,593s | 0m 0,878s |
| 28 | 0m 0,909s | 0m 1,363s |
| 29 | 0m 1,467s | 0m 2,179s |
| 30 | 0m 2,326s | 0m 3,561s |
| 31 | 0m 3,659s | 0m 5,796s |
| 32 | 0m 5,977s | 0m 9,469s |
| 33 | 0m 9,477s | 0m14,108s |
| 34 | 0m15,981s | 0m24,799s |
| 35 | 0m26,933s | 0m43,855s |
| 36 | 0m42,236s | 1m 7,766s |
| 37 | 1m12,161s | 1m49,319s |
| 38 | 1m53,534s | 3m 0,748s |
| 39 | 3m18,046s | 4m54,461s |
| 40 | 5m17,664s | 7m54,811s |
+----+-----------+-----------+
```

The average of slowdown $\boldsymbol{sd_n} = \frac{\boldsymbol{t_n}^{\text{Typed}}}{\boldsymbol{t_n}^{\text{Unyped}}}$ from the point of actual slowdown registered $\boldsymbol{n = 21}$ is:

$$\frac{\mathbf{1}}{\mathbf{20}}\sum_{\boldsymbol{n}=\mathbf{21}}^{\mathbf{40}} \boldsymbol{s\,d_n} = \frac{\mathbf{1}}{\mathbf{20}}\sum_{\boldsymbol{n}=\mathbf{21}}^{\mathbf{40}} \frac{\boldsymbol{t_n}^{\text{Typed}}}{\boldsymbol{t_n}^{\text{Unyped}}} \approx \mathbf{1.45}.$$

As we can see, section of code with active gradual typing runtime type checking exhibit almost $\times$ $\mathbf{1.5}$ slowdown. Thus, we have reproduced the result of an article [2] but in the case Lama semantics using this artificially small example.

6. Conclusion

We introduced type system with following properties:

- Monomorphic;
- Gradual.

It would be nice to introduce such features in type system as:

- Polymorphism;

- Recursive types.

In the future work it is desired to use type equations and Hindley-Milner style inference with unification algorithm as presented in [17] and [19].

It is worth to mention the reproduction of the result of a recent article about industrial-level languages that use gradual types unsoundly [2]. We have modeled the situation of values constantly transiting from untyped part to typed parts of program and expectedly acquired slowdown of execution.

In addition, we have provided a simple and powerful, yet dangerous, method of maintaining trade-off between type safety and execution performance: let programmer choose areas of code where he needs extra performance and where he needs static and runtime type safety guaranties, either with `#NoTypecheck`, or better with `#StaticTypecheck` and `#GradualTyping` annotations.

The idea goes further. It would be nice to introduce some other sections of static verification that programmers can apply at their taste. For instance, live variable analysis #LiveVarAnalysis, or memory access safety. Thus, programmer acquire framework with bunch of static verifiers and the ability to choose what guaranties is the most important at applied piece of code. To sum up, programmer maintains compilation time and acquires code with the needed guarantees unified in one syntax.

Even though the type system soundness is still questionable and should be proved or improved, several tests are added to codebase to check type system, including not compiling tests, runtime error tests and positive example tests. Introduced type system enhances coding experience and points out at least silly and obvious errors that programmers are frequently making. Moreover, Lama’s facility has been extended by logger to generate warning messages, mostly for case expression coverage.

The implementation of gradual typing for Lama language resides in personal repository within branch named “GraduLama” [20].

References

[1] Jeremy G. Siek and Walid Taha. Gradual Typing for Functional Languages. In Proc. of the Seventh Workshop on Scheme and Functional Programming, 2006, pp. 81-92.

[2] Cameron Moy, Phúc C. Nguyễn et al. Corpse reviver: sound and efficient gradual typing via contract verification. Proceedings of the ACM on Programming Languages, vil. 5, issue POPL, 2021, Article 53, 28 p.

[3] D. Boulytchev. JetBrains-Research/Lama source code. Available at https://github.com/JetBrains-Research/Lama, accessed 27/03/2021.

[4] Ingkarat Rak-amnouykit, Daniel McCrevan et al. Python 3 types in the wild: a tale of two type systems. In Proc of the 16th ACM SIGPLAN International Symposium on Dynamic Languages (DLS 2020), 2020, pp. 57-70.

[5] Guido van Rossum, Ivan Levkivskyi. PEP 483 – The Theory of Type Hints. Available at https://www.python.org/dev/peps/pep-0483/ Request timestamp: 27/03/2021.

[6] Guido van Rossum, Jukka Lehtosalo, Łukasz Langa. “PEP 484 – Type Hints. Available at https://www.python.org/dev/peps/pep-0484/, , accessed 27/03/2021.

[7] Jukka Lehtosalo et al. Mypy: Optional Static Typing for Python. Available at https://github.com/python/mypy, accessed 27/03/2021.

[8] Pytype: A static type analyzer for Python code. Available at https://github.com/google/pytype, accessed 27/03/2021.

[9] Sam Tobin-Hochstadt, Vincent St-Amour et al. The Typed Racket Guide. Available at https://docs.racket-lang.org/tsguide/index.html, accessed 27/03/2021.

[10] D. Boulytchev. Lama language specification v. 1.10. Available at https://github.com/JetBrains-Research/Lama/blob/1.10/lama-spec.pdf, accessed 27/03/2021.

[11] R. Rivest. S-Expressions., 4/05/1997. Available at http://people.csail.mit.edu/rivest/Sexp.txt, accessed 29/03/2021.

[12] Amal Ahmed, Dustin Jamneret al. Theorems for free for free: parametricity, with and without types. Proceedings of the ACM on Programming Languages, vol. 1, issue ICFP, 2017, Article 39, 28 p.

[13] Jack Williams, J. Garrett Morris, and Philip Wadler. The root cause of blame: contracts for intersection and union types. Proceedings of the ACM on Programming Languages, vol. 2, issue OOPSLA, 2018, Article 134, 29 pages.
[14] P. Wadler. A Complement to Blame. In Proc. of the 1st Summit on Advances in Programming Languages (SNAPL 2015), 2015, pp. 309-320.
[15] Henry Gordon Rice. Classes of recursively enumerable sets and their decision problems. Transactions of the American Mathematical Society, vol. 74, no. 2. 1953, pp. 358- 366.
[16] Benjamin C. Pierce. Types and Programming Languages. The MIT Press, 2002, 648 p.
[17] Giuseppe Castagna, Victor Lanvin et al. 2019. Gradual typing: a new perspective. Proceedings of the ACM on Programming Languages, vol. 3, issue POPL, 2019, Article 16, 32 p.
[18] Karla Ramírez Pulido, Jorge Luis Ortega-Arjona et al. Gradual Typing Using Union Typing with Records. Electronic Notes in Theoretical Computer Science, vol.354, 2020, pp. 171-186.
[19] Yusuke Miyazaki, Taro Sekiyama, and Atsushi Igarashi. 2019. Dynamic type inference for gradual Hindley–Milner typing. Proceedings of the ACM on Programming Languages, vol. 3, issue POPL, 2019, Article 18, 29 pp.
[20] V. Kryshtapovich. GraduLama source code Available at https://github.com/kry127/Lama/tree/gradulama, accessed 27/03/2021.

Информация об авторах / Information about authors

Виктор Сергеевич КРЫШТАПОВИЧ, студент магистратуры второго курса. Научные интересы: системы типов, базы данных.

Viktor Sergeevich KRYSHTAPOVICH, second year master's student. Research interests: type systems, databases.

DOI: 10.15514/ISPRAS-2021-33(3)-6

HTTP-Request Classification in Automatic Web Application Crawling

A.V. Lapkina, ORCID: 0000-0002-7249-7672 <amiriya@seclab.cs.msu.ru>
A.A. Petukhov, ORCID: 0000-0002-1427-2440 <petand@seclab.cs.msu.su>
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. The problem of automatic requests classification, as well as the problem of determining the routing rules for the requests on the server side, is directly connected with analysis of the user interface of dynamic web pages. This problem can be solved at the browser level, since it contains complete information about possible requests arising from interaction interaction between the user and the web application. In this paper, in order to extract the classification features, using data from the request execution context in the web client is suggested. A request context or a request trace is a collection of additional identification data that can be obtained by observing the web page JavaScript code execution or the user interface elements changes as a result of the interface elements activation. Such data, for example, include the position and the style of the element that caused the client request, the JavaScript function call stack, and the changes in the page's DOM tree after the request was initialized. In this study the implementation of the Chrome Developer Tools Protocol is used to solve the problem at the browser level and to automate the request trace selection.

Keywords: request classification; application crawling; dynamic web application; Chrome DevTools

For citation: Lapkina A.V., Petukhov A.A. HTTP-Request Classification in Automatic Web Application Crawling. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 77-86. DOI: 10.15514/ISPRAS-2021-33(3)-6.

Классификация HTTP-запросов к серверу в задаче автоматического обхода современных веб-приложений

А.В. Лапкина, ORCID: 0000-0002-7249-7672 <amiriya@seclab.cs.msu.ru>
А.А. Петухов, ORCID: 0000-0002-1427-2440 <petand@seclab.cs.msu.su>
Московский государственный университет имени М.В. Ломоносова,
119991, Россия, Москва, Ленинские горы, д. 1

Аннотация. Задача автоматической классификации запросов приложения, а также задача определения правил маршрутизации запросов на сервере напрямую связана с анализом пользовательского интерфейса динамических веб-страниц и может быть решена на уровне браузера, поскольку он содержит полную информацию о возможных запросах, возникающих при взаимодействии пользователя с каждой из страниц веб-приложения. В данной работе для решения поставленной задачи предлагается использовать данные из контекста выполнения запроса в веб-клиенте с целью выделения дополнительных признаков для классификации запросов. При этом в качестве контекста возникновения или трассы запроса рассматривается совокупность дополнительных идентификационных данных, которые можно получить, наблюдая за выполнением JavaScript-кода на веб-странице или за изменением элементов пользовательского интерфейса в результате активации интерфейсных элементов. К таким данным, например, можно отнести положение и стиль элемента, вызвавшего клиентский запрос, стек вызовов функций JavaScript и изменение в DOM-дереве страницы после запроса. В рамках данной работы для автоматизации выделения трасс запросов и их последующей классификации используется реализация протокола Chrome DevTools.

Ключевые слова: классификация запросов; динамические веб-приложения; автоматический обход приложений; протокол Chrome DevTools.

Для цитирования: Лапкина А.В., Петухов А.А. Классификация HTTP-запросов к серверу в задаче автоматического обхода современных веб-приложений. Труды ИСП РАН, том 33, вып. 3, 2021 г., стр. 77-86 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(3)–6.

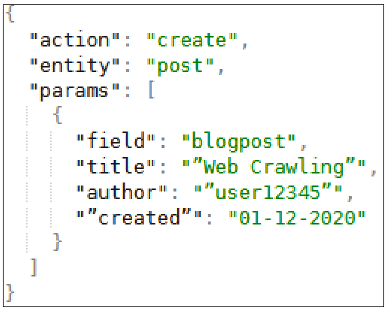
1. Introduction

The problem of classifying the requests from a web application client to a server and correlating them with application functions most often arises while analyzing applications using the black box method [1]. In the case of automated web application testing, the first step is collecting information about it. The structure of the application, its functions, input parameters, and types of requests are investigated. To collect this information, it is required to solve the problem of navigating the web application interface [2] – to find control elements automatically and activate them in order to cause client-server interaction.

To make sensible decisions in the navigation process, it is necessary to determine the results of triggering an action in the web interface: what HTTP request will be sent to the server, which function of the application will be executed, and how the state of the web application will change.

Since modern web interfaces are built with HTML and JavaScript technologies, the problem of navigating the application is reduced to analyzing the web interface (DOM and its visual presentation) and Javascript code. The latter implements the logic for the user and the server interaction: it processes user actions in the web interface, sends requests to the server and displays the results of their execution.

A particular problem in the process of navigating a web application is connected with correlating outgoing requests with the server-side actions of the web application. In traditional web applications, functions were uniquely addressed by URLs, so the problem of matching a request to an action on the server-side was trivial. In modern web applications, especially in single-page applications that implement the RPC concept (JSON RPC, XML RPC), URL can be the same for all server-side actions and the name of the function can be passed in the request parameters (see fig. 1). In order to correlate outgoing requests with the functions of the web application, it is necessary to extract a set of features from outgoing requests that uniquely identify functions of the web application.



```
{
  "action": "create",
  "entity": "post",
  "params": [
    {
      "field": "blogpost",
      "title": ""Web Crawling"",
      "author": ""user12345"",
      ""created"": "01-12-2020"
    }
  ]
}
```

Fig.1 Example of a POST-request with JSON in the body.
The called function is passed in the action field of the JSON structure.

Modern web applications use the concept of incoming requests routing [3]. To associate an incoming HTTP-request with a specific function or class in the application code, the developer defines the request routing rules: a table with predicates for HTTP-requests and function names. To process the next incoming request, the predicate for functions are calculated and the one that returns true will be called (the table is looked up from the top to the bottom until the first routing rule is triggered). The minimum set of request parameters, which values make the predicate true, will be called the discriminant for this request.

The set of specific values of the discriminant's parameters, that allow us to classify the request explicitly, is considered as the request key. In example presented on fig. 3 “*action*”: “*create*” pair is the request key.

For requests with body-parameters in the JSON format, we will consider the ones with Content-Type: *application/x-www-form-urlencoded* and take into account not only the name of the significant parameters, but also the nesting objects degree.

In the paper, sites that use ReactJS library and implement a web interface in accordance with the framework rules specified in the documentation [4, 5] are investigated. This decision was made as ReactJS is one of the most popular framework among sites written with JavaScript.

2. Related work

The problem of classifying web application requests consists of two main subproblems. The first one is connected with a strategy for obtaining a set of outgoing requests of the web application. The second one is connected with determining a strategy for the inductive extraction of classification features.

The strategy of building a set of outgoing requests determines the order the application interfaces would be processed, and the order controls (links, buttons, tabs, scrolling, etc.) implemented in the graphical interface will be activated. The problem of automatic construction of the outgoing requests set can be solved with web crawlers using such methods as depth-first crawling, breadth-first crawling, or random crawling [6]. However, these strategies are ineffective for modern dynamic web applications [7, 8].

In modern surveys, the use of dynamic analysis of the web applications [8, 9], as well as additional properties of the web pages is used to solve this problem and to improve the quality of crawling. For example, they consider using the analysis of the structure of the web page elements and their relative position, as well as the history of elements crawling [10] or the user interface segmentation [11].

Traditionally, such request data elements as a method, target URL, path and GET- or POST-parameters are used as features for classifying outgoing requests. However, in order to facilitate the requests classification, some studies consider additional indicators related to the state of the web application at the moment the request was initialized. For example, the state of a hierarchical finite state machine built in the process of navigating the application [12] or the state of the DOM model of the page [13] is used as such additional features.

3. General design

In this research, the problem of constructing a classifier of outgoing HTTP requests from a web client to a web application, that allows us to restore the routing model on the server-side of the application as part of automatic website crawling is solved by developing the algorithm of classification. The classifier receives a site to crawl as an input. The result of the tool's operation is a set of discriminants. Their combined values are the key to identify the action on the server-side for each request.

Automatic forms filling [14] and navigating the internal zone of a web application are not considered in this paper. The lattest means that if the access to the internal zone of the web application requires authentication [15] is not considered in this paper.

It was assumed that the context of outgoing requests may contain parameters that can be used as identification keys of the actions on the server side. If such parameters are found, it is suggested to use them as additional features for identifying the requests. It was also assumed that it is possible to build an iterative algorithm for classifying outgoing requests based on the found key parameters from the context. Since the URL is provided as an input, elements are activated gradually and the set of requests is formed iteratively. That is the reason it was decided to select the request features gradually.

In the next sections a description of the approach, implementation and results of experiments evaluating the validity of the assumption and the applicability of the approach are situated.

The task of selecting additional features requires a preliminary analytical study of the relations between user actions and the parameters of the request context. The research is performed for applications built on the basis of the ReactJS library [16]. The unified concept of programs that use this framework allows extrapolating the results obtained on the experimental set of sites to other sites based on this technology.

To establish the dependency between the context and the outgoing requests parameters, it is necessary to mark up some data manually and analyze the frequency of occurrence of significant context parameters types. If it turns out that there are such sets of parameters in the context that will have the same set of values (key), when two identical actions from the web interface are triggered, and which values would be different, when different actions are triggered, then we assume that there is a dependency between context parameters and classes of outgoing application requests.

In this paper, such context elements as the DOM state before the request was sent, the DOM state after the request was sent, the identifier of the DOM element node to which the called event handler belongs, the style of this element and the call frames array (the stack trace or the list of called functions with script identifiers and function positions) are examined.

The preliminary experimental reseach consists of several steps. As a first step the same action A is triggered via two different interface elements on the selected site performing interactions A_1 and A_2. Their traces T_1 and T_2 with the sets of parameters DOM_before_1, DOM_after_1, $node\_id_1$, css_1, $callframes_1$ and DOM_before_2, DOM_after_2, $node\_id_2$, css_2, $callframes_2$ are obtained. Then action B with trace T_3, different from actions A is triggered. After that, the values of the traces T_1, T_2 and T_3 are compared. The next step is to determine which parameters from the traces T_1 and T_2 have coinciding values and which parameters in pairs T_1, T_3 and T_2, T_3 have different values. After that the same comparison is made for other actions on the selected site and on other sites from the sites list. If results of the experiment show that there is a set of context parameters where with a high probability the same values are used for the same actions and where different actions result in different values, then they will be used as additional classification features.

Site list for experimental research was obtained from the Built With list [17] and the top sites of Coder Academy [18]. To select significant parameters,sites with different user interface complexity were used: from very complex (airbnb.com, facebook.com) to simpler ones (bbc.com, bleacherreport.com). The list also included sites with different routing schemes, such as routing by URL, routing based on query-parameters or routing based on body-parameters of the POST-requests. These requirements were intended to provide better coverage of various site types used on the Internet.

The experiment of analyzing dependency between significant context parameters and user actions was carried out on 20 target sites. The results are presented in Table 1 and Table 2.

Table 1. Percentage of coincidence between actions and context parameters for identical actions

DOM before action	**DOM after action**	**node id**	**css**	**callframes**
58%	54%	80%	65%	96%

Table2. Percentage of difference between actions and context parameters for different actions

DOM before action	**DOM after action**	**node id**	**css**	**callframes**
81%	92%	99%	73%	100%

The experiment results show that the strongest dependency corresponds to the callframes parameter. In this regard, it was decided to use the callframes array from the request context to classify requests to the server in addition to such request's attributes as its method, URL, path, query-parameters and body-parameters for POST-requests.

To validate the suggested approach, a classification algorithm was composed and tested. It receives a site for processing as an input, and produces a set of request's discriminants as an output.

4. Classification Algorithm

The request classification algorithm implements the idea of inductive constructing a set of significant features. An example of the basic algorithm processing two user events A and B is presented below.

Data structures used:

VP ($valuable\ parameters$): a set of significant request parameters. Consists of elements in the form ($param\_name$: [$val1, val2, val3, \ldots$]. Initially $VP = \emptyset$.

HP ($hint\ parameters$): a set of possibly significant parameters. $HP = \emptyset$.

NVP ($not\ valuable\ parameters$): a set of non-significant query parameters. Initially $NVP = \emptyset$.

AP ($all\ parameters$): set of all request parameters. Consists of elements in the form ($param\_name$: (val_1: $counter\_1$, val_2: $counter\_2$), where $param\_name$ is the name of the parameter, val_i is the i-th value of this parameter, $counter\_i$ is the number of times that the value of the $param\_name$ parameter has been encountered with the value val_i }. Initially $AP = \emptyset$.

RS ($request\ schemes$): A set of application request schemes. Each request scheme is a structure with fields containing the method, hostname, path, callframes, and the names of the get and post parameters. $RS = \emptyset$.

$trace, trace2$: the trace of the request. Consists of hostname, path, callframes, query-parameters (if any) and body-parameters (if any).

$P, P2$ ($parameters$): variable to store the parameters of the current request.

$counter$: requests counter. Initially $counter\ =\ 0$.

Used procedures:

$CheckScheme$ (S): Checks the presence of Scheme S in the RS set. Returns true if schema S is present in RS, false otherwise (see Algorithm 1).

```
Data: scheme S, set of all schemes RS
Result: boolean value that indicates if S is present in RS
1 for scheme in RS do
2     if ((hostname in S = hostname in scheme) and
      (path in S = path in scheme) and (method in S = method in scheme)
      then
3        return true;
4     end
5     if (callframes in S = callframes in scheme)
      then
6        return true;
7     end
8     if ((query-params in S = query-params in scheme)
      and (body-params in S = body-params in scheme))
      then
9        return true;
10    end
11    return false;
12 end
```

Algorithm 1: CheckScheme

Technical aspects such as extracting custom events from the web pages for crawling, navigating between application pages, and triggering custom event handlers, are discussed in the section Implementation.

The basic logic of the algorithm is presented in Algorithm 2.

```
Data: two custom event handlers A, B received from a given site
      for crawling
Result: a set of discriminants for custom events A, B
1 trigger event listener A;
2 intercept trace;
3 counter+ = 1;
4 VP ← hostname, path (where hostname, path ∈ trace);
5 P ← query-params; body-params (where query-params,
   body-params ∈ trace);
6 for param in P do
7     if ((param in AP) and (param.value = AP.param name.val_i))
      then
8         counter i+ = 1
9     else
10        AP ← {param:value : 1}
11    end
12    if param in NVP
      then
13        remove param from P;
14    end
15 end
16 AP ← P;
17 S ← hostname, path, callframes, query-params, body-params
18 (where hostname, path, callframes, query-params, body-params ∈ trace);
19 if checkScheme (S) = true
    then
20    trigger event listener B;
21    counter+ = 1;
22    repeat steps 4-41;
23    HP ← P;
24    trigger event listener A;
25    intercept trace2;
26 else
27 end
28 P2 ← query-params; body-params
   (where query-params, body-params ∈ trace2);
29 VP ← PP2;
30 NVP ← (PP2)/(PP2);
31 for param in NVP do
32    remove param from VP;
33    remove param from S;
34    for scheme in RS do
35        remove param from scheme;
36    end
37 end
38 RS ← S;
39 trigger event listener B;
40 counter+ = 1;
41 repeat steps 4 -41;
```

Algorithm 2. Basic classification algorithm

In a general case, the algorithm sequentially processes all activated user events for a given site. When the work is complete, the number of parameters and their values are recalculated from the set of all application parameters. In this case, the parameters that had the same value for all processed requests are moved from the list of significant parameters (if they were there) to the list of insignificant ones, and are also removed from the request schemes (see Algorithm 3).

```
Data: sets AP, VP, NVP, SR and counter variable
Result: a set VP for application requests
1 for param in AP do
2     if (length(param) = 1) and (counter = param.counter)
      then
3        remove param from VP
4        remove param from SR
5        NVP ← param
6     end
7 end
```

Algorithm 3: Algorithm for recalculating the significance of parameters

The output of the algorithm is a set of significant request parameters. In this case, the key from the values of these discriminants allows the outgoing application request to be uniquely identified.

To validate that the constructed algorithm is applicable, a tool was developed that implements the suggested classifier. It iteratively constructs the set of outgoing requests for the application and extracts the classification features.

5. Implementation

The constructed tool automatically performs the following actions in the process of building a set of outgoing requests in automatic mode:

- collects custom event handlers used on the page;
- activates the handlers obtained in step 1, thus initiating the HTTP request from the client to the server;
- determines the content of emerging HTTP requests;
- defines the context of emerging requests;
- monitors dynamic changes in the DOM of a web page;
- extracts the discriminants of request taking into account the requests' context according to the basic algorithm.

From an architectural point of view, the classifier can be divided into the following logical components (see fig. 2).

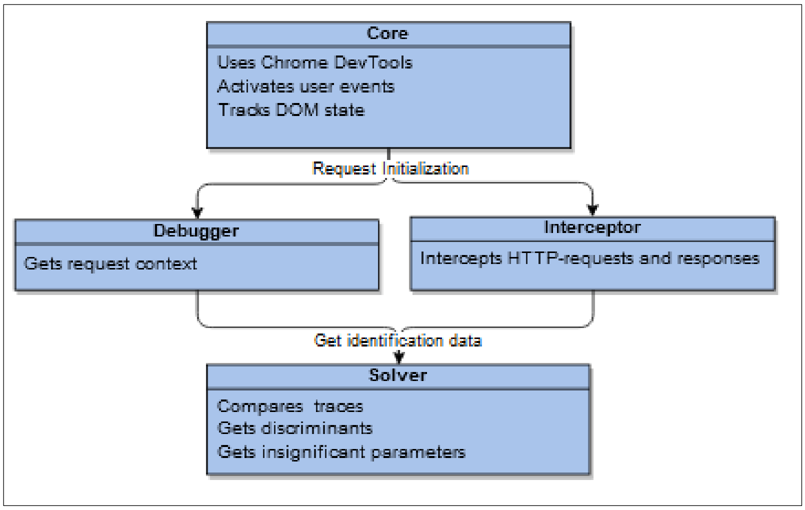


Fig. 2. Tool components

The core of the classifier is responsible for interacting with the browser and using the Chrome DevTools protocol. This protocol is a programmable version of the developer's toolkit for Chromium browsers. In the study it is used to navigate a web application by automatically activating user events on a web page, as well as to track the state of the browser context at the time when HTTP-requests are performed.

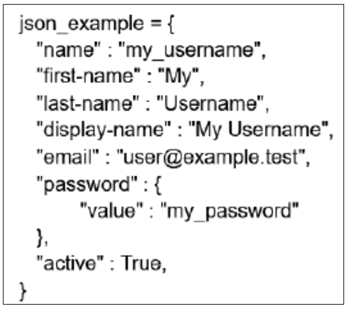
Debugger is used to get the context of the HTTP request and extract the callframes for further processing.

An interceptor is used to intercept requests from the client side of the application, as well as to obtain request elements such as URL, path, and parameters.

Solver represents the classifier itself. It compares the request traces received from the debugger and the interceptor. This part is also responsible for making decisions about the significance of the received features for the classification. It selects discriminants of requests and forms a list of parameters that are not significant for subsequent classification.

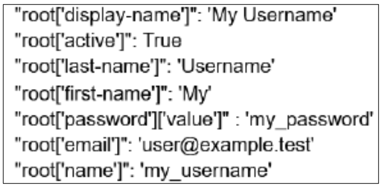
Possible complexity of the parameters' structure must be considered while examining request elements and their contexts.

For a more convenient representation of data transmitted in JSON format in the current study the DeepDiff library was used. It allows users to represent data as a set of fields and values, taking into account nested elements (see fig. 3 and fig. 4).



```
json_example = {
    "name" : "my_username",
    "first-name" : "My",
    "last-name" : "Username",
    "display-name" : "My Username",
    "email" : "user@example.test",
    "password" : {
          "value" : "my_password"
    },
    "active" : True,
}
```

Fig. 3. Data in JSON format



```
"root['display-name']": 'My Username'
"root['active']": True
"root['last-name']": 'Username'
"root['first-name']": 'My'
"root['password']['value']" : 'my_password'
"root['email']": 'user@example.test'
"root['name']": 'my_username'
```

Fig. 4. Same JSON data after DeepDiff processing

6. Experiments

The implemented classifier was firstly tested manually on 10 sites built with ReactJS. For this experiment the activation of user events was performed manually through interaction with the web interface of the application. The requests interception, their contexts selection and subsequent classification were performed automatically. The analysis of the discriminants extracted during the classification showed their 100\% completeness. In other words, there were no parameters that have been mistakenly marked as insignificant based on the classification results. The results of this experiment support the suggested method of solving the problem and allow proceeding to an automatic experiment.

To test the classifier in automatic mode, from the constructed set of 100 sites built with ReactJS, sites using Captcha were excluded. As a result, the final set consisted of 96 sites. The subsequent analysis of the received discriminants of requests also showed their completeness and confirmed the possibility of classifying the requests of the web application using their context. Moreover, usage of callframes helped to classify requests for 73\% of the sites crawled. Therefore, the experiment was considered as successful and the suggested approach was verified and showed its applicability in case of sites, written with React. Nevertheless, to expand the research results to the sites built with other frameworks, additional experiments are required.

In addition, due to the approach of activating custom events twice, using their context and removing insignificant request elements, it was possible to reduce the number of distinguished request discriminants for 52% in comparison with the total number of parameters received. This means that the number of parameters for fuzzing decreased and therefore the process of the subsequent black box testing may become more efficient.

This notwithstanding, in the left 48% of parameters that were marked as valuable, there may be some that were falsely recognized as significant. Nevertheless, the task of identifying was not considered in this study.

Based on the results of the experiments, the influence of request parts on routing was also calculated. Their frequency of occurrence is presented on fig. 5.

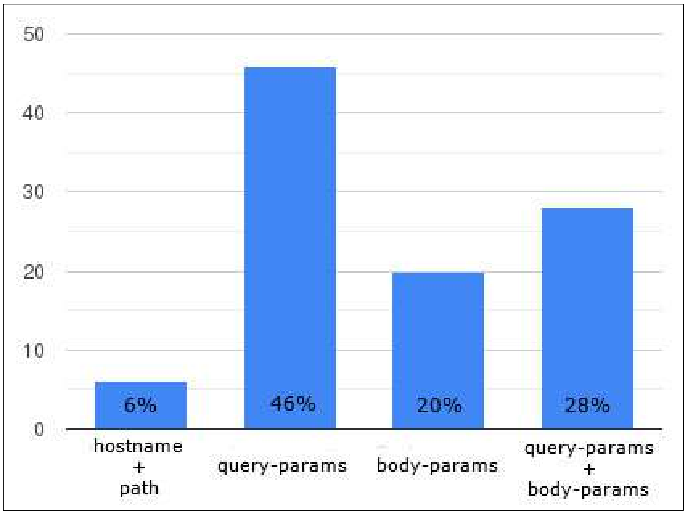


Fig. 5. Influence of request elements on routing in percents

7. Conclusion

The paper suggests a method for classifying requests of web applications with a dynamic interface. The experiments show that the suggested method, based on the usage of request context as a source for additional classification features solves the problem of classifying requests with the same level of completeness as the naive method that takes into account only the request content. The constructed classifier helps to reduce the number of insignificant parameters among the discriminants of the request, which is a positive achievement in the case of using a tool for determining the parameters of application requests for subsequent black box testing.

Список литературы / References

[1] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell. State of the art: Automated black-box web application vulnerability testing. In Proc. of IEEE Symposium on Security and Privacy, 2010, pp. 332-345.
[2] A.M. Reina-Quintero. Surveying navigation modelling approaches. International Journal of Computer Applications in Technology, vol. 33, no. 4, 2008, pp. 327-336.
[3] P. Himschoot. Single Page Applications and Routing. In Blazor Revealed. Building Web Applications in .NET. Apress, 2019, pp. 187-212.
[4] ReactJS official web page. Available at http://www.ReactJs.org, accessed 10.03.2021.
[5] A. Fedosejev. React.js Essentials. A fast-paced guide to designing and building scalable and maintainable web apps with React.js. Packt Publishing, 2015, 208 p.

[6] C. Olston and M. Najork. Web crawling. Foundations and Trends in Information Retrieval, vol. 4, no. 3, 2010, pp. 175–246.
[7] S. Khalid, S. Khusro, and I. Ullah. Crawling ajax-based web applications: Evolution and state-of-the-art. Malaysian Journal of Computer-Science, vol. 31, no. 1, 2018, pp. 35–47.
[8] Г.М. Носеевич, А.А. Петухов. Поиск входных точек для веб-приложений с динамическим пользовательским интерфейсом. Безопасность информационных технологий, том 6, no.1, 2013 г., стр. 13-20 / G.M. Noseevich, A.A. Petuhov. Determining Data Entry Points for Javascript-rich Web applications. IT Security (Russia), vol. 6, no. 1, 2013, pp. 13-20 (in Russian).
[9] T. Pandikumar, Tseday Eshetu. Detecting Web Application Vulnerability using Dynamic Analysis with Penetration Testing. International Research Journal of Engineering and Technology, vol. 03, no. 10, 2016, pp. 430-433.
[10] А.А. Петухов, Н.Б. Матюнин. Автоматический обход веб-приложений с динамическим пользовательским интерфейсом. Проблемы информационной безопасности. Компьютерные системы, no. 3, 2014 г., стр. 43-49 / A.A. Petuhov, N.B. Matjunin. Automatic crawling of web applications with dynamic user interface. Information Security Problems. Computer Systems, no. 3, 2014, pp 43-49 (in Russian).
[11] И.С. Говорков. Оптимизация обхода страниц динамических веб-приложений, построенных с использованием библиотеки ReactJS. Тезисы конференции «Ломоносов-2018», 2018 г., стр. 1-3 / I.S. Govorkov. Optimization of crawling pages of dynamic web applications built using the ReactJS library. Abstracts of the conference «Lomonosov-2018», 2018, pp. 1-3 (in Russian).
[12] C. H. Liu, C. J. Wu, and H. M. Chen. Testing of AJAX-based Web applications using hierarchical state model. In Proc. of the IEEE 13th International Conference on e-Business Engineering (ICEBE), 2016, pp. 250-256.
[13] X. Zhang and H. Wang. AJAX Crawling Scheme Based on Document Object Model. In Proc. of the Fourth International Conference on Computational and Information Sciences (ICCIS), 2012, pp. 1198-1201.
[14] W.-K. Chen, C.-H. Liu, and K.-M. Chen. A web crawler supporting interactive and incremental user directives. Lecture Notes in Electrical Engineering, vol. 464, 2017, pp. 64–73
[15] H.Z.U. Khan. Comparative Study of Authentication Techniques. International Journal of Video Image Processing and Network Security, vol. 10, no. 04, 2010, pp. 9-13.
[16] S. Aggarwal. Modern Web-Development Using ReactJS. International Journal of Recent Research Aspects, vol. 5, no. 1, 2018, pp.133-137
[17] Websites using React. Available at https://trends.builtwith.com/websitelist/React, accessed 10.03.2021.
[18] Top 32 Sites Built with ReactJS. Available at https://medium.com/@coderacademy/32-sites-built-with-reactjs-172e3a4bed81, accessed 10.03.2021.
[19] Thends in JavaScript frameworks. Available at https://trends.google.com/trends/explore?q=vue.js,react,angular, accessed 10.03.2021

Информация об авторах / Information about authors

Анна Вадимовна ЛАПКИНА, магистр ВМК МГУ, выпуск 2021. Научные интересы: безопасность веб-приложений, обнаружение уязвимостей, автоматическая навигация по веб-приложениям.

Anna Vadimovna LAPKINA, master's graduate in 2021, CS department, MSU. Research interests: application security, vulnerability analysis, automatic application navigaton.

Андрей Александрович ПЕТУХОВ. Младший научный сострудник в Лаборатории интеллектуальных систем кибербезопасности ВМК МГУ. Научные интересы: тестирование веб-приложений методом черного ящика, обнаружение уязвимостей, анализ программ.

Andrew Alexandrovitch PETUKHOV, researcher in Cybersecurity Lab, CS department, MSU. Research interests: black-box testing of web applications, vulnerability analysis, program analysis.

DOI: 10.15514/ISPRAS-2021-33(3)-7

High Performance Distributed Web-Scraper

D.S. Eyzenakh, ORCID: 0000-0003-1584-1745 <eisenachdenis@gmail.com>
A.S. Rameykov, ORCID: 0000-0001-7989-6732 <arcane561@gmail.com>
I.V. Nikiforov, ORCID: 0000-0003-0198-1886 <igor.nikiforovv@gmail.com>
Peter the Great St.Petersburg Polytechnic University
29, Polytechnicheskaya, St.Petersburg, 195251, Russia

Abstract. Over the past decade, the Internet has become the gigantic and richest source of data. The data is used for the extraction of knowledge by performing machine leaning analysis. In order to perform data mining of the web-information, the data should be extracted from the source and placed on analytical storage. This is the ETL-process. Different web-sources have different ways to access their data: either API over HTTP protocol or HTML source code parsing. The article is devoted to the approach of high-performance data extraction from sources that do not provide an API to access the data. Distinctive features of the proposed approach are: load balancing, two levels of data storage, and separating the process of downloading files from the process of scraping. The approach is implemented in the solution with the following technologies: Docker, Kubernetes, Scrapy, Python, MongoDB, Redis Cluster, and CephFS. The results of solution testing are described in this article as well.

Keywords: web-scraping; web-crawling; distributed data collection; distributed data analysis

For citation: Eyzenakh D.S., Rameykov A.S., Nikiforov I.V. High Performance Distributed Web-Scraper. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 87-100. DOI: 10.15514/ISPRAS-2021-33(3)-7

Высокопроизводительный распределенный веб-скрапер

Д.С. Эйзенах, ORCID: 0000-0003-1584-1745 <eisenachdenis@gmail.com>
А.С. Рамейков, ORCID:\ 0000-0001-7989-6732 <arcane561@gmail.com>
И.В. Никифоров, ORCID: 0000-0003-0198-1886 <igor.nikiforovv@gmail.com>
Санкт-Петербургский Политехнический университет Петра Великого,
Россия, Санкт-Петербург, 195251, ул. Политехническая, д. 29

Аннотация. За последнее десятилетие Интернет стал гигантским и богатейшим источником данных. Данные используются для извлечения знаний путем выполнения машинного анализа. Чтобы выполнить интеллектуальный анализ данных веб-информации, данные должны быть извлечены из источника и помещены в аналитическое хранилище. Это ETL-процесс. Разные веб-источники имеют разные способы доступа к своим данным: либо API по протоколу HTTP, либо парсинг исходного кода HTML. Статья посвящена подходу к высокопроизводительному извлечению данных из источников, не имеющих API для доступа к данным. Отличительными особенностями предлагаемого подхода являются: балансировка нагрузки, двухуровневая подсистема данных и отделение процесса загрузки файлов от процесса парсинга. Подход реализован в решении со следующими технологиями: Docker, Kubernetes, Scrapy, Python, MongoDB, Redis Cluster и CephFS. Результаты тестирования решения также описаны в этой статье.

Ключевые слова: веб-скрапинг; веб-краулинг; распределенный сбор данных; распределенный анализ данных

Для цитирования: Эйзенах Д.С., Рамейков А.С., Никифоров И.В. Высокопроизводительный распределенный веб-скрапер. Труды ИСП РАН, том 33, вып. 3, 2021 г., стр. 87-100 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(3)–7.

1. Introduction

Due to the rapid development of the network, the World Wide Web has become a carrier of a large amount of information. The data extraction and use of information has become a huge challenge nowadays. Traditional access to the information through browsers like Chrome, Firefox, etc. can provide a comfortable user experience with web pages. Web sites have a lot of information and sometimes haven't got any instruments to access over the API and preserve it in analytical storage. The manual collection of data for further analysis can take a lot of time and in the case of semi-structured or unstructured data types the collection and analyzing of data can become even more difficult and time-consuming. The person who manually collects data can make mistakes (duplication, typos in the text, etc.) as far as the process is error-prone.

Web-scraping is the technique which is focused on solving the issue of the manual data processing approach [1]. Web scraping is the part of ETL-process and is broadly used in web-indexing, web-mining, web data integration and data mining. However, many existing solutions do not support parallel computing on multiple machines. This significantly reduces performance, limiting the system's ability to collect large amounts of data. A distributed approach allows you to create a horizontally scalable system performance of which can be increased depending on the user's needs.

The article proposes an approach to organize distributed, horizontally scalable scraping and distributed data storage. Using an orchestration system greatly simplifies the interaction with the system, and the support of automatic load balancing avoids overloading individual nodes.

2. Existing Web Scraping Techniques

Typically, web scraping applications imitate a regular web user. They follow the links and search for the information they need. The classic web scraper can be classified into two types: web-crawlers and data extractors (fig. 1).

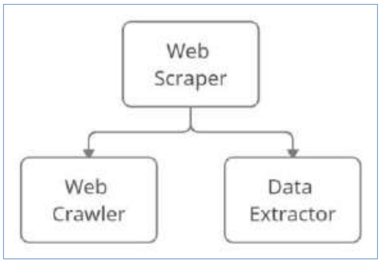


Fig. 1 Web-scraper structure

A web-crawler (or called a spider, spiderbot) is the first type of data web-scraping. The crawler is a web robot also known as an Internet bot that scans the World Wide Web typically operated by search engines for the purpose of Web indexing [2]. The crawling procedure starts with the list of seed URLs. The program identifies all the links that exist on seed pages and stores them. After that, the list of all links is recursively visited. This process continues until all URLs will be visited. There are several types of web-crawlers, but all of them can be divided into a common crawler and focused crawler.

Focused crawler searches for the most suitable pages according to the topic that is defined by the user. This goal is achieved by using algorithms of intelligent text analysis. It ensures that web pages can only be crawled for information related to the specific topic. In the server's perspective, there are single machine crawlers or distribution crawlers. The information crawling can be achieved by dividing into several nodes and their cooperation, which improves the efficiency and performance of the crawler.

The second type of web scraper is a data extractor [3]. The website contains a large amount of information and the analyst cannot spend a lot of time manually collecting and converting this data into the desired format. Besides that, a web page can contain a lot of unstructured data that means it can contain noise or redundant data. Data extractors can easily extract large and unstructured data and convert them into a comprehensive and structured format. The extraction process starts with indexing or crawling. In the crawling process, the crawler finds a list of the relevant URLs that the data extractor will process. In these web pages a lot of junk and useful data is mixed. The data extractor extracts the needed information from the web-pages. Data extractor contains a lot of techniques [4] for extraction data from HTML pages.

3. Comparison Analysis of Systems

Here is an overview and comparison of web scraping frameworks for fast scanning any kind of data, distributed scraping systems for increasing the performance, and orchestration systems.

3.1 Scraping tools

There are various tools for working with web scrapers. They can be divided into three categories: libraries, frameworks, and desktop or web-based environments.

3.1.1 Libraries

Modern web resources may contain various information. Due to this circumstance, certain flexibility is required for configuring and implementing web scraping tools. The libraries guarantee access to the web resource. Most of the library implement the client side of the http protocol, then the resulting web page is parsed and the data is retrieved using string functions such as regular expressions, splitting and trimming, etc. [5]. Also, third-party libraries can help with implementing more complex analysis, for example, building an html-tree and XPATH mappings.

One of the most popular site access libraries is *libcurl*. It supports the major features of the HTTP protocol, including SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form-based upload, proxies, cookies and HTTP authentication. Moreover, it can work with many programming languages. In Java, the Apache HttpClient package emulates HTTP main features, i.e., all request methods, cookies, SSL and HTTP authentication, and can be combined with HTML parsing libraries. Java also supports XPath and provides several HTML cleaning libraries, such as *jsoup*. Programs like *curl* (libcurl) and *wget* implement the HTTP client layer, while utilities such as *grep*, *awk*, *sed*, *cut* and *paste* can be used to parse and transform contents conveniently.

3.1.2 Desktop or web application

Desktop applications are implementations of web scrapers that are designed for noncoding professionals. This kind of web scraper contains a graphical shell that makes it easier to create and support web robots. Typically, these applications include an embedded web browser, where the user can navigate to a target web resource and interactively select page elements to extract them, avoiding any kind of *regex*, *XPath* queries, or other technical details. In addition, modules are capable of generating several kinds of outputs, such as CSV, Excel and XML files, and queries that are inserted into databases. The main disadvantages of desktop solutions are commercial distribution and limited API access, which make it difficult to embed these web scrapers into other programs.

Table 1. Comparison of scraping frameworks

Feature/ Framework	**Scrapy**	**PySpider**	**NodeCralwer**	**Apify SDK**	**Selenium**
Builtin Data Storage Supports	Customizable	CSV, JSON	CSV, JSON, XML	JSON, CSV, XML, HTML	Customizable

Suitable for Broad Crawling	Yes	No	Yes	Yes	No
Builtin Scaling	Yes	Yes	No	Yes	No
Support AJAX	No	Yes	No	Yes	Yes
Available Selectors	CSS, Xpath	CSS, Xpath	CSS, Xpath	CSS	CSS, Xpath
Builtin Interface for Periodic Jobs	No	Yes	No	Yes	No
Speed (Fast, Medium, Slow)	Fast	Medium	Medium	Medium	Very slow
CPU Usage (Fast, Medium, Slow)	Medium	Medium	Medium	Medium	High
Memory Usage (High, Medium, Low)	Medium	Medium	Medium	High	Medium
Github Forks	9000	3600	852	182	6200
Github Stars	39600	14800	5800	2700	19800
License	BSD License	Apache License 2.0	MIT	Apache License 2.0	Apache License 2.0

3.1.3 Frameworks

Programming libraries have their limitations. For example, you need to use one library for accessing a web page, another for analyzing and extracting data from HTML pages. The architecture designing and the compatibility of the library's checking process can take a significant amount of time. Frameworks are a complete solution for developing web scrapers. Comparison results of popular frameworks for implementing web scrapers are presented in the article as well (Table 1).

Comparison is made according to the following criteria. Built-in Data Storage Supports - supporting types of files or other storage.

Suitable for Broad Crawling – this type of crawler covers a large (potentially unlimited) number of domains, and is only limited by time or another arbitrary constraint, rather than stopping when the domain has already been crawled to completion or when there are no more requests to perform. These are called *broad crawls*, which are the typical crawlers used by search engines. Speed, CPU usage, and memory usage can represent system performance. GitHub Forks, GitHub Starts, Last Update can inform the state of the framework, support and community activity.

3.2 Orchestration and containerization systems

Since our system should support horizontal and vertical scaling, it must support the orchestration system. The orchestration system will monitor the status of services, distribute the load among the nodes in the cluster, taking into account the resources of each of these nodes. An orchestration system is a support for the compatibility of software products that communicate with each other through remote procedure calls (RPC). There are many solutions on the market today, but some of them are bound to specific companies. Such systems can impose many different restrictions such as: territorial limitations, bounded choice of cloud computing service, the chance to be left without data due to any external factors. And so, at the moment, there are the following tools: Kubernetes, Docker Swarm, Apache Mesos.

Based on the paper [6], we can conclude that the Kubernetes orchestration system has a large coverage of the required technologies. It is also the de facto standard today, as evidenced by the fact that it is the only orchestration system that has been accepted into the Cloud Native Computing Foundation [7]. It is also used by such large companies as Amazon, Google, Intel, etc.

We were also faced with the choice of virtualization or containerization system. Referring to the fact that Kubernetes can work with virtual machines, we chose containerization, due to it consuming less resources, which was confirmed by research [8]. Also, Kubernetes in its delivery recommends to work with containers. Thus, our web scraper will be delivered as a container running Docker.

3.3 Distributed scraping system review

3.3.1 Research

Scrapy does not provide any built-in facility for running spiders in a distributed (multi-server) manner. However, there are several ways to organize work. Some of the popular solutions are Frontera, Scrapy Redis, Scrapy Cluster, and Scrapyd.

Frontera is a distributed crawler [9] [10] system. Based on the description of the project, we can say that the system is a separately distributed web crawler. It is designed to collect a large number of URLs as data sources. The system does not have a built-in data extractor and it is not known whether it is possible to add one. Hence, we can say that the system is intended for other tasks.

Scrapy Redis [11] [12] is a system architecture that supports distributed web-scraping. In the process of crawling, the Redis database can mark the links that have been already crawled and add to the queue links that haven't crawled yet, avoiding the repeated crawling problem in the distributed process. From the Scrapy Redis description follows that Redis database is used as main storage. Redis, as the main storage, does not satisfy the ACID theorem, namely CD, which carries the consequences of losing part of the tasks, if the cluster is in an emergency state, the consequences of the loss of tasks can carry different types of damage, from re-scanning the page, which entails a decrease in production.

Scrapy Cluster [13] was taken after Scrapy Redis, which offloads Requests to a Redis instance. It has the same problem with the Redis database. Based on the project description we can find out that the system does not imply the usage of an orchestration system, this opportunity is provided to the user. It is a big disadvantage due to the reason that it is not clear how the system will behave when the basic functions of maintaining the cluster will be launched by the orchestrator. For instance, operation “liveness check” could not behave correctly and conflict with internal monitoring of the system or doesn’t work at all. Scrapy Cluster also uses the Apache Kafka message broker as a connection between the system components. The parallelism of Kafka lies in the number of sections in the topic [14]. All data that falls into the topic is balanced between sections. From one section, following the documentation, only one instance of the application can read data. Several disadvantages can be distinguished from this:

- Adding new topics will semantically separate the data, which means that the data that has been already stored in Kafka is not rebalanced internally [15]. This means that if you try to add a new instance, the spider will slow down the system until all new jobs will be balanced against the new partitions.
- In the case of Scrapy Cluster, the spiders are bound to the partition, and according to the scrapy cluster documentation, control signals for the spider can be sent to the partition (stop the scraping job) [16]. This makes it impossible to add new partitions until the end of the complete scraping session. Since concerning to the balancing formula in Kafka:
    - $hash(key)\ \%\ number\_of\_partition$

the control signal can be sent to the wrong spider.

- Removing topics from the Kafka is impossible, which can lead to a situation when several dozen spiders stop in the cluster, the producer (sending the task) cannot rebalance the queue between the remaining spiders. This can lead to imbalance – overloading one spider and no load on others.

One of the most popular distributed scraping solutions using the Scrapy framework is Scrapyd [17]. Scrapyd allows you to deploy and manage multiple Scrapy projects using the json API. The wrapper provides the ability to store a queue in a database, and also allows you to manage several machines at once. With all the visible advantages, the system has not been without drawbacks. The lack of a balancing system does not allow the use of a larger number of nodes. When sending a request for scraping, you must specify the ip address of a specific node and before that make sure that it is not 100% loaded.

3.3.1 Conclusion

After conducting research and analysis of various web scraping tools, it was decided to use the Scrapy framework. It does not restrict the developer by its license or capabilities and is also used in many companies [18]. At the same time, it has a convenient architecture for building any web scraper. A convenient mechanism for adding additional software easily compensates drawbacks of the framework. Such as support of JavaScript, etc

Considering all of the above, we decided to develop a system that would solve the problems of existing solutions, allowing programs written in the Scrapy framework to work, and also use the Kubernetes orchestration system.

4. Methodology

4.1 Overall architecture of our distributed scraping system

Distributed scraper architecture presents 3 functional layers (fig. 2):

User interface layer;

Web scraping layer;

Data storage layer.

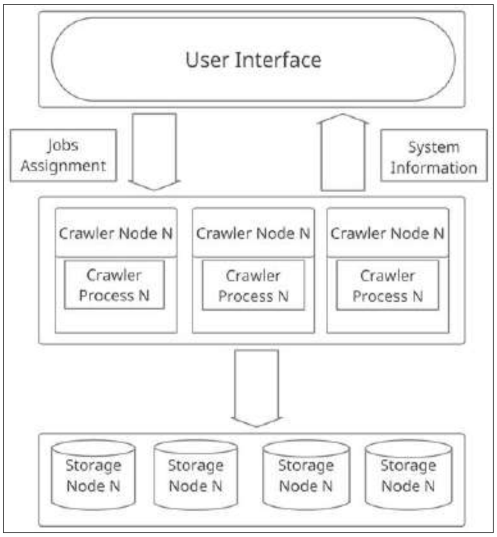


Fig. 2 Distributed approach for data extraction

The user interface layer is responsible for interacting with the end-user, the user can send control commands to the cluster, receive a response and see the scraping statistics at a given point of time. The web scraping layer is a layer of distributed web spiders that do not store state by themselves, that is to say, they receive it from the user interface layer, so they can be multiplied, so this layer is responsible for all the scraping logic of sites. The data storage layer is responsible for storing all collected information, which includes texts and media content.

4.2 System design

Fig. 3 shows a detailed diagram of the system operation.

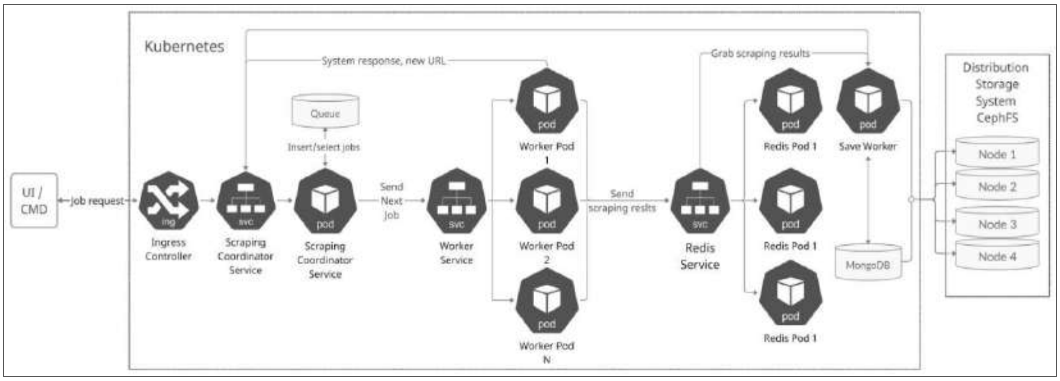


Fig. 3. System design

4.2.1 Ingress Controller

An ingress controller is an element of the orchestrator infrastructure, the main task of which is to proxy external traffic to services within the cluster. It also performs other tasks such as SSL termination and balancing and routing of traffic based on names and URLs.

Insert reference to the figure, the ingress controller is the entry point to the cluster for the end-user and, based on the hostnames, redirects requests to a particular environment.

4.2.2 Services

There are three services in our system: Scrapy Coordinator Service, Spider Service, Redis Service. Pods in the cluster are not permanent, they can be stopped, they can change the IP address, they can be moved to another element of the cluster (node). Because of this, we faced the problem of controlled access to pods. Service solves this problem by acting as an abstraction that provides access to pods, has access sharing mechanisms, and also has mechanisms for discovering all services that match the conditions. The end-user or program does not need to know the IP address of a particular pod, he can refer to the domain name (for example crawler.company 1, where crawler is the generic pod name, and company 1 is the namespace name) and gain access to one of the pods.

Service also balances traffic using the Round-robin method. This algorithm is already built into the standard delivery of Service Kubernetes and allows you to evenly distribute tasks across all working nodes.

4.2.3 Scraping Coordinator and Workers

As we saw in fig. 3, the scraping coordinator and workers run in separate containers. Scraping coordinator working with PostgreSQL [19] database. PostgreSQL works in Master Slave mode, namely with asynchronous replication, asynchronous replication allows you not to wait for a response from the master, thereby not slowing down its work, and replication will not stop if the slave is disabled, this solution is quite simple and reliable in the sense that this replication mode is shipped with the database distribution and solves most fault tolerance problems. It works as a storage of the queue of requests and processed links.

Worker consists of scrapy spiders written in scrapy and HTTP-wrapper. The wrapper is a web server written in Python with micro-framework Klein. Klein framework based on Twisted – the most powerful Python asynchronous framework, provides easy integration with Scrapy that uses Twisted for all HTTP traffic.

The following is the sequence followed by the system when initiating a scraping request from the perspective of the scraping coordinators and workers.

1) Each scraping coordinator receives a list of URLs to crawl and the number of workers instances a user would like to use. The scraping coordinator checks in the PostgreSQL database if the seed URL has already been processed. If not, the coordinator creates a job that includes the seed URL and custom settings for the spider. After that, it adds those jobs to a queue along with a user scrape endpoint of the worker based on the number of received from the user.
2) The scraping coordinator pops jobs from the queue and passes them to the scraper's URLs.
3) HTTP-wrapper takes a job, takes settings, and starts Scrapy spider. If a worker is free and accepts the URL, it sends back an acceptance message. If it is busy and has no free threads to handle the request, it replies with a rejected message. The scraping coordinator adds those URLs for which it received a rejection message, back into the queue.
4) Spider scrape data and send it to the Redis cluster. If the user has enabled the deep scraping function, extracts all the child URLs (HREF elements in the web page) and passes them to the scraping coordinator.

4.2.4 Scrapy Spider

It is a standalone spider program using the Scrapy framework. Has ample opportunities, such as: JavaScript processing. Since most pages have dynamically generated content these days, it is no longer enough to just browse static pages. If this factor is not taken into account, a large amount of data can be lost during screening. Scrapy in the standard delivery cannot work with dynamic content. But there is a possibility of connecting additional modules – headless web browsers

Supports processing web pages with pagination. Unlike traditional search engines, which write every next page to the seed URL queue. This can lead to high code interfacing, poor readability, and spider startup costs. Using Scrapy's system call-back mechanism as a bridge, URL queue creation and content crawling operation are performed separately, which solves the shortcomings of traditional crawlers [20].

The system also can use middleware to dynamically change IP proxy, as well as the User-Agent value. All this significantly reduces the chance of blocking by a web resource.

4.2.5 Redis-Cluster

A memory database is needed to quickly save results, thereby blocking save operation minimizes waiting on the part of the scraper, increasing its performance. Redis Cluster acts as an intermediate caching layer, it provides fast storage of information, since all data is stored in RAM. Spiders do not stand idle waiting for information to be saved, this is important because the write speed in distributed file systems is rather low [21]. Redis stores information primarily as a dictionary, that is, on a key-value basis. The key is the site URL, and the value is the result of scraping in the form of a JSON file.

4.2.6 Save Worker

Save Worker performs the task of post-processing information. It scans keys in Redis at a certain frequency. After the information reaches the desired size (recommended 4+ Gb [21]) or is not updated, it starts downloading information to itself and simultaneously deleting data from Redis. After that, it starts scanning the scraping result, looking for certain marks in it in order to load the missing files into a distributed file storage. Thus, the spiders unload the waiting time for downloading large files with an indefinite download time. After that SaveWorker saves all results to the database with one request.

4.2.7 MongoDB

MongoDB [22] is well adapted to our problem. It is a document-oriented database management system and does not require a description of the schema and tables.

MongoDB has the ability to scale horizontally.

4.2.8 CephFS

It is a software-defined distributed file system that can scale flexibly to store petabytes of data. Ceph is able to replicate data between nodes, as well as balance the load between them. When a node fails, Ceph can self-heal without downtime, thereby improving system availability. Ceph offers 3 types of interface for interacting with storage, a POSIX compatible [23] file system, an S3 storage and a block device, thus providing higher compatibility with already written software.

4.2.9 Sharing resources between users

To restrict manual cluster management, unique user environment settings are specified. After that, they are automatically added to the declarative description of the cluster configuration file. This file is stored alongside the project output in YML format in the Git [24] repository.

The system supports resource sharing using the tools provided by the orchestration system, namely namespace and ingress controller. Ingress controller redirects the user to a particular namespace, according to the URL. The software located in one namespace does not have direct access to the resources of another namespace, just as each namespace can be allocated quotas for processor time and the amount of RAM.

5. Approbation and Testing

In order to verify the effect of distributed crawlers builds an experimental stand. The three servers use the 64-bit Debian 9 operating system.

5.1 Scalability

Scrapers do not have an internal state, this is confirmed by the fact that the coordinator transfers the state to each individual scraper and does not store the subsequent state, but writes it to the in-memory database and to the coordinator, therefore such a system scale well horizontally.

5.2 Chaos monkey testing

Chaos monkey is a set of software tools that allows you to simulate crashes on a live system. It analyzes the system for all its critical components and disables them in different sequences. This allows you to observe the actions of the system during emergency operation, as well as identify critical points of the system. During testing, a situation occurred when, in aggressive mode, the testing software disabled both the Master and Slave PostgreSQL servers.

5.3 System speed test

The first experiment examines the overall speed of the system. For this, the Scrapy project was developed and deployed in the system, data collection takes place within a single website. The main database for storing data is the MongoDB database management system based on CephFS distributed file storage. Text data is stored as JSON files, images and other data are downloaded directly to file storage.

Testing was carried out for 5 hours, the same number of seed-URLs were chosen as input parameters, which makes it possible to ensure that the web crawler will follow the same route from the links. The performance test results are presented in the table. Each node ran 5 instances of Scrapy. This value was chosen empirically based on the load on the systems, and also on the network.

Table 2. Result of performance test

Time/h	**One**	**Three**	**Five**
(Single node) Pages	6650	19267	31 920
Elements	77200	223 880	370 560
(Two-node) Pages	11970	34 114	58 653
Elements	131500	368 200	631 200
(Tree-node) Pages	15960	46 922	76 927
Elements	257000	724 740	1 259 300

It can be seen by reproducing the above experiments (table 2) that the data acquisition experiment results of the distributed scraping shows that the efficiency of a node crawling is lower than that of two nodes crawling at the same time. Compared with the stand-alone spider, it can get more pages and run more efficiently.

5.4 Balancing test

Since the system contains several fail-safe elements and has the ability to distribute the load, its stability should be checked.

At first, the work of the system for distributing the load on the nodes was checked. The launch was carried out on 100 seed URLs, which contain tabular data, text, as well as data stored in various types of files. During the scanning process, the crawler worked in a limited wide scan mode, that is, it could click additional links within the same domain.

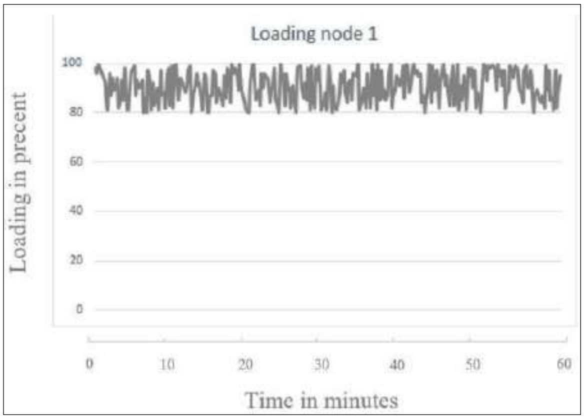


Fig. 4. Loading node №1

Testing was carried out within 5 hours. During testing, there were no emergencies.

On the graphs of the load of nodes (fig. 4-6), we can see that the balancer copes well with its task; the system is working quite stably, there are no strong drawdowns and spikes in performance.

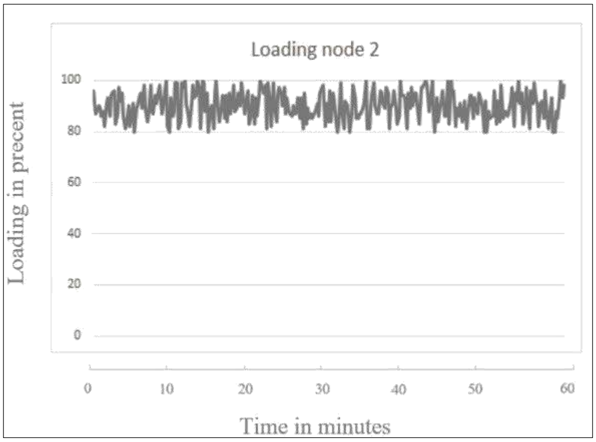


Fig. 5 Loading node №2

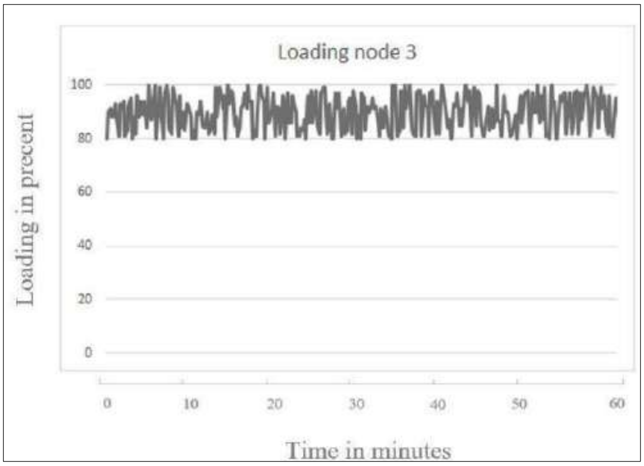


Fig. 6 Loading node №3

Table 3. Sability test

Time/h	**One**	**Two**	**Three**	**Four**	**Five**
Pages	15343	14903	16013	14850	15025

On the table 3, we can see that the number of collected items for each hour is almost the same. This fact shows the stability of the system. This is facilitated by the work of the balancing algorithm, which allows not to overload the system nodes and the optimal load.

6. Conclusion

As part of the work, the following work has been done:

- Classification of information extraction solutions has been introduced;
- A review of existing distributed web scrapers implementations has been conducted;
- A comparative analysis of the considered solutions is carried out.

In the course of the comparative analysis, deficiencies were found in existing solutions, namely, the lack of an orchestration system, problems with horizontal scalability implementations, and deployment of applications.

An architecture has been proposed, which is headed by the Kubernetes orchestration system, which monitors the health of each element of the cluster and shares access to resources.

During the analysis of the work of the resulting system, it showed its viability. Reducing the time required to retrieve data from web resources, connecting additional work nodes to work. And so, the increased stability of the system due to replication of the data storage, the use of a load balancer, and an intermediate storage layer in the Redis Cluster.

References

[1] Deepak Kumar Mahto, Lisha Singh. A dive into Web Scraper world. In Proc. of the 3rd International Conference on Computing for Sustainable Global Development (INDIACom), 2016, pp. 689-693.

[2] Web Cralwer. Available at: https://webbrowsersintrodu ction.com/.

[3] Momin Saniya Parvez, Khan Shaista Agah Tasneem, Shivankar Sneha Rajendra, Kalpana R. Bodke. Analysis of Different Web Data Extraction Techniques. In Proc. of the International Conference on Smart City and Emerging Technology (ICSCET), 2018, pp. 1-7.

[4] Anand V. Saurkar, Kedar G. Pathare, Shweta A. Gode. An Overview On Web Scraping Techniques and Tools. International Journal on Future Revolution in Computer Science & Communication Engineering, vol. 4, no. 4, 2018, pp. 363-367.

[5] Rohmat Gunawan, Alam Rahmatulloh, Irfan Darmawan, Firman Firdaus. Comparison of Web Scraping Techniques: Regular Expression, HTML DOM and Xpath. In Proc. of the International Conference on Industrial Enterprise and System Engineering, 2018, pp. 283-287.

[6] Isam Mashhour Al Jawarneh, Paolo Bellavista et al. Container Orchestration Engines: A Thorough Functional and Performance Comparison. In Proc. of the 2019 IEEE International Conference on Communications (ICC), 2019, pp. 1-6.

[7] CNCF certificate. Available at: https://www.cncf.io/certification/software-conformance/.

[8] S. Vestman. Cloud application platform - Virtualization vs Containerization. Student Thesis. Blekinge Institute of Technology, Sweden, 2017, 45 p.

[9] Distributed Frontera: Web crawling at scale. Available at: https://www.zyte.com/blog/distributed-frontera-web-crawling-at-large-scale/.

[10] Frontera documentation. Available: at https://frontera.r eadthedocs.io/en/latest/.

[11] Scrapy-Redis documentation. Available at: https://scrapy-redis.readthedocs.io/en/v0.6.x/readme.html#.

[12] Fulian Yin, Xiating He, Zhixin Liu. Research on Scrapy-Based Distributed Crawler System for Crawling Semi-structure Information at High Speed. In Proc. of the 2018 IEEE 4th International Conference on Computer and Communications (ICCC), 2018, pp. 1356-1359.

[13] Scrapy-Cluster documentation. Available at: https://scrapy-cluster.readthedocs.io/en/latest/.

[14] Kafka documentation. Intro. Available at: https://kafka.apache.org/documentation/#introduction.

[15] Kafka official documentation. Basic_ops_modify_topic. Available: at https://kafka.apache.org/documentation.html #basic_ops_modify_topic.

[16] Scrapy-Cluster documentation. Core Concepts. Available at: https://scrapy-cluster.readthedocs.io/en/latest/topics/introduction/overview.html.

[17] Scrapyd documentation. Available at: https://scrapyd.re adthedocs.io/en/stable/.

[18] Official Scrapy framework web-site. List of companies using Scrapy. Available at: https://scrapy.org/companies/.

[19] Regina O. Obe, Leo S. Hsu. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database. 3rd Edition, O'Reilly Media, Inc., 2017, 314 p.

[20] Deng Kaiying, Chen Senpeng, Deng Jingwei. On optimisation of web crawler system on Scrapy framework. International Journal of Wireless and Mobile Computing, vol. 18, no. 4, 2020, pp. 332-338.

[21] Jia-Yow Weng, Chao-Tung Yang. Chih-Hung Chang. The Integration of Shared Storages with the CephFS. In Proc. of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), 2019. pp. 93-98

[22] Shannon Bradshaw, Eoin Brazil, Kristina Chodorow. Customers who viewed MongoDB: The Definitive Guide: Powerful and Scalable Data Storage. 3rd edition, O'Reilly Media, Inc., 2019, 514 p.

[23] Abutalib Aghayev, Sage Weil et al. File systems unfit as distributed storage backends: lessons from 10 years of Ceph evolution. In Proc. of the 27th ACM Symposium on Operating Systems, 2019. pp. 353–369

[24] N. Voinov, K. Rodriguez Garzon et al. Big data processing system for analysis of GitHub events. In Proc. of the 22nd International Conference on Soft Computing and Measurements, 2019, pp. 187-190.

Информация об авторах / Information about authors

Денис Сергеевич ЭЙЗЕНАХ, студент. Область интересов: технологии контейнеризации, распределенные системы, параллельная обработка данных.

Denis EYZENAKH, student. Research interests: containerization technologies, distributed systems, parallel data processing.

Антон Сергеевич РАМЕЙКОВ, студент. Область интересов: технологии сбора данных, параллельная обработка данных, системы распределенных вычислений.

Anton RAMEYKOV, student. Research interests: data collection technologies, parallel data processing, distributed computing systems.

Игорь Валерьевич НИКИФОРОВ, к.т.н., доцент. Область интересов: параллельная обработка данных, системы распределенного хранения данных, большие данные, верификация программного обеспечения, автоматизация тестирования.

Igor NIKIFOROV, PhD (Computer Science), Assistant Professor. Research interests: parallel data processing, distributed storage systems, big data, software verification, test automation.

DOI: 10.15514/ISPRAS-2021-33(3)-8

Power Fx: Low-code Language for Collaboration Tools

I.A. Voronkov, ORCID: 0000-0001-5620-1002<iliaftk@outlook.com>
S.E. Saradgishvili, ORCID: 0000-0002-1291-1675 <ssarad@ya.ru>

Peter the Great St.Petersburg Polytechnic University
29, Polytechnicheskaya, St.Petersburg, 195251, Russia

Abstract. The paper provides an overview of the first impression of the language for implementation low code of approach. About a month has passed since the release date.

Keywords: low-code; collaboration; power platform

For citation: Voronkov I.A., Saradgishvili S.E. Power Fx: Low-code Language for Lollaboration Tools. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 101-108. DOI: 10.15514/ISPRAS-2021-33(3)-8

Power Fx: Low-code язык для инструментов совместной работы

И.А. Воронков, ORCID: 0000-0001-5620-1002<iliaftk@outlook.com>
С.Э. Сараджишвили, ORCID: 0000-0002-1291-1675 <ssarad@ya.ru>

Санкт-Петербургский Политехнический университет Петра Великого,
Россия, Санкт-Петербург, 195251, ул. Политехническая, д. 29

Аннотация. В статье представлен обзор первого впечатления от языка для реализации подхода low code. Со дня релиза прошло около одного месяца.

Ключевые слова: автоматизированное тестирование; обеспечение качества; статический анализ исходного кода.

Для цитирования: Воронков И.А., Сараджишвили С.Э. Power Fx: Low-code язык для инструментов совместной работы. Труды ИСП РАН, том 33, вып. 3, 2021 г., стр. 101-108 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(3)–8.

1. Introduction

The IT market is now experiencing a new round of its rapid development. The demand for specialists in this area remains at least high [1].

According to the rules of the market, the cost of projects and services is growing following demand. Numerous platforms operating on the principles of low-code and zero-code have become one of the options for solving problems for automation and digitalization [2]. The general approach in such solutions is that most of the processes can be represented in the form of a graphical designer that works in accordance with UML/BPMN. The main feature is that the developer does not need to think about how interactions occur at the level of data structures, what algorithms are used for selection, sorting, and how computations are parallelized. Main objective: Implementation of the business requirement by creating a process state diagram. From a business point of view, this paradigm is the most effective way to solve a problem, since business users are accustomed to thinking precisely by the tasks that a particular system must solve.

Unfortunately, at this stage of technology development, we are not able to completely switch to such technological solutions. The reasons may be different: the inability to completely solve the problem using only design tools, the lack of specialists with modeling, programming and business experience, the presence of a large layer of legacy code. Nevertheless, it is worth noting the growth of such decisions not only in business, but also in the scientific community [3, 4]. Along with the development of technologies, the question of the place of these solutions in the future is becoming more acute.

Many owners of large IT companies are promoting the idea that programming and software development should cease to be a highly specialized area [5]. The main idea is that in the future, people will use software tools to solve their personalized tasks. To do this, you do not have to get an education, take special courses. All you need to know is what you want to achieve with the program and how it can make your life easier. Of course, such ideas may seem utopian and unrealizable on the horizon of the next 10–20 years but looking back at the history of the development of the IT industry, we begin to think that all this may happen in the near future [6].

In this article, we provide an overview of an intermediate: a language that can empower people who are familiar with Excel syntax to develop software.

2. Description

At the beginning, we emphasize that this language is part of the Power platform. This is a relatively new vision of Microsoft corporation about the business data warehouse as a single point of connection and storage of data and tools. Together with the Power platform, the Power automate product demonstrates its development - a workflow designer, partly the successor to SSIS, SharePoint workflow engine. Together, these tools are able to close most of the tasks of automating enterprise activities.

To enhance the ability to develop using third-party technologies and programming languages, Microsoft introduced the Microsoft Graph API [7]. Microsoft Graph is the gateway to data and intelligence in Microsoft 365. It provides a unified programmability model that you can use to access the tremendous amount of data in Microsoft 365, Windows 10, and Enterprise Mobility + Security. Use the wealth of data in Microsoft Graph to build apps for organizations and consumers that interact with millions of users. The Microsoft Graph API offers a single endpoint, https://graph.microsoft.com, to provide access to rich, people-centric data and insights in the Microsoft cloud, including Microsoft 365, Windows 10, and Enterprise Mobility + Security. You can use REST APIs or SDKs to access the endpoint and build apps that support Microsoft 365 scenarios, spanning across productivity, collaboration, education, people and workplace intelligence, and much more.

Microsoft Graph also includes a powerful set of services that manage user and device identity, access, compliance, security, and help protect organizations from data leakage or loss. Microsoft Graph connectors (preview) work in the incoming direction, delivering data external to the Microsoft cloud into Microsoft Graph services and applications, to enhance Microsoft 365 experiences such as Microsoft Search. Connectors exist for many commonly used data sources such as Box, Google Drive, Jira, and Salesforce. Microsoft Graph data connect provides a set of tools to streamline secure and scalable delivery of Microsoft Graph data to popular Azure data stores. The cached data serves as data sources for Azure development tools that you can use to build intelligent applications.

Three key traits of Power Fx:

- The future of programming is open. Microsoft has embraced the pace of open innovation that has accelerated the adoption of languages like C# and Typescript. With Power Fx. Microsoft will open-source Power Fx, making the language available for open contribution by the broader community on GitHub.

- Power Fx is based on Microsoft Excel. Using formulas that are already familiar to hundreds of millions of users, Power Fx allows a broad range of people to bring skills they already know to low code solutions. Power Fx becomes a common ground for business users and professional developers alike to express logic and solve problems.
- Power Fx is built for low code. Power Fx is already the foundation of the Microsoft Power Apps canvas.

2.1 Data types [8]:

- **Boolean**. A true or false value. Can be used directly in If, Filter and other functions.Example: *false*
- **Color**. A color specification, including an alpha channel. Example: *ColorValue("#102031")*
- **Currency**. A currency value that's stored in a floating-point number. Example: *333*
- **Date**. A date without a time, in the time zone of the app's user. Example: *Date(2021, 5, 16)*
- **DateTime**. A date with a time, in the time zone of the app's user. Example: *DateTimeValue("May 21, 2019 11:00:09 PM")*
- **GUID**. A Globally Unique Identifier. Example: *GUID()*
- **Hyperlink**. A text string that holds a hyperlink. Example: ma*ke.powerapps.com*
- **Image**. A Universal Resource Identifier (URI) text string to an image in .jpeg, .png, .svg, .gif, or other common web-image format.
- **Media**. A URI text string to a video or audio recording.
- **Number**. A floating-point number. Example: *8.903e121*
- **Option set**. A choice from a set of options, backed by a number. This data type combines a localizable text label with a numeric value. The label appears in the app, and the numeric value is stored and used for comparisons. Example: *ThisItem.OrderStatus*
- **Record**. A record of data values. This compound data type contains instances of other data types that are listed in this topic. More information: Working with tables.
- **Record reference**. A reference to a record in an entity. Such references are often used with polymorphic lookups.
- **Table**. A table of records. All of the records must have the same names for their fields with the same data types, and omitted fields are treated as blank. This compound data type contains instances of other data types that are listed in this topic. More information: Working with tables.
- **Text**. A Unicode text string. Example: *"Hello, World"*
- **Time**. A time without a date, in the time zone of the app's user. Example: Example: *Time(12, 13, 35)*
- **Two option.** A choice from a set of two options, backed by a boolean value. This data type combines a localizable text label with a boolean value. The label appears in the app, and the boolean value is stored and used for comparisons.

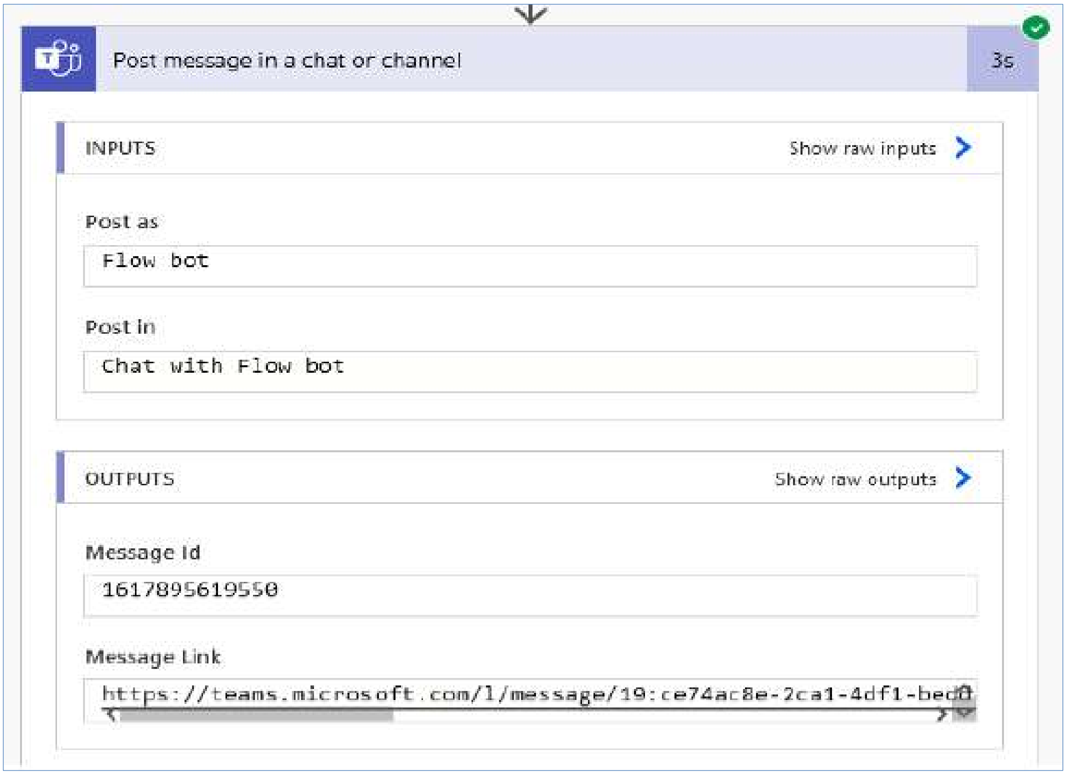
This set of types allows you to store business information of any complexity. Expansion of primitives is planned to support translation of types from one to another.

2.2 Working with datasets

Power Fx supports the If / else construct, but first-time users of this tool may be confused by the lack of a For loop. We tend to attribute this semantic idea to the fast-paced idea of promoting LINQ expressions [9]. Working with sets is based on the following commands:

- **Clear**. The Clear function deletes all the records of a collection. The columns of the collection will remain.Note that Clear only operates on collections and not other data sources. You can use RemoveIf(DataSource, true) for this purpose. Use caution as this will remove all records from the data source's storage and can affect other users. You can use the Remove function to selectively remove records.Clear has no return value. It can only be used in a behavior formula. Example: Clear(DataSource).

- **ClearCollect**. The ClearCollect function deletes all the records from a collection. And then adds a different set of records to the same collection. With a single function, ClearCollect offers the combination of Clear and then Collect.
- **Delegation**. When used with a data source, these functions can't be delegated. Only the first portion of the data source will be retrieved and then the function applied. The result may not represent the complete story. A warning may appear at authoring time to remind you of this limitation and to suggest switching to delegable alternatives where possible. Nevertheless, if it is necessary to organize a For loop to iterate over the data, we can use the extension mechanism. . The most common use case for this mechanism can be represented in the form of interaction with the Microsoft Teams component, where the message acts as a triggering event. Fig. 1 demonstrates a step for triggering an event with receiving/ sending a message to a special channel.


Post message in a chat or channel 3s

INPUTS Show raw inputs

Post as
Flow bot

Post in
Chat with Flow bot

OUTPUTS Show raw outputs

Message Id
1617895619550

Message Link
https://teams.microsoft.com/l/message/19:ce74ac8e-2ca1-4df1-bed

Fig. 1. Send teams message

In this example (listing 1), we can observe the possibility of traversing the collection with a search by values from another source. The key feature here is the fact that the data types in collections can differ in both type and number of values. One of the most common examples is searching on columns with multiple values (User () AAD) [10].

```
CountRows(
            Filter(
                ForAll(
                    'TestArea.Value,
                    If(
                        Value in TArea1.SelectedItems.Value,
                        {t: 1},
                        {t: 0}
                    )
                ),
                t = 1
            )
        ) >= CountRows(TArea1.SelectedItems.Value),
        CountRows(
            Filter(
                ForAll(
                    Product.Value,
                    If(
                        Value in ProductFilterChoice.SelectedItems.Value,
                        {x: 1},
                        {x: 0}
                    )
                ),
                x = 1
            )
        ) >= CountRows(ProductFilterChoice.SelectedItems.Value)
If(
    !IsBlank(CommentInput),
    Patch(
        TEST_Comments,
        Defaults(TEST_Comments),
        {
            TEST_Comments_ID_BP: displayItem.ID,
            TEST_Comments_Comment: CommentInput.Text,
            Title: CommentInput.Text,
            TEST_Comments_User: {
                '@odata.type':
"#Microsoft.Azure.Connectors.SharePoint.SPListExpandedUser",
                Claims: "i:0#.f|membership|" & Lower(User().Email),
                Department: "",
                DisplayName: User().FullName,
                Email: User().Email,
                JobTitle: ".",
                Picture: "."
            }
        }
    );
    Collect(
        Comments,
        {
            Title: CommentInput.Text,
            Created: Now(),
            TEST_Comments_User: {
                '@odata.type':
"#Microsoft.Azure.Connectors.SharePoint.SPListExpandedUser",
                Claims: "i:0#.f|membership|" & Lower(User().Email),
                Department: "",
                DisplayName: User().FullName,
                Email: User().Email,
                JobTitle: ".",
                Picture: "."
            }
        }
    );UpdateIf(
    TestSPCollection,
    ID = displayItem.ID,
    {
        Number_of_Comments: CountRows(Comments)

    }
);
    Reset(CommentInput);

)
collection_originalStatuses = ["New", "Pending", "Complete"]


ForAll(
// loop through the original collection
collection_originalStatuses,
// copy each one to a new collection
Collect(
collection_indexedStatuses,
// create an object with the value (or other props) and an index
{
Value: Value,
// the index starts at 0 and increments as each item is copied
Index: CountRows(collection_indexedStatuses)
}
)
)
collection_indexedStatuses = [
{Value: "New", Index: 0},
{Value: "Pending", Index: 1},
{Value: "Complete", Index: 2}
]
```

Listing 1. Traversing a collection

3. Interim results of the research

In this paper, we have described the main features of the language and ways to expand the functionality. To automate tasks in enterprises, this language is well used, and there is also a practice of reusing components: connectors, XML code. To compare the speed of delivery development, we plan to implement an experiment on the implementation of typical tasks using this language. On a specific example of updating the structure of a multiple field with users and their profiles, the language, as well as the low-code approach, showed excellent results.

References

[1] Rezaee Jordehi. Dynamic environmental-economic load dispatch in grid-connected microgrids with demand response programs considering the uncertainties of demand, renewable generation and market price. International Journal of Numerical Modelling: Electronic Networks, Devices and Fields, vol. 34, no. 1, 2021, 17 p.

[2] R. Sanchis, Ó. García-Perales, F. Fraile, R. Poler. Low-Code as Enabler of Digital Transformation in Manufacturing Industry. Applied Science, vol. 10, no. 1, 2019, 12 p.

[3] M.-L. How. Artificial Intelligence for Social Good in Responsible Global Citizenship Education: An Inclusive Democratized Low-Code Approach. In Proc. of the 3rd World Conference on Teaching and Education, 2021, pp. 81-89.

[4] N. Rauschmayr et al. Amazon SageMaker debugger: a system for real-time insights into machine learning model training. In Proc. of the 4th MLSys Conference, 2021, 13 p.

[5] E. Straschnov. You Shouldn't Have to Learn How to Code. Available at https://www.huffpost.com/entry/you-shouldnt-have-to-lear_b_6111914, accessed Apr. 08, 2021.

[6] L. Floridi and M. Chiriatti. GPT-3: Its Nature, Scope, Limits, and Consequences. Minds and Machines, vol. 30, no. 4, 2020, pp. 681-694.

[7] Overview of Microsoft Graph. Available at https://docs.microsoft.com/en-us/graph/overview, accessed Apr. 08, 2021.
[8] Microsoft Power Fx. Data types. Available at https://github.com/microsoft/Power-Fx/blob/main/docs/data-types.md, accessed Apr. 08, 2021.
[9] Y. Bai. Introduction to Language Integrated Query (LINQ). In SQL Server Database Programming with Visual Basic.NET, Wiley, 2020, pp. 123-213.
[10] T. Shimayoshi, Y. Kasahara, and N. Fujimura. Challenge for Consolidation of Individual Email Services into a Cloud Service. In Proc. of the ACM SIGUCCS Annual Conference, Mar. 2021, pp. 26-29.

Информация об авторах / Information about authors

Илья Александрович ВОРОНКОВ, аспирант. Научные интересы: платформы для совместной работы, ETL.

Ilia Alexandrovich VORONKOV, PhD Student. Research Interests: collaboration platforms, ETL.

Сергей Эрикович САРАДЖИШВИЛИ, кандидат технических наук, доцент. Научные интересы: обработка многомерных сигналов, комплексные системы.

Sergey Erikovich SARADGISHVILI, Candidate of Technical Sciences, Associate Professor. Research interests: multidimensional signal processing, complex systems.

DOI: 10.15514/ISPRAS-2021-33(3)-9

Method of Performance Analysis of Time-Critical Applications Using DB-Nets

A.M. Rigin, ORCID: 0000-0003-4081-9144 <amrigin@edu.hse.ru>
S.A. Shershakov, ORCID: 0000-0001-8173-5970 <sshershakov@hse.ru>

HSE University,
20, Myasnitskaya st., Moscow, 101000, Russia

Abstract. These days, most of time-critical business processes are performed using computer technologies. As an example, one can consider financial processes including trading on stock exchanges powered by electronic communication protocols such as the Financial Information eXchange (FIX) Protocol. One of the main challenges emerging with such processes concerns maintaining the best possible performance since any unspecified delay may cause a large financial loss or other damage. Therefore, performance analysis of time-critical systems and applications is required. In the current work, we develop a novel method for a performance analysis of time-critical applications based on the db-net formalism, which combines the ability of colored Petri nets to model a system control flow with the ability to model relational database states. This method allows to conduct a performance analysis for time-critical applications that work as transactional systems and have log messages which can be represented in the form of table records in a relational database. One of such applications is a FIX protocol-based trading communication system. This system is used in the work to demonstrate applicability of the proposed method for time-critical systems performance analysis. However, there are plenty of similar systems existing for different domains, and the method can also be applied for a performance analysis of these systems. The software prototype is developed for testing and demonstrating abilities of the method. This software prototype is based on an extension of Renew software tool, which is a reference net simulator. The testing input for the software prototype includes a test log with FIX messages, provided by a software developer of testing solutions for one of the global stock exchanges. An application of the method for quantitative analysis of maximum acceptable delay violations is presented. The developed method allows to conduct a performance analysis as a part of conformance checking of a considered system. The method can be used in further research in this domain as well as in testing the performance of real time-critical software systems.

Keywords: performance analysis; time-critical applications; db-nets; FIX protocol; software modeling; software testing

For citation: Rigin A.M., Shershakov S.A. Method of Performance Analysis of Time-Critical Applications Using DB-Nets. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 109-122. DOI: 10.15514/ISPRAS-2021-33(3)-9

Acknowledgements. This work is supported by the Basic Research Program at the HSE University.

Метод анализа производительности критичных по времени приложений с помощью DB-Nets

А.М. Ригин, ORCID: 0000-0003-4081-9144 <amrigin@edu.hse.ru>
С.А. Шершаков, ORCID: 0000-0001-8173-5970 <sshershakov@hse.ru>

Национальный исследовательский университет «Высшая школа экономики», 101000, Россия, г. Москва, ул. Мясницкая, д. 20.

Аннотация. В настоящее время большинство критичных по времени бизнес-процессов выполняются с использованием компьютерных технологий. В качестве примера можно рассмотреть финансовые процессы, включая торговлю на фондовых биржах, использующие такие протоколы передачи информации, как Financial Information eXchange (FIX) Protocol. Один из основных вызовов, возникающих относительно таких процессов, – это поддержка наилучшей производительности, так как любая задержка, не установленная спецификацией, может привести к большим финансовым потерям и иному ущербу. Следовательно, необходимо проводить анализ производительности критичных по времени систем и приложений. В данной работе предложен новый метод для анализа производительности критичных по времени приложений, основанный на формализме db-net. Этот формализм позволяет моделировать поток управления системой с использованием цветных сетей Петри, а также моделировать состояния реляционной базы данных. Метод позволяет проводить анализ производительности критичных по времени приложений, которые работают как транзакционные системы и создают логи с сообщениями, представимыми в форме записей в таблице реляционной базы данных. Примером таких приложений является коммуникационная система для торговли на фондовой бирже, работающая на основе протокола FIX. Эта система рассматривается в данной работе для демонстрации применимости предложенного метода. В то же время существует множество подобных систем в различных предметных областях, и предложенный метод может быть также применён и для анализа производительности таких систем. Для тестирования и апробирования метода разработан программный прототип. Он основан на расширении программного инструмента Renew – симулятора ссылочных сетей Петри. Прототип протестирован на логе, содержащем сообщения протокола FIX, предоставленном разработчиком решений для тестирования программного обеспечения одной из мировых фондовых бирж. Показано применение метода для количественного анализа превышений максимально допустимых задержек между сообщениями. Разработанный метод позволяет выполнять анализ производительности как часть проверки соответствия свойств системы заданной модели (conformance checking). Метод может быть использован как для научно-исследовательских целей, так и для анализа производительности реальных информационных систем.

Ключевые слова: анализ производительности; критичные по времени приложения; db-nets; протокол FIX; моделирование ПО; тестирование ПО.

Для цитирования: Ригин А.М., Шершаков С.А. Метод анализа производительности критичных по времени приложений с помощью DB-Nets. Труды ИСП РАН, том 33, вып. 3, 2021 г., стр. 109-122 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(3)–9.

Благодарности. Работа выполнена в рамках Программы фундаментальных исследований НИУ ВШЭ.

1. Introduction

Nowadays, most of time-critical business processes are performed using computer technologies. Nuclear reactor control, medical equipment control, spaceship control are some obvious examples of such processes. However, different financial processes including trading on stock exchanges also can demand strict performance requirements.

In the previous century, trading on stock exchanges was primarily performed through phone calls and with use of paper-based order books [1]. Working this way did not allow traders to compete for the best price that is generally offered during very short period. This was the reason of beginning of automatization of trading on stock exchanges. In order to guarantee compatibility of software systems of different traders, brokers, and exchanges, there were financial protocols for electronic communication between trading participants created. Financial Information eXchange (FIX)

Protocol maintained by the FIX Trading Community [2] is one of the most known and widely used protocols of such type. There exist different approaches to encode messages transferring with the FIX protocol. In this paper we focus on the *FIX TagValue Encoding*, which is the main standard of encoding FIX messages [2].

The FIX protocol allows traders, brokers, and exchanges to create and fill (execute) orders for buying or selling securities in several milliseconds using electronic communication channels such as Internet [2]. It is a great driver for competence in the global stock markets, however it creates new challenges for financial software vendors. One of such challenges is maintaining the best possible performance. Any unspecified delay may cause a large financial loss for a trader due to the best price is missed. Such delays may create unequal and unfair conditions for different participants, lead to local or global economic problems as well as public scandals and reputational problems for the exchange or some traders or brokers.

Financial protocol-based communication systems are considered in this work to demonstrate applicability of the proposed method for time-critical systems performance analysis. However, there are plenty of similar systems existing for different domains, and the method can also be applied for a performance analysis of these systems.

Any FIX message consists of a set of tag-value pairs [3]. In fact, it means that we can represent these messages in the form of records of a table in some relational database. Therefore, some methods of system modeling, which rely on relational database states, can be considered here. The same is valid not only for messages of the FIX protocol, but for any messages of transactional systems that are represented as sets of tag-value pairs.

In 2020, we developed a software simulator for the db-net formalism [4] introduced by Montali and Rivkin in 2017. This formalism is represented by the layer with modified colored Petri net modeling a control flow of a process system, and two inner layers for working with an attached relational database modeling a persistent storage [5] as shown in Fig. 1. This simulator is developed as a plugin for Renew (Reference Net Workshop) software tool which is a Java-based reference net simulator [6].

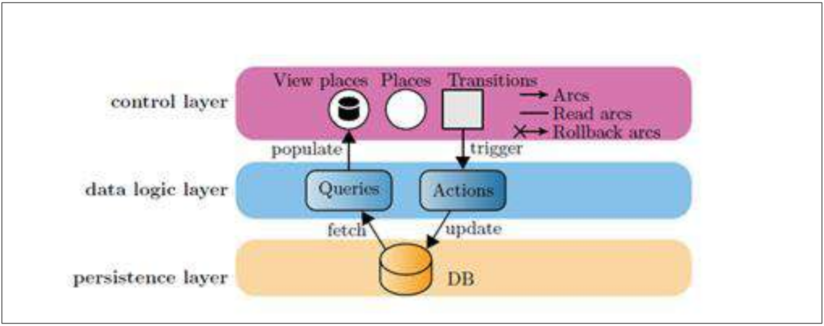


Fig. 1. The db-net structure [5].

Generally, the lowest layer of the db-net (the persistence layer) is represented by an ordinary relational database [5]. However, it can be replaced with any other information storage, which is accessible through a custom relational DML interface that is to be implemented.

One can model a tag-value message sending by using the “insert” database operation, where tags are represented as attributes of a relational table and values are represented as attributes of a record in the table. A tag-value message receiving can be modeled similarly using the “select” database operation.

In the current time, there are some performance analysis research works focused on distributed software systems such as [7, 8], however performance analysis using db-nets has its advantages for transactional systems which send and receive messages that are representable in the form of records in relational tables.

Firstly, it allows to integrate performance analysis into conformance checking of a system. A performance property can be considered together with other checked properties of the system to check all of them simultaneously. Therefore, it allows to abstract away from the performance and to combine performance analysis of transactional systems with other methods for their verification and validation, based on Petri nets and their modifications, especially db-nets (e.g., checking safety, liveness, fairness, and similar properties). Moreover, colored Petri net models, that are automatically generated from event logs using process discovery algorithms, may be extended with db-net elements and time constraints, and used for performance analysis.

Secondly, this method allows to apply well-known approaches used in the relational database domain to the wide set of transactional systems supporting time-critical applications.

All the above provides the motivation for the research.

The purpose of the research is development of a method of performance analysis of time-critical applications using db-nets.

The objectives of the research are as follows.

1) Developing a method for performance analysis of time-critical applications using db-nets.
2) Developing a software prototype for performance analysis of time-critical application logs using db-net models.
3) Checking the method by testing the developed software prototype on a test log of FIX messages provided by a software developer of testing solutions for one of the global stock exchanges.

The rest of the paper is organized as follows. The Section 2 presents the theoretical foundations and concepts of the work and the developed method. In the Section 3, the developed software prototype and its testing are described. After this, the main points of the paper are summarized in the conclusion.

2. Performance Analysis Using DB-Nets

2.1 DB-Nets

The db-net formalism is a modification of the colored Petri net, which allows to model a system control flow together with relational database states. The db-net consists of three layers: (1) the control layer, (2) the data logic layer which connects the control layer and the persistence layer together, and (3) the persistence layer [5]. The scheme of db-net structure is shown in Fig. 1.

The persistence layer allows to store the persistent data and is formally defined by a relational database schema and constraints that declare the data consistency rules [5].

The data logic layer is defined by two sets: (1) set of *queries* for retrieving records from a database in the persistence layer and (2) set of *actions* for insertion and deletion of records in the persistence layer database. Each action includes sets of added and deleted *facts* (records in relational tables) [5].

The control layer allows to model a system control flow and is defined by a colored Petri net with the following modifications [5].

1) Queries defined in the data logic layer are assigned to places of a colored Petri net in the control layer. Such places are called view places. View places cannot contain tokens (resources modeled in a Petri net) such as other places, but they produce new tokens by retrieving data from the persistence layer through assigned queries.
2) Actions defined in the data logic layer are assigned to transitions of the net in the control layer. When a transition with the assigned action is fired (executed), the action is performed on a database in the persistence layer.
3) In addition to traditional Petri net arcs, there exist read arcs and rollback arcs in the db-net control layer. The former is used for connecting view places with transitions and the latter is used for defining a flow for a case of rollback of an action due to violation of the data consistency rules in a database of the persistence layer after performing the action.

The db-net control layer's net and persistence layer's database schema example for the taxi booking software system is shown in Fig. 2.

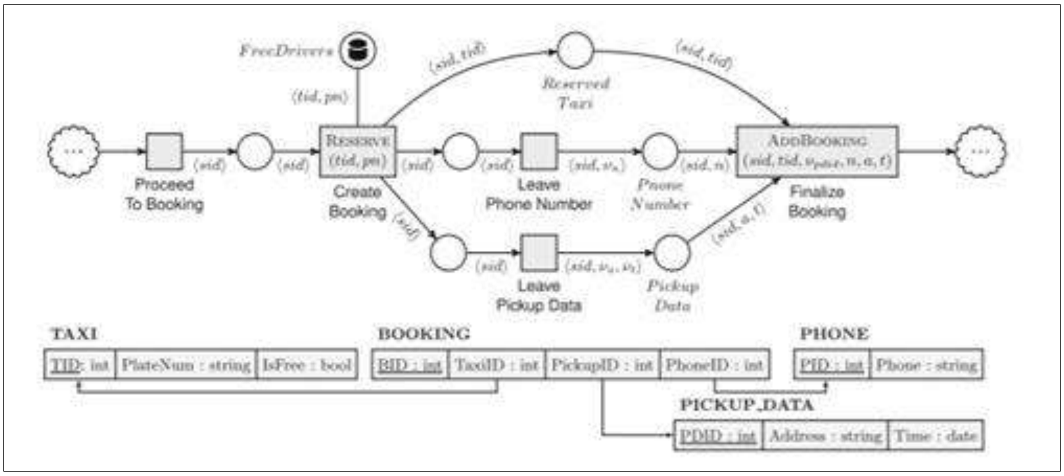


Fig. 2. The control layer's net and persistence layer's database schema example for the taxi booking software system [5].

2.2 Conformance Checking

Conformance checking allows to verify that a considered system satisfies desirable properties through ensuring that an event log produced by the system fits a designed model [9]. These properties include safety, liveness, fairness, and similar ones. For example, safety properties guarantee that the system does not achieve certain undesirable states.

The method proposed in this paper allows to check performance simultaneously with checking other properties of a system. A performance property is considered in this work as a safety property for satisfying that a system should not achieve the state where a delay between two messages exceeds a maximum acceptable one. Therefore, performance can be checked in a process model designed for checking other properties by extending the model with information about time constraints [9].

Since the db-net formalism extends colored Petri net, it is possible to check all properties using a db-net if these properties can be checked using a traditional colored Petri net. The performance in the proposed method is checked using db-net elements.

The set of properties $S = \{s_1, s_2, \dots, s_n\}$, where $s_i, i = \overline{1,n}$ is the i-th checked property, is considered as an example. Some of these properties may be performance properties. The set of performance properties is $P = \{p_1, p_2, \dots, p_m\}$, where $P \subseteq S$. Each performance property $p_j, j = \overline{1,m}$ contains time constraints in the form of a maximum acceptable delay for pairs of messages of particular types as specified in the proposed method (the subsection 2.3). Each property $p_j, j = \overline{1,m}$ such that $p_j \in P$ is checked by the proposed method using db-nets. Other properties $s_i, i = \overline{1,n}$ such that $s_i \in S \setminus P$ are checked by other methods utilizing colored Petri nets and db-nets.

As a result, performance analysis can be conducted as a part of conformance checking of a system, where performance properties are among of all checked properties. This allows to abstract away from the performance properties and check all properties simultaneously.

2.3 Method of Performance Analysis Using DB-Nets

The developed method implies analyzing messages sent or received by the application or its modeled part (*request* and *response messages*, respectively) and stored in a log of the application. We analyze those messages for which a maximum delay between sent message and received response is restricted. The method utilizes the db-net formalism.

The method consists of two parts: (1) set of requirements for implementing the method in a software tool and (2) sequence of stages and steps for using the method after being implemented.

2.3.1 Implementing the Method in a Software Tool

The following set of requirements specifies how the method of performance analysis using db-nets is to be implemented as a software tool. These requirements extend general principles of the db-net behavior, which are described in [5].

1) When a *request message* is inserted in an action assigned to a db-net transition, it should be stored in the memory (RAM or persistent storage) for further retrieving when the corresponding response message is retrieved.
2) When a *response message* is retrieved by a "select" query assigned to a db-net view place *and* the connected by a read arc db-net transition contains parameters for performance analysis as specified in the step 6 of the stage 1 of the method (the Section 2.3.2), the following sequence of steps is to be executed:
    a) The corresponding request message (with the same *id* attribute value) is to be retrieved through the specified query from the memory/storage (as specified in the item 1 of the current set of requirements).
    b) If there is no stored corresponding request message, then this sequence is to be stopped and the token with the response message is to be moved to the places connected by output arcs.
    c) The *sending timestamps* of the request and response messages are to be parsed using a specified pattern or a regular expression.
    d) A *delay* that is a difference (in milliseconds) between these two *sending timestamps* is to be calculated. If it exceeds the specified *maximum acceptable value of a delay*, then the validation is to be considered as failed – information about the *id* and *message type* of the problematic messages is to be displayed or stored in the report (depending on the requirements and implementation), for the first violation or for each violation (also depending on the requirements and implementation).
3) If there are several response messages for one request message, only the first response message is considered.
4) If the simulation is finished (no transitions can be *fired* – executed) and the validation did not fail, then such validation is considered as succeeded.

2.3.2 Use of the Method

After implementing the software tool, the method is to be used by following the sequence of steps divided into three stages, as follows.

Stage 1. Modeling a DB-Net. A db-net that matches a system/a modeled part of a system is to be modeled using the following steps.

1) A scope of the modeled system is to be defined. It should include considered components of the system which send *request messages* (messages sent by the system or its considered component) and get responses to them (*response messages*), and considered types of *request messages* and corresponding types of *response messages*. From now on, we will call a modeled system/part of the system a *time-critical application* (or just an *application*).
2) It is necessary to make sure that the application works as a transactional system and satisfies the ACID (atomicity, consistency, isolation, durability) properties [10], and a log with its request and response messages can be represented in the form of tables in a relational database. It means that each message includes a set of *tags* (attributes) together with their *values*. *Tags* are represented as attributes of a relational table, messages are represented as records of the

table, and *values* are represented as attributes of a record in the table. Types of messages and parts of the application which do not satisfy these properties, if any, are to be removed from the scope.

3) A persistence layer of the modeled db-net is to be defined. To do this, a relational database schema is to be created and populated with necessary tables. The table attributes reflect the tags of considered request and response messages.
4) A data logic layer of the modeled db-net is to be defined. The «insert» queries, which model insertion of the request messages into the modeled relational database, are to be specified. The «select» queries, which model retrieving the request and response messages from the modeled relational database, should similarly be specified.
5) A model of a system control flow (a control layer of the modeled db-net) is to be defined. After that, «insert» and «select» queries from the modeled data logic layer are assigned to transitions and view places, respectively.
6) For each db-net transition connected by a read arc with a view place that is assigned with a «select» query for retrieving the response messages, the following parameters for conducting a performance analysis are to be specified:
   a) The name of a variable in the control layer that stores a value of the *id* attribute of a response message, which allows to find a corresponding request message by the same value of the same *id* attribute.
   b) The name of a variable in the control layer that stores a value of the *sending timestamp* attribute of a response message.
   c) An ordering number of the *sending timestamp* attribute of a message in results of a "select" query for retrieving the corresponding request message, that is mentioned in the item “f” of the current list.
   d) A pattern or a regular expression for parsing the *sending timestamp* string in a message.
   e) An ordering number of the *message type* attribute of a message in results of a "select" query for retrieving the corresponding request message, that is mentioned in the item “f” of the current list.
   f) The name of a declared "select" query for retrieving the corresponding request message.
   g) The *maximum acceptable value of a delay* between *sending timestamps* of corresponding request and response messages (in milliseconds).

Stage 2. Preprocessing the Log. Preparing a log of the application includes the following steps.

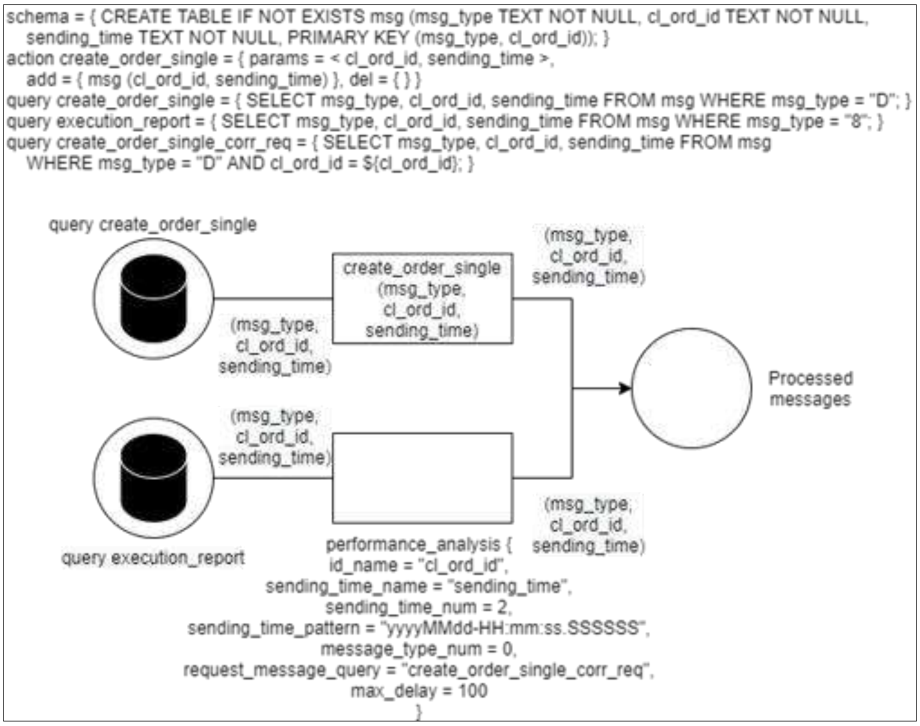
1) It is necessary to make sure that the messages in a log are represented in a form satisfying properties described in the step 2 of the stage 1. Any messages that are not represented in a valid form as well as broken messages are to be removed.
2) The log should be prepared in a format compatible with a software tool implementing the method.

Stage 3. Conducting a Performance Analysis Using DB-Nets. A simulation of the modeled db-net is to be run in the software tool implementing the method.

2.4 Example of Performance Analysis Using DB-Nets for the FIX Protocol

The developed method is illustrated by an example modeling a trading order creation with use of the FIX protocol. The example includes the analysis of two types of FIX messages: (1) *create_order_single* (*msg_type = “D”*) which is used for request messages sent from a trader or a broker to the exchange, to create an order for buying or selling securities, and (2) *execution_report* (*msg_type = “8”*) which is used for response messages sent from the exchange to the trader or the broker as a confirmation of the order creation (or information about the order rejection with clarification of a reason). For each message, the attributes *msg_type*, *cl_ord_id* and *sending_time* are considered in the model. The *msg_type* attribute defines a type of the message. The corresponding request/response messages are connected by a key (id), whose role is played by the *cl_ord_id* attribute. The *sending_time* attribute is a sending timestamp of the message.

The db-net modeling this example is shown in Fig. 3. A schema of a relational database in the db-net persistence layer includes a *msg* relational table for storing FIX messages. The table contains *msg_type*, *cl_ord_id* and *sending_time* attributes. The *create_order_single* action models the "insert" DML query for insertion of the *msg_type*, *cl_ord_id* and *sending_time* attributes of the *create_order_single* FIX message. The *create_order_single* and *execution_report* queries model the "select" SQL query for retrieving the same attributes of the *create_order_single* and *execution_report* FIX messages, respectively. The *create_order_single_corr_req* query models the "select" SQL query for retrieving the same attributes of the *create_order_single* FIX message by the given *cl_ord_id*. It is used for retrieving the corresponding request message for a previously retrieved response message.



```
schema = { CREATE TABLE IF NOT EXISTS msg (msg_type TEXT NOT NULL, cl_ord_id TEXT NOT NULL,
   sending_time TEXT NOT NULL, PRIMARY KEY (msg_type, cl_ord_id)); }
action create_order_single = { params = < cl_ord_id, sending_time >,
   add = { msg (cl_ord_id, sending_time) }, del = { } }
query create_order_single = { SELECT msg_type, cl_ord_id, sending_time FROM msg WHERE msg_type = "D"; }
query execution_report = { SELECT msg_type, cl_ord_id, sending_time FROM msg WHERE msg_type = "8"; }
query create_order_single_corr_req = { SELECT msg_type, cl_ord_id, sending_time FROM msg
   WHERE msg_type = "D" AND cl_ord_id = ${cl_ord_id}; }
```

```
performance_analysis {
id_name = "cl_ord_id",
sending_time_name = "sending_time",
sending_time_num = 2,
sending_time_pattern = "yyyyMMdd-HH:mm:ss.SSSSSS",
message_type_num = 0,
request_message_query = "create_order_single_corr_req",
max_delay = 100
}
```

query create_order_single
create_order_single (msg_type, cl_ord_id, sending_time)
(msg_type, cl_ord_id, sending_time)
(msg_type, cl_ord_id, sending_time)
Processed messages
(msg_type, cl_ord_id, sending_time)
(msg_type, cl_ord_id, sending_time)
query execution_report

Fig. 3. Example of a db-net model for a performance analysis of a FIX protocol-based system.

The view place assigned with *create_order_single* query (fig. 3) is responsible for retrieving messages of *create_order_single* type. The following transition executes the *create_order_single* action, modeling insertion of the messages into the *msg* table. Then the transition transfers the messages to the *Processed messages* place.

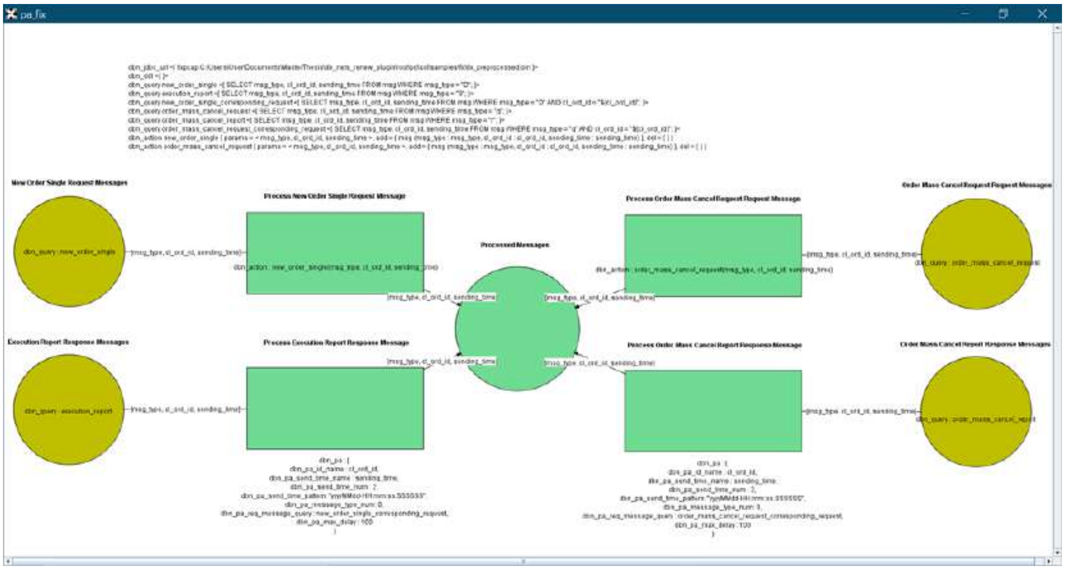
The view place assigned with *execution_report* query (fig. 3) is responsible for retrieving messages of *execution_report* type. After retrieving an *execution_report* message, the following transition retrieves the corresponding *create_order_single* message (with *msg_type = "D"* and the same *cl_ord_id*) using the *create_order_single_corr_req* query and calculates a delay between these two messages as a difference between their sending timestamps (the *sending_time* attribute). If the calculated delay exceeds *max_delay* (it is 100 ms in the example), then the validation fails. Otherwise, the *execution_report* message is transferred to the *Processed messages* place. After all messages are retrieved from the log, and validation did not fail, the validation of the log is considered succeeded.

We consider two following FIX messages (these messages are presented below in the human-readable form, not in the original FIX tag-value form): (1) *create_order_single (msg_type = "D", cl_ord_id = "12345", sending_time = "20190218-02:14:45.490000")* and (2) *execution_report (msg_type = "8", cl_ord_id = "12345", sending_time = "20190218-02:14:45.492787")*. Firstly, the *create_order_single* message is retrieved by the view place assigned with the *create_order_single* query. The following transition performs the *create_order_single* action with the "insert" DML query for this message and transfers the message to the *Processed messages* place. Secondly, the *execution_report* message is retrieved by the view place assigned with the *execution_report* query. By the *cl_ord_id = "12345"* attribute value of the message, the following transition retrieves the corresponding *create_order_single* message (with *msg_type = "D"* and the same *cl_ord_id = "12345"*) using the *create_order_single_corr_req* query and calculates a delay between these two messages as a difference between their sending timestamps (the *sending_time* attribute). This delay equals 3 ms (rounding up). A maximum acceptable delay linked with the transition is defined to 100 ms. The delay does not exceed the maximum acceptable delay, so the validation does not fail, and the *execution_report* message is transferred to the *Processed messages* place. However, if the *sending_time* attribute value of the *execution_report* message was, for example, *"20190218-02:14:45.592787"*, then the delay would be equal to 103 ms (rounding up) and the maximum acceptable delay would be exceeded which would lead the validation to fail.

3. Software Prototype

3.1 Software Prototype Features and Implementation

For testing and illustrating abilities of the method, the latter is implemented in the form of a software prototype. For doing this, we developed the db-net software simulator (Renew DB-Nets Plugin) in 2020 [4] and then extended it with features for conducting a performance analysis of time-critical applications using the proposed method. The simulator has a form of a plugin for Renew software tool which is a Java-based reference net simulator [6]. The simulator has a graphical user interface as shown in the screenshot in fig. 4.


pa_fix
New Order Single Request Messages
Process New Order Single Request Message
Processed Messages
Process Order Mass Cancel Request Request Message
Order Mass Cancel Request Request Messages
Execution Report Response Messages
Process Execution Report Response Message
Process Order Mass Cancel Report Response Message
Order Mass Cancel Report Response Messages
dbn_query : new_order_single
dbn_query : execution_report
dbn_query : order_mass_cancel_request
dbn_query : order_mass_cancel_report
dbn_action : new_order_single(msg_type, cl_ord_id, sending_time)
dbn_action : order_mass_cancel_request(msg_type, cl_ord_id, sending_time)
[msg_type, cl_ord_id, sending_time]

```
dbn_jdbc_url<{ tsp:sap:C:/Users/User/Documents/Master/Thesis/db_nets_renew_plugin/rootof/dist/samples/fix/fix_preprocessed.csv }>
dbn_ddl<{ }>
dbn_query new_order_single<{ SELECT msg_type, cl_ord_id, sending_time FROM msg WHERE msg_type="D"; }>
dbn_query execution_report<{ SELECT msg_type, cl_ord_id, sending_time FROM msg WHERE msg_type="8"; }>
dbn_query new_order_single_corresponding_request<{ SELECT msg_type, cl_ord_id, sending_time FROM msg WHERE msg_type="D" AND cl_ord_id="${cl_ord_id}"; }>
dbn_query order_mass_cancel_request<{ SELECT msg_type, cl_ord_id, sending_time FROM msg WHERE msg_type="q"; }>
dbn_query order_mass_cancel_report<{ SELECT msg_type, cl_ord_id, sending_time FROM msg WHERE msg_type="r"; }>
dbn_query order_mass_cancel_request_corresponding_request<{ SELECT msg_type, cl_ord_id, sending_time FROM msg WHERE msg_type="q" AND cl_ord_id="${cl_ord_id}"; }>
dbn_action new_order_single [ params=<msg_type, cl_ord_id, sending_time>, add={ msg (msg_type : msg_type, cl_ord_id : cl_ord_id, sending_time : sending_time) }, del={ } ]
dbn_action order_mass_cancel_request [ params=<msg_type, cl_ord_id, sending_time>, add={ msg (msg_type : msg_type, cl_ord_id : cl_ord_id, sending_time : sending_time) }, del={ } ]
```

```
dbn_pa {
dbn_pa_id_name : cl_ord_id,
dbn_pa_send_time_name : sending_time,
dbn_pa_send_time_num : 2,
dbn_pa_send_time_pattern : "yyyyMMdd-HH:mm:ss.SSSSSS",
dbn_pa_message_type_num : 0,
dbn_pa_req_message_query : new_order_single_corresponding_request,
dbn_pa_max_delay : 100
}
```

```
dbn_pa {
dbn_pa_id_name : cl_ord_id,
dbn_pa_send_time_name : sending_time,
dbn_pa_send_time_num : 2,
dbn_pa_send_time_pattern : "yyyyMMdd-HH:mm:ss.SSSSSS",
dbn_pa_message_type_num : 0,
dbn_pa_req_message_query : order_mass_cancel_request_corresponding_request,
dbn_pa_max_delay : 100
}
```

Fig. 4. Screenshot of a graphical user interface of the developed software prototype.

The prototype allows to (1) model a db-net for a considered system, (2) specify parameters for conducting a performance analysis of time-critical applications, as described in the step 6 of the stage 1 of the described method (the Section 2.3.2), (3) conduct a performance analysis of an application in parallel with a db-net model simulation using the proposed method and (4) work with a FIX log (raw binary data of the FIX protocol packages captured as a Wireshark PCAP file [11] with further filtering) through a relational DML interface.

An implementation of the developed db-net simulator is described in [4]. This implementation is based on an implementation of Renew software tool, a reference Petri net simulator. The Renew code which was suitable for the db-net behavior is reused. Other code is overridden by a custom db-net implementation. Classes representing elements of the db-net control layer are inherited from Renew classes representing similar elements of traditional colored Petri nets and necessary methods are overridden. The prototype is implemented as a pure plugin for Renew tool, without modifying existing Renew source code [4]. The plugin code, UML class diagram and documentation are available in the project GitHub repository[1].

For working with a FIX log through a relational DML interface, the alternative implementation of the database connection interface is created. It is used if the JDBC URL in a db-net model starts from the *"fixpcap:"* prefix. All messages that are read from file through this connection are stored in RAM (in the *java.util.HashMap* container, where keys, which are pairs of *message type* and *id*, are stored in a hashtable). When the message is being retrieved through this connection, it is firstly searched in RAM. If it is found in RAM, it is returned and removed from RAM. If it is not found in RAM, then the file is scanned until finding this message (and all scanned messages are stored in RAM). This approach allows to scan each line of the file only once and to minimize the RAM usage.

For goals of a performance analysis, the prototype follows the set of requirements described in the subsection 2.3.1. When the first *maximum acceptable delay* violation is detected while simulating a db-net model, the dialog window with an information message describing this violation is shown and the corresponding CSV report is created. All *maximum acceptable delay* violations that are detected during the current simulation are written into the created CSV report. The format of a CSV report is presented in Table. 1.

Table. 1. Columns of the CSV Report

Column Name	**Description**	**Type**	**Example**
#	Order number of the row in the CSV report (starting from 1)	Integer	1
Request Message Type	Type of the request message	String	*D*[2]
Message ID	ID of the request and response message pair	String	15504
Delay	Difference (in milliseconds) between request and response message sending timestamps	Integer	493
Max Delay	Maximum acceptable delay	Integer	100
Diff	Difference between detected delay and maximum acceptable delay	Integer	393

[1] Link: https://github.com/Glost/db_nets_renew_plugin

[2] In the FIX Protocol, the *D* message type is used for the *New Order Single* messages.

3.2 Testing the Prototype on the FIX Log and Quantitative Analysis of Maximum Acceptable Delay Violations

The developed software prototype is tested on a log with FIX protocol messages, which is represented by the raw binary data extracted from a Wireshark PCAP file with some FIX protocol messages captured in the testing environment. The file is provided by a software developer of testing solutions for one of the global stock exchanges.

The screenshot in fig. 4 shows the db-net model for performance analysis applied to the FIX protocol messages for the *New Order Single* scenario (request message: *New Order Single*, message type: *"D"*; response message: *Execution Report*, message type: *"8"*) and the *Order Mass Cancel Request* scenario (request message: *Order Mass Cancel Request*, message type: *"q"*; response message: *Order Mass Cancel Report*, message type: *"r"*). The total number of processed messages in this model equals 321671.

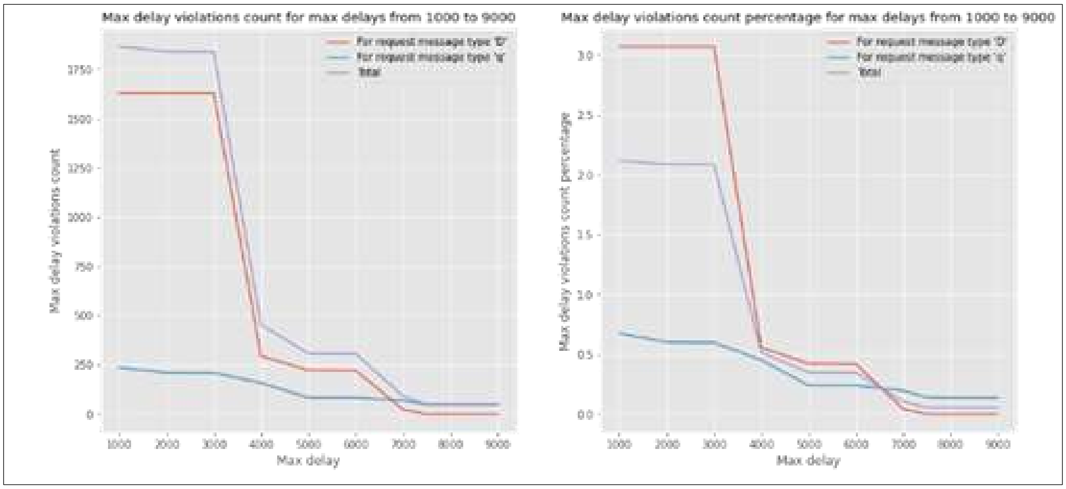


Fig. 5. Quantitative analysis of maximum acceptable delay violations for maximum acceptable delay values from 1000 ms to 9000 ms.

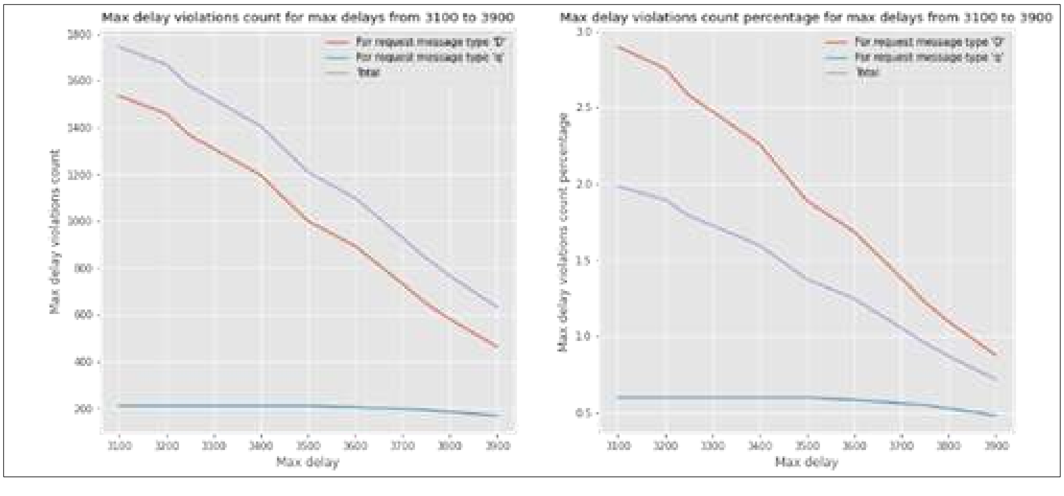


Fig. 6. Quantitative analysis of maximum acceptable delay violations for maximum acceptable delay values from 3100 ms to 3900 ms.

Using this model, the quantitative analysis of *maximum acceptable delay* violations is conducted based on the CSV reports with information about violations. The plots in Fig. 5 show (1) counts of *maximum acceptable delay* violations and (2) percentages (ratios) of message pairs with *maximum acceptable delay* violations (where 100 % is all processed message pairs), in the db-net model described above, with a breakdown to the request message types (*"D"* is used for the *New Order Single* messages and *"q"* is used for the *Order Mass Cancel Request* messages) for maximum acceptable delay values from 1000 ms to 9000 ms.

The significant decrease in count of violations between *maximum acceptable delay* values 3000 ms and 4000 ms is notable. The plots in Fig. 6 show the same metrics for *maximum acceptable delay* values from 3100 ms to 3900 ms. We can conclude that the most of delays larger than 1 second are between 3 and 4 seconds.

Such quantitative analysis is an example of possible applications of the developed method. For instance, requirements and service level agreements (SLAs) can be specified and adjusted basing on some statistics on ratio of message pairs violating each *maximum acceptable delay*. This information with a breakdown to the request message types allows to focus on improving the speed of the most critical scenarios.

4. Conclusion

In the current work, a novel method of performance analysis of time-critical applications based on the db-net formalism is developed. This method allows to integrate performance analysis into conformance checking of a system. Therefore, it allows to abstract away from performance and to combine performance analysis of transactional systems with other methods for their verification and validation, based on Petri nets and their modifications, especially db-nets (e.g., checking safety, liveness, fairness, and similar properties). Colored Petri net models, that are automatically generated from event logs using process discovery algorithms, may be extended with db-net elements and time constraints, and used for performance analysis. Moreover, the method allows to apply well-known approaches used in the relational database domain to a wide set of transactional systems supporting time-critical applications.

A software prototype implementing the method is developed. The prototype is checked on a test log with FIX messages provided by a software developer of testing solutions for one of the global stock exchanges. A quantitative analysis of maximum acceptable delay violations is conducted based on this log. This demonstrates how the method can be applied for similar analysis.

The developed method can be used in research in this domain as well as in testing performance of real time-critical software systems. Further steps include extending the method for use with hierarchical Petri nets and more complex variants of performance analysis of transactional systems. Approbation of the method for integrating performance analysis into conformance checking of a real software system is planned.

The developed software prototype is to be improved for being more usable. This will make the prototype a new software tool in the pool of open-source solutions for conformance checking and performance analysis.

References

[1]. Harris L. Back Office Operations. Trading and Exchanges: Market Microstructure for PractitionersOxford Univyversity Press, 2003, chapter 7, section 7.2.2, pp. 148-149.

[2]. Introduction, FIX Trading Community, Available at: https://www.fixtrading.org/online-specification/introduction/, accessed 28.03.2021.

[3]. FIX TagValue Encoding, FIX Trading Community, Available at: https://www.fixtrading.org/standards/tagvalue-online/, accessed 28.03.2021.

[4]. Rigin A., Shershakov S. Data and Reference Semantic-Based Simulator of DB-Nets with the Use of Renew Tool. Lecture Notes in Computer Science, vol. 12602, 2021, pp. 453-465, DOI: 10.1007/978-3-030-72610-2_34.

[5]. Montali M., Rivkin A. DB-Nets: On the Marriage of Colored Petri Nets and Relational Databases. Lecture Notes in Computer Science, vol. 10470, 2017, pp. 91-118.
[6]. Renew – The Reference Net Workshop. Renew.de, Available at: http://www.renew.de/, accessed 28.03.2021.
[7]. Vetter J. Performance analysis of distributed applications using automatic classification of communication inefficiencies. In Proc. of the 14th international conference on Supercomputing (ICS '00), 2000, pp. 245-254.
[8]. Marsan M. A., Bianco A. et al. A LOTOS extension for the performance analysis of distributed systems. IEEE/ACM Transactions on Networking, vol. 2, no. 2, 1994, pp. 151-165.
[9]. van der Aalst W., Adriansyah A., van Dongen B. Replaying history on process models for conformance checking and performance analysis. WIREs Data Mining and Knowledge Discovery, vol. 2, no. 2, 2012, pp. 182-192.
[10]. Haerder T., Reuter A. Principles of transaction-oriented database recovery. ACM Computing Surveys, vol. 15, no. 4, 1983, pp. 287-317.
[11]. 5.2. Open Capture Files, Wireshark.org, Available at: https://www.wireshark.org/docs/wsug_html_chunked/ChIOOpenSection.html, accessed 28.03.2021.

Информация об авторах / Information about authors

Антон Михайлович РИГИН получил степень магистра в области системной и программной инженерии в 2021 г. в Национальном исследовательском университете «Высшая школа экономики» (Москва, Россия). Его исследовательские интересы включают программную инженерию, извлечение и анализ процессов (process mining), верификацию программного обеспечения, алгоритмы и структуры данных и их применение в задачах индексирования и хранения данных в системах управления базами данных.

Anton Mikhailovich RIGIN received his master’s degree in System and Software Engineering from the National Research University Higher School of Economics (Moscow, Russia) in 2021. His research interests include software engineering, process mining, software verification, algorithms and data structures and their usage in problems of data indexing and storage in database management systems.

Сергей Андреевич ШЕРШАКОВ получил степень кандидата компьютерных наук Национального исследовательского университета «Высшая школа экономики» (Москва, Россия) в 2020 году. В настоящий момент он является доцентом департамента больших данных и информационного поиска и научным сотрудником научно-учебной лаборатории процессно-ориентированных информационных систем (Лаборатории ПОИС) факультета компьютерных наук Высшей школы экономики. В число научных интересов входят извлечение и анализ процессов (process mining), верификация программного обеспечения, архитектуры информационных систем и преподавание программной инженерии.

Sergey Andreevich SHERSHAKOV received his PhD degree in Computer Science from the National Research University Higher School of Economics (Moscow, Russia) in 2020. He is currently an Associate Professor at the Big Data and Information Retrieval School and a research fellow at the Laboratory of Process-Aware Information Systems (PAIS Lab) of the Faculty of Computer Science at the HSE University. His research interests include process mining, software verification, information system architectures and teaching software engineering.

DOI: 10.15514/ISPRAS-2021-33(3)-10

Historical Civil Registration Record Transcription Using an eXtreme Model Driven Approach

[1] *R. Khan, ORCID: 0000-0001-9006-6748 <Rafflesia.Khan@ul.ieg>*
[1]*A. Schieweck, ORCID: 0000-0002-5008-9168 <Alexander.Schieweck@ul.ieg>*
[1,2] *C. Breathnach, ORCID: 0000-0002-4065-0660 <Ciara.Breathnach@ul.ieg>*
[1,2] *T. Margaria, ORCID: 0000-0002-5547-9739 <Tiziana.Margaria@ul.ieg >*
[1] *University of Limerick,*
Limerick, V94 T9PX, Ireland
[2] *Lero: The Irish Software Research Centre*
Tierney Building, University of Limerick, Ireland

Abstract. Modelling is considered as a universal approach to define and simplify real-world applications through appropriate abstraction. Model-driven system engineering identifies and integrates appropriate concepts, techniques, and tools which provide important artefacts for interdisciplinary activities. In this paper, we show how we used a model-driven approach to design and improve a Digital Humanities dynamic web application within an interdisciplinary project that enables history students and volunteers of history associations to transcribe a large corpus of image-based data from the General Register Office (GRO) records. Our model-driven approach generates the software application from data, workflow and GUI abstract models, ready for deployment.

Keywords: Software and System Engineering; Model-Driven Development; Web Application; Historical Civil Record; Digital Humanities; XMDD; DIME

For citation: Khan R., Schieweck A., Breathnach C., Margaria T. Historical Civil Registration Record Transcription Using an eXtreme Model Driven Approach. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 123-142. DOI: 10.15514/ISPRAS-2021-33(3)-10

Acknowledgments. We are grateful for the full cooperation of the Registrar General of Ireland for permission to use these data for research purposes. This research is funded by the Irish Research Council Laureate Award 2017/32 and by Science Foundation Ireland through the grants 13/RC/2094 to Lero - the Irish Software Research Centre (www.lero.ie).

“Death and Burial Data: Ireland 1864-1922” is a project funded by Irish Research Council Laureate Award IRCLA/2017/32 to Dr. Ciara Breathnach (Department of History – DH), in cooperation with Prof. Tiziana Margaria (Software Systems, Dept of Computer Science and Information Systems – CSIS) at the University of Limerick.

Транскрипция исторических записей актов гражданского состояния с использованием экстремального модельно-управляемого подхода

[1] *Р. Хан, ORCID: 0000-0001-9006-6748 <Rafflesia.Khan@ul.ieg>*
[1] *А. Шивек, ORCID: 0000-0002-5008-9168 <Alexander.Schieweck@ul.ieg>*
[1,2] *С. Бретнах, ORCID: 0000-0002-4065-0660 <Ciara.Breathnach@ul.ieg>*
[1,2] *Т. Маргария, ORCID: 0000-0002-5547-9739 <Tiziana.Margaria@ul.ieg >*
[1] *Университет Лимерика,*
Ирландия, V94 T9PX, Лимерик
[2] *Lero: Ирландский центр исследований программного обеспечения*
Ирландия, Лимерикский университет, Здание Тирни

Аннотация. Моделирование считается универсальным подходом к определению и упрощению реальных приложений с помощью соответствующей абстракции. Системная инженерия, управляемая моделями, определяет и объединяет соответствующие концепции, методы и инструменты, которые обеспечивают важные артефакты для междисциплинарной деятельности. В этой статье мы покажем, как мы использовали подход, основанный на моделях, для разработки и улучшения динамического веб-приложения цифровых гуманитарных наук в рамках междисциплинарного проекта, который позволяет студентам-историкам и волонтерам исторических ассоциаций транскрибировать большой корпус изображений документов Управления записью актов гражданского состояния. Наш подход, основанный на моделях, генерирует программное приложение на основе абстрактных моделей данных, рабочего процесса и графического интерфейса пользователя, готовых к развертыванию.

Ключевые слова: программная и системная инженерия; модельно-управляемая разработка; Web-приложение; исторические записи актов гражданского состояния; цифровые гуманитарные науки; XMDD; DIME

Для цитирования: Хан Р., Шивек А., Бретнах С., Маргария Т. Транскрипция исторических записей актов гражданского состояния с использованием экстремального модельно-управляемого подхода. Труды ИСП РАН, том 33, вып. 3, 2021 г., стр. 123-142 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(3)–10.

Благодарности. Мы благодарны Генеральному регистратору Ирландии за разрешение использовать данные в исследовательских целях. Это исследование финансируется Ирландским исследовательским советом по лауреатским премиям 2017/32 и Научным фондом Ирландии через гранты 13/RC/2094, предоставленные Lero – Ирландскому исследовательскому центру программного обеспечения (www.lero.ie).

1. Introduction

Historical data concerning individual life events, combined with wider socio-economic records provide excellent sources for analysis and reflection. Accordingly, the digitalisation of corpora of historical data concerning various aspects of the life and activities of individuals and communities is an essential precondition for the ease of analysis, for example using modern data analytics and AI techniques. The Digital Humanities Manifesto 2.0 (DH) [1] presents DH as a discipline which studies the intersection of the disciplines of computing and humanities. DH currently combines methods, tools, and technologies provided by the computing sciences (such as data visualization, information retrieval, text mining etc.) with the perspectives and methodologies stemming from the humanities disciplines (such as history, trend analysis etc).

One of the increasingly popular means of using digitally available data foots on the concept of a Digital Twin (DT) [2]. A Digital Twin is a virtual and abstract model of a physical entity (an engine, a patient, a student, a plant or a city) that serves as the enabler means for simulation, analysis, prediction, and real-time analysis of the system it represents. It has gained enormous relevance and popularity in recent years as it provides a handy virtual model of a physical process or service. In the Industry 4.0 context, it often leverages technologies such as the Internet of Things (IoT), Artificial Intelligence (AI), Cyber-Physical Systems (CPS) and Big Data for digitization. By definition, digital twins refer to a «live» model that continuously updates and changes as its physical counterpart changes [3]. In the Humanities, the DT concept unfold a massive potential to transform the landscape of how DH methods can assist in the representation, analysis and understanding of our past, which in turn can provide useful learnings for the present and future. It promises a tremendous innovation potential, and most of the current research on digital twins is focusing on specific implementations for concrete use cases and the generalization towards reusable abstract models [4]. Developing a mirror of a traditional Digital Humanities record system through the digital twin lens is time-consuming, complicated and requires deep interdisciplinary knowledge in the humanities domain and model creation and software development. Too often, this induces a knowledge gap, giving rise to fundamental research questions on how to connect the two disciplines in such a way that a «lingua franca» can bridge the concepts and the means of expression and analysis of both disciplines.

We use a specific kind of Model-Driven Design, called XMDD for (eXtreme Model-Driven Design) [5] to bridge this gap. Model-Driven Development (MDD) specifically focuses on supporting the collaborative (software) development process by using abstract representations of data and processes. Using these models, we combine computing knowledge with the formal descriptions of the historian’s knowledge, and this way succeed in reducing complexity and improve productivity, as described by [6].

To reduce the discipline-specific knowledge gap between humanities and technology, the project «DBDIrl[1] - Death and Burial Data: Ireland 1864-1922» [7] adopts a data-driven public-history and digital-humanities research methodology which uses advanced MDD for application development. DBDIrl is an interdisciplinary project that combines historians' understanding of Big old data with computer analysts' tools and methodologies. Its objective is to build an extensible and reusable Big Data interoperability and analysis framework that supports flexible Big Data integration between different historical data sources and provides a web-based platform for the analysis of its underlying corpora. The corpora stem from various sources of national records, like the civil registration records of the General Register Office, the individual level census returns of 1901 and 1911, and various coroner's court records within the period 1864 to 1922, i.e., from the introduction of civil registration records in 1864 to 1922, when the Irish Free State was established. This Digital Humanities platform needs to be robust and easily evolvable, able to integrate different data and interpreted terms, able to manage and analyze various data representations and enrichments, all in a transparent and FAIR (i.e., Findability, Accessibility, Interoperability, and Reuse of digital assets) [8] data context.

This paper focuses on developing an efficient and flexible data access mechanism to make the heterogeneous sources of historical data available to a wider range of researchers through adequate user interfaces.

DBDIrl applies the eXtreme Model-Driven Approach for complete design, development and execution of a Big Data interoperability framework. The first component of that framework is a Web application that supports efficient and correct data entry. We refer to it as the *Historian DIME app* or *Historian app* in short, and it is completely developed following a model-driven approach.

Key contributions of this work are:

- A model-driven Web application for input and storage of Irish Civil Registration data, specifically death registration data, from 1864 to1922, introducing a database for subsequent digital data analysis.
- Producing a systematic and clean data source for (relevant subsets of) the death records. Massive information regarding the death records was previously collected as images of the original registers stored as TIFF files, plus an excel index summary. The page-by-page images of the

[1] https://www.dbdirl.com/

handwritten records were digital, but it was impossible to analyze them. The database of systematic and clean data can now be processed for further research, concerning the discovery of information, its evolution, trends over time, and finding insightful patterns about individuals and families.

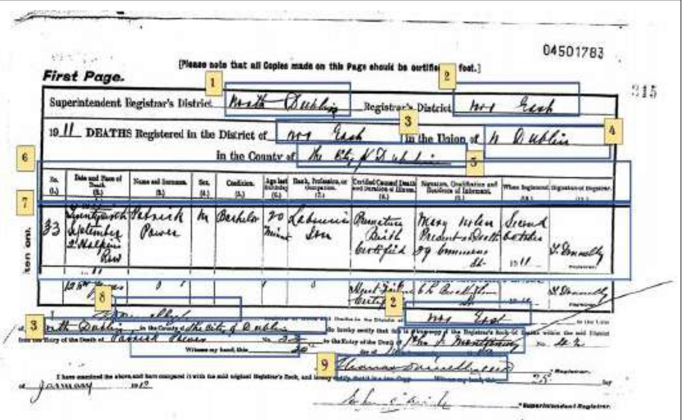
- Illustrating the impact of the MDD approach on the adaptation and evolution of the Historian App, from its first version to the current one, including the re-usability of components and the refinement of its organization, to support increasing levels of error prevention and embedded error checking. It is essential if we want to gradually build a platform, where such applications and data analytics applications can be quickly and correctly assembled from a service-oriented Domain Specific Language that covers the functionalities and the data occurring in a history research context.
- Showcasing the use of DIME [9], a specific low-code application design framework, where stakeholders can develop their specific application without any coding knowledge.

To our knowledge, this is the first attempt to work with Historians as customers using a model-driven approach.

The paper is organized as follows: Section 2 presents the project background and motivation. Section 3 describes related work in the fields of MDD and Big data analysis. Section 4 discusses the co-development methodology and its life-cycle along with an explanation of the abstract architecture and workflow of the proposed XMDD based application. Section 5 illustrates the model types of the XMDD technology and the concrete design of the Historian App. Section 6 describes some major challenges with corresponding proposed solutions. Finally, Section 7 concludes the paper and highlights some future work.

2. DBDIrl Project Background

The General Register Office (GRO) is responsible for recording Irish civil information of birth, marriage and death. In 2016 it placed historical data online for free on irishgenealogy.ie. To initiate a search at the site, some basic personal information is required, but it has limited functionalities. This site holds civil data sets regarding individuals, but for the fundamental objectives of DBDIrl, a centralized data storage containing complete and correct data is needed for future research and exploration.



04501783

[Please note that all Copies made on this Page should be certified at foot.]

First Page.

Superintendent Registrar's District North Dublin Registrar's District No 1 Each

1911 DEATHS Registered in the District of No 1 Each in the Union of N Dublin in the County of the City of Dublin

No. (1.)	Date and Place of Death. (2.)	Name and Surname. (3.)	Sex. (4.)	Condition. (5.)	Age last Birthday. (6.)	Rank, Profession, or Occupation. (7.)	Certified Cause of Death and Duration of Illness. (8.)	Signature, Qualification and Residence of Informant. (9.)	When Registered. (10.)	Signature of Registrar. (11.)
33	Twentyfourth September 2 Halpin Row	Patrick Power	M	Bachelor	20 Mins	Labourer's Son	Premature Birth Certified	Mary Nolan Present at Death 29 [illegible]	Second October 1911	T. Donnelly Registrar

1 -> Superintendent Registrar's District Union, 2 -> Registrar's District, 3 -> Registrar's City, 4 -> Registrar's Union, 5 -> Registrar's County, 6 -> Required Properties for a Legitimate Death Registration, 7 -> Properties for an Individual Death Registration, 8 -> Full name of Registrar, 9 -> Registrar's Signature.

Fig. 1: Death record of Irish civil registration: the GRO original register page (TIFF file available at irishgenealogy.ie) with properties highlighted

As the primary data, DBDIrl uses the Death Registration Data (DRD) from 1864 to 1922 directly shared by from the GRO. We received approximately 4.3 million individual Civil Register records of death registration in two different formats. Over 1 TB were images produced through high-resolution scans of the original register pages and provided as .TIFF files. We also received .csv files with group id, name, age, superintendent's district and .TIFF file path of all individual death records. Fig. 1 shows a page from the death register. Each scan captures a full register page, including up to 10 individual records, each recording an individual death.

In the absence of complete metadata and a fully digital version of the image's contents, the .TIFF file is de facto just a picture, i.e., an unstructured analog image of the page, and useless for the purpose of automatic analysis of the contained information. A human eye sees easily that every record has 11 index properties (identified and numbered in fig. [1] describing the death event and its circumstances. This set of complex properties collectively represents the individual's death event along with its essential information. Their complete digitization, meaning the transformation of the TIFF images into a curated repository of clean and faithful data that is fully automatically searchable and analyzable, is the aim of the current phase of DBDIrl.

For essential quality guarantee, the historical digital data collection must maintain with certainty the overall integrity of the original historical data. Additionally, the technology needs to enable domain experts, like historians and archivists, to handle the maintenance of the data collection and the evolution of the applications. These experts are mostly not programmers, and most certainly not experienced in all of web development, databases, software architectures, UI design and development, privacy and security, testing and deployment. So we adopted a programming-less low code approach based on an Integrated Modelling Environment (IME) that subsumes most of these characteristics in the development platform of choice.

The goal consists of three main tasks:

- transform the TIFF files into a digital curated repository;
- achieve this transformation in a low-code environment that is easily maintainable and evolvable, effectively building a new generation data entry, storage and management platform for digital humanities;
- make historical data from heterogeneous sources available to a wider range of researchers through adequate user interfaces and easy-to-use analysis tools.

Currently we are working on tasks 1 and 2.

2.1 Automated Digitization Attempt

DBDIrl started with attempts to transcribe the .TIFF files to an operable, structured data format. A widespread approach would use OCR or Natural Language Processing (NLP) tools to extract the text from each .TIFF file. While the state of the art tools work quite well for printed texts, they severely failed in our case. In fact well-known language processing tools could not produce any useful results. There are many reasons for this failure: (1) death records are handwritten texts, which is a difficult problem; (2) they were written by different registrars and their superintendents, with considerable variation of handwriting pattern; (3) tools have difficulty handling the data variety, (4) for some writers the corpus of records is very small and insufficient for a good training set; (5) very few existing tools extract the text as individual properties, thus even in case of success a significant manual post-processing would be needed; (6) accurate text extraction needs a well-trained model with a huge and precisely labelled data sets for training, which is not available here; (7) there should be reliable methods to combine all the individual property texts into correct death record entries, which is difficult when most properties are not correctly recognized; and (8) there is a scalability issue when uploading millions of records into a server.

2.2 Supported Digitization

As a consequence, we abandoned an automated recognition approach for the time being, in favour of a manual, but highly assisted and supported data enrichment through a web based application. In this sense, the first and second task now align much more closely: We have now an XMDD based web application for the historians' data entry, where application developers and historians work side by side in application design and development within a model-driven, low-code environment. This application development approach helps the historians to further develop and maintain their own application at the model level, without the need of any programming knowledge.

3. Related Work

Since the emergence of UML and its predecessors, several MDD approaches have been proposed in the literature to address the generation of code from models representing various aspects of the system [10], including for telecommunications [11], web and client applications [12-15]. MDD techniques are mainly used for decreasing the effort needed for application development and maintenance and increase the portability to new platforms. The eXtreme Model-Driven Development (XMDD) [5] approach is a low-code approach that combines several software designs and programming paradigms such as agility, model-driven development, service orientation, domain-specific languages, data management, data flow and control flow design, Formal models and methods, generative programming, eXtreme programming, aspect orientation and full code generation [5], [16]. According to [17],

> «Models allow sharing a common vision and knowledge among technical and non-technical stakeholders, facilitating and promoting the communication among them.»

In terms of specific MDD approaches and applications, [18] proposed automated extraction, analysis, and visualization of data and metrics on model-driven artifacts. In cyber-physical systems, [19], and [20] demonstrate the use of MDD in robotics. [21] proposed a DSL for service customization for telecommunications sytems. [22] proposed a Domain-Specific Modelling Language for smart home applications with two transformation templates that generate code from instances of SmartHomeML for SmartThings and Alexa. They designed the transformation using an MDD approach in a platform-specific model-to-code implementation artefact.

In e-learning, [23] propose a course management system that stores a course model as machine-readable components that generates a final course in different platform-specific target models.

In web applications, modern Single-Page Applications (SPA) use MDE to connect between client and server of a web application, and [24] present a model-driven approach for the consumption of RESTful Web services in SPA.

Ref [25] defines a Machine Learning based MDE approach that analyzes Big Data for probabilistic modelling by defining a domain-specific modelling language. In Big Data, [26] introduced SkyViz, a model-driven approach for automating the translation of user objectives to visualize the Big Data Analytics' results into a set of most suitable and concrete visualizations. [27] proposed a design method to specify, deploy, and monitor Big Data Analytics solutions using MDD.

While all this shows that MDD is applied in a variety of relevant areas for the DBDIrl project, as per our study there are no MDD based context-aware web applications that work with real-world big data archiving, management and analysis.

4. The Historian App as a MDD Application

The Historian App we developed and evolved in a number of iterations is the DBDIrl solution to data entry, storage and management for the historical civil registration (i.e., death) data of Ireland from 1864 to 1922. We adopt the eXtreme Model-Driven Development (XMDD) [28], which provides a fast turnaround of easily modifiable prototypes understandable to the non-IT experts. In this way, a more collaborative approach between domain experts (here the historians as central stakeholders) and developers establishes itself along the entire project life cycle.

The agile model-based approach helps repeat the feedback and co-design cycles with the historians in a continuous refinement process. In addition, using models also helped the developer team when reflecting, presenting and explaining the work progress to the historians and the historians when understanding and monitoring the development.

We chose the DIME Integrated Modelling Environment [9, 29], based on Domain-Specific Libraries (DSLs), as the XMDD framework for our project. DIME provides reusable features, and functionalities [20, 30] where developers can develop web applications within a low-code environment without having any programming knowledge. DIME supports model types for processes, services, data, and the UI that are integrated and kept consistent to a reasonable extent by the platform. Many domain-specific libraries (DSLs) are already available, for example, for the GUI design of the web applications. New services as well as entire new DSLs can be introduced in an easy way. These characteristics help the IT specialists and the domain specialists to better understand and monitor the development throughout the project life cycle on the basis of the domain knowledge.

4.1 he IME-based co-development lifecycle

The application development life cycle of the Historian App is illustrated in fig. 2.

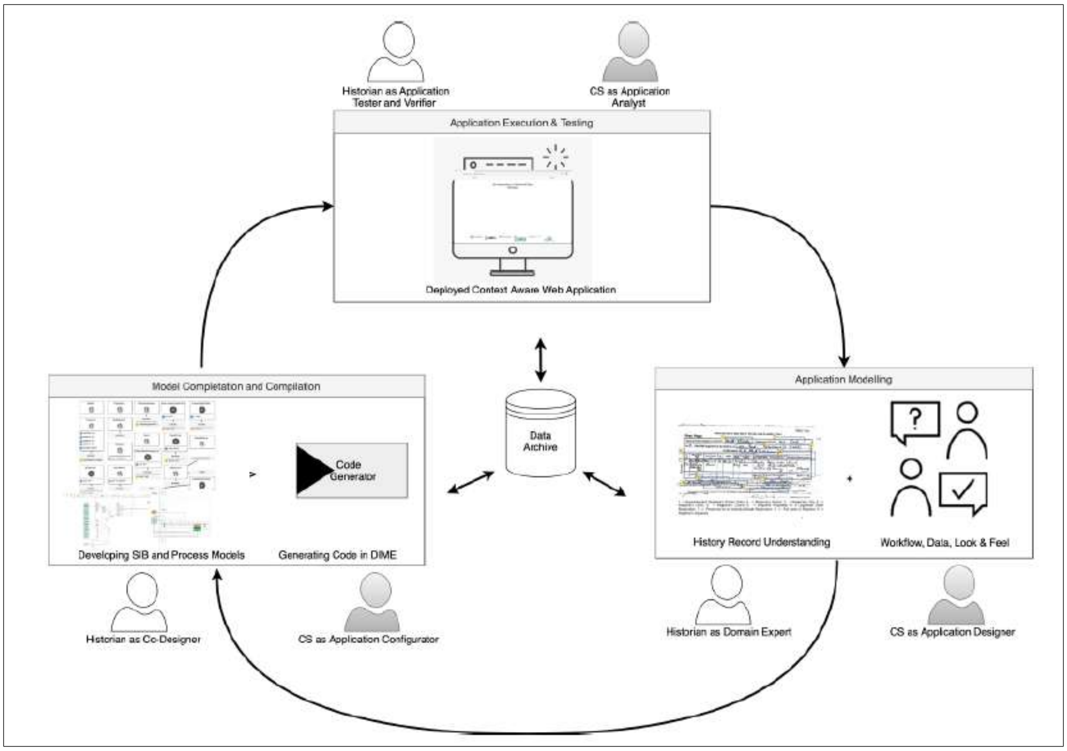

Historian as Application Tester and Verifier
CS as Application Analyst
Application Execution & Testing
Deployed Context Aware Web Application
Model Completaton and Compilaton
Code Generator
Developing SIB and Process Models
Generating Code in DIME
Data Archive
Application Modelling
History Record Understanding
Workflow, Data, Look & Feel
Historian as Co-Designer
CS as Application Configurator
Historian as Domain Expert
CS as Application Designer

Fig. 2: Collaborative development lifecycle in an IME: agile iterative phases, roles of Historians and Computer Scientists (CS).

The project development life cycle involves in each phase both the computer scientists and historians, in different roles. The historians become successively more skilled in dealing with the models and application design. At project completion the historians may be able to modify and evolve, or even design and implement, their own web applications on the basis of the existing DSLs, without any coding knowledge.

4.1.1 Phase 1 – Application Modelling

As illustrated in fig. 2, the project life cycle starts by collecting and collaboratively analyzing the historians' requirements. They are materialized as abstract workflow models with the corresponding (unique and coherent) data model. Data and processes go hand in hand in DIME, so they are typically co-developed and co-evolved in an XMDD approach. In this phase, the historians used their expert knowledge about handling historical data and the correctness of the data and the records. The historians collected the data and analysed their characteristics. While historians were finalizing the data properties that they need for their further analysis, the computer scientists started designing the data models including entities, attributes and relations. Then the historians specified how they want the data to be stored, explaining what is already there and usable, what else needs to be added, and how. Next, the CS team designed the corresponding data and workflow models, expressing the high-level application logic and the elementary operations required for application development. Gathering this expert knowledge in terms of workflows and properties or conditions (on the individual data item, the record, the workflows) corresponds to gathering the historian data entry application's static and behavioral requirements. At this phase, the historians also validated the models and helped in finalizing them.

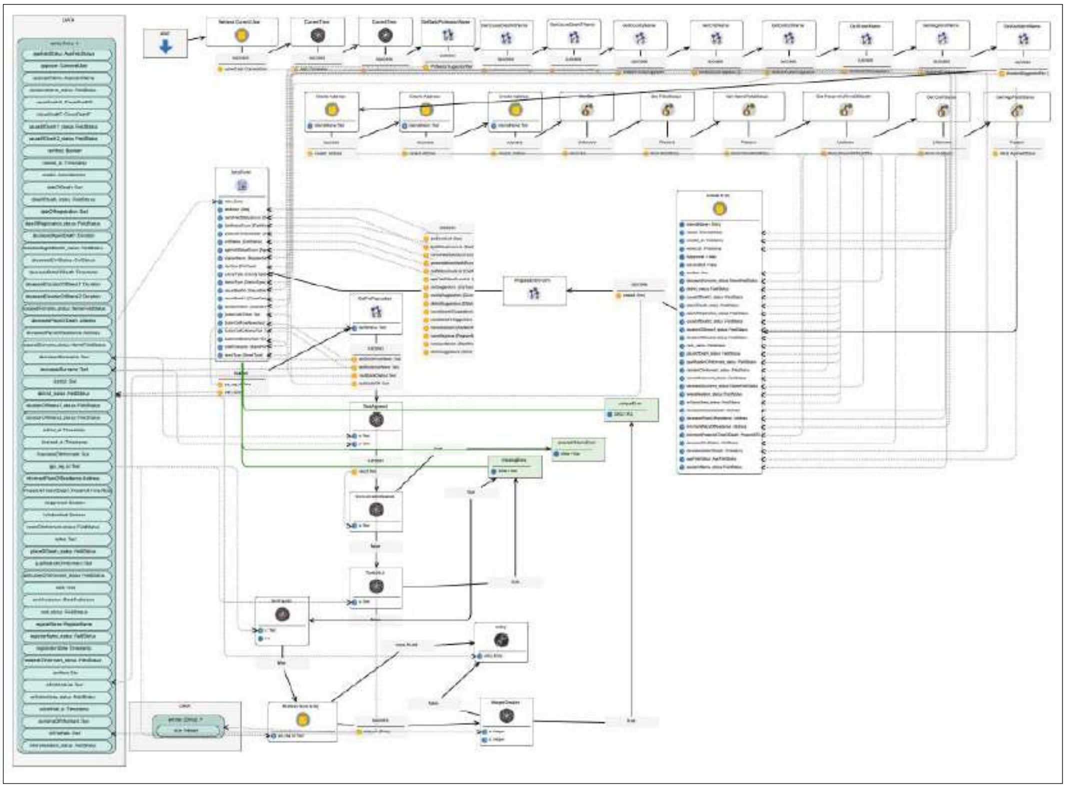


Fig. 3. Create Entry process of Historian DIME app including all other processes that successfully stores a death record with all its attributes and event listeners.

4.1.2 Phase 2 – Model Completion and Compilation

The second phase includes all the XMDD: model refinement, followed by DSL extension and implementation of new functionalities. Here the historians participated as stakeholders for detailed questions, the CS team as fine granular designers and developers. The CS team extended the DSLs where functionalities were missing, implemented them in a reusable, service-oriented way and modelled the Web application GUI. A growing hierarchy of nested workflows structure the application logic in behavioural features. For the business logic they acted as application configurators on the basis of these models and services. We reuse existing DIME process models such as *RetrieveEnumLiteralSIB* that gets a field status (illustrated in fig. 3) but also designed new processes for further required operations such as *GetPrePopulated* to load in the application a predefined set of data from a file. This phase also includes the models-to-code generation phase from the collection of validated models, and the deployment on a standard web stack. It produces a deployed, running application, that is further examined, updated, recompiled and redeployed.

4.1.3 Phase 3 – Application Execution and Testing

In this phase, the Historians and other end users (like history students and volunteers in the transcribathons for the data entry) test and use the application, as shown in fig. 4. Small adjustments and optimizations may be carried out as a consequence of live testing. This is the validation and use phase of the current version of the application. It includes live debugging, error handling and fixing, as well as the definition of new features and changes for the next development phase.

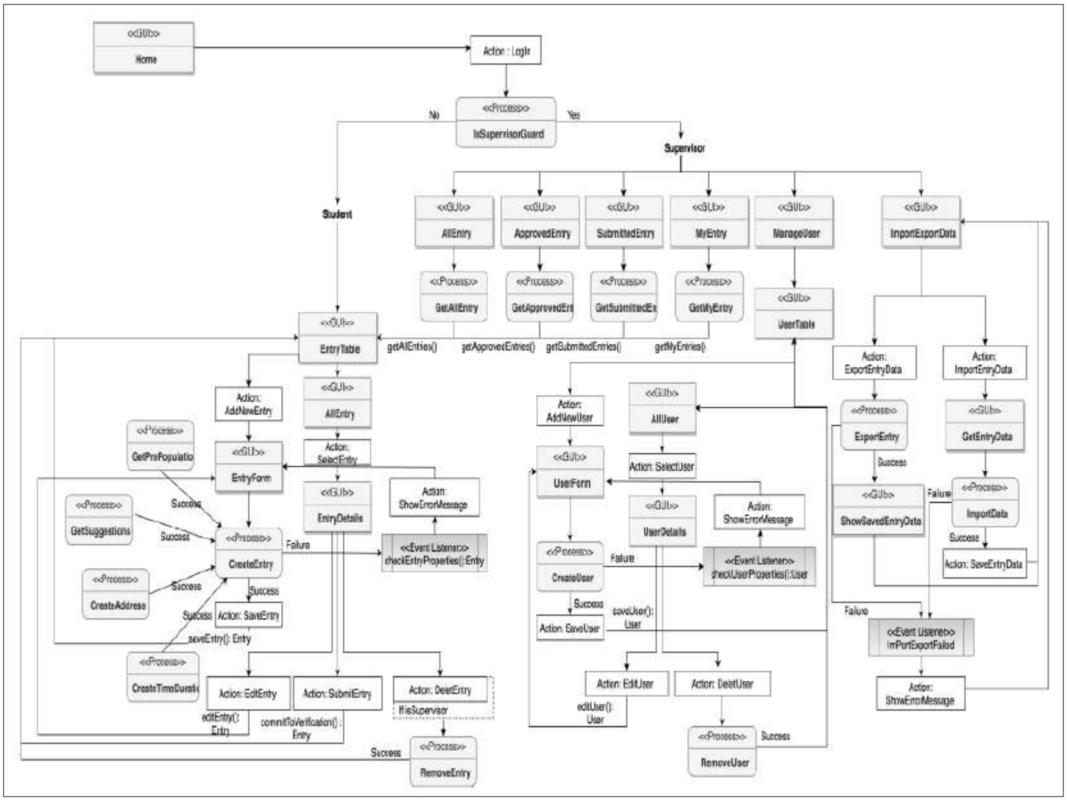


Fig. 4: Model-driven abstract architecture of DBDIrl's data entry web application – Feature level

The whole cycle follows an agile software development procedure.

4.2 Modeling the Historian Web Application: The Full Workflow

We describe now the application workflow along with the explanation of the main processes and GUI models developed for the application. The Model-Driven Development (MDD) of DBDIrl starts with listing and developing process models, the data model and identifying user roles. Fig. 4 illustrates the feature-level abstract architecture of DBDIrl's data entry web application in terms of *Processes*, *GUIs*, *Actions* and *Event Handlers* (as indicated by the respective stereotypes `<<Process>>`, `<<Action>>` etc.) along with the connections among them.

The application homepage[2] is a GUI model where users can login. Its action Login calls the process IsSupervisorGuard, that checks the login credentials and establishes the user role: Supervisor or Student.

In the **Student** role, a successful login directly links to the *EntryTable* page, a GUI model where the student sees all the entries recorded by him/her. Action *AddNewEntry* leads the user to the *EntryForm* page, a GUI model which calls the *CreateEntry* process shown in Fig. 3.

On the *EntryForm* page, users enter the data of all the records from the .TIFF file of the death record register page, by filling up field by field the record's properties in the corresponding fields on the web page.

CreateEntry is a big process: it receives the data entered by the user and to do so in an error-free way it calls other processes that provide support functions. For example, it uses the *GetPrePopulation* process for reading pre-populated data from a file, *GetSuggestions* to provide pre-populated options in the Web form as drop-down menu for certain data attributes, *CreateAddress* to create a new (complex) address object with the individual attributes city, county, district and street.

Similarly, *CreateTimeDuration* creates a duration object from various time properties. The *GetPrePopulation* process receives the group id of a death record provided by the user, and it reads name, age, superintendent's district and .TIFF file path from the .csv files we received from the GRO. It also auto-fills the corresponding fields of the *EntryForm*.

The *GetSuggestions* process reads large lists of pre-populated, validated values for a number of properties. It displays those options as a drop-down menu in the form, to ease the input of attributes like cause of death, registrar name, assistant name, rank profession and street names of Ireland from 1864 to 1922. These data collections are pre-validated, as the Historians collected them from 18's Ireland records.

CreateEntry performs all the individual operations needed to successfully save an entry with all its values entered by the user (either by hand or by selecting pre-populated fields), property by property. Event listeners on *CreateEntry* process help check data validity and show alert messages in case of a wrong entry (incorrect value or format).

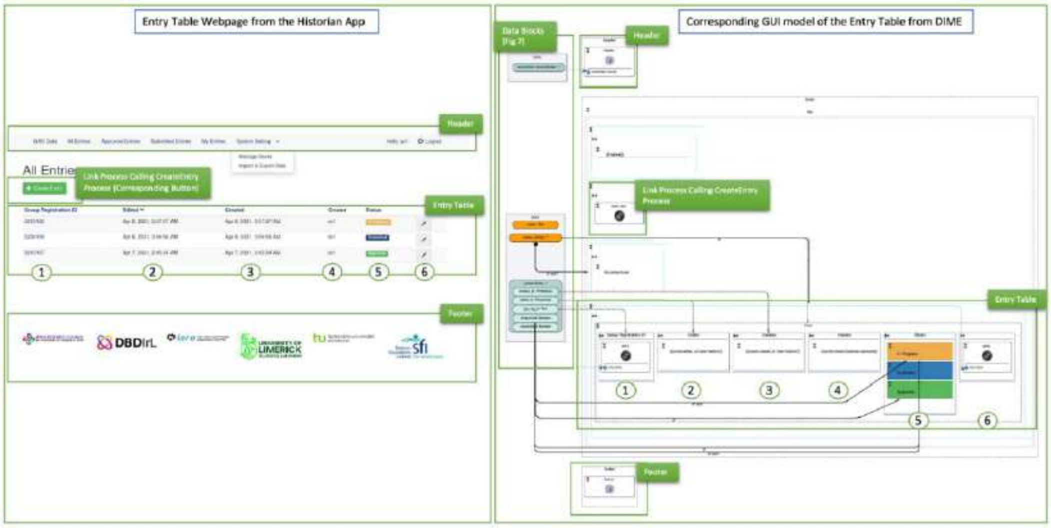

Entry Table Webpage from the Historian App
Header
All Entries
Link Process Calling CreateEntry Process (Corresponding Button)
Entry Table
1
2
3
4
5
6
Footer
DBDIrL
UNIVERSITY OF LIMERICK
sfi
Corresponding GUI model of the Entry Table from DIME
Data Blocks (Fig 7)
Header
Link Process Calling CreateEntry Process
Entry Table
1
2
3
4
5
6
Footer

Fig. 5. The Entry Table of the Historian App: Web page (Left) and its corresponding GUI model (Right)

[2] The Historian App is available at https://civilreg.dbdirl.com/home, it is accessible to predefined, verified users.

Finally, the action *SaveEntry* from the *EntryForm* successfully saves an entry and sends the user back to the *EntryTable*, to process the next record.

Fig. 5 shows the entry table as it is displayed on the Historian App web page, with the corresponding GUI model in DIME. We see here that the structure and look and feel are very recognizable. The data flow is explicitly modelled, and we recognize buttons (like the *CreateEntry button*) and other elements like fields filled from the database and status indicators that are color coded (orange, green and blue).

From the *EntryTable*, selecting an entry leads the users to the *EntryDetails* GUI: there they can (re)view the entered entry details and choose to edit the entry (this brings them back to the *EntryForm*, filled with all the previously entered data), or submit the entry for review. The *Entry Delete* option is only available to the Supervisor role, who can delete an entry from the database.

In the **Supervisor** role, a successful login directly links to the *EntryTable* GUI. The supervisor is displayed the user's entries and also has other options, like seeing all submitted and approved entries individually. The Supervisors have a validation and approval function: they can see the details of all the entries stored by Student users, and have actions to perform edits, approve, as well as remove each entry.

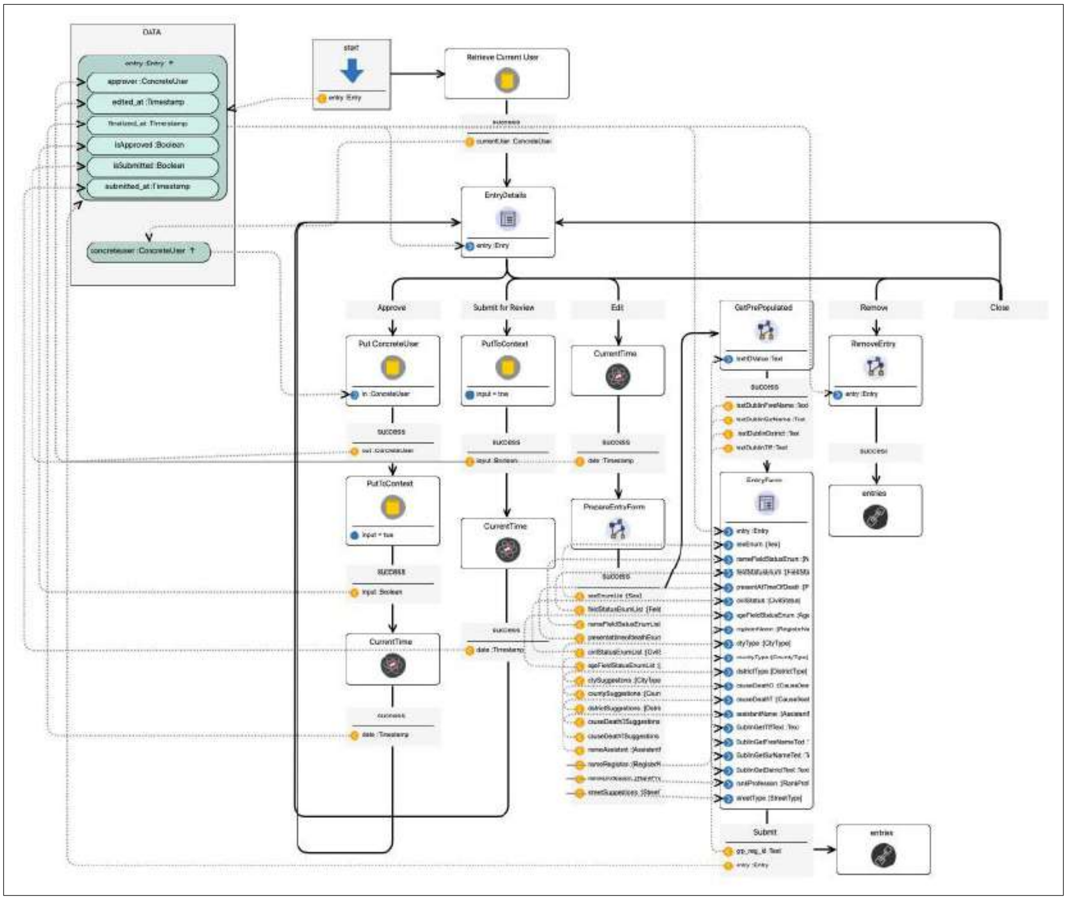


Fig. 6: Entry table management process of Historian app, with flows for the Approve, Submit for Review, Edit, Remove, Close entry operations

Fig. 6 shows the *ShowEntry* process, with flows for the *Approve*, *Submit for Review*, *Edit*, *Remove*, *Close* entry operations. Selecting *ManageUser* leads the supervisor to a *UserTable* page that includes the *AllUser* GUI (displaying all the users) and the *AddNewUser* action. *AddNewUser* calls the *UserForm* GUI, containing all the fields required to creating an user.

Similar to *CreateEntry*, the *CreateUser* process performs all the operations necessary to create a user. It also includes an event listener that alerts the supervisor in case wrong or ill-formatted information is provided. After saving a user, the action *EditUser* calls the *UserForm* again with the previously provided data to edit, while *DeleteUser* calls the *RemoveUser* process to delete the user. Supervisor can also import and export data to and from the application. An event listener is used to check issues regarding import/export operations, together with the processes *ExportEntry* and *ImportData* that export and import data, respectively.

Altogether, Fig. 4 presents an overview of how the Historian App is organized, and shows the interplay among all the Processes, GUIs, (GUI)Actions and Event Handlers.

5. Model Types and Concrete Models

We describe now the main model types, model elements and models of the Historian App.

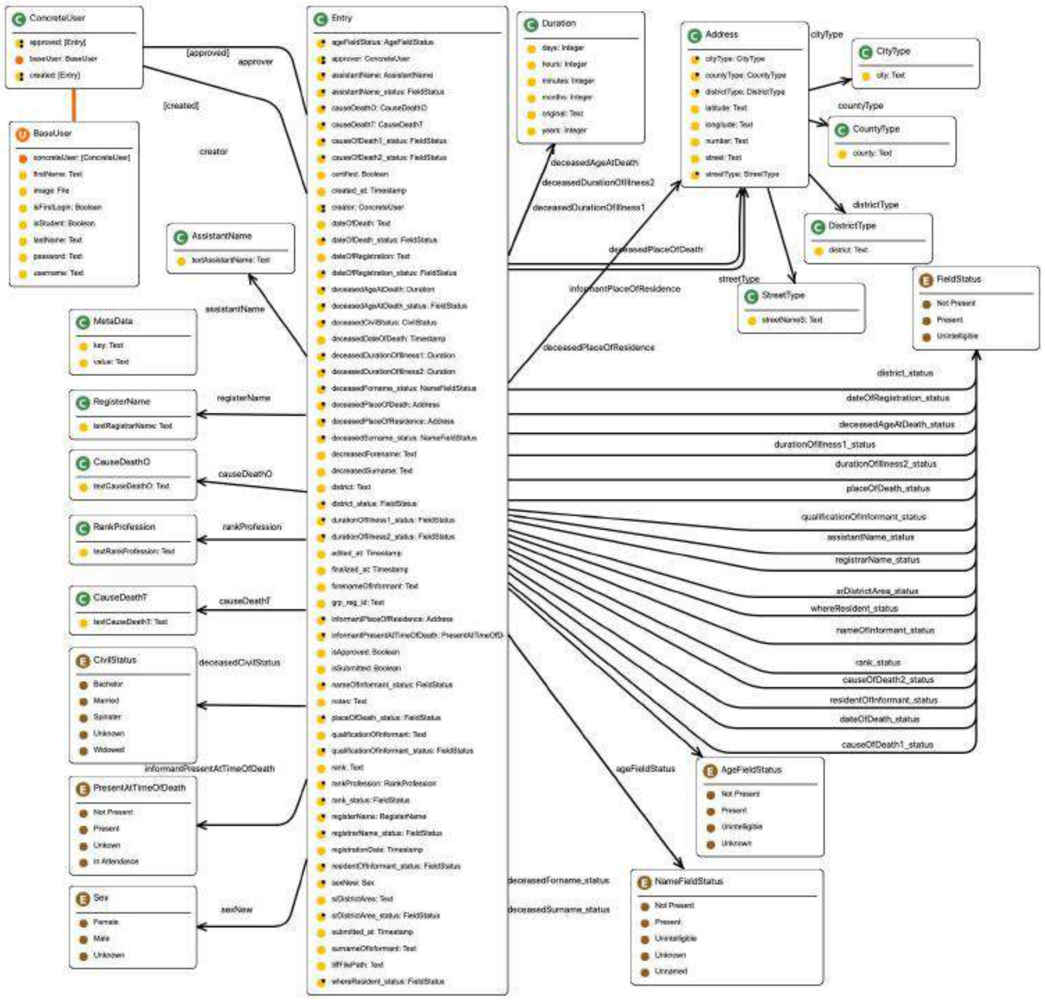


Fig. 7: Data model of Irish civil registration in DIME

5.1 Data Model

Fig. 7 shows the data model of the application, representing both the concrete and abstract data. It contains both unidirectional and bidirectional relations such as association and inheritance. In our finer granular representation of a record, every Entry has 27 individual properties. Some properties like Sex, Address, Age include sub properties, which are at the elementary granularity needed for data analysis. The decision of moving to these 27 properties from the original 11 properties of fig. 1 is an example of the design choices for the Historian App stemming from the co-design practice. As shown in fig. 7, every concrete user can have base user who as act as creator (only students) or approver of an entry. The Entry itself is a concrete type data at DIME (green data objects at fig. 7). Most of the attributes of entry are stored as text or number i.e. primitive attributes of DIME (small yellow components at fig. 7). Some are Enum type attributes e.g. Civil Status with some optional values (brown data objects at fig. 7). Some properties may not be present in the original record: the corresponding cases are captured by the FieldStatus. Some attributes e.g. Registrar Name are created as concrete type object so that they can receive list of data options and presented as drop-down menu to the application and user can choose the correct information from provides options. Duration and Address are also concrete type objects with required values.

5.2 Graphical User Interface Models

In DIME, the GUI model type represents the structure (layout and contents) of the Historian app's individual web pages. A collection of GUI models defined, therefore the abstract and concrete «look» of the presentation layer of a DIME application. We see in fig. 4 that the created GUI models connect the GUI and Process Models. Every GUI models of Historian DIME app is created using components from DIME palette. The GUI models call process models to execute an operation. These GUI models are also reusable, for example we use EntryForm at fig. 4 for both create and update operation of each entry.

5.3 Native DSLs

In DIME, the actions and services are collected in domain specific palettes that are basically a service or component oriented DSL. The DSL elements correspond to (calls to) individual functionalities that are either directly implemented or provided by an external service provider, like e.g. the database. The individual functionalities are modelled as special native types called SIBs, for service-independent building blocks, where service-independent means that they are widely reusable across applications. These Native SIBs enable interoperability on a structural level. Within the Historian app, besides the pre-existing DIME SIBs we create Native SIBs for different operations such as data pre-population, CSV file import and export and to get field suggestions etc.

5.4 Data-flow

In DIME data flow is explicitly modelled within the process models. The input/output ports of SIBs can either be connected directly with each other or used to read and write from/to variables placed in a dedicated container representing the data context [9]. Fig. 4 shows the data flow connections of the proposed application using arrows. All other figures of the Historian DIME app shows data flow connection.

5.4 Process models

Process models express the business logic in a fashion roughly similar to Activity Diagrams, but with a clean formal semantics. There are several process types: basic, interactable and interaction processes. Each process type follows certain rules regarding which kind of SIBs they contain and the kind of tasks they express. The graphical syntax and general handling are the same for all the types of SIBs and processes. Fig. 4 shows the process models that are created for the Historian application. The collection of process models together with the connected GUI, actions and data flow expresses the behaviour of the application in terms of its operation. Fig. 3 shows the actual process model CreateEntry of Historian app (also presented in fig. 4) that performs all necessary operations and successfully stores an entry to the database.

Process models can also be of different type such as

a) *Basic Processes*: Basic Processes consist of native SIBs and built-in SIBs, and express the smallest processes of the application's business logic. In the Historian app, basic processes models are the CRUD (i.e., create, read, update, and delete) operations and data operations. In fig. 4 CreateUser and RemoveEntry are two examples of basic processes.

b) *Interactable Processes*: Interactable Processes work as interfaces between the front-end layer and the backend of the application. They are similar to Basic processes, but are restricted to non-native type. The StartUp process is the only interactable process in the Historian app. This process includes operation of successful login of user with different role.

c) *Interaction Processes*: Interaction Processes are used to define the immediate interaction between user and application, accordingly they can be seen as a sitemap [9]. Where interactable SIBs communicate with the backend, interaction SIBs establish a new hierarchy level with the frontend. As the Historian app is essentially a sophisticated daTa entry app, most of its processes are developed as interaction process, like GetPrePopulation, GetSuggestions, CreateAddress etc.

d) *Security Processes*: Security Processes realize the (role based) access control with a predefined interface. The IsSupervisorGuard and ExportFileGuard processes are two examples of security processes in the Historian application. In IsSupervisorGuard the start node must include the currently signed-in user (i.e., the Supervisor) as an input, and all following nodes are restricted to be labelled with «granted».

6. Challenges and Solutions

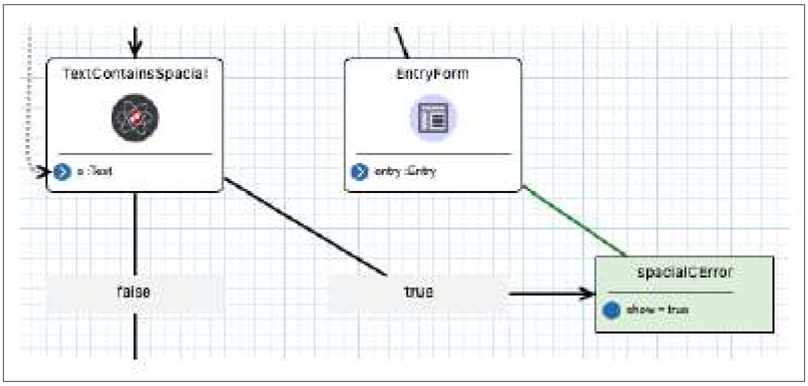
6.1 Defining the major context parameters

In a context-dependent application, a DSL should enable modelling the different context situations that may occur during user interface usage. This DSL will eventually help developers to separately specify context-specific services to monitor various parameters and react accordingly. For example, the application's abstract GUI rules cover various adaptation dimensions: layout, navigation, reusability. Accordingly, modelling, adaptation, transformation and execution of processes and GUIs take into consideration the context management and the corresponding adaptation. In particular, the processes and functionalities are associated with responsive GUIs.

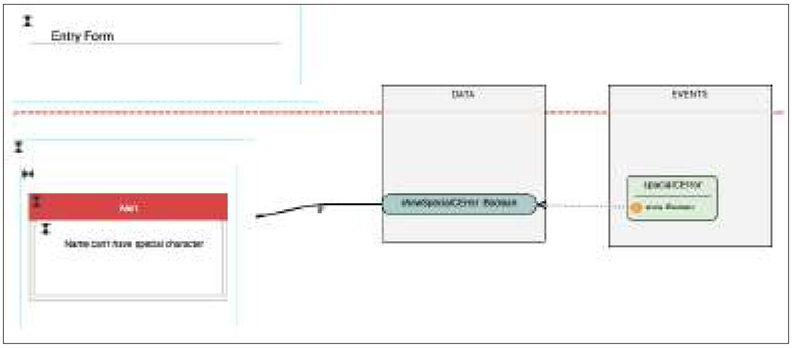
6.2 User interface adaptation at runtime

To achieve a responsive web application, the integrated execution environment must be equipped to generate adaptation services in dependence of the context. For this, the generated adaptation processes need to be coupled with generated code that enables an automatic dynamic reaction of the runtime UI to the context-of-use. In the Historian App, the data entered by the user is the predominant part of the dynamic context to which the app reacts. The reaction manifests itself in a validation of the entry or an error message if problems are detected. To address various run-time errors, we introduced 'Alert' models with the event handler. To this aim, we introduced native SIBs for several condition checks, detecting e.g., whether an unintentional special character is entered, or a date entered in an incorrect format, a required field is left unattended, the ID not unique etc. Process events are connected with those detections, and respective event-listeners are introduced at the corresponding GUI models, enabling this was a run time error handling. Fig. 8(a) shows an event and corresponding event listener model connected with respective alert that warns the user that

'Name can't have special character' (fig. 8(b)). We also proposed a rule-based classifier [31] for overall data monitoring and error detection. The integration of the classifier with the DIME application is currently ongoing.



(a) Detection: Event Listener in the CreateEntry Process



Name can't have special character

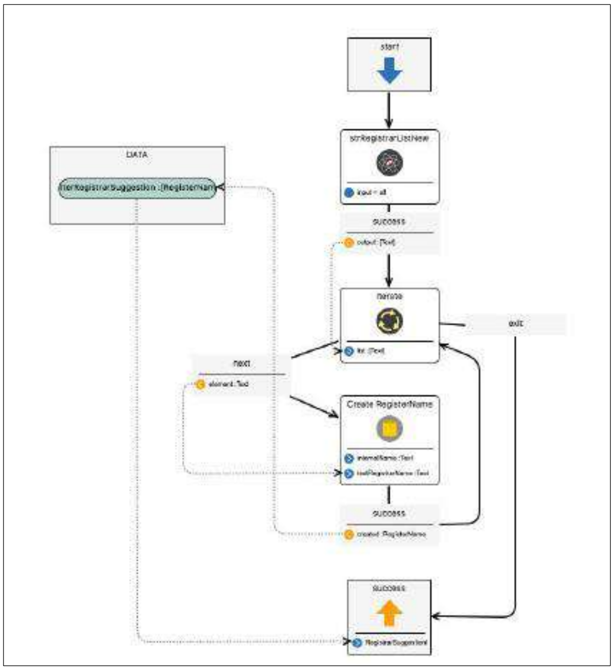
(b) Handling: Event with corresponding alert in the EntryForm GUI

Fig. 8. Handling a run-time error in DIME: unexpected special character in the name field

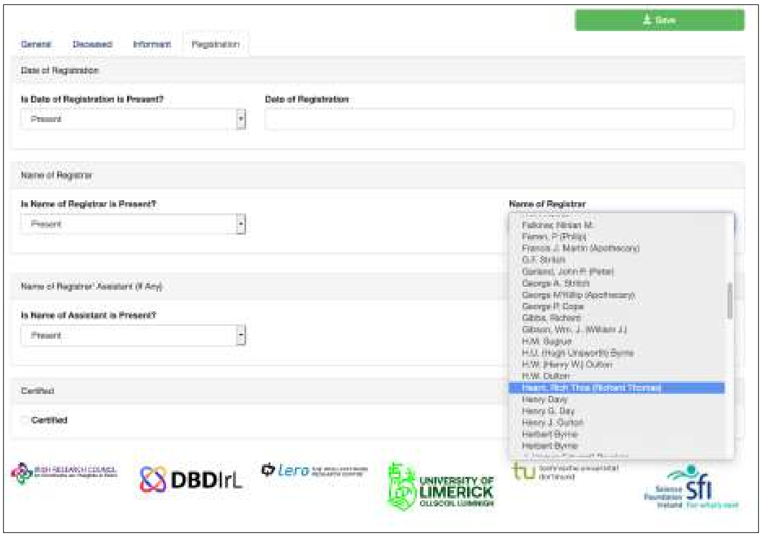
6.2 Data Entry with minimum error

In 2020 we conducted a pilot Transcribathon using the Historian application version 1, which was not a responsive application with built-in data checks. Examining the resulting data entry, it emerged that most of the wrong entries occurred at the fields Cause of death, Address, Age/Duration and at the Registrar names. The date format also posed problems.

As a solution, we worked together to create drop-down lists of cause of death, registrar name and street name which can be used to provide a predefined list of suggestions, thereby eliminating the free text entry, and reducing error rate. For registrar name prediction, it is possible to create a registrar names list for the period and location of interest. Using the method employed by 18 we ordered the geographical data like street, city and county names of Ireland, which occur in various address fields. All these lists are added to the application, by augmenting it with native SIBs and adequate GUI and process elements. Fig. 9 shows an example. The Web page screenshot of fig. 9b shows the Registrar data entry page, and in particular the pre-populated data field for the registrar name of a death record. This drop-down menu or combo-box lists all the registrar names collected from the early 1900s Dublin Street Directory. Once the historians found and verified the names of the registrars who registered the death records relevant to this specific time and place, the IT specialists created the GetSuggestion process shown in fig. 9(a). This process shows the list as a drop-down menu in the right location of the application page. Thus, instead of inputting a free string that needs to be validated, in version 2 for this field users can select the correct option from this menu, avoiding errors instead of repairing them.



(a) GetSuggestions process


Save

General | Deceased | Informant | Registration

Date of Registration

Is Date of Registration is Present?: Present

Date of Registration:

Name of Registrar

Is Name of Registrar is Present?: Present

Name of Registrar:
Falkiner, Ninian M.
Farren, P (Philip)
Francis J. Martin (Apothecary)
G.F. Stritch
Garland, John P. (Peter)
George A. Stritch
George M'Killip (Apothecary)
George P. Cope
Gibbs, Richard
Gibson, Wm. J. (William J.)
H.M. Sugrue
H.U. (Hugh Unsworth) Byrne
H.W. (Henry W.) Dutton
H.W. Dutton
Heard, Rich Thos (Richard Thomas)
Henry Davy
Henry G. Day
Henry J. Dutton
Herbert Byrne
Herbert Byrne

Name of Registrar' Assistant (If Any)

Is Name of Assistant is Present?: Present

Certified

Certified

(b) Registrar name field with suggestions

Fig. 9. Data entry error handling in the Historian App by providing suggestions

6. Conclusions

In this paper, we presented the first MDD based application developed in the DBDIrl project to support the correct and reliable data entry of Civil registration records. As this project deals with a large dataset, we need a reliable application that prevents as much as possible errors. To this aim, we co-developed with the Historians a model-driven application using XMDD as an agile version of MDD and the DIME integrated modelling environment. We briefly introduced the various model types and showed how they are used in conjunction to create a coherent data, process, GUI and role-based access model. The main advantage of these choices is the ability to quickly react to the findings, exemplified here by the evolution from the V1 to V2 of the App. In particular, the V2 greatly improves the achieved data quality by making the Application reactive to context-specific events, and equipping most of the data entry fields with pre-populated lists of plausible options, as for addresses causes of death and registrar names, and with context-specific rule checks, as for date and age formats. The main lesson learned is that such an application is necessarily long lived, due to the sheer enormous amount of data to be digitized over time, by many groups of volunteers, and never really «finished». In such a context of continuous improvement, the ability to collaborate with the Historians on the basis of models rather than code is an essential asset, producing successive versions of the app that improve or customise specific aspects of the functionality and the presentation.

The work currently in progress concerns on the one side the inclusion of rule-based classifiers in the application, and on the other side the development in the same paradigm of a data analytics application. The Analysis App needs to be as flexible and customizable as this one, because it will serve the certainly diverse and specialized analysis needs of a growing community of researchers working on big data archival systems in the digital humanities.

References

[1] J. Schnapp, L. Peter, and T. Presner. Digital humanities manifesto, 2008. Available at https://tcp.hypotheses.org/category/manifeste.

[2] M. Grieves and J. Vickers. Digital twin: Mitigating unpredictable, undesirable emergent behavior. In Transdisciplinary perspectives on complex systems, Springer, 2017, pp. 85-113.

[3] Y. Lu, C. Liu et al. Digital twin driven smart manufacturing: Connotation, reference model, applications and research issues. Robotics and Computer-Integrated Manufacturing, vol. 61, 2020, article 101837.

[4] M. Dalibor, J. Michael et al. Towards a model-driven architecture for interactive digital twin cockpits. Lecture Notes in Computer Science, vol. 12400, 2020, pp. 377-387.

[5] T. Margaria and B. Steffen. Extreme model-driven development (xmdd) technologies as a hands-on approach to software development without coding. Encyclopedia of Education and Information echnologies, 2020, pp. 732-750.

[6] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In Proc. of the Conference on Future of Software Engineering (FOSE’07), 2007, pp. 37-54.

[7] C. Breathnach, N. M. Ibrahim et al. Towards model checking product lines in the digital humanities: An application to historical data. Lecture Notes in Computer Science, vol. 11865, 2019, pp. 338-364.

[8] M.D. Wilkinson, M. Dumontier et al. Addendum: The fair guiding principles for scientific data management and stewardship. Scientific data, vol. 6, no. 1, 2019, pp. 1-2.

[9] S. Boßelmann, M. Frohme et al. DIME: A Programming-Less Modeling Environment for Web Applications. Lecture Notes in Computer Science, vol. 9953, 2016, pp. 809-832.

[10] B. Steffen, T. Margaria et al. The metaframe’95 environment. Lecture Notes in Computer Science, vol. 1102, 1996, pp. 450-453.

[11] B. Steffen, T. Margaria et al. An environment for the creation of intelligent network services. In IN/AIN Technologies, Operations, Services, and Applications – A Comprehensive Report, International Engineering Consortium, 1996, pp. 287-300.

[12] P. Fraternali, S. Comai et al. Engineering rich internet applications with a model-driven approach. ACM Transactions on the Web (TWEB), vol. 4, no. 2, 2010, pp. 1-47.

[13] T. Margaria. Web services-based tool-integration in the eti platform. Software & Systems Modeling, vol. 4, no. 2, 2005, pp. 141-156.

[14] T. Margaria, C. Kubczak et al. Model-based design of distributed collaborative bioinformatics processes in the jabc. In Proc. of the 11th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'06), 2006, 8 p.

[15] A.-L. Lamprecht, T. Margaria et al. Genefisher-p: variations of genefisher as processes in bio-jeti. BMC bioinformatics, vol. 9, no. 4, 2008, pp. 1-15.

[16] D. Withers, E. Kawas et al. Semantically-guided workflow construction in Taverna: the SADI and

BioMoby plug-ins. Lecture Notes in Computer Science, vol. 6416, 2010, pp. 301-312.
[17] A R. da Silva. Model-driven engineering: A survey supported by the unified conceptual model. Computer Languages, Systems & Structures, vol. 43, 2015, pp. 139-155.
[18] J.G. Mengerink, A. Serebrenik et al. Automated analyses of model-driven artifacts: obtaining insights into industrial application of mde. 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement, 2017, pp. 116–121.
[19] S. J¨orges, C. Kubczak et al. Model Driven Design of Reliable Robot Control Programs Using the jABC. In Proc. of the Fourth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASe'07, 2007), pp. 137–148.
[20] T. Margaria and A. Schieweck. The digital thread in industry 4.0. Lecture Notes in Computer Science, vol. 11918, 2019, pp. 3-24.
[21] B. Steffen, T. Margaria et al. A constraint-oriented service creation environment. Lecture Notes in Computer Science, vol. 1055, 1996, pp. 418-421.
[22] P. Mikulecky. Formal models for ambient intelligence. In Proc. of the 2010 Sixth International Conference on Intelligent Environments, 2010, pp. 370-371.
[23] G. Savić, M. Segedinac et al. A model-driven approach to e-course management. Australasian Journal of Educational Technology, vol. 34, no. 1, 2018, pp. 14-29.
[24] A. Hernandez-Mendez, N. Scholz, and F. Matthes. A model-driven approach for generating restful web services in single-page applications. In Proc. of the 6th International Conference on Model-Driven Engineering and Software Development, 2018, pp. 480-487.
[25] D. Breuker. Towards model-driven engineering for big data analytics – an exploratory analysis of domain-specific languages for machine learning. In Proc. of the 47th Hawaii International Conference on System Sciences, 2014, pp. 758-767.
[26] M. Golfarelli and S. Rizzi. A model-driven approach to automate data visualization in big data analytics. Information Visualization, vol. 19, no. 1, 2020, pp. 24-47.
[27] C. Castellanos, B. Pérez et al. A model-driven architectural design method for big data analytics applications. In Proc. of the 2020 IEEE International Conference on Software Architecture Companion (ICSA-C), 2020), pp. 89-94.
[28] T. Margaria and B. Steffen. Agile IT: Thinking in User-Centric Models. Communications in Computer and Information Science, vol. 17, 2009, pp. 490–502.
[29] S. Boßelmann, D. Kühn, and T. Margaria. A fully model-based approach to the design of the secube community web app. In Proc. of the 12th International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS), 2017, pp. 1–7.
[30] S. Jörges, A.-L. Lamprecht et al. A Constraint-based Variability Modeling Framework. International Journal on Software Tools for Technology Transfer (STTT), vol. 14, no. 5, 2012, pp. 511-530.
[31] E. O'Shea, R. Khan et al. Towards automatic data cleansing and classification of valid historical data an incremental approach based on mdd. In 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 1914-1923.

Информация об авторах / Information about authors

Rafflesia KHAN, MSc in Computer Science and Engineering from Khulna University, Khulna, Bangladesh, currently Research PhD Student at Computer Science and Information System Department. Her research interests are Big Data, Model-Driven Development, Digital Humanities, Historical Data Analysis, Pattern Recognition, Image classification, Object detection & recognition, Facial behaviour analysis, Image and Video Processing, Internet of Things, Security of IoT, Machine learning, Computer Graphics and Artificial Intelligence.

Раффлезия ХАН, магистр компьютерных наук и инженерии из Университета Кхулна, Кхулна, Бангладеш, в настоящее время аспирант-исследователь факультета компьютерных наук и информационных систем. Ее исследовательские интересы: большие данные, разработка на основе моделей, цифровые гуманитарные науки, анализ исторических данных, распознавание образов, классификация изображений, обнаружение и распознавание объектов, анализ поведения лиц, обработка изображений и видео, Интернет вещей, безопасность Интернета вещей, машинное обучение, гомпьютерная графика и искусственный интеллект.

Alexander SCHIEWECK, Master in Computer Science form the TU Dortmund University, Germany, currently Research PhD Student at Computer Science and Information System Department. Research Interests: Low-Code and Formal Methods for Software Verification as part of the Software Engineering process.

Александр ШИВЕК, магистр компьютерных наук Технического университета Дортмунда, Германия, в настоящее время аспирант-исследователь факультета компьютерных наук и информационных систем. Научные интересы: малокодовые и формальные методы верификации программного обеспечения как часть процесса разработки программного обеспечения.

Ciara BREATHNACH, PhD in History, University College Cork (UCC), Associate Professor in History at the University of Limerick, Ireland. Research Interest: modern Ireland, social history, gender, medicalization, death, migration, health-history; social determinants of health in nineteenth and early twentieth Irish history; infant and maternal mortality; social history of medicalisation; the social function of modern medicine in acculturating Irish immigrants in New York and Boston, 1860- 1912.

Сиара БРЕТНАХ, кандидат исторических наук, Университетский колледж Корка (UCC), доцент кафедры истории. Область исследования: современная Ирландия, социальная история, пол, медикализация, смерть, миграция, история здоровья; социальные детерминанты здоровья в истории Ирландии девятнадцатого и начала двадцатого века; младенческая и материнская смертность; социальный анамнез медикализации; социальная функция современной медицины в воспитании ирландских иммигрантов в Нью-Йорке и Бостоне, 1860-1912 гг.

Tiziana MARGARIA, PhD in Computer and Systems Engineering, Politecnico di Torino, Italy, Professor at Computer Science and Information System Department at the University of Limerick, Ireland. Research Interest: eXtreme Model Driven Design, lightweight formal methods, automatic program synthesis, system correctness, in particular compliance and security, future education in SE and IT.

Тициана МАРГАРИА, кандидат компьютерных наук и системной инженерии, Туринский политехнический университет, Италия, профессор кафедры компьютерных наук и информационных систем. Область научных интересов: проектирование на основе экстремальных моделей, упрощенные формальные методы, автоматический синтез программ, правильность системы, в частности соответствие и безопасность, будущее образование в области программной инженерии и информационной технологии.

DOI: 10.15514/ISPRAS-2021-33(3)-11

Mechanized Theory of Event Structures: A Case of Parallel Register Machine

[1] *V.P. Gladstein, ORCID: 0000-0001-9233-3133 <vovaglad00@gmail.com>*
[1] *D.V. Mikhailovskii,* ORCID: *0000-0002-1026-1170 <mikhaylovskiy.dmitriy@gmail.com>*
[1,2] *E.A. Moiseenko, ORCID: 0000-0003-2715-1143 <e.moiseenko@2012.spbu.ru>*
[3] *A.A. Trunov, ORCID: 0000-0003-0719-4744 <anton@zilliqa.com>*

[1] *Saint Petersburg State University,*
14 line of V.O., 29B, St. Petersburg, 199178, Russia
[2] *JetBrains Research,*
Kantemirovskaya st. 2, room 422, Saint Petersburg, 197342, Russia
[3] *Zilliqa Research*
12 Marina View, Asia Square Tower 2, #11-01, 018961, Singapore

Abstract. The true concurrency models, and in particular event structures, have been introduced in the 1980s as an alternative to operational interleaving semantics of concurrency, and nowadays they are regaining popularity. Event structures represent the causal dependency and conflict between the individual atomic actions of the system directly. This property leads to a more compact and concise representation of semantics. In this work-in-progress report, we present a theory of event structures mechanized in the COQ proof assistant and demonstrate how it can be applied to define certified executable semantics of a simple parallel register machine with shared memory.

Keywords: semantics; event structures; interactive theorem proving; Coq

For citation: Gladstein V.P., Mikhailovskii D.V., Moiseenko E.A., Trunov A.A. Mechanized Theory of Event Structures: A Case of Parallel Register Machine. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 143-154. DOI: 10.15514/ISPRAS-2021-33(3)-11

Acknowledgements. Evgenii Moiseenko was supported by RFBR according to the research project № 20-31-90088

Механизированная теория структур событий: случай параллельной регистровой машины

[1] *В.П. Гладштейн, ORCID: 0000-0001-9233-3133 <vovaglad00@gmail.com>*
[1] *Д.В. Михайловский, ORCID: 0000-0002-1026-1170 <mikhaylovskiy.dmitriy@gmail.com>*
[1,2] *Е.А. Моисеенко, ORCID: 0000-0003-2715-1143 <e.moiseenko@2012.spbu.ru>*
[3] *А.А. Трунов, ORCID: 0000-0003-0719-4744 <anton@zilliqa.com>*

[1] *Санкт-Петербургский государственный университет,*
Россия, 199178, 14 линия В.О., Санкт-Петербург, 29б
[2] *JetBrains Research,*
Россия, 197342, Санкт-Петербург, Кантемировская ул. 2, каб. 422
[3] *Zilliqa Research*
Сингапур, 018961

Аннотация. Модели истинной конкурентности и, в частности, структуры событий были представлены в 1980-ых как альтернатива операционным семантикам с чередованием, и на сегодняшний день эти модели вновь обретают популярность. Структуры событий позволяют явно выразить отношения причинно-следственной связи и конфликта между атомарными событиями системы, что приводит к более компактному и лаконичному представлению семантики. В данной отчете о текущей работе мы представляем теорию структур событий, механизированную в системе интерактивного доказательства теорем COQ и демонстрируем пример применения этой теории к проблеме задания сертифицированной исполняемой семантики простой параллельной регистровой машины с разделяемой памятью.

Ключевые слова: семантика; структуры событий; интерактивное доказательства теорем; Coq

Для цитирования: Гладштейн В.П., Михайловский Д.В., Моисеенко Е.А., Трунов А.А. Механизированная теория структур событий: случай параллельной регистровой машины. Труды ИСП РАН, том 33, вып. 3, 2021 г., стр. 143-154 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(3)–11.

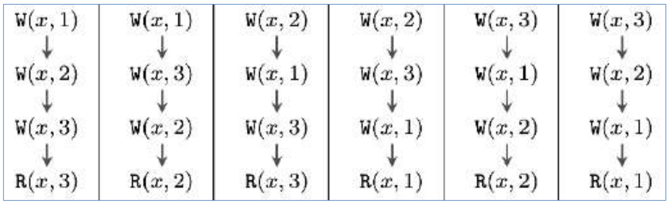
Благодарности: Евгений Моисеенко выполнял данное исследование при финансовой поддержке РФФИ в рамках научного проекта № 20-31-90088.

1. Introduction

Event structures is a mathematical formalism introduced by Winskel [1] as a semantic domain of concurrent programs. In recent years there has been renewed interest in event structures, with the applications of the theory ranging from relaxed memory models [2-4] to model-based mutation testing [5]. The main advantage of event structures compared to traditional interleaving semantics is that they give a more compact and concise representation of programs' behaviors. For example, consider the following code snippet of a simple parallel program.

$$x := 1 \,\|\, x := 2 \,\|\, x := 3$$

$$r := x$$



W(x, 1)	W(x, 1)	W(x, 2)	W(x, 2)	W(x, 3)	W(x, 3)
↓	↓	↓	↓	↓	↓
W(x, 2)	W(x, 3)	W(x, 1)	W(x, 3)	W(x, 1)	W(x, 2)
↓	↓	↓	↓	↓	↓
W(x, 3)	W(x, 2)	W(x, 3)	W(x, 1)	W(x, 2)	W(x, 1)
↓	↓	↓	↓	↓	↓
R(x, 3)	R(x, 2)	R(x, 3)	R(x, 1)	R(x, 2)	R(x, 1)

Fig. 1. Example of program traces

Under the interleaving semantics, it has 3! = 6 traces with each trace consisting of 4 events, as depicted in fig.1. Events themselves represent atomic side-effects produced by instruction executions. In our case, an event is either a write of a value `a` to a shared variable `x` denoted as `W(x,a)`, or a read of a value `a` from a shared variable `x` denoted as `R(x,a)`. The same information can be encoded in a single event structure containing 6 events in total (see fig.2). In the event structure, there are two types of edges between the events. The grey arrows represent the causality relation, a partial order reflecting the causal relationship between the atomic events of computation. The red edges represent the conflict relation which is a symmetric and irreflexive relation encoding mutually exclusive events. Each particular trace can be extracted from the event structure as a linearization of some configuration, that is a causally-closed and conflict-free subset of events, which additionally should satisfy the constraint that each read is preceded by a matching write.

The programming languages theory and formal semantics research communities are moving to increase the usage of proof assistants like COQ [6], AGDA [7], ISABELLE/HOL [8], AREND [9], and others, to complement theoretical studies with their mechanization, as this process increases the reliability and reproducibility of scientific results. Yet, to the best of our knowledge, there is little work on mechanization of the theory of event structures. The present report aims to close the gap. We have chosen COQ as the proof assistant because it's a mature formal proof management tool with a rich ecosystem of libraries, plugins, documentation, and existing applications including the certification of properties of programming languages: the verified C compiler CompCert [10], the Verified Software Toolchain [11] for verification of C programs, and the Iris framework [12] for concurrent separation logic, to name a few.

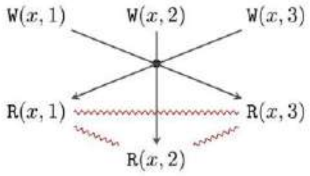


Fig. 2. Example of program event structure

Our end goal is to develop a COQ library containing a comprehensive set of common definitions, lemmas, and tactics that would allow researchers to utilize the theory of event structures for the needs of their domain. In this work-in-progress report, we sketch the common design principles behind our library and give a concrete example of its usage by developing a formal mechanized semantics of a simple register machine with shared memory. Our library together with the examples of its usage is available online at https://github.com/event-structures/event-struct.

2. Related Work

Event structures were introduced by Winskel to study the semantics of the calculus of communicating systems [1], [13]. Several modifications of event structures [14], [15] were later proposed to tackle similar problems. More recently, event structures were applied in the context of relaxed memory models [2–4], [16].

Among this line of work, we are aware of only one paper [16] that was accompanied by a mechanization in a proof assistant. The authors formalized the WEAKESTMO [4] memory model in COQ. However, this memory model uses a custom variant of event structures, that does not obey the axioms of any conventional class of event structures [13–15]. This fact makes it harder to reuse and adapt it to other applications of the theory.

3. Background

There exist several modifications of event structures. Currently, we have implemented only the prime event structures [1] in our library. We give some background on this class of event structures below.

Definition 3.1: A prime event structure (PES) is a triple $(E, \leq, \#)$ where

- E is a set of events
- $\leq$ is a causality relation on E such that
  - $(E, \leq)$ is a partial order
  - for every $e \in E$ its causality prefix $\lceil e \rceil \overset{\text{def}}{=} \{e' : e' \leq e\}$ is finite, i.e., every event is caused by a finite set of events.
- # is a conflict relation on E such that
  - # is irreflexive and symmetric
  - it satisfies hereditary condition: $e_1 \# e_2$ and $e_2 \leq e_3$ implies $e_1 \# e_3$. That is, if two events are in conflict, then all their causal successors are necessarily in conflict.

A single prime event structure can encode multiple runs of a program. Each individual run can be extracted as a configuration. In other words, configurations are used to model a history of computation up to a certain point.

Definition 3.2: A configuration of PES $(E, \leq, \#)$ is a set of events $X \subseteq E$ such that

- it is causally closed: $e_1 \leq e_2$ and $e_2 \in X$ then $e_1 \in X$
- and conflict-free: if $e_1, e_2 \in X$ then $e_1 \# e_2$ is false

4. Overview of Our Library

In this section, we sketch the design principles of our library. We build our mechanization on top of the MATHCOMP [17] library which is an extensive and coherent repository of formalized mathematical theories, whose implementation is based on the SSREFLECT [18] extension of the COQ system. By using MATHCOMP, we draw on the large corpus of already formalized algorithms and mathematical results: its core mod-ules feature support for a range of useful data structures, e.g. numbers, sequences, finite graphs, and also interfaces: types with decidable equality, subtypes, finite types, and so on. We also use the small-scale reflection methodology [18], [19], a key ingredient of SSREFLECT. The small-scale reflection approach is based on the pervasive use of symbolic representations intermixed with logical ones within the confines of the same proof goal, as opposed to large-scale reflection which does not allow such mixing. Symbolic representations are connected to the corresponding logical ones via user-defined reflect predicates. The symbolic representation can be manipulated by the computational engine of the language, allowing the user to automate low-level routine proof management by using various decision and simplification procedures. Whenever the user needs to guide the proof they can switch to the logical representation and perform some proof steps manually. To achieve better automation and e.g. get proof irrelevance for free, one is encouraged to use decision procedures whenever possible. For example, in the context of our library, we encode the binary relations of the event structures as decidable bool-valued relations, i.e., $\leq, \# : E \to E \to bool$, as opposed to propositional relations of type $\leq, \# : E \to E \to Prop$. Encoding computable relations in COQ, especially their (computable) transitive closures, can be quite challenging since COQ is a total language and its termination checker only understands termination patterns going slightly beyond simple structural recursion. To make it easier, we employ the EQUATIONS function definition plugin [20] which provides both notations for writing programs by dependent pattern-matching and good support for well-founded recursion. In fact, binary relations are omnipresent in our formalization. This quickly manifested in a substantial amount of proof overhead and we sought for tools to automate our proofs. Since binary relations form a Kleene Algebra with Tests (KAT) [21] ,we have chosen to use the RELATION-ALGEBRA [22] package which provides a number of tactics to solve goals using decision procedures for a number of theories, such as partially ordered monoids, lattices, residuated Kleene allegories and KATs.

We also favor the computational encoding of semantics. Similar to the recent related works on mechanization of operational semantics [23–25], we encode the semantics as monadic interpreters. This allows us to extract [26] the semantics as a functional program and run it. We believe that the possibility to run the semantics is a very useful feature, as it allows to debug the formal semantics and helps to develop better intuition about it.

To facilitate computable semantics, we define a subclass of finitely supported event structures as a finite sequence of events combined with a finitely supported function which enhances events with additional information, such as their labels, causality predecessors, etc. Encoding finitely supported functions is not a trivial endeavor in a proof assistant and for this task we use the FINMAP library which is an extension of MATHCOMP providing finite sets and finite maps on types with a choice operator (rather than on finite types).

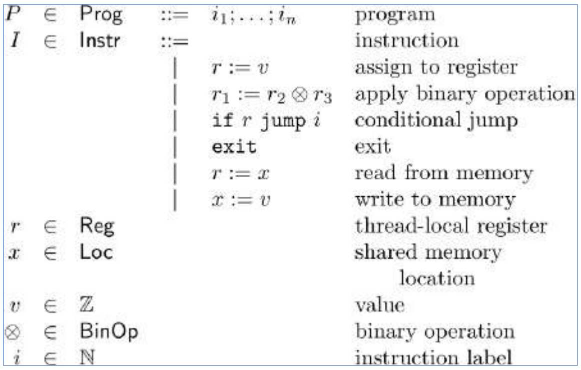
Finally, to encode the algebraic hierarchy of various classes of event structures we use yet another feature of MATHCOMP—packed classes [27], which is a design pattern providing multiple inheritance, maximal sharing of notations and theories, and automated structure inference.

5. Case Study

In this section, we provide a case study demonstrating an application of our mechanized theory of event structures. We show how it can be used to encode the semantics of a parallel register machine equipped with shared memory.

5.1 Register Machine

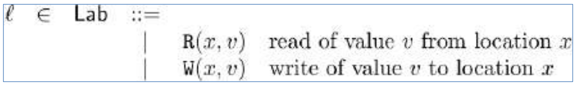
For our case study, we use a simple idealized model of a register machine, which consists of a finite sequence of instructions, an instruction pointer, and an infinite set of registers. The syntax of the machine's language is shown in fig. 3.



P	$\in$	Prog	$::=$	$i_1; \ldots ; i_n$	program
I	$\in$	Instr	$::=$		instruction
			$\mid$	$r := v$	assign to register
			$\mid$	$r_1 := r_2 \otimes r_3$	apply binary operation
			$\mid$	$\texttt{if } r \texttt{ jump } i$	conditional jump
			$\mid$	$\texttt{exit}$	exit
			$\mid$	$r := x$	read from memory
			$\mid$	$x := v$	write to memory
r	$\in$	Reg			thread-local register
x	$\in$	Loc			shared memory location
v	$\in$	$\mathbb{Z}$			value
$\otimes$	$\in$	BinOp			binary operation
i	$\in$	$\mathbb{N}$			instruction label

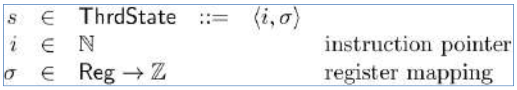
Fig. 3. Syntax of the register machine

We first present the semantics of a single-threaded program. Under this semantics, memory access instructions do not operate on shared memory but rather produce a label denoting the side-effect of the operation (see fig. 4). This encoding allows us to decouple the semantics of the register machine from a memory model.



ℓ	$\in$	Lab	$::=$		
			$\mid$	$\texttt{R}(x, v)$	read of value v from location x
			$\mid$	$\texttt{W}(x, v)$	write of value v to location x

Fig. 4. Syntax of labels



s	$\in$	ThrdState	$::=$	$\langle i, \sigma \rangle$	
i	$\in$	$\mathbb{N}$			instruction pointer
σ	$\in$	$\mathsf{Reg} \to \mathbb{Z}$			register mapping

Fig. 5. Thread state of the register machine

The semantics is given in the form of a labelled transition system: $P \vdash s \rightarrow_l s'$, where P is a program, l is a label, s and s' are states of the machine. The state of the machine itself consists of an instruction pointer i and a map from registers to their values σ, as shown in fig. 5. The rules of the semantics are standard (see fig. 6). As we have mentioned, in our COQ development we actually use the monadic encoding of the operational semantics. The labelled transition system can be derived from this encoding.

$$\frac{P[i] = \ r := v}{P \vdash \langle i, \sigma\rangle \xrightarrow{\epsilon} \langle i+1, \sigma[r \mapsto v]\rangle}\ \texttt{Assign} \qquad \frac{P[i] = \ r_1 := r_2 \otimes r_3 \qquad v = \sigma(r_2) \otimes \sigma(r_3)}{P \vdash \langle i, \sigma\rangle \xrightarrow{\epsilon} \langle i+1, \sigma[r_1 \mapsto v]\rangle}\ \texttt{Binop}$$

$$\frac{P[i] = \ x := v}{P \vdash \langle i, \sigma\rangle \xrightarrow{\texttt{W}(x,v)} \langle i+1, \sigma\rangle}\ \texttt{Store} \qquad \frac{P[i] = \ \texttt{exit} \qquad \mathsf{len}(P) = n}{P \vdash \langle i, \sigma\rangle \xrightarrow{\epsilon} \langle n, \sigma\rangle}\ \texttt{Exit}$$

$$\frac{P[i] = \ r := x}{P \vdash \langle i, \sigma\rangle \xrightarrow{\texttt{R}(x,v)} \langle i+1, \sigma[r \mapsto v]\rangle}\ \texttt{Load} \qquad \frac{P[i] = \ \texttt{if}\ r\ \texttt{jump}\ j \qquad \sigma(r) = 0}{P \vdash \langle i, \sigma\rangle \xrightarrow{\epsilon} \langle i+1, \sigma\rangle}\ \texttt{CJump}_z$$

$$\frac{P[i] = \ \texttt{if}\ r\ \texttt{jump}\ j \qquad \sigma(r) \neq 0}{P \vdash \langle i, \sigma\rangle \xrightarrow{\epsilon} \langle j, \sigma\rangle}\ \texttt{CJump}_{nz}$$

Fig. 6. Thread semantics of the register machine

5.2 Event Structure of Register Machine

In this section we present operational semantics which constructs a prime event structure encoding a set of possible behaviors of the register machine. The event structure is constructed incrementally in a step-by-step fashion by adding a single event on each step. In order to generate a new event on each step, we require that events behave as identifiers.

Definition 5.1: We say that a set E together with strict partial order $\prec$ form an identifier set if

- there exists a distinguished initial identifier $e_0 \in E$
- there exists a function $fresh : E \to E$ which generates a new fresh identifier, such that
$$e \prec fresh(e)$$

We will encode the event structure as a tuple $(\mathcal{E},\ lab,\ f_{po},\ f_{rf})$ and explain below the meaning of each component, and how they together form a prime event structure.

The first component $\mathcal{E}$ is a sequence of events $e_1 \succ \ldots \succ e_n$ in reverse order w.r.t the order in which events get added to the structure. The second component is a labelling function $lab : E \to L$, assigning a label to each event.

Next, following the theory of axiomatic weak memory models [28], we define the causality relation of the register machine's event structure as the reflexive transitive closure of the union of two relations —*program order* and *reads-from*, denoted as *po* and *rf* correspondingly.

$$\leq \overset{\text{def}}{=} (po \cup rf)^*$$

The program order relation tracks precedence of events within a single thread. The reads-from relation captures the flow of values from write events to read events and ensures that values do not appear out of thin air [28], [29].

In order to construct *po* and *rf* incrementally we represent them via their inverse covering functions f_{po} and f_{rf}.

Definition 5.2 (Covering): Let $\leq$ be a partial order. Then $\lessdot$ is covering relation w.r.t. $\leq$ whenever $x \lessdot y$ is true if and only if $x < y$ and there is no z, s.t. $x < z$ and $z < y$. A (non-deterministic) function f from A to the set of finite subsets of A is a covering function if its corresponding relation, i.e., $f^{\uparrow} \overset{\text{def}}{=} \{\langle x, y\rangle \mid y \in f(x)\}$, is a covering relation.

We use the inverse covering function because it is more convenient in our setting. Indeed, the semantics adds a new event at each step. Then it is convenient to require that, in addition, the small-step relation is provided with the *po* and *rf* predecessors of a new event.

$$\lessdot_{po} \overset{\text{def}}{=} {f_{po}^{\uparrow}}^{-1} \qquad\qquad po \overset{\text{def}}{=} \lessdot_{po}^{+}$$

$$<_{rf} \overset{\text{def}}{=} {f_{rf}^{\uparrow}}^{-1} \qquad rf \overset{\text{def}}{=} <_{rf}^{+}$$

We define the conflict relation in two steps. First, we define the primitive conflict relation $\sim_{\#}$ which is generated by the f_{po} function. The two events are considered to be in primitive conflict if they are not equal and have a common *po* predecessor. For this definition to work properly, we also need to assume that each thread has a special initial event labelled by a distinguished thread start label `TS`.

$$e_1 \sim_{\#} e_2 \overset{\text{def}}{=} e_1 \neq e_2 \land f_{po}(e_1) = f_{po}(e_2)$$

Second, we extend the primitive conflict along the causality relation:

$$e_1 \# e_2 \overset{\text{def}}{=} \exists e'_1, e'_2 \in E.\, e'_1 \sim_{\#} e'_2 \land e'_1 \leq e_1 \land e'_2 \leq e_2$$

We also need a way to reconcile the event structure with the states of the machine's threads. To do so, we use a function $\Sigma : E \rightarrow ThrdState$ which maps an event to a thread state obtained as the result of the execution of the event's side-effect.

Let us consider an example. Given the program below, our semantics builds the corresponding event structure as shown in fig. 7.

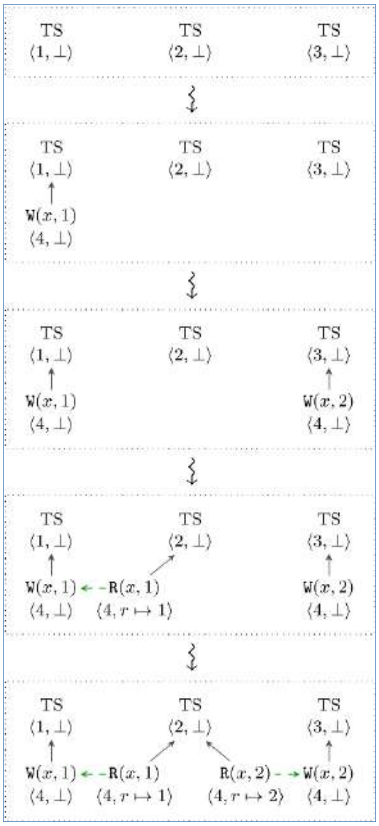


Fig. 7. Example of the event structure construction

The construction starts from an initial event structure containing, for each thread, an event labelled by `TS`. We depict the corresponding thread state below each label. Initially, each event is mapped to an initial thread state consisting of an instruction pointer pointing to the first instruction to be executed and an initial mapping of registers denoted as ⊥. The first step executes the store instruction from the leftmost thread and exits the program, since the execution of this thread terminates (we omit the `exit` instructions at the end of each thread for brevity). Next, the store from the rightmost thread is executed and the corresponding write event gets added to the structure. After that, the load instruction from the middle thread is executed. Since there are two matching write events in the event structure, two conflicting reads are conjoined to the event structure. Note that the events can be added non-deterministically in any order respecting causality. We could have first executed the rightmost thread and added write `W(x,2)` before `W(x,1)`, or we could have added the read with label `R(x,2)` before another read `R(x,1)`.

$$\frac{e = \mathsf{fresh}(\mathsf{first}(\mathcal{E})) \qquad e_{\mathrm{po}} \in \mathcal{E} \qquad e_{\mathtt{rf}} \in \mathcal{E}}{\langle \mathcal{E}, \mathtt{lab}, f_{\mathrm{po}}, f_{\mathtt{rf}}\rangle \xhookrightarrow{\langle e,\ell,e_{\mathrm{po}},e_{\mathtt{rf}}\rangle} \langle e :: \mathcal{E}, \mathtt{lab}[e \mapsto \ell], f_{\mathrm{po}}[e \mapsto e_{\mathrm{po}}], f_{\mathtt{rf}}[e \mapsto e_{\mathtt{rf}}]\rangle}\ \text{(Add Event)}$$

$$\frac{e \in S.\mathcal{E} \qquad s = \Sigma(e) \qquad P \vdash s \xrightarrow{\epsilon} s'}{P \vdash \langle S, \Sigma\rangle \xRightarrow{\epsilon} \langle S, \Sigma[e \mapsto s']\rangle}\ \text{(Idle)}$$

$$\frac{s = \Sigma(e_{\mathrm{po}}) \qquad P \vdash s \xrightarrow{\ell} s' \qquad S \xhookrightarrow{\langle e,\ell,e_{\mathrm{po}},\bot\rangle} S' \qquad l = \mathtt{W}(x,v)}{P \vdash \langle S, \Sigma\rangle \xRightarrow{\langle e,\ell,e_{\mathrm{po}},\bot\rangle} \langle S', \Sigma[e \mapsto s']\rangle}\ \text{(Store)}$$

$$\frac{s = \Sigma(e_{\mathrm{po}}) \qquad P \vdash s \xrightarrow{\ell} s' \qquad S \xhookrightarrow{\langle e,\ell,e_{\mathrm{po}},e_{\mathtt{rf}}\rangle} S' \qquad \neg(e \# e_{\mathtt{rf}}) \qquad \begin{array}{c} l = \mathtt{R}(x,v) \\ \mathtt{lab}(e_{\mathtt{rf}}) = \mathtt{W}(x,v) \end{array}}{P \vdash \langle S, \Sigma\rangle \xRightarrow{\langle e,\ell,e_{\mathrm{po}},e_{\mathtt{rf}}\rangle} \langle S', \Sigma[e \mapsto s']\rangle}\ \text{(Load)}$$

$$\frac{s = \Sigma(e_{\mathrm{po}}) \qquad P \vdash s \xrightarrow{\ell} s' \qquad S \xhookrightarrow{\langle e,\ell,e_{\mathrm{po}},\bot\rangle} S' \qquad l = \mathtt{R}(x,\bot)}{P \vdash \langle S, \Sigma\rangle \xRightarrow{\langle e,\ell,e_{\mathrm{po}},\bot\rangle} \langle S', \Sigma[e \mapsto s']\rangle}\ \text{(Load-Bottom)}$$

Fig. 8. Semantics of register machine's event structure construction

The rules of operational semantics constructing the event structure are presented in fig. 8. The first auxiliary rule (Add Event) adds a new event, sets its label, *po* and *rf* predecessors. The (Idle) handles the case when a thread of the register machine performs an internal step without anyside effect. It chooses an event *e* together with the thread state *s* corresponding to it and performs one step reduction to a new state *s′*. It then updates the mapping of events to thread states. The last three rules (Store), (Load), and (Load-Bottom) correspond to store and load performed by some thread. Similarly to (Idle), an event e_{po} is selected and one reduction is performed from the corresponding thread state *s*. Unlike the (Idle) case, however, a new event *e* is also generated. In the case of (Load), additionally, an event e_{rf} is selected, such that it has a write label matching the read label of the new event. The rule (Load-Bottom) corresponds to a case when load is performed "too early", before any write to the given location is available.

The following theorem asserts that the event structure built this way indeed satisfies the axioms of the prime event structure.

Theorem 1: The tuple $(E, \leq, \#)$, where $\leq$ and # are defined as described above, forms prime event structure.

We sketch the proof below (one can also find mechanized proof in our COQ development).

First, we need to show that $\leq \stackrel{\text{def}}{=} (po \cup rf)^*$ is a partial order. Reflexivity and transitivity follows immediately from the definition of the reflexive-transitive closure. To show antisymmetry note that $<_{po} \subseteq \prec$ and $<_{rf} \subseteq \prec$ by construction. Therefore $\leq$ is a subset of the reflexive closure of $\prec$. Since $\prec$ is a partial order, it is antisymmetric, and thus $\leq$ should also be antisymmetric. The axiom of finite cause, i.e., $\lceil e \rceil$ is finite for every event *e*, follows from the fact that at each step of the construction the set of possible predecessors of the new event can be over-approximated by the finite sequence $\mathcal{E}$.

Second, we need to show that the conflict relation # defined as described above obeys the laws of the conflict relation. Trivially, this relation is symmetric, and obeys the hereditary property. The side condition $\neg\ (e\ \#\ e_{rf})$ of the rule (Load) ensures that the conflict relation is irreflexive.

In fig. 9 one can see the prime event structure obtained as a result of the incremental construction depicted in fig. 7.

$$\begin{tikzcd}
\substack{\mathtt{TS}\\ \langle 1,\bot\rangle} \arrow[d] & & & \substack{\mathtt{TS}\\ \langle 2,\bot\rangle} \arrow[dl] \arrow[dr] & & & \substack{\mathtt{TS}\\ \langle 3,\bot\rangle} \arrow[d] \\
\substack{\mathtt{W}(x,1)\\ \langle 4,\bot\rangle} \arrow[rr] & & \substack{\mathtt{R}(x,1)\\ \langle 4,r\mapsto 1\rangle} \arrow[rr, squiggly, no head] & & \substack{\mathtt{R}(x,2)\\ \langle 4,r\mapsto 2\rangle} & & \substack{\mathtt{W}(x,2)\\ \langle 4,\bot\rangle} \arrow[ll]
\end{tikzcd}$$

Fig. 9. Example of prime event structure

Once the event structure is constructed, one can extract the configurations corresponding to the particular runs of the parallel register machine, and further filter them via the consistency predicate defining the memory consistency model.

Our construction of event structures allows to encode a wide class of so-called $po \cup rf$ acyclic relaxed memory models [28].

For example, a predicate corresponding to sequential consistency [30] requires that the causality order can be extended to a total order on all events of the configuration, such that for each read event the last preceding write event to the same location has the same value as the read.

6. Future Work

There are several directions for future work.

First, we plan to apply our library to a wider range of problems. We are going to develop a mechanized semantics of some long-established languages used to model concurrency, in particular the calculus of communicating systems (CCS) [31] and π-calculus [32].

We also plan to continue our work on expressing various relaxed models of shared memory [28], [33], [34] in terms of event structures. Second, we want to cover other classes of event structures in our library, in particular bundle [14], flow [15], and stable [1], [13] event structures. We plan to use them to develop mechanized denotational semantics of concurrent languages and relaxed shared memory models [35].

Finally, we plan to mechanize in COQ classical results that connect various classes of event structures [15], [36]. It would allow us to easily establish the connection between operational and denotational semantics of concurrent languages.

References

[1]. G. Winskel. Event structures. Lecture Notes in Computer Science, vol. 255, 1986, pp. 325-392.
[2]. A. Jeffrey and J. Riely. On thin air reads: Towards an event structures model of relaxed memory. In Proc. of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, 2016, pp. 759-767.
[3]. J. Pichon-Pharabod and P. Sewell. A concurrency semantics for relaxed atomics that permits optimisation and avoids thin-air executions. ACM SIGPLAN Notices, vol. 51, issue 1, 2016, pp. 622-633.
[4]. S. Chakraborty and V. Vafeiadis. Grounding thin-air reads with event structures. Proceedings of the ACM on Programming Languages, vol. 3, issue POPL, 2019, pp. 1-28.
[5]. A. Fellner, T. Tarrach, and G. Weissenbacher. Language inclusion for finite prime event structures. Lecture Notes in Computer Science, vol. 11990, 2020, pp. 314-336.
[6]. The Coq Development Team. The Coq Proof Assistant, 2021. Available at https://coq.inria.fr/, accessed 7-May-2021.
[7]. Agda language reference. Available at https://agda.readthedocs.io/, accessed 7-May-2021.
[8]. T. Nipkow, L. C. Paulson, and M. Wenzel. Isabelle/HOL: a proof assistant for higher-order logic. Lecture Notes in Computer Science, vol. 2283, 2002, 240 p.
[9]. Arend theorem prover. Available at https://arend-lang.github.io/, accessed 7-May-2021.

[10]. X. Leroy. Formal verification of a realistic compiler. Communications of the ACM, vol. 52, no. 7, 2009, pp. 107–115.
[11]. A.W. Appel. Verified software toolchain. Lecture Notes in Computer Science, vol. 6602, 2011, pp. 1-17.
[12]. R. Jung, R. Krebbers et al. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. Journal of Functional Programming, vol. 28, 2018, article e 20.
[13]. G. Winskel. Event structure semantics for CCS and related languages. Lecture Notes in Computer Science, vol. 140, 1982, pp. 561-576.
[14]. R. Langerak. Bundle event structures: a non-interleaving semantics for LOTOS. In Proc. of the 5th International Conference on Formal Description Techniques for Distributed Systems and Communications Protocols, 1992, pp. 331-346.
[15]. G. Boudol and I. Castellani. Flow models of distributed computations: event structures and nets. Research Report RR-1482, INRIA, 1991, 40 p.
[16]. E. Moiseenko, A. Podkopaev et al. Reconciling event structures with modern multiprocessors. In Proc. of the 34th European Conference on Object-Oriented Programming, 2020, 26 p.
[17]. A. Mahboubi and E. Tassi. Mathematical components, 2017. Available at http://doi.org/10.5281/zenodo.4457887, accessed 7-May-2021.
[18]. G. Gonthier, A. Mahboubi, and E. Tassi. A small scale reflection extension for the Coq system. Research Report RR-6455, Inria Saclay Ile de France, 2016, 69 p.
[19]. G. Gonthier and A. Mahboubi. An introduction to small scale reflection in Coq. Journal of formalized reasoning, vol. 3, no. 2, 2010, pp. 95-152.
[20]. M. Sozeauand, C. Mangin. Equations reloaded: High-level dependently-typed functional programming and proving in Coq. Proceedings of the ACM on Programming Languages, vol. 3, 2019, article no. 86.
[21]. D. Kozen. Kleene algebra with tests. ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 19, no. 3,1997, pp. 427-443.
[22]. D. Pous. Kleene algebra with tests and Coq tools for while programs. Lecture Notes in Computer Science, vol. 7998, 2013, pp. 180-196.
[23]. L.-y. Xia, Y. Zakowski et al. Interaction trees: representing recursive and impureprograms in Coq. Proceedings of the ACM on Programming Languages, vol. 4, issue POPL, 2019, pp. 1-32.
[24]. T. Letan and Y. R ́egis Gianas. Freespec: specifying, verifying, and executing impure computations in Coq. In Proc. of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, 2020, pp. 32-46.
[25]. R. Affeldt, D. Nowak, and T. Saikawa. A hierarchy of monadic effects for program verification using equational reasoning. Lecture Notes in Computer Science, vol. 11825, 2019, pp. 226-254.
[26]. P. Letouzey. Extraction in Coq: An overview. Lecture Notes in Computer Science, vol. 5028, 2008, pp. 359-369.
[27]. F. Garillot, G. Gonthier et al. Packaging mathematical structures. Lecture Notes in Computer Science, vol. 5674, 2009, pp. 327-342.
[28]. O. Lahav, V. Vafeiadis et al. Repairing sequential consistency in C/C++11. In Proc. of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2017, pp. 618-632.
[29]. H.-J. Boehm and B. Demsky. Outlawing ghosts: Avoiding out-of-thin-air results. In Proc. of the Workshop on Memory Systems Performance and Correctness, 2014, article no. 7.
[30]. L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Transactions on Computers, vol. 28, no. 9, 1979, pp. 690-691.
[31]. R. Milner. A calculus of communicating systems. Springer-Verlag, 1980, 260 p.
[32]. R. Milner. Communicating and mobile systems: the pi calculus. Cambridge university press, 1999, 176 p.
[33]. O. Lahav, N. Giannarakis, and V. Vafeiadis. Taming release-acquire consistency. ACM SIGPLAN Notices, vol. 51, no. 1, 2016, pp. 649-662
[34]. A. Podkopaev, O. Lahav, and V. Vafeiadis. Bridging the gap between programming languages and hardware weak memory models. Proceedings of the ACM on Programming Languages, vol. 3, no. POPL, 2019, pp. 1-31.
[35]. M. Dodds, M. Batty, and A. Gotsman. Compositional verification of compiler optimizations on relaxed memory. Lecture Notes in Computer Science, vol. 10801, 2018, pp. 1027-1055.
[36]. M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, part I. Theoretical Computer Science, vol. 13, no. 1, 1981, pp. 85-108.

Информация об авторах / Information about authors

Владимир Петрович ГЛАДШТЕЙН – студент бакалавриата Санкт-Петербургского Государственного Университета. Сфера научных интересов: методы формальной верификации программ, системы интерактивного доказательства теорем, теория зависимых типов.

Vladimir GLADSTEIN – bachelor student at Saint Petersburg State University. Research interests: formal verification of programs, interactive theorem proving, dependent types theory.

Дмитрий Владимирович МИХАЙЛОВСКИЙ – студент бакалавриата Санкт-Петербургского Государственного Университета. Сфера научных интересов: методы формальной верификации программ, системы интерактивного доказательства теорем, теория зависимых типов.

Dmitrii MIKHAILOVSKII – bachelor student at Saint Petersburg State University. Research interests: formal verification of programs, interactive theorem proving, dependent types theory.

Евгений Александрович МОИСЕЕНКО – аспирант Санкт-Петербургского Государственного Университета, исследователь в JetBrains Research. Сфера научных интересов: методы формальной верификации программ, семантика конкурентных программ.

Evgenii MOISEENKO – PhD student at Saint Petersburg State University, Researcher at JetBrains Research. Research interests: formal verification of programs, semantics of concurrency.

Антон Александрович ТРУНОВ – инженер-исследователь в Zilliqa Research. Сфера научных интересов: методы формальной верификации программ, системы интерактивного доказательства теорем.

Anton TRUNOV – research engineer at Zilliqa Research. Research interests: formal verification of programs, interactive theorem proving.

DOI: 10.15514/ISPRAS-2021-33(3)-12

Generation of Petri Nets Using Structural Property-Preserving Transformations

[1,2] R.A. Nesterov, ORCID: 0000-0002-4162-9070 <rnesterov@hse.ru>
[1] S.Yu. Savelyev, ORCID: 0000-0003-1660-2615 <syusavelev@edu.hse.ru>
[1] HSE University,
11, Pokrovsky boulevard, Moscow, 109028, Russia
[2] Univerist`a degli Studi di Milano-Bicocca,
1 Piazza dell'Ateneo Nuovo, 20126 Milan, Italy

Abstract. In this paper, we present an approach to the generation of Petri nets exhibiting desired structural and behavioral properties. Given a reference Petri net, we apply a collection of local refinement transformations, which extends the internal structure of the reference model. The correctness of applying these transformations is justified via Petri net morphisms and by the fact that transformations do not add new deadlocks to Petri nets. We have designed two Petri net refinement algorithms supporting the randomized and fixed generation of models. These algorithms have been implemented and evaluated within the environment of the Carassius Petri net editor. The proposed approach can be applied to evaluate and conduct experiments for algorithms operating with Petri nets.

Keywords: Petri nets; morphisms; property-preserving transformations; generation of models

For citation: Nesterov R.A., Savelyev S.Yu. Generation of Petri Nets Using Structural Property-Preserving Transformations. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 155-170. DOI: 10.15514/ISPRAS-2021-33(3)-12

Acknowledgments. This work is supported by the Basic Research Program at the National Research University Higher School of Economics (HSE University), Russia.

Генерация сетей Петри с помощью структурных трансформаций, сохраняющих поведенческие свойства

[1,2] Р.А. Нестеров, ORCID: 0000-0002-4162-9070 <rnesterov@hse.ru>
[1] С.Ю. Савельев, ORCID: 0000-0003-1660-2615 <syusavelev@edu.hse.ru>
[1] Национальный исследовательский университет Высшая школа экономики,
109028, Россия, Москва, Покровский бульвар 11
[2] Миланский университет-Бикокка
20126, Италия, г. Милан, пл. Атенео Нуово, 1

Аннотация. В работе предложен подход к генерации сетей Петри, обладающих желаемыми структурными и поведенческими свойствами. На вход подается эталонная сеть Петри, к которой применяется набор локальных трансформаций, расширяющих ее внутреннюю структуру. Корректность применения этих трансформаций обуславливается тем, что они порождают морфизмы на сетях Петри, а также не добавляют новых тупиков при расширении эталонных сетей Петри. Таким образом, сохраняются поведенческие и структурные свойства, которыми обладает эталонная сеть Петри. Были разработаны алгоритмы фиксированной и случайной генерации сетей Петри. Эти алгоритмы реализованы в виде расширения для редактора сетей Петри Carassius. Кроме того, проведена экспериментальная оценка разработанных алгоритмов. Предлагаемый подход к генерации сетей Петри и разработанные алгоритмы могут применяться для проведения комплексной экспериментальной оценки и нагрузочного тестирования алгоритмов, на вход которым подаются модели поведения процессов в виде сетей Петри.

Ключевые слова: сети Петри; морфизмы; трансформации; сохранение свойств; генерация моделей

Для цитирования: Нестеров Р.А., Савельев С.Ю. Генерация сетей Петри с помощью структурных трансформаций, сохраняющих поведенческие свойства. Труды ИСП РАН, том 33, вып. 3, 2021 г., стр. 155-170 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(3)–12.

Благодарности. Работа поддержана Программой фундаментальных исследований Национального исследовательского университета «Высшая школа экономики» (НИУ ВШЭ), Россия.

1. Introduction

Petri nets are widely used to formally represent the behavior of distributed systems for their precise semantics, which helps to prove many crucial behavioral properties, including boundedness, deadlock-freeness, covering by place invariants, and others [1]. The automated verification of these properties is supported by different algorithms. For instance, covering by place invariants can be decided using linear algebraic techniques [2].

The software implementation of algorithms operating with Petri nets naturally requires the preparation of model sets that exhibit the specific structural and behavioral properties. Such sets of models are then used for the thorough evaluation of algorithms under development. Firstly, the manual generation of Petri nets with specific properties is a time-consuming activity. Moreover, if one has to prepare a particularly large-scale model, then there arises an additional task to verify the necessary properties of this model. The computational cost of such a verification can grow too fast due to the well-known *state-explosion* problem of distributed systems, when the number of reachable states grows exponentially compared to the size of a system model.

In our paper, we propose an approach to the generation of Petri nets based on structural transformations. Firstly, a *reference* Petri net is constructed. This model has all the target structural and behavioral properties. Secondly, applying a collection of *local* transformations that extend the internal structure of a reference Petri net, we obtain a *refinement* exhibiting the same properties as an initial reference model. The general scheme of this approach is schematically shown in fig. 1, where a refinement is a result of applying *k* local transformations to a reference Petri net. Note that a refinement has the more sophisticated structure than a reference model. Transformations, considered in our study, are called *local*, since they change only the specific part of a model, while the rest of the model remains untouched. The mathematical framework of these transformations is responsible for preserving the structural and behavioral properties of a reference Petri net. In addition, the application of transformations requires only the local checks of structural constraints.

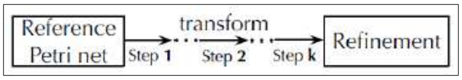


Fig. 1. Step-wise generation of a Petri net

We consider two generation schemes: *fixed* and *randomized*. Within the fixed generation of Petri nets, a specific sequence of transformations is applied to an initial reference model. Conversely, the randomized generation is a based on a non-deterministic choice of transformations.

Thus, the main *results* of our paper are as follows:

the algorithms for the fixed and randomized generation of Petri nets from a given reference model;

the software implementation and evaluation of these algorithms within the environment of the *Carassius* Petri net editor [3].

The remainder of this paper is structured as follows. The next section discusses the related research. In Section 3, we define a class of Petri nets considered in our paper. Section 4 describes the mathematical framework behind a collection of structural transformations that are used to refine

Petri nets. The algorithms for the fixed and randomized Petri net generation are presented in Section 5. In Section 6, we describe a software implementation as well as evaluation of these algorithms, and Section 7 concludes the paper.

2. Related Work

Process Log Generator PLG2 [4] is a well-known software used for the random generation of process models. It supports different notations, including Petri nets and Business Process Model and Notation (BPMN). As shown in [5], the specific classes of BPMN models correspond to Petri nets and vice versa. PLG2 generates process models based on randomly generated context-free grammars and parameters such as the maximum model size, the frequencies of standard behavioral patterns, and others. Compared to our approach, PLG2 offers only the fully randomized model generation and guarantees the behavioral correctness of constructed models. However, within our approach, a reference model may have, for instance, deadlocks, which will be preserved in its refinement.

The generation of BPMN process models has also been considered in [6]. The authors of this approach allow specifying the parameters such as the size of models, the frequencies of behavioral patterns, the types of activities. Similar to our approach, they have also used a collection of initial BPMN models to generate a set of synthetic models.

PTandLogGenerator [7] is another tool supporting the randomized generation of process models. It produces so-called process trees, which specify relations among process activities, for example, sequential, alternative, or concurrent. Process trees can be converted to Petri nets. The prime objective of PTandLogGenerator and the previously mentioned PLG2 is to simulate the behavior of randomly generated process models.

An approach to the generation of *benchmarks*, using random step-wise Petri net refinements, has been presented in [8]. Within this approach, the authors have also defined a set of refinement transformations similar to those used in our study. Based on the proposed transformations, different Petri net classes have been identified and studied. It has been shown what transformations can be used to generate all Petri nets representing a given class.

Structural *transformations* of Petri nets have been first studied in the works [9-11], describing simple yet powerful reduction and extension transformations, s.t. liveness, boundedness, home states, and other behavioral properties are preserved.

Morphisms on Petri nets provide a formal and natural framework to express structural property-preserving relations between Petri nets [12-14]. Using morphisms, one can consider more sophisticated problems of property preservation, including, for instance, bisimulations between Petri nets, as discussed in [13]. For *elementary net systems* [15] – a fundamental class of Petri nets also considered in our paper – *α-morphisms* have been introduced in [16]. They help to formalize structural relations between abstract models and their *refinements*. Concerning our approach to the Petri net generation, a reference Petri net represents an abstract model. In addition, α-morphisms preserve the behavioral properties (reachable markings) as well as reflect them under the specific local requirements.

Since the direct application of α-morphisms is rather difficult for the sophisticated constraints to be checked, a collection of *local* transformations proposed in [17] can be used to define α-morphisms systematically in a step-by-step way. These transformations are used in our study to generate Petri nets, which preserve the properties of an initial reference model. Correspondingly, the mathematical framework behind transformations, which provide the property preservation, is based on α-morphisms.

The existing open-source Petri net editors, among the others, include *Platform Independent Petri Net Editor* [18, 19], *PNEditor* [20], *WoPeD* [21, 22], *Wolfgang* [23], *Carassius* [3]. They allow modeling, simulating and analyzing the behavior of Petri nets. The problem of the model generation has not been considered within these editors. In our study, we will extend the functionality of the

Carassius Petri net editor to provide the generation of Petri nets with the desired structural and behavioral properties.

3. Elementary Net Systems

In our study, we consider the generation of *elementary net systems* (EN-systems). They form the fundamental class of Petri nets used to model the *control-flow* of distributed systems, while other aspects such as data and time are not considered. The structure of EN-systems is modeled using bipartite graphs with two kinds of nodes: *places* and *transitions*. Places in an EN-system can carry at most a single *token*. Thus, they can be interpreted as boolean conditions, truth values of those are changed by transition *firings*. Below we provide the formal definitions based on [15] concerning the structural and behavioral aspects of EN-systems.

Let S be a set. The set of all finite non-empty sequences over S is denoted by S^+, and $S^* = S^+ \cup \{\varepsilon\}$, where ε is the empty sequence.

Definition 1 (Net): A *net* is a triple $N = (P, T, F)$, where P and T are two disjoint sets of places and transitions, and $F \subseteq (P \times T) \cup (T \times P)$ is flow relation. For any node $x \in P \cup T$:

1) $\bullet x = \{y \in X \mid (y, x) \in F\}$ is the *preset* of x.
2) $x \bullet = \{y \in X \mid (x, y) \in F\}$ is the *postset* of x.
3) $\bullet x \bullet = \bullet x \cup x \bullet$ is the *neighborhood* of $x \in X$.

The standard graphical notation is adopted: places are shown with *circles*, and transitions are shown with *boxes*.

In our work, we consider nets without self-loops, i.e., $\forall\, x \in P \cup T$: $\bullet x \cap x \bullet = \emptyset$ and isolated transitions, i.e., $\forall\, t \in T$: $|\bullet t| \geq 1$ and $|t \bullet| \geq 1$.

The $\bullet$-notation can also be extended to subsets of nodes. $N = (P, T, F)$ be a net, and $Y \subseteq (P \cup T)$. Then $\bullet Y = \bigcup_{y \in Y} \bullet y$, $Y \bullet = \bigcup_{y \in Y} y \bullet$ and $\bullet Y \bullet = \bullet Y \cup Y \bullet$. $N(Y)$ denotes the subnet of N *generated* by Y, i.e., $N(Y) = (P \cap Y, T \cap Y, F \cap (Y \times Y))$.

A *marking* (state) m in a net $N = (P, T, F)$ is a subset of its places, i.e., $m \subseteq P$. Pictorially, markings are depicted by placing black dots inside corresponding places. A marking m in a net $N = (P, T, F)$ has a *contact* if $\exists t \in T$: $\bullet m$ and $m \cap t \bullet \neq \emptyset$.

Definition 2 (EN-system): An *elementary net system* (EN-system) is a couple (N, m_0), where $N = (P, T, F)$ is a net, and $m_0 \subseteq P$ is the *initial* marking.

The behavior of EN-system is defined by the *firing rule*. A marking m in a net $N = (P, T, F)$ *enables* transition $t \in T$, denoted $m[t\rangle$, iff $\bullet t \subseteq m$ and $m \cap t \bullet = \emptyset$. Enabled transitions may *fire*. Firing t at m evolves N to a new marking $m' = (m \setminus t \bullet) \cup t \bullet$, denoted $m[t\rangle m'$.

A sequence $w \in T^*$ is a *firing sequence* in an EN-system $N = (P, T, F, m_0)$ if $w = t_1 t_2 \ldots t_n$ and $m_0[t_1\rangle m_1[t_2\rangle \ldots m_{n-1}[t_n\rangle m_n$. Then we write $m[w\rangle m_n$. The set of all firing sequence in N is denoted by $FS(N)$.

A marking m in $N = (P, T, F, m_0)$ is *reachable* exists $\exists w \in FS(N)$: $m_0[w\rangle m$. The set of all markings reachable from m will be denoted $[m\rangle$.

A reachable marking $m \in [m_0\rangle$ in $N = (P, T, F, m_0)$ is a *deadlock* iff it does not enable any transitions. An EN-system is deadlock-free iff there are no reachable deadlocks.

A *state machine* is a connected net $N = (P, T, F)$, where $\forall t \in T : |\bullet t| = |t \bullet| = 1$. A subnet of an EN-system $N = (P, T, F, m_0)$ generated by $Y \subseteq P$ and $\bullet Y \bullet$, i.e., $N(Y \cup \bullet Y \bullet)$ is a *sequential component* of N if it it is a state machine and has a single token in the initial marking. N is covered by sequential components if every place belongs to at least one sequential component. In this case, N is state machine decomposable (SMD). Reachable markings in SMD-EN systems are free from contacts.

State machine decomposability is a basic feature bridging the structural and behavioral properties of EN-systems [15]. The example shown in fig. 2 provides an SMD-EN system with three sequential components: *A* (dotted line), *B* (dashed line), and *C* (dash-dotted line). Sequential components *A*, *B*, *C* have independent parts (concurrent behavior) and synchronous transitions, e.g., transition t_4, which will be executed by *A* and *B* simultaneously. Each token of a reachable marking in an SMD-EN system can be characterized by sequential components. For instance, a token in p_7, shown in fig. 2, belongs to two of three sequential components: *A* and *B*.

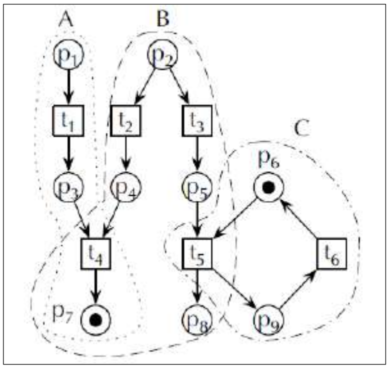


Fig. 2. SMD-EN system with three sequential components

Further, we work with SMD-EN-systems unless otherwise stated explicitly. Thus, we omit the SMD abbreviation in their descriptions.

4. Refinement of EN-systems

In this section, we discuss the mathematical framework behind our approach to the generation of EN-systems using refinement transformations. Firstly, we consider α-morphisms formalizing relations between abstract and refined EN-systems [16]. Secondly, we describe a set of local EN-system transformations that induce corresponding α-morphisms and define them in a step-wise manner [17].

4.1 Morphisms

A class of α-morphisms has been introduced in [16] to formalize relations between an abstract EN-system and its refinement, where subnets in a refined model can substitute places in an abstract model. Using the example shown in fig. 3, we briefly discuss the main intuition behind α-morphisms.

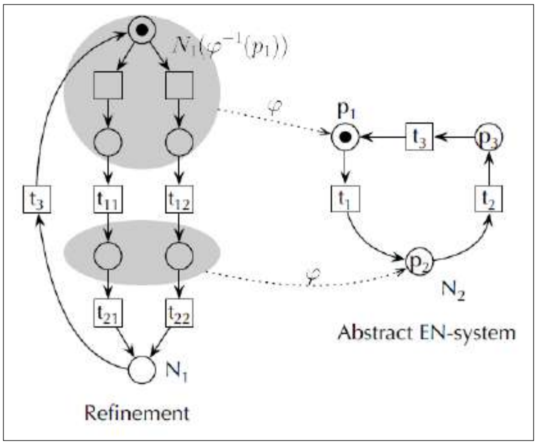


Fig. 3. The α-morphism $\varphi: N_1 \to N_2$

An α-morphism $\varphi: N_1 \to N_2$ is a total surjective map from the set of nodes of a refined EN-system N_1 on the set of nodes of an abstract EN-system N_2. Places in an abstract EN-system can be refined with \emph{acyclic} subnets in its refinement. For example, subnet N_1 ($\varphi^{-1}(p_1)$) refines place p_1 in N_2 shown in fig. 3. The refinement of places can also result in a split of transitions, e.g., transition $t_1$in N_2 is split into two transitions, t_{11} and t_{12}, in N_1, as shown in fig. 3.

An α-morphism $\varphi: N_1 \to N_2$ is defined in terms of how transitions in N_1 are mapped to nodes in N_2. If the image of transition in N_1 is also a transition in N_2, then their neighborhoods should correspond as well. For instance, since the image of transition t_{11} in N_1 shown in fig. 3 is transition t_1 in N_1, the image of $\bullet\, t_{11} \bullet$ is $\bullet\, t_1 \bullet$. If the image of transition in N_1 is a place in N_2, then the image of its neighborhood is this place as well. For instance, transitions in subnet N_1 ($\varphi^{-1}(p_1)$) are mapped to place p_1 inN_2 as well as their neighborhoods.

These constraints combined with several other structural restrictions imposed on subnets in a refined EN-system, discussed in detail in [16], assure the main motivation behind α-morphisms: a refinement should behave «in a similar way» as an abstract model does. Whenever there is a token in a place in abstract EN-system, there exists the possibility to fire a transition that puts a token into a corresponding subnet in a refined EN-system, s.t. the other input transitions remain disabled afterwards (see Lemma 1 in [16]).

The direct application of α-morphisms is rather difficult for their sophisticated structural constraints. An approach based on the subsequent application of local structural transformations [17] comes to the aid of this problem. It is discussed in the following section, where we redefine the refinement notion through these transformations.

4.2 Refinement Transformations

The main idea of structural transformations, defined in [17], lies in a step-by-step construction of a refined model from an abstract one. These transformations are called *local* because they change only a specific subnet in an initial model, while the rest of the model remains untouched.

As shown in [17], every step of applying a transformation to an EN-system induces a corresponding α-morphism from a transformed model to an initial one. Moreover, after a series of transformations is applied to an EN-system, there will be an α-morphism from a result EN-system towards an initial EN-system. Fig. 4 shows the main idea of this approach, where R is a refinement obtained from A by a sequential application of k transformations, s.t. there is an α-morphism $\varphi: R \to A$, and R preserves the behavioral properties of A, especially the presence or absence of deadlocks.

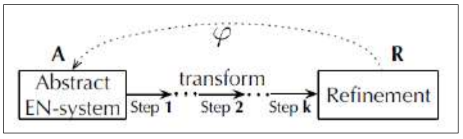


Fig. 4. Refinement based on transformations and α-morphisms

Structural transformations help us to reconsider the notion of a refinement without referring to the formal definition of α-morphisms. In addition, within the framework of our approach to the Petri net generation, transformations play a crucial role. A reference model (see fig. 1) is an abstract EN-system, and its refinement is a result of applying transformations.

We next briefly consider the key aspects of refinement transformations, described in [17].

A *transformation* is a tuple $\rho = (L, R, c_L, c_R)$, where:

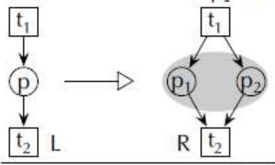
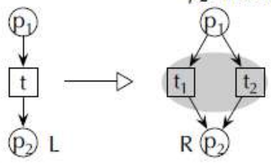
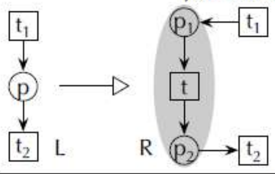
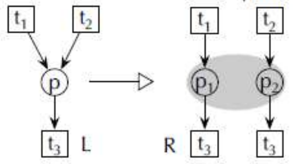
1) L is the *left* part – a subnet to be transformed.
2) R is the *right* part – a subnet replacing L.
3) c_L – constraints imposed on L.
4) c_R – constraints imposed on R.

Constraints c_L and c_R are structural and marking restrictions. They are responsible for corresponding α-morphisms.

The application of a transformation ρ to an EN-system N involves (1) finding a match for L in N according to c_L, i.e., subnet $N(X_L)$ with $X_L \subseteq P \cup T$ and (2) replacing $N(X_L)$ with R according to c_R. The result of applying ρ to N is denoted by $\rho(N, X_L)$. We write $N \overset{\rho}{\to} N'$ if $N' = \rho(N, X_L)$ and the specification of an affected subnet is not important.

The set of four refinement transformations $RT = \{\rho_1, \rho_2, \rho_3, \rho_4\}$ is described in Table 1, where we provide their constraints as well. Intuitively, ρ_1 adds concurrency, ρ_2 and ρ_4 introduce and extend choices, while $\rho_3$adds a new transition into an initial model.

Table 1. Refinement transformations

Transformation	Constraints
ρ_1: Place duplication	
	1. $^\bullet p_1 = {}^\bullet p = {}^\bullet p_2$; 2. $p_1{}^\bullet = p^\bullet = p_2{}^\bullet$; 3. ($p_1 \in m_0'$ and $p_2 \in m_0'$) iff $p \in m_0$.
ρ_2: Transition duplication	
	1. $^\bullet t_1 = {}^\bullet t = {}^\bullet t_2$; 2. $t_1{}^\bullet = t^\bullet = t_2{}^\bullet$.
ρ_3: Transition introduction	
	1. $^\bullet t = \{p_1\}$, $t^\bullet = \{p_2\}$; 2. $p_1{}^\bullet = {}^\bullet p_2 = \{t\}$; 3. $^\bullet p_1 = {}^\bullet p$, $p_2{}^\bullet = p^\bullet$; 4. $p_1 \in m_0' \Leftrightarrow p \in m_0$;
ρ_4: Place split	
	1. $^\bullet p_1 \subset {}^\bullet p$, $^\bullet p_2 \subset p$; 2. $^\bullet p_1 \cup {}^\bullet p_2 = {}^\bullet p$; 3. $^\bullet p_1 \cap {}^\bullet p_2 = \varnothing$; 3. $p_1{}^\bullet$, $p_2{}^\bullet$ are two complete copies of $p^\bullet$; 4. $^\bullet(p_i{}^\bullet) \setminus \{p_i\} = {}^\bullet(p^\bullet)$; 5. if $p \in m_0$, then $p_1 \in m_0'$ iff $p \in m_0$;

Then we can define a refinement as an EN-system that is obtained by applying a sequence π ∈ RT^* of refinement transformations to another EN-system, as formally given below.

Definition 3 (Refinement): Let $N_i = (P_1, T_i, F_i, m_0^i)$ be an EN-system with $i = 1, 2$. N_1 is a refinement of N_2 iff there is a sequence of transformations $\langle \rho_1 . \rho_2 \dots \rho_k \rangle \in RT^*$, s.t. $N_2 \overset{\rho_1}{\to} N'_2 \overset{\rho_2}{\to} \dots \overset{\rho_k}{\to} N_1$.

Let us consider the example of applying transformation ρ_3 to place ρ_9 in the EN-system shown in fig. 2. According to Table 1, there are no specific restrictions imposed on a place in the left part of ρ_3. Then, we can replace place ρ_9 with a subnet, corresponding to the right part of ρ_3, as shown in fig. 5. Since ρ_9 is not marked, added places are also not marked.

As proven in [17], refinement transformations do not introduce new deadlocks, unless they are already present in an initial EN-system, i.e., the deadlocks in a transformed EN-system are the inverse images of the deadlocks, present in an initial EN system (under the corresponding α-morphism).

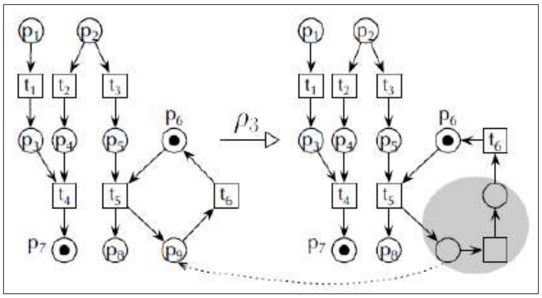


Fig. 5. Application of _3 to the EN-system from fig. 2

Thus, the following proposition holds.

Proposition 1: Let $N_i = (P_1, T_i, F_i, m_0^i)$ be an EN-system with $i = 1, 2$ s.t. N_1 is a refinement of N_2. If N_2 is deadlock-free, then N_1 is deadlock-free as well.

Now we can proceed to the design and implementation of algorithms, which use the set of refinement transformations to generate EN-systems.

5. Generation Algorithms

In this section, we discuss two algorithms that support the automated generation of EN-systems, using the structural transformations, described in Table 1, according to Definition 3. The first algorithm corresponds to the generation of an EN-system by applying a fixed sequence of the refinement transformations to an initial model. The second algorithm corresponds to the randomized EN-system generation, parameterized with the probability of applying each transformation.

5.1 Fixed Generation of an EN-system

Algorithm 1 corresponds to a direct implementation of Definition 3. There is a fixed finite sequence, $\pi = \langle \rho_1 . \rho_2 \dots \rho_n \rangle \in RT^*$, of refinement transformation to be applied to an EN-system $N = (P, T, F, m_0)$.

```
Algorithm 1: Fixed generation
Input: EN-system $N = (P, T, F, m_0)$
       Transformations $RT = \{\rho_1, \rho_2, \rho_3, \rho_4\}$
       Sequence $\pi = \langle \rho_1, \rho_2, ..., \rho_n \rangle \in RT^*$
Output: EN-system $R = (P', T', F', m'_0)$ – a
        refinement of $N$

$R \leftarrow N$
$i \leftarrow 1$
foreach $\rho_i \in \pi$ do
    if $\exists X'_L \in P' \cup T'$ and $\rho_i$ is applicable to subnet
     $R(X'_L)$ in $R$ then
        $R \leftarrow \rho_i(R, X'_L)$
    end
    $i \leftarrow i + 1$
end
```

If a current transformation ρ_i can be applied to some subnet generated by $X_L{}' \in P' \cup T'$, then we replace R with a result of applying ρ to R. If a current transformation ρ_i can be applied to different subnets, the choice is made non-deterministically (it may be determined by the specific implementation of Algorithm 1). Otherwise, if a current transformation ρ_i cannot be applied to a subnet in R, we skip it and pass on to the next transformation in a sequence π.

The correctness of the fixed generation algorithms follows from (a) the finitness of π (the algorithm always terminates) and (b) Proposition 1, i.e., an obtained refinement R preserves the deadlock-freeness of N.

5.2 Randomized Generation of an EN-system

Within the randomized generation algorithm, presented in this paragraph (see Algorithm 2), a sequence of refinement transformations is not known in advance, as opposed to the fixed generation. A specific sequence of refinement transformations is constructed with respect to the parameters defined by a user.

```
Algorithm 2: Randomized generation
Input: EN-system $N = (P, T, F, m_0)$
       Transformations $RT = \{\rho_1, \rho_2, \rho_3, \rho_4\}$
       Probabilities $prob\colon RT \to [0, 1]$, s.t.
       $\forall \rho \in RT\colon \sum freq(\rho) = 1$
       Maximum mumber of nodes $maxSize$
       Maximum number of steps $maxSteps$
Output: EN-system $R = (P', T', F', m'_0)$ – a
        refinement of $N$

$R \leftarrow N$
$totalSteps \leftarrow 0$
while $|P' \cup T'| < maxSize$ OR
 $totalSteps \leq maxSteps$ do
    $AT \leftarrow$ FINDAPPLICABLE($R, RT$)
    $sumProb \leftarrow \sum_{\forall \rho \in AT} prob(\rho)$
    foreach $\rho \in AT$ do
        $prob'(\rho) = \frac{prob(\rho)}{sumProb}$
    end
    order $AT$ in the descending order of $prob'$;
    $r \leftarrow$ RANDOMNUMBER(0, 1)
    $cumulProb \leftarrow 0$
    $i \leftarrow 1$
    while $cumulProb < r$ do
        $cumulProb \leftarrow cumulProb + prob'(\rho_i)$
        $i \leftarrow i + 1$
    end
    $R \leftarrow \rho_i(R, X'_L)$
    $totalSteps \leftarrow totalSteps + 1$
end
```

The randomized generation parameters include:

1) The *maximum size* of a refinement – the number of places and transitions;
2) The *maximum number* of steps – the number of applied transformations;
3) The *probability* of choosing a specific refinement transformation – the value in the interval [0, 1].

Probabilities are set for the four refinement transformations, s.t. the sum of all four probabilities is equal to 1. Below we describe how the specific refinement transformation is chosen at each step of Algorithm 2.

Firstly, we find a set of refinement transformations AT that can be applied to a given EN-system (function $\text{findApplicable}(R, RT)$), according to constraints given in Table 1. Secondly, we normalize the probabilities of the applicable transformations in AT and obtain the values of $prob'$ function. Then, by generating a random number r, we choose the specific refinement transformation ρ_i. Intuitively, we divide the interval [0, 1] into sub-intervals, according to the normalized probabilities of applicable refinement transformations, and check where the value of r is. We assume to use the uniform distribution for the random number generation.

The correctness of Algorithm 2 follows from the fact that (a) the total number of steps (the actual length of an applied transformation sequence) is bounded by the maximum size of a refinement and by the maximum number of steps that can be done; and from (b) Proposition 1, i.e., a constructed refinement preserves the deadlock-freeness of N.

5.3 Example: the Fixed Refinement of an EN-System with a Deadlock

Here we consider an example of applying the fixed generation algorithm to the EN-system that has a deadlock (see fig. 6, where N has the deadlock $\{p_2\}$ reachable from its initial marking $\{p_1, p_2\}$). Let $\pi = \langle \rho_4\ \rho_3\rho_1\rho_3\rho_4\rho_4\ \rho_2\ \rangle \in RT^*$ be a sequence of refinement transformation to be applied to N.

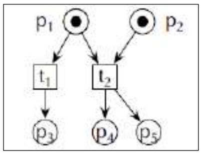


Fig. 6. EN-system with a deadlock

A possible result of applying π to N is provided in fig. 7, where we show transformations affecting disjoint subnets as a single step. It can be seen that none of the ρ_4-elements in π have been applied to N, since there are no places with two or more input transitions. That is why they have been skipped in this example.

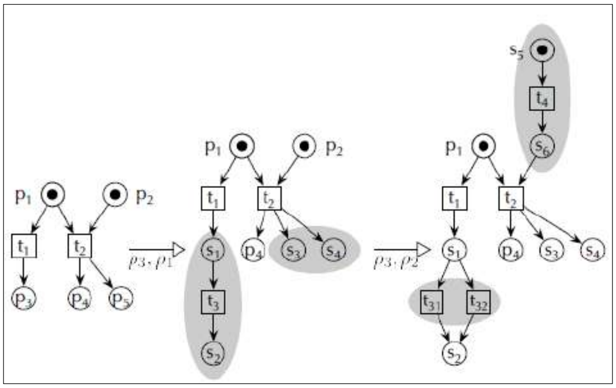


Fig. 7. A result of applying π to the EN-system from Fig. 6

What is more important is that the reachable deadlock $\{p_2\}$ have not been lost in the transformed EN-system. The inverse image of $\{p_2\}$ (under the corresponding α-morphism, refer to fig. 4) in the transformed EN-system is also the reachable deadlock $\{s_6\}$, as formally proven in [17]. New deadlocks have not been introduced into the transformed EN-system.

5.4 Example: a Step in the Randomized Refinement of a Deadlock-Free EN-System

In this paragraph, we consider a step of Algorithm 2 in more detail. Given the EN-system N shown in fig. 8 and the following probabilities: $prob(\rho_1) = 0.15$, $prob(\rho_2) = 0.10$, $prob(\rho_3) = 0.05$, $prob(\rho_4) = 0.7$, we will show how the choice of a refinement transformation is performed.

We start with finding the applicable transformations. Here we have that only ρ_1, ρ_2, and ρ_3 can be applied to the EN-system from fig. 8. Their normalized probabilities are: $prob(\rho_1) = 0.50$, $prob(\rho_3) = 0.33$, and $prob(\rho_3) = 0.17$.

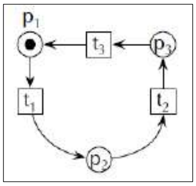


Fig. 8. Deadlock-free EN-system

Then we generate a random number r. Let $r = 0.73$. We check where the value of r is in the interval [0,1$ concerning the cumulative normalized probabilities (see fig. 9).

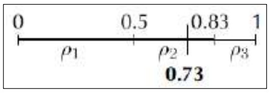


Fig. 9. Checking the placement of the random number r

The value of r is in the interval [0.5, 0.83], corresponding to the refinement transformation ρ_2. Thus, we apply this transformation to the EN-system from fig. 8, and a possible result is shown in fig. 10, if we choose transition t_2 to be transformed.

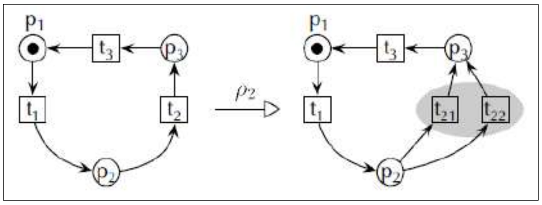


Fig. 10. Applying the chosen transformation to the EN-system from fig. 8

Then, according to Algorithm 2, we continue choosing refinement transformations, according to their probabilities, until we reach either the limit of the size or the limit of the total number of applied transformations.

6. Software Implementation and Evaluation

In this section, we describe details concerning the implementation of the two generation algorithms discussed in the previous section. We have evaluated the randomized generation algorithm using Petri net models for interaction patterns described in [24] as reference models. They provide a highly abstract view of typical asynchronous agent interactions, whereas a refinement of an interface pattern can be seen as the model of a specific system implementing this pattern.

6.1 Carassius Petri Net Editor

The *Carassius* software tool has been presented in [3]. It supports various modeling notations, including (communicating) finite state machines and Petri nets. The Carassius allows one to simulate Petri nets according to the transition firing rule, import and export files in different formats, visualize process behavior. Apart from that, the Carassius has a modular architecture, and it can be easily extended with new features. For example, in [25], the authors have described an extension to the Carassius that supports the simulation of Petri nets with two special types of arcs: reset and inhibitor. The main window of the editor is shown in fig. 11.

We have introduced the following features into the Carassius Petri net editor:

1) the internal storage of refinement transformations;
2) the choice and application of a single transformation to a given EN-system;
3) the generation of an EN-system by applying a fixed transformation sequence (Algorithm 1);
4) the generation of an EN-system by applying a randomly constructed transformation sequence (Algorithm 2).

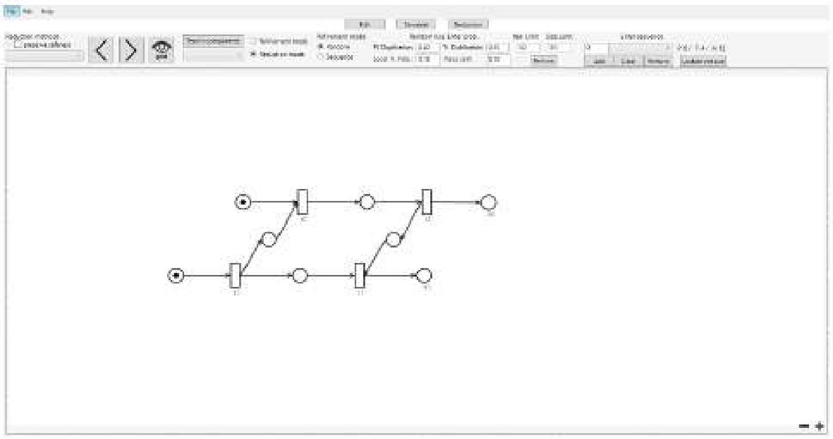
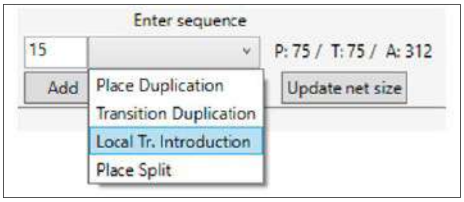


Fig. 11. Carassius process model editor

The implementation of the generation algorithms has also been enriched with the possibility to «roll back» following a transformation sequence to check intermediate results.

The parameters necessary for the fixed and randomized generation are configured in the top panel. A fixed transformation sequence (Algorithm 1 is constructed using a drop-down menu, where one may choose a transformation and assign the corresponding number of occurrences to it (see fig. 12). The configuration of probabilities and other parameters of Algorithm 2 is shown in fig. 13.



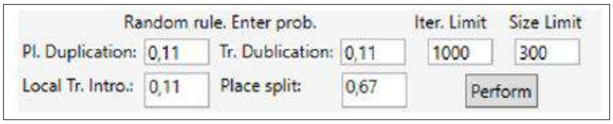
Enter sequence

15 | P: 75 / T: 75 / A: 312

Add | Update net size

Place Duplication
Transition Duplication
Local Tr. Introduction
Place Split

Fig. 12. Constructing a sequence of transformations



Random rule. Enter prob. | Iter. Limit | Size Limit

Pl. Duplication: 0,11 | Tr. Dublication: 0,11 | 1000 | 300

Local Tr. Intro.: 0,11 | Place split: 0,67 | Perform

Fig. 13. Parameters of the randomized generation

As described in the following paragraph, we have considered the application of Algorithm 2 to construct refinements of so-called *interface patterns*.

6.2 Evaluation: Randomized Refinement of Interaction Patterns

Modeling complex information systems is a rather difficult task due to the coordination of several interacting components. *Service interaction patterns*, introduced in the Business Process

Management (BPM) community [26], provide generic solutions for designing composite systems with several interacting entities. The patterns give a highly abstract view of component interactions.

The identification of the typical interface patterns and their modeling using Petri nets have been considered in [24], where the seven asynchronous interaction patterns have been discussed. The models of these patterns are shown in fig. 14. For instance, IP-4 describes the simple message exchange, when the first component sends a message to the second one, and the latter sends back an acknowledgment.

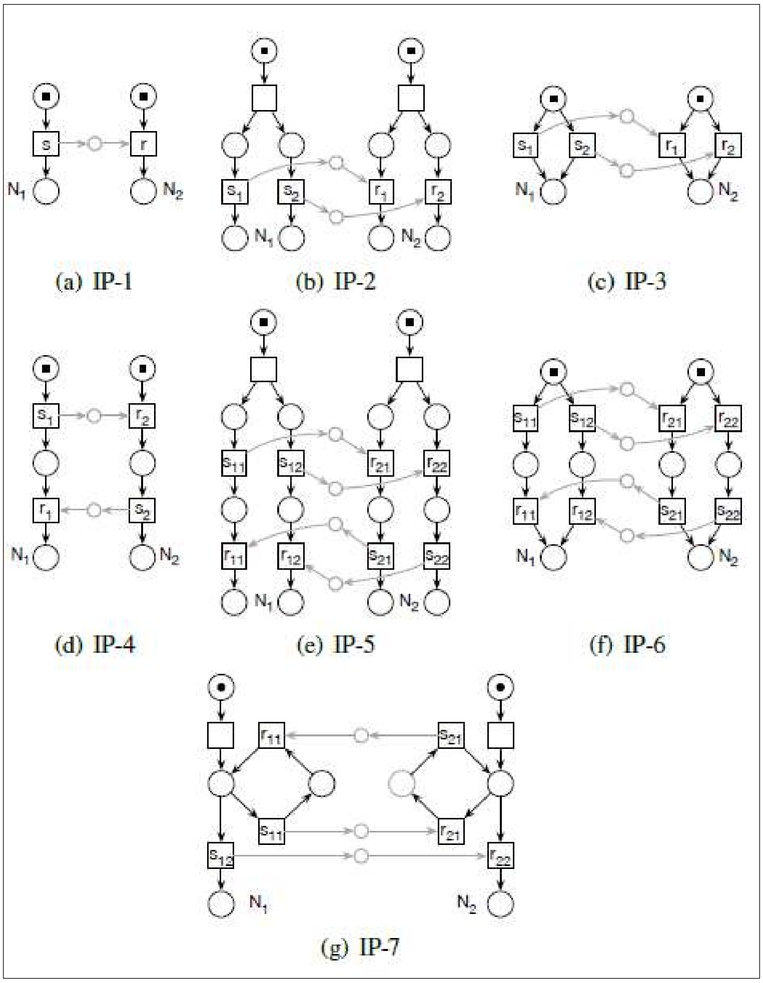


Fig. 14. Interaction patterns: reference models

We have used these interaction patterns to evaluate the randomized generation algorithm. Given an interface pattern, we apply Algorithm 2 and obtain a possible refinement of this pattern. A refinement of an interface pattern inherits its structural and behavioral properties. Intuitively, such a refinement represents a possible system model implementing an interaction pattern.

The results of applying the randomized refinement to the interaction patterns with different parameters are provided in Table 2, where we show the number of places and transitions in the reference model and the obtained refinements. Correspondingly, we have considered five different cases:

- the randomized refinement with equal probabilities for each transformation ($\rho_i = 0.25$);

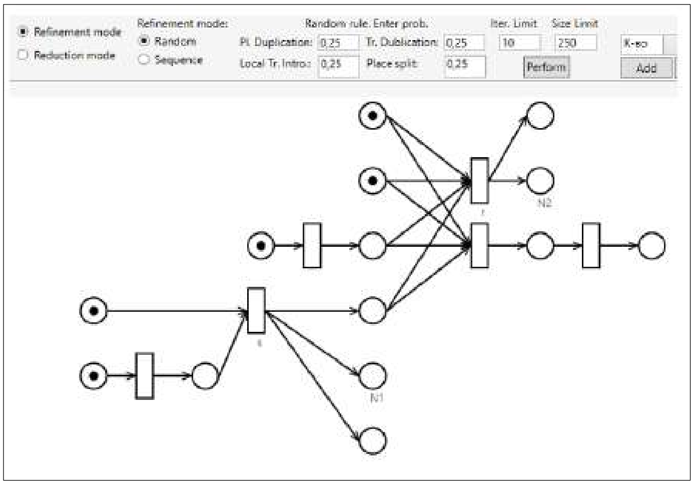
- the four cases when the probability of one transformation (0,67) outweighs the equal probabilities of the other three transformations (0,11).

Table 2. Randomized refinement of interaction patterns

	Reference		**Randomized generation** ($maxSize$=300, $maxSteps$ = 1000)									
			$\rho_i = 0,25$		$\rho_1 = 0,67$		$\rho_2 = 0,67$		$\rho_3 = 0,67$		$\rho_4 = 0,67$	
	$\|P\|$	$\|T\|$	$\|P\|$	$\|T\|$	$\|P\|$	$\|T\|$	$\|P\|$	$\|T\|$	$\|P\|$	$\|T\|$	$\|P\|$	$\|T\|$
IP-1	5	2	134	166	234	66	76	224	156	144	141	166
IP-2	12	6	147	153	216	84	66	256	155	145	146	154
IP-3	6	4	154	149	212	88	85	215	154	147	156	144
IP-4	8	4	132	168	217	83	71	229	152	148	144	156
IP-5	18	10	139	163	207	94	78	222	156	145	157	143
IP-6	12	8	107	193	218	83	72	232	158	142	158	143
IP-7	11	8	140	161	190	110	59	256	143	158	85	215

As it can be seen from Table 2, the number of places and transitions in the constructed refinements is consistent with transformation application probabilities. Within all transformations being equally probable, we do not observe notable differences in the number of places and transitions in the obtained refinements. However, when the place (transition) duplication has the highest probability, we have that the number of places (transitions) significantly outweighs the number of transitions (places) in the refinement. The predominance of the transition introduction (ρ_3) and place split (ρ_4) also does not lead to substantial differences in the number of places and transitions. The application of ρ_4 requires places with two more input transitions, which may not be present in the reference model.

In addition, fig. 15 provides a possible result of applying ten refinement transformations to the interface pattern IP-1, where the transformations have equal probabilities.


(● Refinement mode) ○ Reduction mode

Refinement mode: (● Random) ○ Sequence

Random rule. Enter prob.: Pl. Duplication: 0,25; Tr. Dublication: 0,25; Local Tr. Intro.: 0,25; Place split: 0,25

Iter. Limit: 10; Size Limit: 250

Fig. 15. Refinement of IP-1: 10 steps, equal probabilities

6. Conclusions

In this paper, we have presented an approach to the generation of Petri nets using structural property-preserving transformations. We have considered the generation of elementary net systems, which form the basic class of Petri nets. Elementary net systems reflect the *control-flow* of a process, while data and time aspects are ignored. Given a reference model, we apply a sequence of refinement transformations to obtain a Petri net with similar structural and behavioral properties valid for the reference Petri net. Refinement transformations extend a reference model by adding new places and transitions, i.e., make the structure of a reference model more sophisticated. The proposed approach can be applied for a complex evaluation of algorithms operating with Petri nets requiring the preparation of model sets containing Petri nets with the specific structural and behavioral properties.

The correctness of applying these transformations is based on two observations. Firstly, the transformations induce morphisms between reference and transformed Petri nets. Secondly, the transformations do not introduce new deadlocks, unless they are already present in reference models.

We have designed two algorithms supporting the automated generation of Petri nets with the help of structural property-preserving transformations. The *fixed* generation corresponds to the direct application of a fixed sequence of refinement transformations. Within the *randomized* generation, a user chooses the maximum size of a target model and sets the probability of applying each transformation. We have conducted a series of experiments to evaluate the developed algorithms using Petri net models of service interaction patterns. The experimental results confirm the consistency of the randomized generation algorithm, according to changes in the number of places and transitions with respect to probability values. These algorithms have also been implemented in the existing *Carassius* Petri net editor.

The main limitation to the proposed approach, based on transformations, is that it is impossible to generate a cyclic Petri net from a reference model without cycles. In the future, we plan to relax these constraints and to extend the collection of property-preserving transformations correspondingly. In this light, we also plan to develop a «designer» of structural Petri net transformations that will allow us to construct new transformations. Another direction for the future research is the development of property-preserving transformations for different extensions of Petri nets, including, e.g., colored Petri nets, where tokens can carry data, or timed Petri nets, where transitions are assigned firing time intervals. Note that certain extensions of Petri nets can also be «unfolded» to elementary net systems.

References

[1] W. Reisig. Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies. Springer, 2013, 257 p.

[2] J. Desel. Basic linear algebraic techniques for place/transition nets. Lecture Notes in Computer Science, vol. 1491. Springer, 1998, pp. 257-308.

[3] N. Nikitina, A. Mitsyuk. Carassius: A Simple Process Model Editor. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 3, 2015, pp.219-236. DOI: 10.15514/ISPRAS-2015-27(3)-15.

[4] A. Burattin. Multiperspective process randomization with online and offline simulations. In Proc. of the BPM Demo Track 2016, CEUR Workshop Proceedings, vol. 1789, 2016, pp. 1-6.

[5] A. Kalenkova, W. van der Aalst, I. Lomazova, and V. Rubin. Process mining using BPMN: relating event logs and process models. Software & Systems Modeling, vol. 16, 2017, pp. 1019-1048.

[6] Z. Yan, R. Dijkman, and P. Grefen. Generating process model collections. Software & System Modeling, vol. 16, 2017, pp. 979-995.

[7] T. Jouck and B. Depaire. PTandLogGenerator: A generator for artificial event data. In Proc. of the BPM Demo Track 2016, CEUR Workshop Proceedings, vol. 1789, 2016, pp. 23-27.

[8] K. van Hee and Z. Liu. Generating benchmarks by random stepwise refinement of petri nets. In Proc. of the Workshops of the 31st International Conference on Application and Theory of Petri Nets and Other Models of Concurrency (PETRI NETS 2010) and of the 10th International Conference on Application of Concurrency to System Design (ACSD 2010), CEUR Workshop Proceedings, vol. 827, 2010, pp. 403-417.

[9] G. Berthelor. Checking properties of nets using transformations. Lecture Notes in Computer Science, vol. 222, 1986, pp. 19-40.

[10] T. Murata. Petri nets: Properties, analysis and applications. Proceedings of the IEEE, vol. 77, no. 4, 1989, pp. 541-580.

[11] T. Murata and I. Suzuki. A method for stepwise refinement and abstraction of Petri nets. Journal of Computer and System Sciences, vol. 27, issue 1, 1983, pp. 51-76.

[12] [12] G. Winskel. Petri nets, algebras, morphisms, and compositionality. Information and Computation, vol. 72, no. 3, 1987, pp. 197-238.

[13] G. Winskel and M. Nielsen. Petri nets and bisimulations. Theoretical Computer Science, vol. 153, pp. 211-

244, 1996.

[14] J. Desel and A. Merceron. Vicinity respecting homomorphisms for abstracting system requirements. Lecture Notes in Computer, vol. 6550, 2010, pp. 1-20.

[15] L. Bernardinello and F. De Cindio. A survey of basic net models and modular net classes. Lecture Notes in Computer Science, vol. 609, 1992, pp. 304-351.

[16] L. Bernardinello, E. Mangioni, and L. Pomello. Local state refinement and composition of elementary net systems: An approach based on morphisms, Lecture Notes in Computer Science, vol. 8100, 2013, pp. 48-70.

[17] L. Bernardinello, I. Lomazova, R. Nesterov, and L. Pomello. Property preserving transformations of elementary net systems based on morphisms. In Proc. of the International Workshop on Petri Nets and Software Engineering, CEUR Workshop Proceedings, vol. 2651, 2020, pp. 49-67.

[18] N. Dingle, W. Knottenbelt, and T. Suto. PIPE2: A tool for the performance evaluation of generalised stochastic petri nets. ACM SIGMETRICS Performance Evaluation Review, vol. 36, no. 4, 2009, pp. 34-39.

[19] Platform Independent Petri Net Editor. Available at https://github.com/sarahtattersall/PIPE, accessed: 2021-03-20.

[20] PNEditor (Petri Net Editor). Available at https://github.com/matmas/pneditor, accessed: 2021-03-20.

[21] T. Freytag and M. Sänger. WoPeD - An Educational Tool for Workflow Nets. in Proc. of the BPM Demo Sessions, ser. CEUR Workshop Proceedings, vol. 1295. CEUR-WS.org, 2014, pp. 31-36.

[22] WoPeD (Workflow Petri Net Designer. Available at https://woped.dhbwkarlsruhe.de/, accessed: 2021-03-20.

[23] WOLFGANG - Petri Net Editor. Available at https://github.com/iig-unifreiburg/WOLFGANG, accessed: 2021-03-20.

[24] R. Nesterov and I. lomazova. Asynchronous interaction patterns for mining multi-agent system models from event logs. Proceedings of the MACSPro Workshop 2019, CEUR Workshop Proceedings, vol. 2478, 2019, pp. 62-73.

[25] P.A. Pertsukhov, A.A. Mitsyuk. Simulating Petri Nets with Inhibitor and Reset Arcs. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 4, 2019. pp. 151-162. DOI: 10.15514/ISPRAS-2019-31(4)-1.

[26] A. Barros, M. Dumas, and A. ter Hofstede. Service interaction patterns. Lecture Notes in Computer Science, vol. 3649, 2005, pp. 302-318.

Информация об авторах / Information about authors

Роман Александрович НЕСТЕРОВ – аспирант факультета компьютерных наук НИУ ВШЭ и департамента информатики, системной инженерии и коммуникаций Миланского университета Бикокка (UNIMIB), младший научный сотрудник лаборатории ПОИС НИУ ВШЭ. Область его научных интересов составляют теория параллелизма, сети Петри, формальные методы для моделирования мультиагентных информационных систем.

Roman NESTEROV is a postgraduate student at the Faculty of computer science, HSE University and at the Department of Informatics, System Engineering and Communications, University of Milano-Bicocca, Italy (UNIMIB), a junior research fellow at the Laboratory of Process-Aware Information Systems (PAIS Lab), HSE University. His research interests are the theory of concurrency, Petri nets and formal methods for modeling multi-agent information systems.

Семен Юрьевич САВЕЛЬЕВ – студент четвертого курса бакалавриата факультета компьютерных наук НИУ ВШЭ, стажер-исследователь лаборатории ПОИС НИУ ВШЭ. Область научных интересов составляют: объектно-ориентированное программирование, архитектура информационных систем, моделирование и анализ поведения процессов с помощью сетей Петри и других формализмов.

Semyon SAVELYEV is a fourth-year undergraduate student at the Faculty of computer science, HSE University, a research assistant at the PAIS Lab, HSE University. His research interests include object-oriented programming, architecture of information systems, modeling and analyzing process behavior using Petri nets and other formalisms.

DOI: 10.15514/ISPRAS–2021–33(3)–13

Аналитика в реальном времени, гибридная транзакционная/аналитическая обработка, управление данными в основной памяти и энергонезависимая память

[1,2,3,4,5] С.Д. Кузнецов, ORCID: 0000-0002-8257-028X <kuzloc@ispras.ru>
[6] П.Е. Велихов, ORCID: 0000-0002-0644-8047 <pavel.velikhov@huawei.com>
[6] Ц. Фу, ORCID: 0000-0003-0244-1718 <fqiang.fuqiang@huawei.com>

[1] Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, Москва, ул. А. Солженицына, д. 25
[2] Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1
[3] Московский физико-технический институт, 141700, Россия, Московская область, г. Долгопрудный, Институтский пер., 9
[4] НИУ «Высшая школа экономики», 101978, Россия, Москва, ул. Мясницкая, д. 20
[5] Российский экономический университет имени Г.В. Плеханова, 117997, Москва, Стремянный пер., 36
[6] Техкомпания Хуавэй. 121614, Россия, Москва, ул. Крылатская, д. 17, к. 2

Аннотация. В наши дни аналитика в реальном времени – одно из наиболее часто используемых понятий в мире баз данных. В широком смысле этот термин означает очень быструю аналитику очень свежих данных. Обычно этот термин используется вместе с другими популярными терминами – *гибридной транзакционной / аналитической обработкой (HTAP)* и *обработкой данных в основной памяти*. Причина в том, что самый простой способ предоставить свежие оперативные данные для анализа – это объединить в одной системе как транзакционную, так и аналитическую обработку. Самый эффективный способ обеспечить быструю транзакционную и аналитическую обработку – хранить всю базу данных в основной памяти. Итак, с одной стороны, эти три термина связаны, но с другой стороны, каждый из них имеет собственное право на жизнь. В этой статье мы даем обзор нескольких систем управления данными в памяти, которые не являются системами HTAP. Некоторые из них являются чисто транзакционными, некоторые – чисто аналитическими, а некоторые поддерживают аналитику в реальном времени. Затем мы рассмотрим девять HTAP-СУБД с хранением баз данных в основной памяти, некоторые из которых не поддерживают аналитику в реальном времени. Существующие HTAP-СУБД реального времени с хранением баз данных в основной памяти имеют очень разнообразную и интересную архитектуру, хотя они используют ряд общих подходов: многоверсионное управление параллелизмом, многоядерное распараллеливание, расширенная оптимизация запросов, своевременная компиляция запросов и т.д. Кроме того, нас интересует, используют ли эти системы энергонезависимую память, и если да, то каким образом. Мы пришли к выводу, что появление нового поколения NVM будет значительно стимулировать использование энергонезависимой основной памяти в системах HTAP с хранением баз данных в основной памяти.

Ключевые слова: аналитика в реальном времени; гибридная транзакционная/аналитическая обработка; обработка данных в основной памяти; энергонезависимая память.

Для цитирования: Кузнецов С.Д., Велихов П.Е., Фу Ц. Аналитика в реальном времени, гибридная транзакционная/аналитическая обработка, управление данными в основной памяти и энергонезависимая память. Труды ИСП РАН, том 33, вып. 3, 2021 г., стр. 171-198. DOI: 10.15514/ISPRAS–2021–33(3)–13

Real-Time Analytics, Hybrid Transactional/Analytical Processing, In-Memory Data Management, and Non-Volatile Memory

[1,2,3,4,5] *S.D. Kuznetsov, ORCID: 0000-0002-8257-028X <kuzloc@ispras.ru>*
[6] *P.E. Velikhov, ORCID: 0000-0002-0644-8047 <pavel.velikhov@huawei.com>*
[6] *Q. Fu, ORCID: 0000-0003-0244-1718 <fqiang.fuqiang@huawei.com>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*
[3] *Moscow Institute of Physics and Technology (State University), 9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia*
[4] *National Research University, Higher School of Economics 20, Myasnitskaya Ulitsa, Moscow, 101978, Russia*
[5] *Plekhanov Russian University of Economics, 36, Stremyanny lane, Moscow, 117997, Russia*
[6] *Huawei Technologies Co., Ltd., 17, building 2, st. Krylatskaya, Moscow, 121614, Russia*

Abstract. These days, *real-time analytics* is one of the most often used notions in the world of databases. Broadly, this term means very fast analytics over very fresh data. Usually the term comes together with other popular terms, *hybrid transactional/analytical processing (HTAP)* and *in-memory data processing*. The reason is that the simplest way to provide fresh operational data for analysis is to combine in one system both transactional and analytical processing. The most effective way to provide fast transactional and analytical processing is to store an entire database in memory. So on the one hand, these three terms are related but on the other hand, each of them has its own right to life. In this paper, we provide an overview of several in-memory data management systems that are not HTAP systems. Some of them are purely transactional, some are purely analytical, and some support real-time analytics. Then we overview nine in-memory HTAP DBMSs, some of which don't support real-time analytics. Existing real-time in-memory HTAP DBMSs have very diverse and interesting architectures although they use a number of common approaches: multiversion concurrency control, multicore parallelization, advanced query optimization, just in time compilation, etc. Additionally, we are interested whether these systems use non-volatile memory, and, if yes, in what manner. We conclude that an emergence of new generation of NVM will greatly stimulate its use in in-memory HTAP systems.

Keywords: real-time analytics; hybrid transactional/analytical processing; in-memory data processing; non-volatile memory

For citation: Kuznetsov S.D., Velikhov P.E., Fu Q. Real-Time Analytics, Hybrid Transactional/Analytical Processing, In-Memory Data Management, and Non-Volatile Memory. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 171-198 (in Russian). DOI: 10.15514/ISPRAS–2021–33(3)–13

1. Введение

Термины и концепции в области управления данными постоянно меняются, и аналитика в реальном времени в настоящее время является одной из самых популярных концепций. В целом, аналитика в реальном времени предполагает *быструю* аналитическую обработку *свежих* данных. Оба ключевых слова в приведенном выше предложении, *быстрая обработка* и *свежие данные*, не имеют абсолютного смысла. Аналитическая обработка в режиме реального времени должна происходить настолько быстро, насколько этого требуют клиенты (предприятия), а данные могут быть настолько свежими, насколько они могут быть предоставлены базовой системой управления данными.

В любом случае, наиболее естественным источником свежих корпоративных данных являются данные, генерируемые транзакциями того же предприятия. Обычно все корпоративные транзакции обрабатываются некоторыми СУБД, ориентированными на OLTP, а корпоративная аналитика поддерживается некоторыми СУБД, ориентированными на OLAP. Таким образом, для обеспечения аналитики в реальном времени необходимо очень быстро передавать данные из хранилища транзакционной системы в хранилище аналитической системы. Другими словами, чтобы обеспечить аналитику свежих транзакционных данных, необходимо предоставить очень быстрый механизм ETL (извлечение-преобразование-загрузка). Но никто не знает, как реализовать такой быстрый ETL, и для того, чтобы сделать аналитику в реальном времени возможной, была введена концепция *гибридной транзакционной/аналитической обработки* (hybrid transactional/analytical processing, HTAP).

В широком смысле *HTAP* означает, что мы доставляем свежие транзакционные данные для аналитической обработки без какого-либо ETL посредством тесной интеграции транзакционных и аналитических хранилищ данных. Чтобы добиться такой интеграции магазинов, необходимо также интегрировать транзакционные и аналитические подсистемы. Легко видеть, что такая интеграция была естественной до начала эры «один размер не подходит для всех» [1], потому что универсальные СУБД полностью поддерживали стандарт SQL и теоретически могли поддерживать смешанные рабочие нагрузки OLTP / OLAP.

Основным преимуществом специализированных СУБД было использование структур данных и алгоритмов, наиболее подходящих для соответствующих видов рабочей нагрузки. Например, специализированные аналитические СУБД, такие как Vertica [2], используют разделенные хранилища данных с поколоночным хранением таблиц и оптимизированы для обработки сложных аналитических запросов с несколькими соединениями, в то время как специализированные транзакционные СУБД, такие как VoltDB [3], поддерживают хранилище данных в памяти на основе строк и являются оптимизированными для обработки коротких и простых транзакций. Безусловно, специализированная транзакционная СУБД выигрывает у универсальной СУБД, когда рабочая нагрузка является транзакционной, и специализированная аналитическая СУБД выигрывает, когда рабочая нагрузка является аналитической.

Поэтому «идеалистическая» цель подхода HTAP – одновременно обеспечить в рамках одной системы функциональность и производительность специализированных OLTP- и OLAP-СУБД – кажется недостижимой. Прагматическая цель этого подхода – создание системы, которая обеспечивает разумную пропускную способность для транзакционных рабочих нагрузок и одновременно аналитику в реальном времени по достаточно свежим данным - несомненно достижима (как мы покажем в этой статье).

Очевидно, что требования разумной пропускной способности для транзакционных рабочих нагрузок и свежести данных для аналитических запросов противоречат друг другу. Чтобы обеспечить максимальную актуальность данных, необходимо сделать доступными для анализа все данные, генерируемые текущими обрабатываемыми транзакциями. Однако в этом случае транзакции будут конкурировать с аналитическими запросами, и обработка транзакций станет медленнее. Все известные системы HTAP используют тот или иной компромисс для разумного (в некоторой степени) удовлетворения этих противоречивых требований путем разделения аналитической и транзакционной частей базы данных и, возможно, преобразования данных из представления, хорошо подходящего для обработки транзакций (обычно таблицы с хранением по строкам) в более подходящее для аналитики представление (обычно поколоночное представление таблиц). (Если хотите, это можно назвать облегченным ETL.)

Другое измерение – это носители для хранения данных. Большинство современных СУБД категории HTAP оптимизированы для хранения данных в основной памяти. Это означает, что все структуры данных и алгоритмы, используемые в таких системах, разработаны в соответствии с предположением, что вся обработка данных будет выполняться в основной памяти без ввода-вывода с использованием внешних запоминающих устройств. Вообще говоря, концепция СУБД с хранением баз данных в основной памяти (in-memory СУБД) не нова. Согласно [4], первая попытка реализовать систему базы данных, которая хранит все данные в памяти, была предпринята IBM в 1976 году (IMS Fast Path). Новая волна in-memory СУБД с оперативной памятью наблюдалась в 1990-х годах. Тогда и позже (в начале 2000-х) такие СУБД в основном специализировались на обработке транзакций. Сейчас почти все существующие HTAP-СУБД в той или иной степени относятся к классу систем in-memory. Хранение всех данных (или, по крайней мере, *горячих* данных) в основной памяти обеспечивает более высокую скорость обработки транзакций, что частично сглаживает негативные эффекты одновременно выполняемых аналитических запросов.

Таким образом, понятия аналитики в реальном времени, гибридной транзакционной/аналитической обработки и управления данными в основной памяти являются взаимосвязанными понятиями, хотя каждое из них имеет свой собственный смысл и право на жизнь. Мы обсудим более подробно их происхождение, собственное значение и взаимосвязь в разд. 2.

Еще одна концепция, последняя, но не менее важная для целей этой статьи, – это *энергонезависимая основная память* (non-volatile main, NVM), которая также называется *постоянной памятью* (persistent memory), *памятью класса хранения данных* (storage-class memory) и т.д. NVM на самом деле является обычной памятью с байтовой адресацией (как традиционная RAM), но сохраняет свое состояние после отключения питания. Эти особенности NVM позволяют использовать ее как одноуровневую среду хранения данных в системах управления данными.

NVM наиболее хорошо подходит для чисто транзакционных СУБД, работающих на многоядерных компьютерах [5]. В этом случае основным преимуществом использования NVM является то, что важная функция долговечности транзакций будет обеспечиваться без какой-либо журнализации во внешней памяти. Таким образом, транзакции, обновляющие базу данных, будут обрабатываться с той же скоростью, что и транзакции только для чтения, а общая пропускная способность для транзакционных рабочих нагрузок будет намного выше, чем у традиционных транзакционных in-memory СУБД (эти системы должны журнализовать все операции обновления в энергонезависимой внешней памяти, чтобы обеспечить долговечность транзакций).

Однако в настоящее время наблюдается тенденция к объединению в одной и той же системе баз данных функций обработки транзакций и аналитики в реальном времени, то есть к использованию подхода HTAP. Поэтому представляется интересным и полезным проанализировать, используют ли разработчики имеющихся в настоящее время in-memory СУБД категории HTAP (или собираются ли они использовать) NVM, и, если да, то как они его используют (или планируют использовать).

Основной вклад статьи этой статьи состоит в следующем:

- предоставляются более или менее точные определения и объяснения связанных, но различных понятий аналитики в реальном времени, гибридной транзакционной / аналитической обработки и управления данными в основной памяти;
- обоснована принципиальная важность энергонезависимой памяти в будущих СУБД категории HTAP;
- кратко описан общий ландшафт существующих систем управления данными в основной памяти;
- дан обзор новейших in-memory HTAP-СУБД и использования в них NVM.

Оставшаяся часть статьи имеет следующую структуру. В разд. 2 предлагается некоторая предыстория, включая определения и объяснения терминов и понятий, используемых в статье. В разд. 3 мы приводим общую классификацию современных in-memory СУБД и кратко характеризуем некоторые системы, не принадлежащие к категории HTAP. В разд. 4 более подробно описаны архитектурные и основные функциональные особенности основных коммерческих и академических разработок in-memory HTAP-СУБД. Для каждой системы мы указываем, используется ли в ней NVM, и, если да, то каким образом. Разд. 5 содержит некоторые заключительные соображения авторов и завершает статью.

2. Предпосылки: определения, объяснения и обсуждение

Имеются три концепции и соответствующие термины, которые обычно объединяются в области баз данных, но каждое из них имеет собственное значение и фундаментальное право на отдельную жизнь. Эти термины таковы:

- аналитика в реальном времени;
- гибридная транзакционная / аналитическая обработка (HTAP); а также
- управление данными в памяти.

Мы начнем с рассмотрения смысла каждого из этих терминов, а затем обсудим, почему они в настоящее время объединяются. Наконец, в этом разделе мы также кратко рассмотрим понятие энергонезависимой памяти, текущее состояние соответствующей технологии и ее связь с упомянутыми связанными концепциями.

2.1 Аналитика в реальном времени

Согласно Глоссарию Gartner [6], «Аналитика в реальном времени – это дисциплина, к которой применяется логика и математика к данным, чтобы обеспечить их понимание для быстрого принятия лучших решений. В некоторых случаях использование реального времени просто означает, что аналитика завершается в течение нескольких секунд или минут после поступления новых данных».

Первая (общая) часть этого определения фактически означает, что аналитические инструменты в реальном времени должны удовлетворять потребности лиц, принимающих бизнес-решения. Кажется, это общая цель любого поставщика, предоставляющего аналитические решения. В частности, для достижения этой цели помогают хранение таблиц по столбцам и различные виды разделения базы данных, используемые в специализированных аналитических СУБД, таких как, например, Teradata [7] или Vertica [2].

Второе утверждение в приведенной выше цитате дает более близкую к нашему пониманию, более конкретную и слегка противоречивую интерпретацию концепции аналитики в реальном времени. На самом деле это означает, что:

- данные должны быть доступны для анализа после их создания *как можно скорее*, и
- анализ данных должен быть завершен *как можно быстрее*.

Во-первых, обратите внимание на наличие двух экземпляров оборота «как можно». Для завершения аналитики существует большой разрыв между несколькими секундами и несколькими минутами. Итак, мы видим совершенно особую интерпретацию смысла реального времени: время, потраченное на аналитику, должно быть настолько реальным, насколько это возможно.

Во-вторых, для обеспечения свежести данных, вероятно, нужно отказаться от всей предварительной подготовки данных к анализу. Таким образом, анализ должен проводиться над данными, представленными в их первоначальном представлении (возможно, с небольшим лексическим преобразованием, например, из строковой формы в форму, основанную на столбцах). Непросто обеспечить быстрое выполнение сложных аналитических запросов к неподготовленным данным.

И, в-третьих, приведенное выше определение ничего не говорит о способах достижения цели предоставления аналитики в реальном времени и не указывает каких-либо точных характеристик «реального времени». С одной стороны, такое неопределенное «определение» облегчает разработку системы, удовлетворяющей «требованиям» аналитики в реальном времени. Но с другой стороны, оно делает практически невозможным сравнение любых двух систем этой категории.

2.2 Гибридная транзакционная/аналитическая обработка

Как говорится в Википедии со ссылкой на Gartner [8], «гибридная транзакционная / аналитическая обработка (HTAP) – это новая архитектура приложений, которая "ломает стену" между обработкой транзакций и аналитикой. Это позволяет принимать более информированные решения в режиме реального времени». В соответствии с этим определением стоит сделать несколько комментариев.

(a) Любая универсальная реляционная СУБД, корректно поддерживающая стандарт SQL, относится к категории HTAP в смысле первой части этого определения. Действительно, стандарт SQL предписывает поддержку транзакций ACID со всеми обычными гарантиями. Следовательно, любая СУБД, поддерживающая стандарт SQL, должна быть способна обрабатывать любую транзакционную рабочую нагрузку. Стандарт SQL предусматривает поддержку произвольно сложных аналитических запросов, а также наличие достаточно богатого и расширяемого набора аналитических операций. Таким образом, любая СУБД, поддерживающая стандарт SQL, потенциально должна быть способна обрабатывать любую аналитическую рабочую нагрузку. В определении ничего не говорится о производительности такой обработки.

(b) Фактическая «стена» между OLTP и OLAP появилась только после появления концепции Майкла Стоунбрейкера «один размер не подходит для всех» [1]. Специализированные транзакционные и аналитические СУБД либо не поддерживают весь стандарт SQL, либо поддерживают транзакционные и аналитические функции SQL с неравной производительностью.

(c) Подход HTAP можно рассматривать как шаг назад от специализированных систем баз данных. Одним из основных пунктов критики Стоунбрейкера универсальных СУБД была их сложность. Специализированные DMBS, естественно, намного проще универсальных, поскольку они предоставляют лишь ограниченную функциональность. Большинство усилий сосредоточено на обеспечении производительности. При разработке HTAP-СУБД нам необходимо одновременно обеспечить гибридную функциональность и сохранить (или хотя бы попытаться сохранить) производительность специализированных систем баз данных.

(d) Подход HTAP в смысле приведенного выше определения может помочь решить только первую часть проблемы аналитики в реальном времени – предоставить аналитический доступ к свежим данным. Высокая производительность аналитической обработки может быть достигнута за счет использования оптимизированной архитектуры СУБД категории HTAP, методов взаимодействия транзакционных и аналитических движков внутри системы, а также оптимизации аналитических запросов.

В завершение этого подраздела приведем еще одну цитату [9], которая отражает точку зрения Хассо Платтнера (Hasso Plattner), соучредителя SAP, основателя института Хассо Платтнера (Hasso-Plattner-Institut für Digital Engineering, HPI) и главного инициатора HTAP-СУБД HANA в SAP. Авторы [9] писали, что «Использование общей модели базы данных для корпоративных систем может значительно улучшить эксплуатационные качества и снизить сложность использования систем, обеспечивая при этом доступ к данным в реальном времени и совершенно новые возможности». Кажется, что главная мысль здесь – *эксплуатационные качества*. Очевидно, что если предприятие может использовать одну гибридную базу данных вместо как минимум двух отдельных транзакционных и аналитических баз данных, это будет намного проще и дешевле.

2.3 Управление данными в памяти

В глоссарии Gartner содержится следующее утверждение о HTAP и управлении данными в памяти: «Архитектура гибридной транзакции/аналитической обработки (HTAP) лучше всего обеспечивается методами и технологиями вычислений в основной памяти (in-memory computing, IMC), позволяющими выполнять аналитическую обработку над тем же хранилищем данных (в основной памяти), которое используется для обработки транзакций».

Вероятно, это правда, и в разд. 4 мы увидим несколько примеров, подтверждающих это утверждение. Однако концепции in-memory и HTAP – это не одно и то же. In-memory-СУБД появились намного раньше, чем понятие HTAP, и были ориентированы на очень быструю обработку транзакций. Очень хороший обзор истории и текущего состояния технологии баз данных в основной памяти представлен в статье [4], написанной руководителями нескольких проектов in-memory СУБД.

Даже сейчас существует несколько чисто транзакционных СУБД в оперативной памяти. Некоторые из них являются успешными коммерческими продуктами, такими как VoltDB [3] и SolidDB [10], а другие представляют собой академические прототипы, например, Silo [11]. Среди in-memory СУБД есть и чисто аналитические системы управления данными, например, традиционная массивно-параллельная аналитическая СУБД Exasol [12], многомерные специализированные системы Cognos TM1 [13] и Essbase [14], а также улучшенная версия Spark-PMoF [15] распределенной аналитической среды Spark. Мы кратко обсудим эти системы в следующем разделе.

Как мы упоминали во введении, хранение базы данных полностью в основной памяти в принципе позволяет системе баз данных HTAP сократить время выполнения как транзакций, так и аналитических запросов. Однако для обеспечения быстрой аналитики над свежими данными необходимо также сократить задержку, с которой транзакционные данные становятся доступными для аналитической обработки. Это можно сделать, например, используя для обработки транзакций не традиционное представление таблицы на основе строк, а представление таблицы в виде столбцов, которое лучше всего подходит для аналитической обработки данных.

Это была одна из основных идей Хассо Платтнера при разработке HANA. Как он писал в [16], «Ранние тесты в SAP и HPI с базами данных в основной памяти реляционного типа, основанными на хранении таблиц по строкам, не показали значительных преимуществ перед ведущими СУБД с эквивалентной памятью для кэширования. Здесь родилась альтернативная идея – изучить преимущества использования для OLTP баз данных с поколоночным хранением».

Они сделали это, и HANA в своем обычном, наиболее оптимизированном режиме может выполнять параллельную обработку обоих видов рабочей нагрузки в in-memory хранилище поколоночных таблиц. Однако стоит отметить, что почти все существующие СУБД категории HTAP (включая HANA) поддерживают хранилища таблиц по столбцам и строкам в одной базе данных. Похоже, что хранилище по строкам по-прежнему необходимо, если требуется максимальная пропускная способность для транзакционных рабочих нагрузок.

Наконец, заметим, что все существующие СУБД в памяти, поддерживающие SQL, имеют ряд общих, очень важных технических характеристик, таких как своевременная компиляция запросов в оптимизированный машинный код, агрессивное сжатие данных, активное использование инструкций SIMD процессора, использование структур данных и алгоритмов без блокировок и защелок, многоверсионное управление параллелизмом и т.д. Однако мы почти не будем касаться этих вопросов далее в этой статье, потому что здесь нас больше всего интересуют общие архитектуры и принципы хранения данных в in-memory HTAP-СУБД.

2.4 Энергонезависимая память и ее правильное место в технологии баз данных

Откровенно говоря, основной интерес авторов этой статьи – возможное включение поддержки новой энергонезависимой памяти (NVM) (также называемой постоянной памятью (PMEM) и памятью класса хранения (SCM)) в архитектуры современных (в основном класса in-memory) СУБД. Мы не будем приводить здесь обзор текущего состояния технологии NVM, который можно найти где-либо еще (например, в [5]). Отметим только со ссылкой на [5], что в настоящее время практически доступен только один вид NVM в форм-факторе DIMM – Intel Optane DIMM. Общие характеристики этого продукта – большая задержка (на несколько двоичных порядков выше, чем у DRAM), большая задержка операций записи, чем чтения, и большая емкость (до 512 Гб на модуль). Электронная промышленность обещает предоставить (надеемся, в ближайшее время) следующие поколения NVM, которые будут иметь характеристики задержки и емкости не хуже, чем у текущих DRAM, и характеристики выносливости не хуже, чем у существующих HDD (или, по крайней мере, SSD).

Как уже упоминалось, лучшее место для NVM в области технологий баз данных – это транзакционная in-memory (фактически, in-NVM) СУБД, работающая на многоядерном сервере. Энергонезависимость NVM позволяет использовать одноуровневую архитектуру хранения данных вообще без использования каких-либо внешних запоминающих устройств. Напротив, текущие транзакционные in-memory СУБД вынуждены использовать постоянное внешнее хранилище, чтобы обеспечить важную характеристику долговечности зафиксированных транзакций. Кроме того, одноуровневая архитектура позволяет значительно упростить структуры данных и алгоритмы, используемые для управления базами данных. В такой системе любая транзакция полностью обрабатывается в одном потоке, строго соответствующем ядру процессора. Следовательно, если конкуренция за объекты базы данных между транзакциями рабочей нагрузки не очень высока (транзакции конфликтуют редко), производительность системы может расти почти линейно с увеличением количества ядер.

Однако очевидной текущей тенденцией является подход HTAP с управлением базами данных в основной памяти. Соответствующие системные архитектуры намного сложнее по сравнению с чисто транзакционными системами. Одноуровневое хранилище может иметь недостаточный размер. Поэтому цель данной статьи – проанализировать основные принципы архитектурной организации современных HTAP-СУБД, понять, может ли быть полезной для них энергонезависимая основная память и, если да, то как ее лучше всего использовать. Последний пункт особенно непонятен в отношении аналитики в реальном времени.

Но прежде чем приступить к этому анализу, мы кратко опишем общий ландшафт систем управления данными с хранением баз данных в основной памяти.

3. Пейзаж систем управления данными в памяти

На рис. 1 представлена репрезентативная выборка систем управления данными с хранением баз данных в основной памяти. Точнее, почти все системы в этой таблице, кроме одной (Spark-PMoF), на самом деле являются СУБД разных классов. Три класса здесь включают примеры чисто транзакционных СУБД (синий сектор), гибридных СУБД, поддерживающих как транзакционную, так и аналитическую рабочую нагрузку (темно-красный двойной сектор) и чисто аналитических систем управления данными (красный сектор).

Основное внимание в данной статье уделяется архитектурам HTAP, и мы дадим более подробный анализ решений этого «среднего» класса в следующем разделе. Сейчас мы кратко рассмотрим представителей «крайних» классов – чисто транзакционные и чисто аналитические системы.

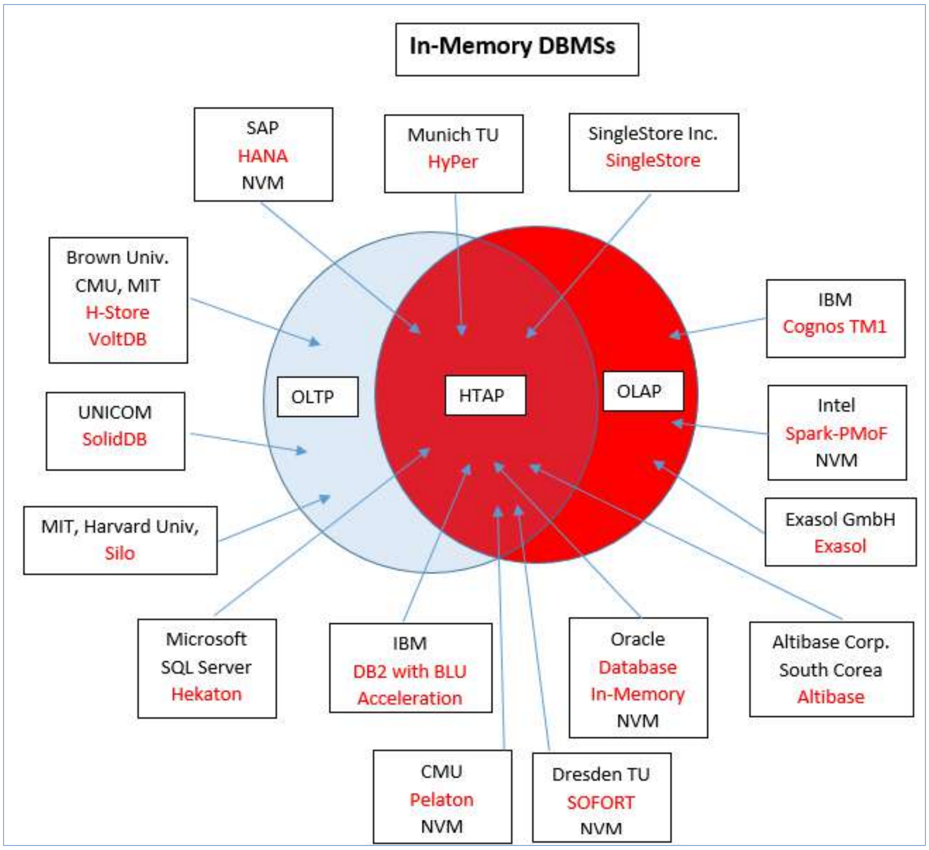

In-Memory DBMSs
SAP HANA NVM
Munich TU HyPer
SingleStore Inc. SingleStore
Brown Univ. CMU, MIT H-Store VoltDB
IBM Cognos TM1
UNICOM SolidDB
OLTP
HTAP
OLAP
Intel Spark-PMoF NVM
Exasol GmbH Exasol
MIT, Harvard Univ, Silo
Microsoft SQL Server Hekaton
IBM DB2 with BLU Acceleration
Oracle Database In-Memory NVM
Altibase Corp. South Corea Altibase
CMU Pelaton NVM
Dresden TU SOFORT NVM

Рис. 1. Системы управления данными с хранением баз данных в основной памяти
Fig. 1. In-memory data management systems

3.1 Транзакционные in-memory СУБД

3.1.1 H-Store/VoltDB

Как мы уже упоминали, самая известная система первого класса – это VoltDB [3]. VoltDB является коммерческим преемником H-Store, академического проекта, инициированного Майклом Стоунбрейкером и реализованного консорциумом четырех крупных университетов США (Браун, Карнеги-Меллон, Массачусетский технологический институт и Йельский университет). Основные идеи H-Store изначально были опубликованы в [17].

Коротко говоря, H-Store/VoltDB – это массивно-параллельная, многораздельная, транзакционная in-memory СУБД с хранилищем таблиц на основе строк. Уникальные особенности этой системы включают отказ от ведения журнала транзакций. Система вообще не использует внешнее хранилище. И надежность транзакций, и надежность базы данных поддерживаются механизмом репликации.

Каждый узел системы выполняет локальные транзакции последовательно, одну за другой, чтобы исключить любую форму конкуренции локальных транзакций и, следовательно, избежать их локальных откатов. Разделение базы данных основано на статическом анализе транзакций, участвующих в рабочей нагрузке (код всех транзакций должен быть доступен заранее). Цель этого анализа (а затем разделения) – минимизировать количество распределенных транзакций, для обработки которых требуются данные из более чем одного раздела.

В действительности система позволяет выполнять распределенные транзакции, которые фиксируются по традиционному двухфазному протоколу. Однако при обработке распределенных транзакций пропускная способность системы естественным образом снижается.

3.1.2 SolidDB

SolidDB [10, 18] – это продукт, который разрабатывался финской компанией Solid Information Technology с 1992 года. Затем он был приобретен IBM в 2007 году и продан компании UNICOM Global в 2014 году. В настоящее время SolidDB в основном используется для поддержки телекоммуникационных и сетевых приложений.

Система одновременно поддерживает два строковых хранилища данных: на дисках и в памяти. Дисковое хранилище полностью традиционное – данные хранятся в блоках дисков; индексы основаны на B + -деревьях. Для хранилища в основной памяти используются специальные, оптимизированные для основной памяти структуры данных. В частности, индексы для таблиц, резидентных в основной памяти, основаны на комбинации B-дерева и Patricia-trie. Решение о размещении таблицы (в памяти или на диске) должно приниматься при создании этой таблицы. Каждая операция запроса или обновления может относиться к таблицам как в памяти, так и на диске с полными транзакционными гарантиями.

Дополнительной особенностью СУБД SolidDB является возможность встраивать ее в приложения. В этом режиме экземпляр СУБД находится в адресном пространстве каждого такого приложения, а часть базы данных в основной памяти находится в сегменте совместно используемой памяти, который присоединен к этой виртуальной памяти. Детали реализации (например, как в этом случае поддерживается защита базы данных от прикладного программного обеспечения (и поддерживается ли она)) не предоставляются.

3.1.3 Silo

Проект Silo [11] был реализован в Гарвардском университете в начале 2010-х годов. Работой руководили лауреат премии Тьюринга Барбара Лисков (Barbara Liskov) и известный исследователь баз данных Сэмюэл Мэдден (Samuel Madden). Дизайн системы кажется предельно современным и интересным. К сожалению, проект мертв, хотя исходный код Silo все еще доступен на GitHub [19].

СУБД Silo ориентирована на транзакционное управление данными в основной памяти на современных многоядерных компьютерах. Хранение таблиц организовано с использованием первичных и вторичных индексов на основе оптимизированной для основной памяти разновидности B-дерева (с элементами префиксного дерева). Первичный индекс обеспечивает поиск кортежа по его первичному ключу; вторичный позволяет найти первичные ключи всех кортежей с заданным вторичным ключом.

Каждая транзакция полностью обрабатывается в одном потоке, выполняемом на отдельном ядре процессора. Все потоки имеют общий доступ ко всей памяти, занимаемой базой данных. Долговечность транзакций обеспечивается с помощью журнала транзакций, который хранится во внешнем постоянном хранилище. Чтобы обеспечить сериализуемость транзакций, Silo использует своего рода оптимистичный контроль параллелизма, разделяющий время на эпохи и использующий блокировки вместе с барьерной синхронизацией во время фиксации транзакции.

3.2 Аналитические системы с хранением баз данных в основной памяти

3.2.1 Exasol

Exasol [12] является основным продуктом немецкой компании Exasol GmbH, основанной в 2000 году. Вкратце, Exasol – это массивно-параллельная СУБД без совместного использования ресурсов (но с возможностью репликации) с хранением баз данных в основной памяти. Все аналитические запросы обычно обрабатываются распределенным образом на основе локальных поколоночных хранилищ таблиц каждого узла. Для обеспечения быстрой вставки новых данных система поддерживает дополнительное хранилище таблиц по строкам. Новые строки сливаются с основным поколоночным хранилищем в фоновом режиме.

Exasol использует ETL для добавления новых данных в базу данных. Источниками данных могут быть внешние транзакционные системы, Hadoop, веб-журналы и т.д. Во время локальной (внутри узла) обработки запроса (или части запроса) система активно использует симметричный параллелизм и инструкции процессора SIMD. Кроме того, система обрабатывает локальные запросы с активным использованием кеширования.

Стоит отметить, что Exasol демонстрирует лучшие результаты в бенчмарке TPC-H при масштабировании до сотен терабайт данных. Однако для справедливости следует сказать, что это единственная СУБД с хранением баз данных в основной памяти, которая участвует в этом соревновании.

3.2.2 Многомерные аналитические in-memory СУБД

В аналитических продуктах IBM и Oracle используются две хорошо известные системы многомерных баз данных в оперативной памяти, Cognos TM1 и Essbase. Эти СУБД кажутся очень похожими по всем характеристикам – они поддерживают многомерные кубы данных в основной памяти с предварительно вычисленными агрегатами, тесно интегрированы с электронными таблицами, такими как Excel, и т.д. Конечно, обе системы используют ETL для загрузки новых данных в базу данных. Еще одна общая черта – отсутствие какого-либо описания внутренней архитектуры системы, структур данных и алгоритмов. Поэтому ниже мы приводим лишь краткую историю этих продуктов.

Как гласит история [13], TM1 была впервые выпущена в начале 1983 года компанией Sinper Corporation, которая была приобретена Applix Inc. в 1996 году. Затем Applix была куплена канадской компанией Cognos в 2007 году, и, наконец, Cognos была приобретена IBM позже в том же году. В настоящее время аналитический сервер TM1 используется в более общем программном обеспечении IBM Planning Analytic.

Первая версия Essbase была разработана в 1992 году компанией Arbor Software. В 1998 году Arbor объединилась с компанией Hyperion Software, и Essbase начала использоваться в составе IBM DB2 OLAP Server, существовавшего до 2005 года. Компания Hyperion, в свою очередь, была приобретена Oracle в 2007 году. В настоящее время Essbase [14] является одним из ключевых компонентов Oracle Exalytics In-Memory Machine, специализированной аналитической комбинации аппаратного и программного обеспечения [20].

3.2.3 Spark PmoF

Популярная среда распределенной обработки данных Spark [21] на самом деле является оптимизированной версией MapReduce, которую можно грубо рассматривать как аналитическую систему в оперативной памяти. Однако при обработке сложных запросов с несколькими фазами map-reduce Spark демонстрирует значительное снижение производительности, поскольку перераспределение данных (перемешивание) выполняется с использованием HDFS (в реализации Apache), то есть через удаленное внешнее хранилище.

Инженеры Intel предложили (и реализовали) еще одну важную оптимизацию Spark, Spark-PMoF [15], используя Persistent Memory over Fabrics (PMoF), которая (опять же грубо говоря) является энергонезависимой памятью с байтовой адресацией, доступной через механизм RDMA (удаленный прямой доступ к памяти). Благодаря этой оптимизации при выполнении конкретного задания Spark не использует внешнее хранилище после начальной загрузки данных и выглядит как настоящая система управления и обработки данных в основной памяти. Наверное, стоит отметить, что это проект с открытым исходным кодом, а код и некоторая документация доступны на GitHub [22].

Приведенные в этом разделе примеры очень разных систем показывают, что ландшафт систем управления данными в памяти чрезвычайно разнообразен, и подкласс in-memory HTAP-СУБД является лишь частью этого ландшафта. Однако именно этот подкласс в данной статье представляет для нас основной интерес, и в следующем разделе мы переходим к обсуждению систем категории HTAP с хранением баз данных основной памяти.

4. In-memory HTAP-СУБД

Основная концепция подхода HTAP – объединение в одной системе баз данных возможностей как обработки транзакций, так и аналитики. Помимо этой общей характеристики, СУБД HTAP должна одновременно обеспечивать:

- очень быструю обработку транзакций;
- очень быстрое предоставление свежих данных, генерируемых транзакциями, для анализа данных; а также
- очень быструю обработку аналитических запросов произвольной сложности.

Одновременно достичь этих претенциозных целей нелегко. Однако есть несколько примеров СУБД, которые демонстрируют желаемые характеристики с использованием различных архитектурных и алгоритмических подходов, хотя все они используют хранилище данных в основной памяти.

В этом разделе мы кратко обсудим происхождение и основные архитектурные особенности этих систем. Мы начинаем с четырех систем основных поставщиков СУБД, а именно SAP, Microsoft, Oracle и IBM. Затем мы рассматриваем пару менее масштабных коммерческих СУБД и, наконец, представляем несколько прототипов академических систем. В каждом случае мы отмечаем, использует ли соответствующая система энергонезависимую память.

4.1 SAP HANA

HANA, аббревиатура от High-Performance Analytic Appliance, является основным продуктом управления базами данных компании SAP SE. Прежде всего, система используется в ERP-решениях самой SAP. Однако HANA приобретает все большую популярность как отдельный инструмент для разработки и поддержки приложений баз данных, которым требуется как транзакционная, так и аналитическая обработка данных. Система доступна в различных воплощениях, например, в кластерной или облачной версии. Однако здесь мы ограничимся основной конфигурацией HANA для одной машины.

4.1.1 Происхождение и предыстория

SAP SE была первой компанией в истории новейшего времени, которая решила объединить транзакционную и аналитическую обработку в одной системе, и Хассо Платтнер, как кажется, был отцом-основателем этого подхода. Еще в 2008 году Вишал Сикка (Vishal Sikka), который был техническим директором SAP с 2007 по 2014 год, написал в своем блоге: «… Хассо Платтнер вдохновил меня на проведение эксперимента, который мы назвали Hana… [и который] продемонстрировал, что возможна совершенно новая архитектура приложения, обеспечивающая комплексную аналитику и агрегацию в реальном времени актуальных транзакционных данных…» [23].

Публикации, посвященные HANA, показывают, что главным архитектором системы с 2009 по 2019 год был Франц Фаербер (Franz Faerber). Активными участниками проекта были (и остаются) Вольфганг Ленер (Wolfgang Lehner) и члены возглавляемой им группы баз данных Дрезденского технического университета.

HANA не создавалась с нуля. На механизм хранения таблиц по столбцам в основной памяти повлияли собственные разработки SAP -- механизм текстового поиска TREX (2001) [24] и SAP Business Warehouse Accelerator (BWA, 2005) [25]. Механизм хранения таблиц по строкам в основной памяти основан на решении P*TIME [26], разработанном компанией Transact in Memory, Inc., которая было приобретено SAP в 2005 году. Механизм MaxDB [27], который был приобретен SAP в 1997 году у Software AG (тогда он назывался Adabas D), использовался для обеспечения уровня постоянного хранения данных. Первый релиз HANA был выпущен SAP в конце 2010 года.

4.1.2 Архитектурные особенности

Рис. 2 демонстрирует общую архитектуру HANA [28]. Мы обсуждаем только часть этой архитектуры, относящуюся к функциям HTAP в основной памяти.

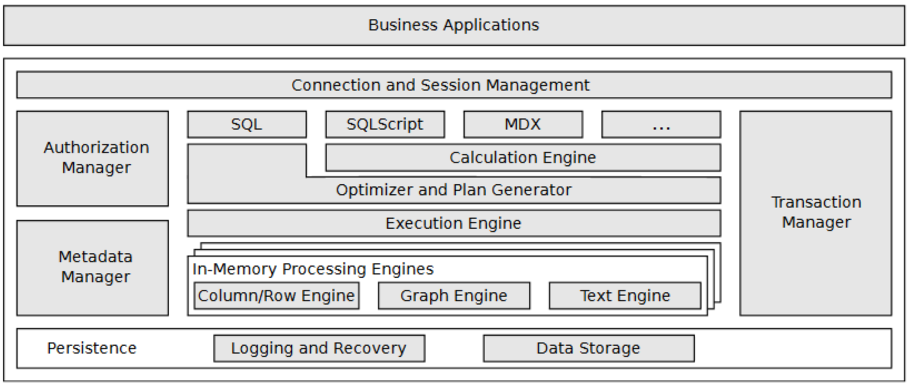


Рис. 2. Архитектура HANA [28]
Fig. 2. HANA architecture [28]

HANA позволяет использовать для одной и той же базы данных хранилища таблиц как по столбцам, так и по строкам. При создании новой таблицы пользователь может выбрать, как ее хранить, по строкам или по столбцам. Согласно первоначальным проектным решениям [16] предпочтительнее использовать хранилище по столбцам, и HANA оптимизирована для такой организации таблиц. Вероятно, хранилище столбцов не позволяет системе продемонстрировать высочайшую производительность обработки транзакций, потому что обновление хранилища столбцов – это операция, требующая много времени. Однако взамен этого поколоночный способ хранения таблиц базы данных снижает затраты на память из-за агрессивного сжатия данных. Чтобы достичь более высокой скорости обработки транзакций, можно использовать дорогостоящее хранилище строк.

Вообще говоря, каждый столбец таблицы хранится в хранилище столбцов с использованием двух структур данных – словаря и индексного вектора. Словарь содержит все различные значения, находящиеся в данный момент в столбце, а вектор индекса связывает каждую строку таблицы с соответствующей записью словаря. Для поддержки запросов по диапазону значений столбца, которые распространены в аналитике, HANA сохраняет отсортированный порядок элементов словаря. Однако это делает операции обновления, часто используемые при обработке транзакций, очень дорогими. Поэтому для каждого столбца поколоночной таблицы имеются две части хранения: основная часть хранит словарь столбца в отсортированном порядке, а в дельта-части такой порядок не поддерживается. Все операции над таблицей используют как основные данные каждого столбца, так и его дельту. Основываясь на информации о размере дельты и текущей рабочей нагрузке, система решает, когда выполнять слияние основной части данных столбца с ее дельтой. Во время выполнения этой операции используется новая дельта.

SAP рекомендует использовать хранилище по строкам для небольших таблиц, в которых строки часто обновляются или удаляются. Таблицы, которые хранятся по строкам, всегда сохраняются в основной памяти. Однако таблицы, которые хранятся по столбцам, могут быть очень большими, и они могут быть разбиты на два раздела, один из которых хранится в памяти, а другой – на дисках. SAP называет раздел таблицы в основной памяти текущим (current), а на диске – историческим (historical) (другие поставщики используют термины горячий (hot) и холодный (cold) соответственно). Имеется в виду, что текущая часть таблицы должна содержать данные, которые в настоящее время активно используются. HANA может распределять данные между этими разделами автоматически в зависимости от времени жизни определенных строк таблицы или с использованием явных подсказок, предоставляемых приложениями. Текущие и исторические разделы имеют разные схемы хранения, но могут быть доступны с помощью одного запроса.

Имеется несколько дополнительных архитектурных решений, позволяющих одновременно и эффективно обрабатывать транзакции и аналитические запросы в одной системе. Во-первых, каждый аналитический запрос компилируется и оптимизируется с учетом самой последней статистики базы данных. Запросы, используемые в повторяющихся (параметризованных) транзакциях, подготавливаются заранее; при обработке транзакции используются готовые исполняемые планы запросов.

Во-вторых, аналитические запросы обрабатываются с высокой степенью распараллеливания с использованием нескольких потоков, выполняемых на разных ядрах ЦП. Распараллеливание транзакционных запросов считается вредным: переключение контекста обходится слишком дорого. Поэтому каждый транзакционный (простой) запрос полностью выполняется в одном потоке.

Наконец, чтобы минимизировать негативные эффекты конкуренции между операциями транзакций и аналитическими запросами, которые могут замедлить обработку транзакций, транзакционные операции всегда имеют более высокий приоритет, чем запросы OLAP.

4.1.3 HANA и энергонезависимая основная память

SAP HANA – первая коммерческая СУБД, активно использующая NVM (Intel Optane PMem) [29-30]. Учитывая особенности Optane NVM (высокая задержка и большая емкость), разработчики HANA решили использовать эту память для замены основных частей поколоночных хранилищ. Основными преимуществами этого подхода являются очень быстрый перезапуск системы после отключения питания (энергонезависимость) и возможность расширения текущих разделов столбчатых таблиц (большая емкость). Минус – очень вероятное замедление аналитики (большая задержка).

Авторы [29] отмечают, что замена DRAM на NVM может:

- сильно упростить и ускорить поддержку долговечности транзакций;
- существенно увеличить объем памяти на один процессор по разумной цене;
- предоставлять доступ к данным непосредственно в NVRAM после перезапуска, без необходимости перезагружать их с диска в DRAM.

Вторая и третья возможности уже реализованы с использованием существующего в настоящее время Optane PMem. Для реализации первой (очень важной) возможности нужны новые версии NVM с лучшими характеристиками задержки.

4.2 Аналитика реального времени в основной памяти в Microsoft

Похоже, что Microsoft внедрила аналитику в реальном времени в SQL Server в три этапа.

Во-первых, они разработали поколоночное хранилище таблиц на дисках [31]. Затем был разработан и реализован механизм OLTP в основной памяти Hekaton [32]. И, наконец, они разработали аналитический механизм для своего хранилища таблиц по столбцам и интегрировали все эти компоненты в SQL Server 2014 (и улучшили в SQL Server 2016), а также предоставили аналитику в реальном времени [33].

Судя по публикациям, Пер-Оке Ларсон (Per-Åke Larson) был руководителем проектов как хранилищ данных в основной памяти, так и систем для аналитики в реальном времени. Он ушел из Microsoft в 2015 году.

4.2.1 Архитектурные решения

Упрощенная архитектура SQL Server 2014 изображена на рис. 3.

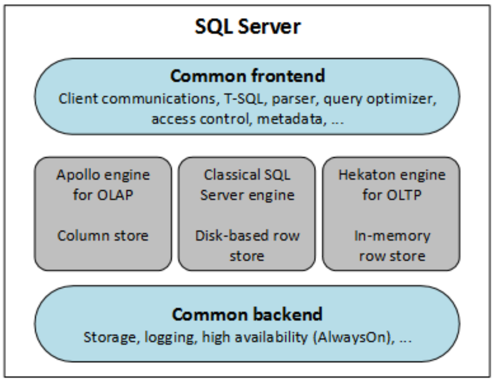

SQL Server
Common frontend
Client communications, T-SQL, parser, query optimizer, access control, metadata, ...
Apollo engine for OLAP
Column store
Classical SQL Server engine
Disk-based row store
Hekaton engine for OLTP
In-memory row store
Common backend
Storage, logging, high availability (AlwaysOn), ...

Рис. 3. Упрощенная архитектура SQL Server [33]
Fig. 3. A simplified architecture of SQL Server [33]

Аналитика в реальном времени обеспечивается компонентами Hekaton и Apollo. Hekaton – это специализированный механизм транзакционной базы данных в памяти, использующий хранилище строк. Apollo – это специализированная аналитическая дисковая подсистема, использующая хранилище по столбцам. Собственно, Apollo – это не название какого-либо продукта, это всего лишь кодовое название проекта. Рабочие данные очень быстро передаются из Hekaton в Apollo и преобразуются из представления на основе строк в представление на основе столбцов. (Последнюю операцию можно рассматривать как своего рода облегченный ETL.)

Основные архитектурные идеи Hekaton заключаются в следующем. Индексы (хэш и своего рода B-Tree) разработаны и оптимизированы для работы с данными, находящимися в основной памяти. Операции с индексами не журнализуются, и все индексы перестраиваются во время восстановления базы данных. Все внутренние структуры данных – распределители памяти, хеш-индексы и диапазоны, а также карта транзакций – полностью свободны от блокировок. Операторы SQL и хранимые процедуры компилируются в настраиваемый высокоэффективный машинный код.

Система определяет *горячие* и *холодные* данные и сохраняет в памяти только горячие данные [34]. Основная идея состоит в том, что не все данные транзакционной базы данных должны быть доступны одинаково быстро; некоторые из них могут быть помещены во внешнее хранилище без замедления обработки транзакций. Hekaton использует LRU-подобный алгоритм для определения того, какие данные не используются активно транзакциями, и перемещает соответствующие части таблиц в дисковое хранилище. Конечно, в памяти и на дисках используются разные физические схемы расположения строк. Поколоночное представление для аналитических целей поддерживается для обеих частей таблиц.

4.2.2 SQL-сервер и энергонезависимая основная память

Первые шаги по использованию NVM были сделаны до выпуска SQL Server 2014 для Linux. В то время Microsoft использовала небольшой объем NVM с прямым доступом, чтобы поместить в NVM буферизованный конец журнала, который еще не сброшен во внешнее хранилище [35]. Потеря хвоста журнала из-за отключения питания – хорошо известная проблема дисковых СУБД. Эта проблема вынуждает использовать очень сложные алгоритмы восстановления базы данных. Использование NVM гарантирует, что весь журнал будет доступен после отключения питания. Это радикально упрощает как поддержку долговечности транзакций, так и процесс восстановления базы данных.

Позже поддержка NVM в SQL Server была значительно расширена. SQL Server 2019 для Linux [36] предоставляет возможность разместить любую базу данных или файл журнала в NVM. Что еще более важно, SQL Server теперь поддерживает так называемый гибридный буферный пул, в котором все очищенные страницы (уже сброшенные на внешнее хранилище) перемещаются в NVM. После этого эти страницы остаются доступными без выполнения операций ввода-вывода.

4.3 Oracle database in memory

Решение Oracle для управления данными в основной памяти основано на in-memory СУБД TimesTen [37]. Проект TimesTen был основан в Hewlett-Packard Labs Мари-Анн Неймат (Marie-Anne Neimat). В то время система называлась SmallDB [38] и предназначалась для использования во внутренних целях HP. В 1996 году Неймат основала стартап TimesTen Inc. С этого времени TimesTen активно использовалась в качестве транзакционной СУБД реального времени во многих приложениях, особенно в области телекоммуникаций.

В 2005 году TimesTen была приобретена Oracle, интегрирована с общей инфраструктурой Oracle и начала использоваться в качестве кеш-памяти основной СУБД Oracle. Около десяти лет Неймат продолжала руководить проектом TimeTen, но затем ушла из Oracle.

Интегрированная опция Oracle Database In-Memory, поддерживающая функциональность HTAP, впервые появилась в Oracle 12c в 2013 году [39]. В настоящее время Тиртханкар Лахири (Tirthankar Lahiri) и Шасанк Чаван (Shasank Chavan) (оба являются вице-президентами Oracle), как кажется, руководят всем направлением Oracle in-memory.

4.3.1 Архитектурные особенности

Система поддерживает два вида хранилищ данных: хранилище строк находится на дисках, а хранилище столбцов – в основной памяти. Можно выборочно объявить важные таблицы или их разделы резидентными в хранилище столбцов при использовании буферного кеша для остальной части таблицы. Таким образом, есть имеются две области основной памяти для размещения частей базы данных: традиционный кеш для блоков внешней памяти и хранилище столбцов.

Части одной и той же таблицы, хранящиеся в разных представлениях (и в разных хранилищах), обновляются синхронно. Хранилище строк обеспечивает быстрый транзакционный доступ к данным с помощью тщательно разработанных индексов и кэширования блоков в памяти. Хранилище столбцов оптимизировано для обработки аналитических запросов. Свежесть данных для аналитики обеспечивается автоматической синхронизацией содержимого хранилищ столбцов и строк при вставке, удалении или обновлении строк. Многоверсионность, используемая в управлении транзакциями, позволяет снизить конкуренцию за данные между транзакциями и аналитическими запросами.

Oracle Database In-Memory может быть развернута в инфраструктуре Oracle Real Application Cluster (RAC) [40]. Общая архитектура представлена на рис. 4.

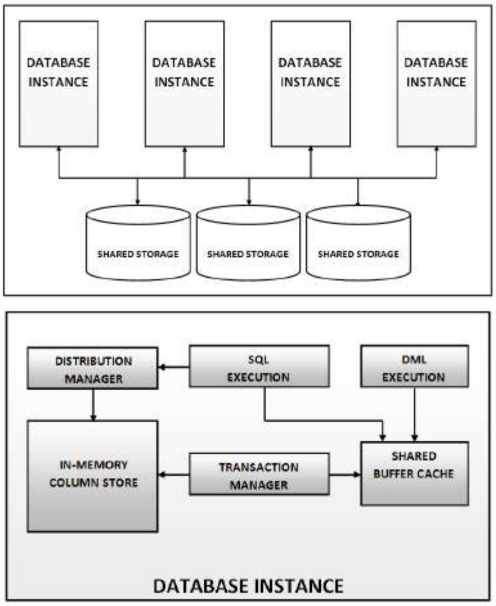


Рис. 4. Архитектура Database In-Memory on Oracle RAC [40]
Fig. 4. Architecture of Database In-Memory on Oracle RAC [40]

В этом случае блоки данных, содержащие части хранилища строк, доступны и изменяются через общий буферный кеш базы данных. Кроме того, для каждого отдельного экземпляра базы данных можно образовать индивидуальное хранилище столбцов в основной памяти. Все таблицы горизонтально разделяются по всем узлам кластера.

Если система работает в режиме in-memory, Exadata Database Machine автоматически переформатирует все данные, к которым осуществляется доступ, в поколоночный формат в основной памяти и сохраняет их в так называемом флэш-кеше (внешнее хранилище большого объема на основе флэш-памяти) [41]. После этого все части запросов, выгружаемые в Exadata, выполняются так же, как если бы они обрабатывались в основной памяти на узле RAC. В частности, для обработки столбцов таблицы Exadata использует инструкции SIMD.

4.3.2 Oracle и энергонезависимая основная память

Oracle предоставляет опцию Persistent Memory Database в своей базе данных Database 20c [42]. Фактически, этот вариант дает возможность разместить всю базу данных или ее часть в NVM. Точнее, они предоставляют средство PMEM Filestore, которое на самом деле обеспечивает файловую систему в энергонезависимой основной памяти, поддерживающую атомарные обновления блоков данных базы данных.

PMEM Filestore предоставляет внешний интерфейс для доступа к базе данных Oracle непосредственно в постоянной памяти. СУБД выполняет ввод-вывод в хранилище PMEM путем копирования памяти намного быстрее, чем ввод-вывод через традиционные вызовы операционной системы. Конечно, доступ к PMEM Filestore предоставляется без традиционного блочного кэширования.

4.4 DB2 с ускорителем BLU

В IBM DB2 ускорение аналитики осуществляется с помощью хранилища столбцов (в памяти), многоядерного распараллеливания и аппаратной векторной обработки. Работа над собственной поколоночной подсистемой в основной памяти в IBM началась в 2006 году. До

2016 года менеджером и техническим руководителем этого направления был Гай Ломан (Guy Lohman) (затем он ушел на пенсию).

Первым продуктом в памяти, разработанным командой Ломана, был Blink [43]. Это чистый механизм хранения столбцов в основной памяти. Он успешно используется и сейчас, например, как Informix Warehouse Accelerator [44].

BLU [45] – это второе поколение продукта. Общая архитектура DB2 с BLU показана на рис. 5 [46].

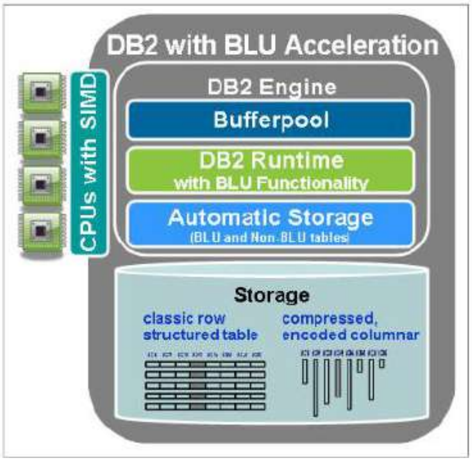

DB2 with BLU Acceleration
CPUs with SIMD
DB2 Engine
Bufferpool
DB2 Runtime with BLU Functionality
Automatic Storage (BLU and Non-BLU tables)
Storage
classic row structured table
compressed, encoded columnar

Рис. 5. Архитектура DB2 с BLU [46]
Fig. 5. The architecture of DB2 with BLU [46]

В базе данных, управляемой DB2 с помощью BLU, каждая таблица может храниться либо в традиционном хранилище строк, либо в хранилище столбцов BLU. Важной особенностью BLU является то, что таблицы в хранилище столбцов могут быть больше, чем размер доступной основной памяти. Фактически, все таблицы хранятся в блоках внешней памяти и доступны через обычный блочный кэш. Однако BLU пытается полностью обработать каждый запрос в основной памяти (хотя технически в этом нет необходимости) и обеспечивает соответствующее управление рабочей нагрузкой. Чтобы обеспечить доступность всех необходимых данных в памяти, BLU использует агрессивную политику предварительной выборки.

Другие характеристики DB2 с BLU кажутся традиционными для современных in-memory СУБД, хотя разработчики BLU подчеркивают очень высокую эффективность их реализации. Система поддерживает возможность сканирования и сравнения сжатых данных. Разработчики избегают блокировок, чтобы обеспечить максимальное распараллеливание. Все структуры данных оптимизированы для минимизации промахов в кэше данных и инструкций. Активно используются векторные команды.

Мы не смогли найти в публикациях и документации по DB2 никаких упоминаний об использовании NVM (или, по крайней мере, о планах использования). Единственное, что касается NVM, – это возможное использование SSD на базе Optane.

4.5 Altibase

Altibase Corporation – южнокорейская компания, основанная в 1999 году. Их in-memory СУБД была приобретена у Южнокорейского научно-исследовательского института электроники и телекоммуникаций (South Korean Electronics and Telecommunications Research Institute). С 2005 года компания предлагает гибридную in-memory и on-disk СУБД Altibase. С 2018 года Altibase является продуктом с открытым исходным кодом [47].

Altibase – это гибридная СУБД в том смысле, что она может управлять данными, хранящимися в памяти, или данными, хранящимися на дисках, или обоими видами данных одновременно. В обоих случаях система хранит таблицы только по строкам [48].

Систему можно рассматривать как разновидность СУБД HTAP, потому что:

- она обеспечивает очень высокую скорость обработки транзакций при работе в режиме основной памяти, и
- дает возможность анализировать свежие операционные данные (в основной памяти) вместе с историческими данными (на дисках) при работе в смешанном режиме.

Однако Altibase, безусловно, не является аналитической системой в реальном времени, поскольку она не имеет хранилища столбцов в основной памяти и, следовательно, не может обеспечивать быструю аналитику. Кроме того, система, вероятно, использует традиционный механизм блокировки, поскольку в документации упоминается обнаружение синхронизационных тупиков.

4.6 MemSQL (SingleStore)

СУБД MemSQL [49], переименованная в SingleStore осенью 2020 г., разработана стартапом MemSQL Software, основанным в 2011 году Эриком Френкилем (Eric Frenkiel) и Никитой Шамгуновым (Nikita Shamgunov) (ранее они оба работали в Facebook). Первый выпуск системы был анонсирован в 2013 году. MemSQL совместима со стандартным протоколом соединения баз данных MySQL (MySQL Wire Protocol). Это означает, что большинство драйверов и продуктов MySQL и MariaDB работают с MemSQL.

На рис. 6 изображена архитектура MemSQL, которая кажется одновременно очень привлекательной для разработчиков современных приложений и немного перегруженной. Здесь нас интересует только основная часть этой архитектуры.

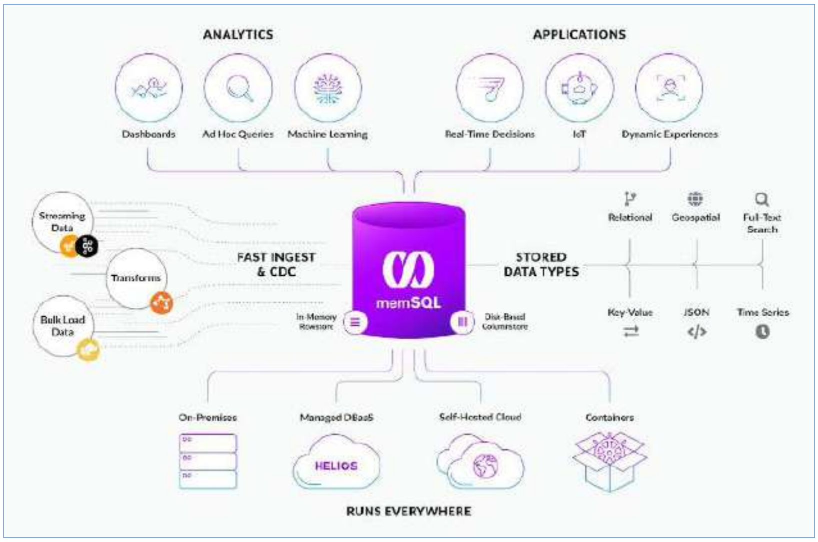


Рис. 6. Архитектура SingleStore [49]
Fig. 6. SingleStore architecture [49]

Система поддерживает хранение таблиц по строкам в основной памяти и по столбцам на дисках. Хранилище строк используется для обработки транзакций. Аналитические запросы могут дополнительно ссылаться на поколононые таблицы, хранящиеся на дисках (они могут содержать, например, исторические данные). Разработчики заявляют, что их система является HTAP и аналитической в реальном времени. Да, очевидно, это СУБД HTAP. Однако мы не уверены, что система действительно может предоставлять аналитику в реальном времени из-за ее основной ориентации на обработку транзакций (хранилище строк в памяти).

Также стоит отметить, что MemSQL позиционируется как распределенная система без совместного использования ресурсов. На наш взгляд, это хорошо для масштабной аналитики, поддерживаемой традиционным способом. Однако в такой архитектуре сложно одновременно поддерживать очень быструю обработку транзакций и очень быструю обработку аналитических запросов. В документации MemSQL ничего не говорится о распределенных транзакциях, двухфазных фиксациях и т.д. А утверждения [50] об очень быстрых распределенных соединениях и агрегации не кажутся убедительными (все это может быть очень быстрым для распределенной СУБД, но слишком медленным для анализа данных в реальном времени).

Разработчики MemSQL думают об использовании постоянной памяти, но не собираются никаким образом переписывать свой код; они предлагают использовать Optane для увеличения емкости основной памяти [51].

4.7 HyPer

Мы заканчиваем этот раздел рассмотрением трех академических HTAP-СУБД, одна из которых, HyPer, является достаточно зрелой системой, а две другие – только исследовательскими прототипами, но они очень интересны. Начнем с краткого обсуждения HyPer.

Проект HyPer реализуется командой баз данных Мюнхенского технического университета под руководством Альфонса Кемпера (Alfons Kemper) и Томаса Ноймана (Thomas Neumann). Хотя это университетский проект, он не является открытым. Исходные тексты системы никогда не публиковались. Более того, в 2015 г. был создан околоуниверситетский стартап HyPer, а в 2016 г. он был поглощен компанией Tableau Software, которая свой вариант СУБД называет Hyper. Детали этой сделки нам неизвестны, но так или иначе развитие HyPer в Мюнхенском университете продолжается. Первый известный отчет по проекту [52] был опубликован в 2010 году.

HyPer – настоящая in-memory СУБД для многоядерных компьютеров. Система поддерживает хранилища таблиц как по строкам, так и по столбцам. Исходный формат таблицы может быть выбран при создании таблицы, и физическая структура этой таблицы может меняться в зависимости от рабочей нагрузки.

Первая версия архитектуры HyPer была очень простой и элегантной, хотя и имела некоторые ограничения. Основные идеи заключались в следующем [53].

Предполагалось, что большинство транзакций могут быть подготовлены заранее, содержать только несколько простых запросов и производить доступ только к нескольким кортежам. Такие транзакции называются короткими. Все короткие транзакции выполняются последовательно в основном процессе OLTP. Вся область основной памяти, в которой находится база данных, отображается в виртуальную памяти этого процесса.

Процесс OLTP периодически (раз в несколько секунд) порождает процесс OLAP после фиксации некоторой только что завершенной транзакции. После этого процесс OLTP включает механизм копирования при записи (copy-on-write), который предоставляет новую страницу основной памяти при любой первой операции записи в соответствующую страницу виртуальной памяти. Таким образом, вновь созданный процесс OLAP видит в своей виртуальной памяти согласованный образ базы данных (согласованный моментальный снимок, consistent snapshot). Процесс OLAP выполняет аналитические запросы над этим моментальным снимком, свежесть данных которого соответствует периодичности создания новых процессов OLAP.

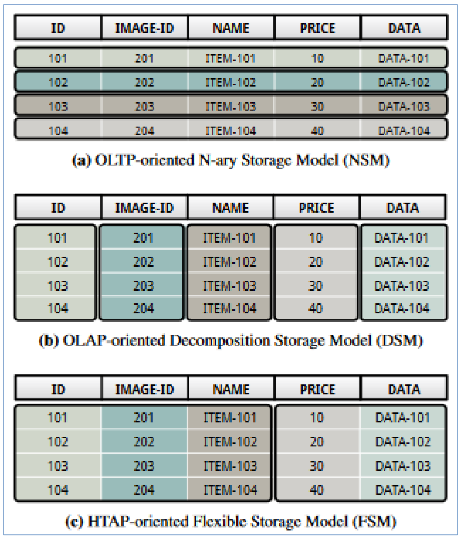
Транзакция, которая выполняет слишком много запросов или обращается к слишком большому количеству кортежей, считается длинной. Выполняемая длинная транзакция откатывается и перенаправляется в текущий процесс OLAP, который имитирует ее выполнение на основе текущего согласованного моментального снимка. Затем эта (частично измененная) транзакция снова перенаправляется в общую очередь транзакций как короткая транзакция. Когда она выполняется, система сначала проверяет все смоделированные обновления по фактическому состоянию базы данных, а затем, если эта проверка прошла успешно, выполняет все эти обновления.

Аналитические запросы распараллеливаются по всем доступным ядрам с использованием массивно-параллельной версии хорошо известного алгоритма сортировки со слиянием (sort-merge join) [54].

Позже разработчики HyPer изменили эту простую архитектуру, чтобы обеспечить более высокую пропускную способность для транзакционных рабочих нагрузок. В частности, они включили в систему свою собственную версию версионного алгоритма управления параллелизмом [55], которая значительно усложняет систему, но улучшает ее транзакционные характеристики.

4.8 Peloton

Проект выполнялся группой баз данных университета Карнеги-Меллон под руководством Эндрю Павло (Andrew Pavlo). Peloton – это in-memory HTAP-СУБД, которая поддерживает хранилища строк и столбцов (см. рис. 7) и обеспечивает самоуправление всех компонентов системы на основе машинного обучения и других технологий искусственного интеллекта [56].



ID	IMAGE-ID	NAME	PRICE	DATA
101	201	ITEM-101	10	DATA-101
102	202	ITEM-102	20	DATA-102
103	203	ITEM-103	30	DATA-103
104	204	ITEM-104	40	DATA-104

(a) OLTP-oriented N-ary Storage Model (NSM)

ID	IMAGE-ID	NAME	PRICE	DATA
101	201	ITEM-101	10	DATA-101
102	202	ITEM-102	20	DATA-102
103	203	ITEM-103	30	DATA-103
104	204	ITEM-104	40	DATA-104

(b) OLAP-oriented Decomposition Storage Model (DSM)

ID	IMAGE-ID	NAME	PRICE	DATA
101	201	ITEM-101	10	DATA-101
102	202	ITEM-102	20	DATA-102
103	203	ITEM-103	30	DATA-103
104	204	ITEM-104	40	DATA-104

(c) HTAP-oriented Flexible Storage Model (FSM)

Рис. 7. Различные схемы хранения таблиц в Peloton [57]
Fig. 7. Different table storage layouts in Peloton [57]

Еще одна важная особенность Peloton – это практически полноценная поддержка NVM вместо (или, вернее, вместе с) энергозависимой основной памяти (или, вернее, вместе с ней). Эту часть проекта реализовал Джой Арулрадж (Joy Arulraj) (совместно с Павло) [58]. За эту работу Арулрадж получил в 2019 году премию Джима Грея за лучшую докторскую диссертацию (Jim Gray Doctoral Dissertation Award).

Наиболее интересными представляются следующие аспекты этой работы:

- архитектура подсистемы хранения использует свойства долговечности и байтовой адресации NVM; обеспечивается экономичное использование энергонезависимой памяти и увеличение срока ее службы за счет уменьшения количества операций записи;

- в системе используется новый протокол журнализации и восстановления, который называется «журнализация с отложенной записью» (Write-Behind Logging, WBL), что позволяет достичь высокого уровня доступности.

Последняя версия Peloton доступна для загрузки из общедоступного репозитория Github [59]. Однако около двух лет назад команда решила отказаться от развития этого репозитория и перейти к созданию новой СУБД. Новый репозиторий создан [60], и ведется работа над проектом под названием NoisePage.

4.8 SOFORT

Проект SOFORT гибридной системы NVM-DRAM с хранением таблиц по стлбцам [61] был выполнен Исмаилом Укидом (Ismail Oukid) в группе баз данных Дрезденского технического университета под руководством Вольфганга Ленера.

В целом SOFORT представляет собой систему с поколоночным хранением таблиц категории HTAP (рис. 8). Система поддерживает одноуровневую подсистему хранения, использующую как NVM, так и традиционную DRAM. SOFORT – это незавершенный исследовательский проект (он мертв).

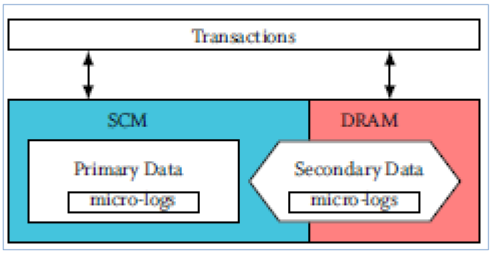


Рис. 8. Гибридная архитектура NVM-DRAM SOFORT [61]
Fig. 8. Hybrid NVM-DRAM architecture of SOFORT [61]

Тем не менее, Укид сделал несколько интересных вещей, которые, безусловно, будут полезны при разработке будущих in-NVM СУБД:

- модель программирования с использованием NVM;
- управление энергонезависимой памятью и ее распределение для нужд СУБД;
- управление транзакциями и восстановление базы данных после сбоев;
- фреймворк для тестирования программного обеспечения, ориентированного на использование энергонезависимой памяти.

5. Обсуждение и заключение

Как видно из приведенного выше обзора, существует множество способов реализации функций HTAP и предоставления аналитики в реальном времени. Общие принципы заключаются в следующем:

- предоставлять свежие данные для анализа путем объединения транзакционной и аналитической обработки в одной системе баз данных;
- сделать обработку транзакций максимально быстрой, чтобы удовлетворить потребности транзакционных клиентов и повысить актуальность данных;
- сделать оперативные данные доступными для анализа как можно быстрее, чтобы удовлетворить первое требование аналитики в реальном времени – свежесть данных;
- как можно быстрее производить оценку аналитических запросов для удовлетворения второго требования аналитики в реальном времени – быстрого анализа данных.

Требования к HTAP-СУБД и аналитике в реальном времени не включают никаких абсолютных чисел, но имеют некоторые неформализованные разумные количественные ограничения. Мы не знаем точно, сколько транзакций (и какого типа) в секунду должна обрабатывать HTAP-СУБД и каково максимально допустимое время для выполнения аналитического запроса. Вот почему этим (полу) качественным требованиям может удовлетворять множество различных архитектур: все они делают все возможное в рамках этих неформальных границ.

Обычный подход к удовлетворению этих требований, насколько возможно их удовлетворить, состоит в том, чтобы хранить всю базу данных в основной памяти. Этот выбор позволяет почти избежать любого ввода-вывода с внешними устройствами хранения (постоянное хранилище используется только для обеспечения долговечности транзакций) и использовать прямые указатели на объекты базы данных в базе данных. Все внутренние структуры данных должны разрабатываться с учетом кеширования.

Для дальнейшего повышения производительности используются передовые методы оптимизации запросов, своевременная компиляция запросов в машинный код, многоядерное распараллеливание, инструкции SIMD и аппаратные ускорители, такие как графические процессоры и FPGA.

Обычно одна СУБД поддерживает два хранилища данных – строковое и столбцовое. Этот подход представляет собой компромисс между возможностью быстрой обработки транзакций (хранилище строк) и быстрой аналитикой запросов (хранилище столбцов), с одной стороны, и необходимостью преобразования данных из строкового формата в поколоночный до того, как данные станут доступны для аналитики, с другой стороны.

Чтобы избежать высокого уровня конкуренции между одновременно выполняемыми транзакциями, а также между транзакциями и параллельно выполняемыми аналитическими запросами, обычно используется какой-то вид многоверсионного управления параллелизмом, часто оптимистический. Все внутренние объекты базы данных, такие как индексы, распределители памяти и т.д., проектируются так, чтобы избежать всех видов блокировок.

Однако хранение базы данных полностью в памяти также является компромиссом между высокой производительностью СУБД и ограниченным размером базы данных. Чтобы смягчить это ограничение, основные производители предпочитают комбинировать хранилища в оперативной памяти и на диске. Они утверждают, что после этого производительность СУБД не ухудшается (непонятно почему), но в любом случае архитектура системы становится намного сложнее.

Есть несколько попыток предоставить функции HTAP и аналитики в реальном времени в массивно-параллельных архитектурах без совместного использования ресурсов. Мы считаем, что такая цель труднодостижима. Во-первых, даже если каждый узел такой системы полностью хранит базу данных в памяти, неизбежное сетевое взаимодействие значительно снизит производительность системы. Во-вторых, как транзакционные, так и аналитические СУБД без совместного использования ресурсов полагаются на подходящее разделение данных по узлам системы. Однако цели разделения для транзакционных и аналитических СУБД различны. Транзакционная СУБД пытается разделить базу данных, чтобы минимизировать количество распределенных транзакций. Аналитическая СУБД при разделении базы данных пытается минимизировать количество распределенных объединений. Сомнительно, чтобы в одной системе можно было достичь обеих целей.

Наконец, значительное число поставщиков СУБД HTAP в оперативной памяти уже используют в своих продуктах доступную в настоящее время энергонезависимую основную память или планируют использовать ее в будущем. Наш обзор демонстрирует достаточно широкий спектр вариантов использования NVM от простого расширения памяти с помощью NVM до наиболее многообещающих одноуровневых архитектур хранения. Однако последний вариант использования сейчас (частично) реализован только в исследовательских проектах.

Причина, вероятно, в том, что в настоящее время на рынке доступен только один вид NVM – Intel Optane, основанный на технологии PCM. Эти модули DIMM на основе NVM имеют довольно высокую задержку и не могут заменить DRAM. Мы полагаем, что широко ожидаемое новое поколение NVM (возможно, на основе STT) значительно ускорит внедрение NVM в HTAP (и чисто транзакционных) СУБД.

Данная статья является русскоязычным (авторским) вариантом ранее опубликованной статьи [62], в которой, кроме смены языка, исправлены некоторые незначительные неточности.

Список литературы / References

[1] Michael Stonebraker, Ugur Cetintemel. "One Size Fits All": An Idea Whose Time Has Come and Gone. Proceedings of the 21st International Conference on Data Engineering, 2005, pp. 2-11.

[2] Andrew Lamb, Matt Fuller, Ramakrishna Varadarajan, Nga Tran, Ben Vandiver, Lyric Doshi, Chuck Bea. The Vertica Analytic Database: C-Store 7 Years Later. Proceedings of the VLDB Endowment, vol. 5, no. 12, 2012, pp. 1790-1801.

[3] Michael Stonebraker, Ariel Weisberg. The VoltDB Main Memory DBMS. Bulletin of the Technical Committee on Data Engineering, vol. 36, no. 2, 2013, pp. 21-27.

[4] Franz Faerber, Alfons Kemper, Per-Åke Larson, Justin Levandoski, Thomas Neumann, Andrew Pavlo. Main Memory Database Systems. Foundations and Trends in Databases, vol. 8, no. 1-2, 2016, pp. 1–130.

[5] С.Д. Кузнецов. В ожидании нативных архитектур СУБД на основе энергонезависимой основной памяти. Труды ИСП РАН, том 32, выпуск 1, 2020 г., стр. 153-180. DOI: 10.15514/ISPRAS-2020-32(1)-9 / Sergey Kuznetsov. Towards a Native Architecture of in-NVM DBMS. Proceedings of the 6th International Conference on Actual Problems of Systems and Software Engineering (APSSE), 2019, pp. 77-89.

[6] Gartner Glossary: Real-time Analytics. URL: https://www.gartner.com/en/information-technology/glossary/real-time-analytics, accessed 08-16-2020.

[7] Mohammed Al-Kateb, Paul Sinclair, Grace Kwan-On Au, Carrie Ballinger. Hybrid Row-Column Partitioning in Teradata. Proceedings of the VLDB Endowment, vol. 9, no. 13, 2016, pp. 1353-1364.

[8] Hybrid transactional/analytical processing. From Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Hybrid_transactional/analytical_processing, accessed 08-17-2020.

[9] Gartner Glossary: HTAP-enabling In-memory Computing Technologies. URL: https://www.gartner.com/en/information-technology/glossary/htap-enabling-memory-computing-technologies, accessed 08-17-2020.

[10] Jan Lindström, Vilho Raatikka, Jarmo Ruuth, Petri Soini, Katriina Vakkila. IBM solidDB: In-Memory Database Optimized for Extreme Speed and Availability. Bulletin of the Technical Committee on Data Engineering, vol. 36, no. 2, 2013, pp. 14-20.

[11] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. Speedy Transactions in Multicore In-Memory Databases. Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, 2013, pp. 18-32.

[12] EXASOL: A Peek Under the Hood. White Paper. URL: https://www.dataviz.sk/wp-content/uploads/2019/09/WP_Exasol_Technical_Peek_under_the_hood.pdf, accessed 08-17-2020.

[13] The Official History of TM1. URL: https://cubewise.com/history/, accessed 08-17-2020.

[14] Michael Schrader, Dan Vlamis, Mike Nader, Chris Claterbos, Dave Collins, Mitch Campbell, Floyd Conrad. Oracle Essbase & Oracle OLAP: The Guide to Oracle's Multidimensional Solution. McGraw-Hill Education, 2009, 524 p.

[15] Yuan Zhou, Haodong Tang, Jian Zhang. Spark-PMoF: Accelerating big data analytics with Persistent Memory over Fabric. Strata Data Conference, 2019 .

[16] Hasso Plattner. A common database approach for OLTP and OLAP using an in-memory column database. Proceedings of the ACM SIGMOD International Conference on Management of data, 2009, pp. 1–2.

[17] Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem, Pat Helland. The End of an Architectural Era (It's Time for a Complete Rewrite). Proceedings of VLDB, 2007, pp. 1150–1160.

[18] solidDB in a Nutshell. URL: https://www.teamblue.unicomsi.com/index.php/download_file/499/660/, accessed 08-19-2020.

[19] gunaprsd/silo: Multicore in-memory storage engine. URL: https://github.com/stephentu/silo, accessed 08-19-2020.

[20] Oracle Exalytics In-Memory Machine: A Brief Introduction. Oracle White Paper, 2013. URL: https://www.oracle.com/technetwork/middleware/bi/overview/whitepaper-exalytics-x3-4-1973011.pdf, accessed 08-19-2020.

[21] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica. Spark: Cluster Computing with Working Sets. Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, 2010, pp. 1-7.

[22] Shuffle Remote PMem Extension for Apache Spark Guide. URL: https://github.com/Intel-bigdata/OAP/tree/master/oap-shuffle/RPMem-shuffle, accessed 08-22-2020.

[23] Vishal Sikka. Timeless Software. URL: http://vishalsikka.blogspot.com/2008/10/timeless-software.html, accessed 08-23-2020.

[24] Frederik Transier, Peter Sanders. Engineering basic algorithms of an in-memory text search engine. ACM Transactions on Information Systems, 2010, Article No. 2.

[25] J. Andrew Ross. SAP NetWeaver BI Accelerator. SAP PRESS, 2008, 260 p.

[26] Sang K. Cha and Changbin Song. P*TIME: Highly Scalable OLTP DBMS for Managing Update-Intensive Stream Workload. Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004, pp. 1033-1044.

[27] André Bögelsack, Stephan Gradl, Manuel Mayer, Helmut Krcmar. SAP MaxDB Administration. SAP PRESS, 2009, 326 p.

[28] Franz Faerber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, Jonathan Dees. The SAP HANA Database – An Architecture Overview. Bulletin of the Technical Committee on Data Engineering, March 2012, vol. 35, no. 1, pp. 28-33.

[29] Mihnea Andrei, Christian Lemke, Günter Radestock, Robert Schulze, Carsten Thiel, Rolando Blanco, Akanksha Meghlan, Muhammad Sharique, Sebastian Seifert, Surendra Vishnoi, Daniel Booss, Thomas Peh, Ivan Schreter, Werner Thesing, Mehul Wagle, Thomas Willhalm. SAP HANA Adoption of Non-Volatile Memory. Proceedings of the VLDB Endowment, vol. 10, no. 12, 2017, pp. 1754-1765.

[30] Intel Optane Persistent Memory and SAP HANA Platform Configuration. Configuration Guide. 2019. URL: https://www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/sap-hana-and-intel-optane-configuration-guide.pdf, accessed 08-26-2020.

[31] Per-Åke Larson, Cipri Clinciu, Eric N. Hanson, Artem Oks, Susan L. Price, Srikumar Rangarajan, Aleksandras Surna, Qingqing Zhou. SQL Server Column Store Indexes. Proceedings of the ACM SIGMOD International Conference on Management of data, 2011, pp. 1177-1184.

[32] Per-Åke Larson, Mike Zwilling, Kevin Farlee. The Hekaton Memory-Optimized OLTP Engine. Bulletin of the Technical Committee on Data Engineering, vol. 36, no. 2, 2013, pp. 34-40.

[33] Per-Åke Larson, Adrian Birka, Eric N. Hanson, Weiyun Huang, Michal Nowakiewicz, Vassilis Papadimos. Real-Time Analytical Processing with SQL Server. Proceedings of the VLDB Endowment, vol. 8, no. 12, 2015, pp. 1740-1751.

[34] Ahmed Eldawy, Justin Levandoski, Per-Åke Larson. Trekking Through Siberia: Managing Cold Data in a Memory-Optimized Database. Proceedings of the VLDB Endowment, vol. 7, no. 11, pp. 931-942.

[35] Bob Dorr. How It Works (It Just Runs Faster): Non-Volatile Memory SQL Server Tail of Log Caching on NVDIMM. URL: https://docs.microsoft.com/ru-ru/archive/blogs/bobsql/how-it-works-it-just-runs-faster-non-volatile-memory-sql-server-tail-of-log-caching-on-nvdimm, accessed 08-27-2020.

[36] Kellyn Gorman, Allan Hirt, Dave Noderer, Mitchell Pearson, James Rowland-Jones, Dustin Ryan, Arun Sirpal, Buck Woody. Introducing Microsoft SQL Server 2019: Reliability, scalability, and security both on premises and in the cloud. Packt Publishing, 2020, 488 p.

[37] Tirthankar Lahiri, Marie-Anne Neimat, Steve Folkman. Oracle TimesTen: An In-Memory Database for Enterprise Applications. Bulletin of the Technical Committee on Data Engineering, vol. 36, no. 2, 2013, pp. 6-13.

[38] Sherry Listgarten and Marie-Anne Neimat. Modelling Costs for a MM-DBMS. Proceedings of the International Workshop on Real-Time Databases, Issues and Applications (RTDB), 1996, pages 72-78.

[39] Tirthankar Lahiri, Shasank Chavan, Maria Colgan, Dinesh Das, Amit Ganesh, Mike Gleeson, Sanket Hase, Allison Holloway, Jesse Kamp, Teck-Hua Lee, Juan Loaiza1, Neil Macnaughton, Vineet Marwah, Niloy Mukherjee, Atrayee Mullick, Sujatha Muthulingam, Vivekanandhan Raja, Marty Roth, Ekrem Soylemez, Mohamed Zait. Oracle Database In-Memory: A dual format in-memory database. Proceedings of the IEEE 31st International Conference on Data Engineering, Seoul, 2015, pp. 1253-1258.

[40] Niloy Mukherjee, Shasank Chavan, Maria Colgan, Dinesh Das, Mike Gleeson, Sanket Hase, Allison Holloway, Hui Jin, Jesse Kamp, Kartik Kulkarni, Tirthankar Lahiri, Juan Loaiza, Neil Macnaughton, Vineet Marwah, Atrayee Mullick, Andy Witkowski, Jiaqi Yan, Mohamed Zait. Distributed Architecture of Oracle Database In-memory. Proceedings of the VLDB Endowment, vol. 8, no. 12, 2015, pp. 1630–1641.

[41] Shasank Chavan, Gurmeet Goindi. Oracle Database In-Memory on Exadata: A Potent Combination. Oracle OpenWorld 2018. URL: https://www.oracle.com/technetwork/database/exadata/pro4016-exadataandinmemory-5187037.pdf, accessed 08-28-2020.

[42] Oracle Database 20c. Database Administrator's Guide. Using Persistent Memory Database. URL: https://docs.oracle.com/en/database/oracle/oracle-database/20/admin/index.html, accessed 08-28-2020.

[43] Ronald Barber, Peter Bendel, Marco Czech, Oliver Draese, Frederick Ho, Namik Hrle, Stratos Idreos, Min-Soo Kim, Oliver Koeth, Jae-Gil Lee, Tianchao Tim Li, Guy Lohman, Konstantinos Morfonios, Rene Mueller, Keshava Murthy, Ippokratis Pandis, Lin Qiao, Vijayshankar Raman, Richard Sidle, Knut Stolze, Sandor Szabo. Business Analytics in (a) Blink. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, vol. 35, no. 1, 2012, pp. 9-14.

[44] IBM Informix Warehouse Accelerator. Technical white paper. URL: https://www.iiug.org/library/ids_12/IWA%20White%20Paper-2013-03-21.pdf, accessed 08-29-2020.

[45] Vijayshankar Raman, Gopi Attaluri, Ronald Barber, Naresh Chainani, David Kalmuk, Vincent KulandaiSamy, Jens Leenstra, Sam Lightstone, Shaorong Liu, Guy M. Lohman, Tim Malkemus, Rene Mueller, Ippokratis Pandis, Berni Schiefer, David Sharpe, Richard Sidle, Adam Storm, Liping Zhang. DB2 with BLU Acceleration: So Much More than Just a Column Store. Proceedings of the VLDB Endowment, Vol. 6, No. 11, 2013, pp. 1080-1091.

[46] Whei-Jen Chen, Brigitte Bläser. Marco Bonezzi, Polly Lau, Jean Cristie Pacanaro, Martin Schlegel, Ayesha Zaka, Alexander Zietlow. Architecting and Deploying DB2 with BLU Acceleration. IBM Redbooks, 2014, 420 p.

[47] Altibase. URL: https://github.com/ALTIBASE, accessed 08-30-2020.

[48] Altibase 7.1 Administrator's Manual. URL: https://github.com/ALTIBASE/Documents/blob/master/Manuals/Altibase_7.1/eng/Administrator's%20Manual%201.md, accessed 08-29-2020.

[49] MemSQL Software. The Cloud-Native Operational Database Built for Speed, Scale, and SQL. URL: https://www.memsql.com/resources/data_sheet-memsql_software/, accessed 08-30-2020.

[50] Jack Chen, Samir Jindel, Robert Walzer, Rajkumar Sen, Nika Jimsheleishvilli, Michael Andrews. The MemSQL Query Optimizer: A modern optimizer for real-time analytics in a distributed database. Proceedings of the VLDB Endowment, Vol. 9, No. 13, 2016, pp. 1401-1412.

[51] Eric Hanson. How to Use MemSQL with Intel's Optane Persistent Memory. URL: https://www.memsql.com/blog/how-to-use-memsql-with-intels-optane-persistent-memory/, accessed 08-30-2020.

[52] Alfons Kemper and Thomas Neumann. HyPer - Hybrid OLTP&OLAP High Performance Database System. Technical Report, TUM-I1010, Munich Technical University, 2010, 29 p.

[53] Alfons Kemper, Thomas Neumann, Jan Finis, Florian Funke, Viktor Leis, Henrik Mühe, Tobias Mühlbauer, Wolf Rödiger. Transaction Processing in the Hybrid OLTP&OLAP Main-Memory Database System HyPer. Bulletin of the Technical Committee on Data Engineering, vol. 36, no. 2, 2013, pp. 41-47.

[54] Martina-Cezara Albutiu, Alfons Kemper, Thomas Neumann. Massively Parallel Sort-Merge Joins in Main Memory Multi-Core Database Systems. Proceedings of the VLDB Endowment, vol. 5, no. 10, 2012, pp. 1064-1075.

[55] Thomas Neumann, Tobias Mühlbauer, Alfons Kemper. Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems. Proceedings of the ACM SIGMOD International Conference on Management of data, 2015, pp. 677–689.

[56] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C. Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun WuF, Ran Xian, Tieying Zhang. Self-Driving Database Management Systems. Proceedings of the 8th Biennial Conference on Innovative Data Systems Research, 2017, 6 p.

[57] Joy Arulraj. Andrew Pavlo. Prashanth Menon. Bridging the Archipelago between Row-Stores and Column-Stores for Hybrid Workloads. Proceedings of the 2016 International Conference on Management of Data, 2016, pp. 583–598.

[58] Joy Arulraj, Andrew Pavlo. Non-Volatile Memory Database Management Systems. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2019, 192 p.

[59] cmu-db / peloton. The Self-Driving Database Management System. URL: https://github.com/cmu-db/peloton, accessed 09-02-2020.

[60] cmu-db / terrier. URL: https://github.com/cmu-db/noisepage, accessed 05-06-2021.

[61] Ismail Oukid. Architectural Principles for Database Systems on Storage-Class Memory. Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn, 2019, pp. 477-486.

[62] S.D. Kuznetsov, P.E. Velikhov, and Q. Fu. Real-time analytics, hybrid transactional/analytical processing, in-memory data management, and non-volatile memory. In Proc. of the Ivannikov ISPRAS Open Conference, 2021, pp. 78-90. DOI: 10.1109/ISPRAS51486.2020.00019.

Информация об авторах / Information about authors

Сергей Дмитриевич КУЗНЕЦОВ – доктор технических наук, профессор, главный научный сотрудник ИСП РАН, профессор кафедр системного программирования МГУ, МФТИ и ВШЭ, старший научный сотрудник РЭУ им. Г.В. Плеханова. Научные интересы: управление данными, архитектуры систем управления данными, модели и языки данных, управление транзакциями, оптимизация запросов.

Sergey Dmitrievich KUZNETSOV – Doctor of Technical Sciences, Professor, Chief Researcher at ISP RAS, Professor at the Departments of System Programming of MSU, MIPT, and HSE, Senior Researcher at REU. Research interests: data management, architectures of data management systems, data models and languages, transaction management, query optimization.

Павел Евгеньевич ВЕЛИХОВ, ведущий инженер ключевых проектов. Научные интересы: массивно-параллельные СУБД, оптимизация запросов, методы искусственного интеллекта для оптимизации СУБД, полуструктурированные модели данных, in-memory СУБД.

Pavel Evgenievich VELIKHOV, Principal Engineer of Key Projects. Research interests: massively parallel DBMS, query optimization, artificial intelligence methods for DBMS optimization, semi-structured data models, in-memory DBMS.

Цян ФУ, представитель бизнес-подразделения. Научные интересы: массивно-параллельные СУБД.

Qiang FU, Business Representative. Research interests: massively parallel DBMS

DOI: 10.15514/ISPRAS-2021-33(3)-14

Применение языковых моделей в задаче извлечения оценочных отношений

Н.Л. Русначенко, ORCID: 0000-0002-9750-5499 <kolyarus@yandex.ru>
Московский государственный технический институт им. Н.Э.Баумана,
105005, Россия, г. Москва, ул. 2-я Бауманская, д. 5, стр. 1

Аннотация. Объемные тексты могут содержать источники взаимосвязанной информации различных типов, передаваемых посредством отношений, некоторые из которых могут быть оценочными. Проведение анализа таких текстов требует установление подобных связей, определении их участников: событий, сущностей, и т.д. В данной работе исследуется применение языковых моделей BERT в задаче извлечения оценочных отношений. Для произвольного документа и списка размеченных в нем именованных сущностей, такая задача предполагает составление списка оценочных отношений между ними. Эффективность применения языковых моделей напрямую зависит от объема обучающих данных. Для увеличения объема обучающего множества применяется подход опосредованного обучения. Такое обучение подразумевает применение алгоритма автоматической разметки оценочных отношений из сторонних источников. Предложенный подход разметки оценочных отношений основан на двухэтапном применении FRAME-BASED фактора в анализе новостных документов, для: (1) составления списка оценочных пар (PAIR-BASED), (2) разметки документов с использованием PAIR-BASED и FRAME-BASED факторов. Полученная на основе такого алгоритма коллекция получила название RuAttitudes2017. Для проведения экспериментов с моделями использовался корпус новостных текстов на русском язык е RuSentRel-1.0. Применение опосредованного обучения с использованием коллекции RuAttitudes2017 повысило качество моделей на 10-13% по метрике F1, и на 25% при сравнении с наилучшими результатами моделей на основе нейронных сетей.

Ключевые слова: анализ тональности; извлечение отношений; опосредованное обучение; нейронные сети; языковые модели

Для цитирования: Русначенко Н. Л. Применение языковых моделей в задаче извлечения оценочных отношений. Труды ИСП РАН, том 33, вып. 3, 2021 г., стр. 199-222. DOI: 10.15514/ISPRAS–2021–33(3)–14

Благодарности: Данная работа выполнена при поддержке гранта РФФИ 20-07-01059

Language Models Application in Sentiment Attitude Extraction Task

N.L. Rusnachenko, ORCID: 0000-0002-9750-5499 <kolyarus@yandex.ru>
Bauman Moscow State Technical University,
5, Building 1, 2-nd Baumanskaya Str., Moscow, 105005, Russia.

Abstract. Large text can convey various forms of sentiment information including the author’s position, positive or negative effects of some events, attitudes of mentioned entities towards to each other. In this paper, we experiment with BERT based language models for extracting sentiment attitudes between named entities. Given a mass media article and list of mentioned named entities, the task is to ex tract positive or negative attitudes between them. Efficiency of language model methods depends on the amount of training data. To enrich training data, we adopt distant supervision method, which provide automatic annotation of unlabeled texts using an additional lexical resource. The proposed approach is subdivided into two stages FRAME-BASED: (1) sentiment pairs list completion (PAIR-BASED), (2) document annotations using PAIR-BASED and FRAME-BASED factors. Being applied towards a large news collection, the method generates

RuAttitudes2017 automatically annotated collection. We evaluate the approach on RuSentRel-1.0, consisted of mass media articles written in Russian. Adopting RuAttitudes2017 in the training process results in 10-13% quality improvement by F1-measure over supervised learning and by 25% over the top neural network based model results.

Keywords: Sentiment Analysis; Relation Extraction; Distant Supervision; Neural Networks; Language Models

For citation: Rusnachenko N.L. Language Models Application in Sentiment Attitude Extraction Task. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 199-222. DOI: 10.15514/ISPRAS-2021-33(3)-14

Acknowledgements. This work was supported by a grant from the RFBR 20-07-01059.

1. Введение

Анализ тональности, т.е. выделение мнения автора о предмете обсуждения в тексте, является одним из наиболее востребованных приложений автоматической обработки текстов за последние годы. Одной из подзадач анализа тональности является задача извлечения оценочных отношений [1], которая предполагает классификацию взаимоотношений между упоминаемыми в тексте именованными сущностями. Извлечение оценочных отношений существенно для анализа тональности новостных и аналитических текстов, поскольку сложным образом влияет на анализ авторской позиции в тексте. В следующем примере приводится фрагмент новостного сообщения, оценочные отношения возникают между сущностями «Россия» и «НАТО» (сущности подчеркнуты): *... Москва$_e$ неоднократно подчеркивала, что ее активность на Балтике$_e$ является ответом именно на действия НАТО$_e$ и эскалацию враждебного подхода к России$_e$ вблизи ее восточных границ ...*

Многие задачи анализа тональности решаются на основе методов машинного обучения, которые, однако, требуют значительного объема обучающих данных. Одним из подходов, направленным на снижение объема ручной разметки данных, является подход опосредованного обучения (Distant Supervision). Опосредованное обучение предполагает выполнение автоматической разметки объемных текстовых коллекций [2] на основе некоторых дополнительных ресурсов, полученная размеченная коллекция далее используется в качестве данных для методов машинного обучения. Несмотря на большое число проведенных исследований подобного подхода разметки документов [3, 4] для задачи анализа тональности и извлечения отношений, область остается изученной лишь частично [5].

В данной работе исследуется применение языковых моделей для извлечения оценочных отношений, предобученных на основе большого автоматического размеченного корпуса извлеченных оценочных отношений по методу опосредованного обучения. Подход основан на использовании лексикона RuSentiFrames [6], который содержит описание оценочных отношений между аргументами слов-предикатов русского языка. Таким образом, вклад настоящей работы следующий:

- исследованы методы машинного обучения для извлечения оценочных отношений из русскоязычных аналитических текстов на уровне документа;
- предложен подход к автоматическому порождению обучающей коллекции для извлечения оценочных отношений, включающий: (1) предварительный этап обработки коллекции для автоматического порождения списка оценочных пар, (2) автоматическую разметку нейтральных отношений;
- проведены исследования извлечения отношений на основе корпуса RuSentRel-1.0 для языковых моделей BERT [7] с применением предложенного подхода порождения обучающей коллекции; согласно полученным результатам исследования, применение опосредованного обучения улучшает качество извлечения оценочных отношений языковыми моделями на 10-13% (трехклассовая классификация) по F1-мере и на 25% при сравнении результатов русскоязычных языковых моделей с аналогичными результатами других архитектур нейронных сетей (кросс-валидационное тестирование).

2. Языковые модели для извлечения отношений

Применение архитектуры *трансформера* [8], оказала огромное влияние в решении многих задач автоматической обработки естественного языка. основана на независимом применении кодировщика [7] и декодировщика [9] трансформера. Применение таких компонентов подразумевает выполнение этапов: (1) предварительного обучения на большом объеме неразмеченных данных и (2) дообучение под конкретную задачу обработки текстов на естественном языке. По завершении первого этапа, модели на основе таких компонентов могут быть интерпретированы как *языковые модели* – вероятностные распределения над последовательностями слов.

Основополагающей моделью на основе *декодировщика* стала GPT [9]. Актуальной на настоящий момент версией является модель GPT-3 [10]. Дообученная версия такой модели на русскоязычных данных получила название **ruGPT-3**[1]. В работе [11] авторы представляют модель для извлечения отношений, которая основывается на классической архитектуре трансформера [8] и дообучении GPT [9], что привело к модели под названием **TRE** [11].

В случае *кодировщиков*, основополагающей моделью стала **BERT** [7]. Такая модель предполагает в качестве входной информации последовательность, опциально разделенную специальным символом [SEP] на две независимые последовательности: TextA и TextB. Учет всех слов контекста достигается благодаря введению задачи *предсказания маскированных токенов* (от англ. Masking Language Modeling): предсказание случайного маскированного слова входной последовательности. Дополнительной задачей, призванной установить связь между TextA и TextB, стала *Natural Language Inference* (NLI), в которой требуется определить[2], является ли TextB продолжением TextA. Применение языковых моделей BERT в задачах классификации выполняется с введением *классификационного слоя*, отвечающего за сопоставление входной последовательности множеству выходных классов задачи. В области аспектно-ориентированного анализа тональности, авторы [12] предлагают подход применения модели BERT в постановке **вопросно-ответный** (Q/A, составление вопроса в TextB для последовательности TextA), и **NLI** (указание ожидаемой информации в TextB, которая должна быть выведена из TextA).

Одним из направлений в развитии BERT-производных архитектур стала публикация предобученных моделей. Исходно доступный набор предобученных моделей[3] делится на: ориентированные под конкретные языки (английский, китайский) и мультиязыковые. Из множества мультиязыковых моделей выделим модель **mBERT**[4], которая предобучена на текстовых данных 104 языков и поддержкой регистра букв в представлении входных последовательностей [7]. Модель mBERT доступна и распространена только в формате base. Для русского языка, авторами проекта DeepPavlov опубликована модель **RuBERT** [13] – дообученная версия mBERT на русскоязычных новостных данных и статьях энциклопедии «Википедия». Модель **SentRuBERT**[5] является дообученной версией RuBERT коллекциями: (1) переведенных на русский язык текстами корпуса SNLI [14] сервисом Google-Translate; (2) русскоязычных текстов корпуса XNLI [15].

Другим направлением в развитии BERT-архитектур стала модификация используемых задач этапа предобучения. В модели **Electra** [16] задача предсказания маскированного слова модифицирована в задачу выявления в контексте специально подмененного слова. **RoBERTa** [17] представляет собой улучшение предобучения (задач на этапе предобучения моделей).

[1] https://github.com/sberbank-ai/ru-gpts

[2] Для проведения классификации в модели предусмотрен специальный токен [CLS] перед началом входной последовательности

[3] https://github.com/google-research/bert

[4] https://huggingface.co/bert-base-multilingual-cased

[5] https://huggingface.co/DeepPavlov/rubert-base-cased-sentence

Построение модели кросс-языкового кодировщика предложений на основе набора текстов ста различных языков [18] стало одним из применений модели RoBERTa. Такая модель получила название **XLM-R**[6]. **SpanBERT** [19] представляет собой модификацию BERT, ориентированную под задачу извлечения отношений (Relation Extraction) [20], посредством изменения алгоритма маскирования частей текста последовательности.

Архитектура классификационного слоя может быть также специфична для конкретной задачи. Например, для задачи извлечения отношений модель **R-BERT** [21] учитывает информацию об объектах отношения входной последовательности.

3. Используемые ресурсы

В подразделах 3.1–3.2 рассматриваются ресурсы, которые использовались для разметки коллекции с целью проведения опосредованного обучения моделей. Основная идея подхода состоит в следующем. Лексикон оценочной лексики RuSentiFrames [6] используется для автоматической разметки оценочных отношений в заголовках большой неразмеченной новостной коллекции. Извлечение отношений производится из заголовков, поскольку они обычно короче, содержат меньше именованных сущностей. Далее размеченные отношения в заголовках фильтруются и используются для разметки отношений внутри текстов новостей.

3.1 Лексикон фреймов RuSentiFrames

Лексикон RuSentiFrames-2.0 описывает оценки и коннотации, передаваемые предикатом в форме отдельного слова (существительного, глагола) или словосочетания. Структура фреймов включает в себя набор специфичных для предикатов ролей и набор различных измерений (характеристик) для описания фреймов. Для обозначения ролей семантические аргументы предикатов нумеруются, начиная с нуля. Для конкретного предиката Arg0 – это, как правило, аргумент (Agent), демонстрирующий свойства агента (активного участника) ситуации [22], в то время как Arg1 это объект (Theme).

В основной части лексикона представлены следующие *измерения*:

- отношение автора текста к указанным участникам (Roles);
- polarity – положительная или отрицательная оценка между участниками отношений;
- effect – положительный или отрицательный эффект для участников;
- state – положительное или отрицательное эмоциональное состояние участников, связанных с описанной ситуацией.

Все утверждения включают доверительную оценку, которая в настоящее время имеет два значения: 1 – утверждение почти всегда верно, или 0.7 – разметка по-умолчанию. Утверждения о нейтральной оценке, эффекте или состоянии участников не учитываются в лексиконе.

Табл. 1. Пример описания фрейма «Одобрить» в лексиконе RuSentiFrames
Table 1. An example of the «Approve» frame description in the RuSentiFrames lexicon

Измерения фрейма «Одобрить»	**Описание**
roles	A0: тот, кто одобряет A1: то, что одобряется
polarity	A0→A1 , pos, 1.0 A1→A0, pos, 0.7
effect	A1, pos, 1.0
state	A0, pos, 1.0 A1, pos, 1.0

[6] https://github.com/pytorch/fairseq/tree/master/examples/xlmr

Табл. 2. Распределение вхождений отношений в лексиконе RuSentiFrames-2.0
Table 2. Distribution of occurrences of relations in the RuSentiFrames-2.0 lexicon

polarity	**Класс тональности**	**Количество**
A0→A1	pos	2558
A0→A1	neg	3289
author→A0	pos	170
author→A0	neg	1581
author→A1	pos	92
author→A1	neg	249

Созданные фреймы связаны также с «семейством» слов и выражений (лексических единиц), которые имеют одинаковые тональности. Лексические единицы, связанные с фреймом, могут быть отдельными словами или словосочетаниями.

RuAttitudes-2.0 сохраняет общую структуру лексикона версии 1.0. В ресурсе описано 311 фреймов, связанных с 7034 лексическими единицами, среди которых 6788 уникальных. Среди уникальных вхождений, 48% составляют глаголы, 14% – существительные, и оставшиеся 38% – словосочетания. Общее число фреймов увеличено на 12% при сравнении с версией 1.0. Пример формата описания фрейма «Одобрить» приведен в Табл. 1. В Табл. 2 представлено распределение вхождений различных типов отношений в RuSentiFrames. Для проведения автоматической разметки в методе опосредованного обучения моделей используется только отношения агента ситуации к объекту (A0→A1).

3.2 Новостные коллекции

Коллекции $\text{NEWS}_{\text{Base}}$ (2,8 млн. новостных текстов) и $\text{NEWS}_{\text{Large}}$ (8,8 млн. новостных текстов), используемые для извлечения отношений, состоят из русскоязычных статей и новостей крупных новостных источников, специализированных политических сайтов, опубликованных в 2017 году. Каждая статья разделена на заголовок и содержание.

4. Описание подхода

Основные предположения подхода состоят в следующем:

- отношения между сущностями, упоминаемыми в новости, в большинстве случаев наиболее четко и просто выражаются в заголовке новости;
- появление предиката из RuSentiFrames (FRAME-BASED) в заголовке позволяет достаточно надежно извлечь отношения между именованными сущностями;
- суммирование выделенных отношений по большой коллекции позволяет выделить основную тональность отношений между сущностями (PAIR-BASED фактор);
- для формирования автоматически размеченной коллекции выбираются заголовки новостей, в которых тональность отношений между сущностями, выделенная на основе фреймов (FRAME-BASED) совпадает с насчитанной тональностью по коллекции для этих сущностей (PAIR-BASED) – так называемые доверенные отношения;
- в размеченную коллекцию также включаются предложения из тела новости с выбранным заголовком, поскольку предполагается, что в среднем тональность отношения между сущностями внутри новости соответствует тональности отношения в заголовке. При этом предложения из тела новости имеют более разнообразную структуру.

Полученный набор данных с автоматически размеченными оценочными отношениями получил название RuAttitudes2017. Рис. 1 иллюстрирует процесс автоматической разметки[7] новостной коллекции.

[7] https://github.com/nicolay-r/RuAttitudes/tree/v2.0

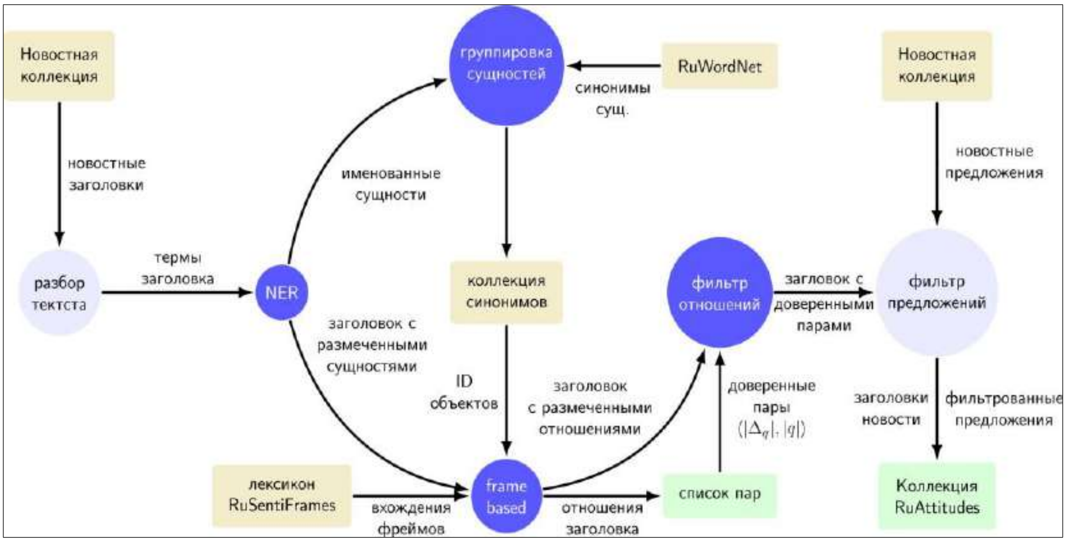


Рис. 1. Диаграмма рабочего процесса извлечения оценочных отношений; прямоугольники – источники информации; кружки – модули обработки потока данных; стрелки – передача информации между модулями с указанием её типа в подписи; для пары q, $|\Delta q|$ – абсолютная разница вероятностей принадлежности положительному и отрицательному классам ($|\Delta q| \in [0,1]$), и $|q|$ – число соответствующих отношений

Fig. 1. Diagram of a workflow for extracting value relationships; rectangles - sources of information; circles - data flow processing modules; arrows - transfer of information between modules with an indication of its type in the signature; for a pair q, $|\Delta q|$ is the absolute difference in the probabilities of belonging to the positive and negative classes ($|\Delta q| \in [0,1]$), and $|q|$ - the number of relevant relationships

4.1 Извлечение оценочных отношений из новостных статей

Процесс извлечения оценочных отношений включает выполнение двух последовательных этапов обработки новостной коллекции:

- этап 1 – автоматическое составление списка пар сущностей с превалирующей тональностью отношений из заголовков новостей из неразмеченного корпуса текстов;
- этап 2 – применение собранного списка пар сущностей с выявленной тональностью отношений для отбора достоверных отношений из новостных заголовков и текстов новостей для формирования автоматически размеченной коллекции RuAttitudes.

Рассмотрим общие компоненты потока обработки информации обеих этапов для заголовка некоторого документа новостной коллекции.

Модуль разбора текста подразумевает выполнение преобразования заголовка представленного последовательностью символов в последовательность термов. Содержимое заголовка разбивается на слова с выделением знаков препинания.

Модуль NER выполняет задачу извлечения именованных сущностей из последовательности термов. Для этого используется предобученная модель BERT$_{\text{Mult-OntoNotes}}$ библиотеки DeepPavlov . Модель обучена на коллекции OntoNotes [23], разметка которой включает 19 типов сущностей. Результатом такого модуля обработки является список Сущностей $E = [e_1, \ldots, e_{|E|}]$, каждый элемент которого представлен последовательностью термов и типом.

Модуль *группировки сущностей* использует множество E для пополнения списка синонимов. Пара сущностей $e_i, e_j \in E, i \neq j$ являются синонимами, если совпадают их *нормальные формы*. Для получения нормальной формы именованной сущности используется:

- лемматизированная форма значения (последовательности термов)[8];
- ресурс RuWordNet [24] для получения названия синонимичной группы лемматизированной формы (если значение найдено).

Табл. 3. Список доверенных пар, извлеченных из новостной коллекции NEWS$_{Large}$, при ограничениях $|\Delta_q| \geq 0.8, |A_q| \geq 150$, где q – произвольная достоверная пара; содержимое упорядочено по Δ_q; пары с одинаковым значением Δ_q упорядочены относительно $|A_q|$: по-убыванию ($\Delta_q > 0$), по-возрастанию
Table 3. The list of trusted pairs retrieved from the NEWS$_{Large}$ news collection, subject to the constraints |Δ_q| ≥ 0.8, | A_q | ≥ 150, where q is an arbitrary valid pair; content is ordered by Δ_q; pairs with the same value of Δ_q are ordered relative to |A_q|: descending (Δ_q> 0), ascending

A**0**	A**1**	$\|\Delta \boldsymbol{q}\|$	$\|\boldsymbol{Aq}\|_{pos}$
МВД России	Российская Федерация	1.00	256
Путин	Министерство внутренних дел	0.91	150
Канада	Украина	0.90	218
Пентагон	Украина	0.90	147
Порошенко	НАТО	0.88	244
Порошенко	Совет национальной безопасности и обороны	0.87	173
Путин	Макрон	0.86	186
Афганистан	Россия	0.85	166
Европейский Парламент	Украина	0.84	273
Украина	МВФ	0.80	204
Трамп	ИГИЛ	-0.79	24
Россия	ИГИЛ	-0.79	60
Гройсман	Донбасс	-0.82	23
Турция	ИГИЛ	-0.83	14
Россия	Siemens	-0.83	19
Израиль	ООН	-0.85	12
Азербайджан	Армения	-0.93	14
Карабах	Азербайджан	-0.94	10
ЕС	Siemens	-1.00	0

В результате можно автоматически сгруппировать такие синонимы, как: (США, Соединенные Штаты), (Россия, Российская Федерация, РФ).

Модуль FRAME-BASED выполняет задачу извлечения достоверных отношений из новостного заголовка с использованием лексикона RuSentiFrames. Для этого на первом шаге из последовательности термов извлекаются вхождения фреймов. Далее составляется множество достоверных пар сущностей. Пара $\langle e_i, e_j \rangle$, где $e_i, e_j \in E$ считается *достоверной*, если выполнены следующие условия:

- именованная сущность e_i упомянута раньше e_j;
- участники e_i и e_j не являются синонимами;
- участники и все именованные сущности между ними принадлежат множеству O_{valid}, которое включает: организации (ORG), людей (PER), геополитические сущности (GPE);
- для всех фреймов, входящих между участниками отношений, определена полярность типа A0→A1.
- отсутствуют предлоги[9] «в» и «на» перед участниками отношений.

[8] Применяется пакет Yandex Mystem

[9] Условие является результатом проведения дополнительного анализа ошибочных результатов.

Касательно условия п. 5, наличие предлогов «в» и «на» в большинстве случаев связано с месторасположением, которое обычно не является субъектом или объектом отношения: *Крым$_e$ бросит вызов Киеву$_e$: «в» ООН$_e$ представят резолюцию о преступлениях против людей «на» Украине$_e$, включающая убийства и похищения.*

Этап 1. Заполнение списка пар. Для некоторого новостного заголовка с множеством размеченных в нем оценочных фреймов, пусть P – множество извлеченных достоверных отношений (результат применения модуля FRAME-BASED, рис. 1). Тогда, каждая пара $\langle e_i, e_j \rangle \in P$ отправляется в *список пар* в следующем формате:

$$a = \langle d, g_i, g_j, l \rangle \qquad g_i, g_j \in G \tag{1}$$

где $d \in N$ – индекс документа рассматриваемого заголовка в новостной коллекции, $g_i, g_j \in N$ – индексы синонимичных групп участников в списке синонимов, а l – оценка пары, которая назначается следующим образом: pos (если для всех вхождений фреймов между e_i и e_j оценка A0→A1 одинакова, и равна pos), neg (иначе).

Таким образом, результирующий *список пар* (см. рис. 1) представляет собой множество достоверных пар $A = \{a_1, a_2, \dots a_{|A|}\}$, извлеченных из всех заголовков документов новостной коллекции.

Извлечение доверенных пар. Из составленного списка пар (см. рис. 1), представленного множеством A, можно выделить наиболее положительно и отрицательно ориентированные пары. *Ориентация* некоторой пары $q = \langle g_i, g_j \rangle$ к классу $c \in \{pos, neg\}$ вычисляется по формуле:

$$p(q|c) = \frac{\left|\{\langle g_i, g_j, l\rangle \middle| l = c\}\right|}{\left|A_q\right|} \tag{2}$$

где A_q– подмножество множества A, элементы которого соответствуют паре q. *Оценочная ориентация* пары q определяется по формуле:

$$\Delta q = p(q|pos) - p(q|neg) \tag{3}$$

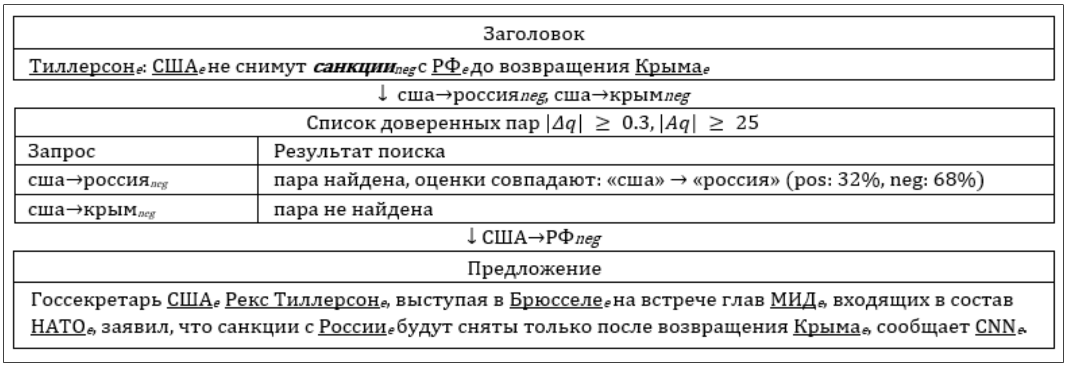
Результирующая оценка для q определяется на основе знака выражения формулы 3: pos ($\Delta_q > 0$), neg ($\Delta_q < 0$). Таким образом, для извлечения и составления множества доверенных пар A' необходимо задать пороговые значения для $|\Delta_q|$ и $|A_q|$. Формат представления доверенной пары q в множестве A' следующий:

$$q = \langle g_i, g_j, \Delta_q \rangle \tag{4}$$

В табл. 3 приведены примеры доверенных пар в результате анализа новостной коллекции NEWS$_{\text{Large}}$ при $|\Delta_q| \geq 0.8$ и $|A_q| \geq 150$.

Этап 2. Извлечение оценочных отношений. Для некоторого новостного заголовка, пусть P – множество извлеченных достоверных отношений (результат применения модуля FRAME-BASED, рис. 1). Модуль *фильтрации отношений* выполняет отбор оценочных отношений среди множества достоверных пар P. Пара $\langle e_i, e_j \rangle \in P$ считается оценочным отношением, если $\langle g_i, g_j \rangle$ содержится во множестве доверенных пар A' и оценка $\langle e_i, e_j \rangle$ совпадает с оценочной ориентацией доверенной пары.

Отобранные оценочные отношения далее передаются на вход модулю *фильтрации предложений* для поиска таких же отношений в предложениях новости. Оценочное отношение заголовка присутствует в предложении новости, если предложение содержит упоминание обеих участников. На рис. 2 рассмотрено применение второго этапа процесса извлечения оценочных отношений для заголовка: *«США$_e$ не снимут* ***санкции***$_{neg}$ *с РФ$_e$ до возвращения Крыма$_e$».*



Заголовок
Тиллерсон$_e$: США$_e$ не снимут ***санкции***$_{neg}$ с РФ$_e$ до возвращения Крыма$_e$

↓ сша→россия$_{neg}$, сша→крым$_{neg}$

Список доверенных пар $\lvert\Delta q\rvert \geq 0.3, \lvert Aq\rvert \geq 25$	
Запрос	Результат поиска
сша→россия$_{neg}$	пара найдена, оценки совпадают: «сша» → «россия» (pos: 32%, neg: 68%)
сша→крым$_{neg}$	пара не найдена

↓ США→РФ$_{neg}$

Предложение
Госсекретарь США$_e$ Рекс Тиллерсон$_e$, выступая в Брюсселе$_e$ на встрече глав МИД$_e$, входящих в состав НАТО$_e$, заявил, что санкции с России$_e$ будут сняты только после возвращения Крыма$_e$, сообщает CNN$_e$.

Рис. 2. Применение метода PAIR-BASED для извлечения достоверных пар из заголовка с последующим выполнением фильтрации отношений и поиском доверенных пар (США→РФ$_{neg}$) в предложениях новости

Fig. 2. Application of the PAIR-BASED method to extract valid pairs from the header, followed by filtering relations and searching for trusted pairs (USA → RF$_{neg}$) in news sentences

Для каждого документа новостной коллекции дополнительно проводится разметка нейтральных отношений. Для некоторого документа с множеством размеченных именованных сущностей E, пара $\langle e_1, e_2\rangle$, $e_1, e_2 \in E$ заголовка или предложения считается нейтральной, если выполнены следующие условия:

- сущность e_1 упомянута в тексте перед e_2 и имеет тип из множества O_{valid};
- сущность e_2 имеет тип LOC и не находится в списке стран/столиц;
- участники e_1 и e_2 не принадлежат одной синонимичной группе, а также отношения $\langle e_1, e_2\rangle$ и $\langle e_2, e_1\rangle$ не содержатся в разметке оценочных отношений.

4.2 Автоматическая разметка отношений и анализ результатов

Поэтапная оценка количества извлеченных данных в результате применения потока обработки (см. рис. 1) к новостным коллекциям приведена в таблице 4. Результатом применения подхода автоматической разметки новостных статей стали коллекции RuAttitudes2017, созданные в различных вариантах независимо в результате обработки NEWS$_{\text{Base}}$ и NEWS$_{\text{Large}}$. Рассмотрим подробнее каждый этап обработки новостных текстов.

На первом этапе список пар заполняется отношениями, которые были извлечены методом FRAME-BASED. Среди всех заголовков отбираются отношения, участники которых имеют тип из множества O_{valid}. Далее, процент отвергнутых отношений относительно такого числа составил 65%, где: 38% отношений без вхождений фреймов между сущностями, 12% отношений, для которых существуют вхождения фреймов с неопределенной полярностью A0→A1, и 15% с наличием предлогов «в» и «на». Таким образом, 35% отношений от начального количества были отобраны как «достоверные» и переданы в *список пар*.

На втором этапе производится фильтрация отношений из заголовков и предложений новостей (см. табл. 4). Для извлечения доверенных пар были выбраны параметры: $|\Delta q| \geq 0.3$, $|q| \geq 25$. В результате, 22-24% достоверных отношений из заголовков были сопоставлены с доверенными парами, среди которых 79% отношений совпадали с оценочной ориентацией соответствующих пар. Новости с такими отношениями в заголовках передавались на этап фильтрации предложений. Дополнительный выбор новостных предложений позволил увеличить объем разметки на 89%.

Объем нейтрально размеченных отношений составил 5-6% от общего числа оценочных отношений коллекций RuAttitudes2017. Расширенные версии коллекций получили названия 2017-Base и 2017-Large для NEWS$_{\text{Base}}$ и NEWS$_{\text{Large}}$ соответственно. Среди объектов таких пар, в большинстве случаев, к сущности типа LOC относятся: моря, озера, острова, реки, и т.д. (см. табл. 5).

Табл. 4. Количественная оценка данных автоматической разметки текстов новостных коллекций NEWS$_{Base}$ и NEWS$_{Large}$; выделенные зеленым цветом результаты соответствуют количественной оценке ресурсов, порожденных в результате обработки новостных коллекций двумя этапами и применения разметки нейтральных отношений

Table 4. Quantification of the data of automatic marking of texts of news collections NEWS$_{Base}$ and NEWS$_{Large}$; the results highlighted in green correspond to a quantitative estimate of the resources generated as a result of processing news collections in two stages and applying the markup of neutral relations

Этап	**Параметр**		
Коллекция	Тип новостной коллекции	NEWS$_{Base}$	NEWS$_{Large}$
	Документы	$2.8 \cdot 10^6$	$8.8 \cdot 10^6$
FRAME-BASED	Отношений с участниками между объектами	867481	2481426
	Отношений без фреймов между участниками	38%	39%
	Отношений без A0→A1	12%	12%
	Отношений, перед участниками которых предлоги «в» и «на»	15%	15%
	Отношений из заголовков	302319	843799
Список пар	Число пар	100329	247876
	Доверенных пар	887	2372
	$\langle |\Delta_q| \geq 0.3, |A_q| \geq 25 \rangle$	1%	1%
	Отношений сопоставленных с доверенными парами	65588	200009
		22%	24%
Фильтрация отношений заголовка	Извлечено	65588	200009
	- Разная оценка	13583	42627
	- Одинаковая оценка	21%	21%
		52005	157382
		79%	79%
Фильтрация	Извлечено предложений	39152	117791
RuAttitudes	Версия	2017-Base	2017-Large
	Новостей	44017	134442
	Отношений на новость	2.28	2.26
	Предложений на новость	0.89	0.88
Нейтральные отношения	Версия	2017-Base	2017-Large
	Добавлено отношений	5428	17790
		5.72%	6.23%
	Отношений на новость	0.12	0.13
	Отношений на предложение	0.03	0.03

Табл. 5. Примеры наиболее частотных, нейтрально размеченных отношений из корпуса RuAttitudes2017Large

Tab. 5. Examples of the most frequent, neutrally marked relations from the corpus RuAttitudes2017$_{Large}$

A0	**A1**	**Вхождений**	**Процент**
КНДР	корейский полуостров	301	1.7%
Россия	ближний восток	232	1.3%
США	Баренцево море	204	1.1%
Иран	ближний восток	189	1.1%
Япония	Курилы	172	1.0%
США	ближний восток	166	0.9%

РФ	Курилы	163	0.9%
Волгоград	река волга	155	0.9%
Правительство РФ	Волга	120	0.7%
Япония	Южный Курилы	115	0.6%
КНДР	Тихий Океан	103	0.6%
Сирия	Тивериадский Озеро	93	0.5%
НАТО	Североатлантический	92	0.5%
Гуам	Тихий Океан	79	0.4%
Израиль	Тивериадский Озеро	74	0.4%
Россия	Арктика	73	0.4%

5. Эксперименты

5.1 Корпус RuSentRel

Корпус представляет собой 75 больших аналитических текстов по международной политике с портала ИНОСМИ (insomi.ru), размеченных с выделением порядка 2000 оценочных отношений между упомянутыми в текстах сущностями. Табл. 6 приводит количественные данные корпуса по фиксированным разделениям документов на обучающее и тестовое множества. В текстах статей автоматически размечены именованные сущности по четырем классам: личности (PER), организации (ORG), места (LOC), геополитические сущности (GEO). Общее число размеченных именованных сущностей составляет 15.5 тысяч.

Разметка отношений поделена на два типа: (1) отношение автора к упомянутой именованной сущности; (2) отношения субъектов, переданное от одних именованных сущностей к другим именованным сущностям. Отношения фиксируются тройками, и рассматриваются не для каждого предложения, а для документа в целом. Оценка отношения может быть отрицательной (neg), либо положительной (pos); например: (Автор, США, neg), (США, Россия, neg). Нейтральные, а также отсутствующие отношения в корпусе не зафиксированы.

Табл. 6. Параметры корпуса RuSentRel-1.0 с фиксированным разбиением на обучающую и тестовые коллекции

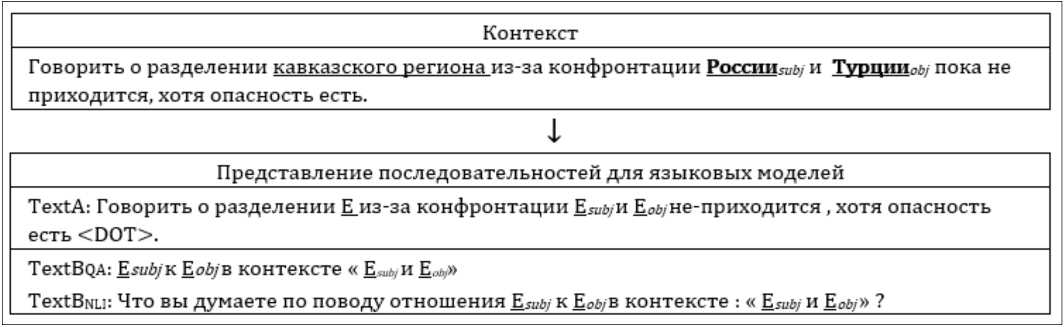
Table 6. Parameters of the RuSentRel-1.0 corpus with a fixed division into training and test collections

Коллекция	**Обучающая**	**Тестовая**
Документов	44	29
Предложений (ср./док.)	74.5	137
Упомянутых сущностей (*NE*) (ср./док.)	194	300
Сущностей (ср. на документ)	33.3	59.9
Положительных пар сущностей (ср./док.)	7.23	14.7
Негативных пар (ср./док.)	9.33	15.6
Расстояние между *NE* в предложении (в словах)	10.2	10.2
Нейтральных пар (ср./док.)	120	276

5.2 Описание эксперимента

Пусть задано подмножество документов коллекции RuSentRel, в котором каждый документ представлен парой: (1) текст, (2) список выделенных именованных сущностей *E*. Используя методы машинного обучения, для каждого документа требуется составить список оценочных отношений между парами сущностей множества *E*. Оценка отношения может быть отрицательной (neg), либо положительной (pos) (согласно п. 5.1). Составление списка выполняется в двух независимых экспериментах:

- *двуклассовый* [5] – необходимо определить оценки заведомо известных пар;
- *трехклассовый* – необходимо извлечь оценочные отношения из документа.



Контекст

Говорить о разделении кавказского региона из-за конфронтации **России**$_{subj}$ и **Турции**$_{obj}$ пока не приходится, хотя опасность есть.

↓

Представление последовательностей для языковых моделей

TextA: Говорить о разделении E из-за конфронтации E_{subj} и E_{obj} не-приходится , хотя опасность есть <DOT>.

TextB$_{QA}$: E_{subj} к E_{obj} в контексте « E_{subj} и E_{obj}»

TextB$_{NLI}$: Что вы думаете по поводу отношения E_{subj} к E_{obj} в контексте : « E_{subj} и E_{obj}» ?

Рис. 3. Пример обработки контекста в последовательности (TextA) и представлений вспомогательной информации (TextB) для подачи на вход языковым моделям BERT; для TextB используются форматы: задание вопроса (QA), вывод по контексту (NLI)
Fig. 3. An example of processing context in sequence (TextA) and representations of auxiliary information (TextB) for input to BERT language models; for TextB, the following formats are used: question asking (QA), output by context (NLI)

Описание подхода. Основное предположение о наличии оценочного отношения между парой сущностей в тексте документа – относительно короткое расстояние между ними. *Контекст* – ограниченный по длине фрагмент предложения, содержащий не менее двух именованных сущностей, в котором выделена пара $\langle e_s, e_o \rangle$ сущностей «субъект→объект». Таким образом, для некоторой пары сущностей можно составить множество контекстов. Контекст рассматривается как *оценочный*, если соответствующая пара $\langle e_s, e_o \rangle$, для которой такой контекст был составлен, присутствует в разметке документа.

Обработка и извлечение контекстов. Пример обработки контекстов для подачи на вход языковым моделям BERT приведен на рис. 3. Входная последовательность может состоять из одной (TextA) или двух последовательностей (TextA+TextB), соединенных разделителем. Если основная часть (TextA) используется для форматированного представления исходного контекста, то дополнительная последовательность TextB может быть использована для передачи вспомогательной информации. В работе рассмотрены следующие форматы входных последовательностей [12]:

Таким образом, процесс извлечения оценочных отношений может быть сведен к классификационной задаче на уровне контекстов с последующим отображением контекстных отношений на уровень документа. Оценка контекста с выделенным в нем парой $\langle e_s, e_o \rangle$ может быть отрицательной (neg), положительной (pos), или *нейтральной* (neu). Для отображения контекстных отношений на уровень документов используется вычисление среднего значения среди полученных оценок по всем контекстам рассматриваемого отношения *методом голосования* [25].

- C – использование последовательности без разделения (TextA);
- QA – дополнение TextA вопросом в TextB;
- NLI – дополнение TextA выводом отношения по контексту в TextB.

В случае нейронных сетей используется контекст без добавления вспомогательной информации. Для контекста применяются дополнительные преобразования: лемматизация термов, разметка знаков препинания, разметка вхождений фреймов [26, 27]. В целях устранения возможности принятия решения моделями на основе слов и словосочетаний сущностей и участников отношения, применяется *маскирование сущностей*. Используются следующие типы масок: E_{subj} (субъект и его синонимы), E_{obj} (объект и его синонимы), и E для остальных сущностей. Табл. 7 приводит количественные данные для извлеченных контекстов[10] из коллекций RuSentRel и RuAttitudes. В опосредованном обучении используются две версии размеченных корпусов: 2017-Base, 2017-Large.

Табл. 7. Число контекстов, извлеченных на этапе подготовки данных из коллекций RuAttitudes и обучающего множества коллекции RuSentRel; максимально допустимое число термов в контексте ограничено значением 50

Table 7. The number of contexts extracted at the stage of data preparation from the RuAttitudes collections and the training set of the RuSentRel collection; the maximum number of terms in a context is limited to 50

Коллекция	**pos**	**neg**	**neu**
RuSentRel (обучающее множество)	551	727	6530
RuAttitudes (2017-Base)	38809	55725	4723
RuAttitudes (2017-Large)	123281	161275	15429

Оценка качества разметки. Для некоторого документа коллекции, оценка качества разметки основана на подсчете метрик точности (P), полноты (R), и F_1-меры для каждого из оценочных классов в отдельности. Для оценки результата на множестве документов размера n фиксируется показатель F_{1-mean}^{PN}, который в свою очередь основан на вычислении макро-усреднений $F_{1-macro}$ над документами по каждому из оценочных классов в отдельности:

$$F_{1-macro}^{pos} = \frac{1}{n}\sum_{i=1}^{n} F_1^{pos}(i) \qquad F_{1-macro}^{neg} = \frac{1}{n}\sum_{i=1}^{n} F_1^{neg}(i) \tag{5}$$

$$F_{1-mean}^{PN} = \frac{\left(F_{1-macro}^{pos} + F_{1-macro}^{neg}\right)}{2} \cdot 100 \tag{6}$$

Форматы обучения моделей. Обучение моделей проводилось в следующих режимах:

- *обучение с учителем* – обучение на составленных контекстах оценочных отношений ручной разметки коллекции RuSentRel обучающего множества документов (см. таблицу 6);
- *применение опосредованного обучения* – обучение моделей на основе контекстов оценочных отношений коллекций RuSentRel (обучающее множество документов) и RuAttitudes.

Опосредованное обучение выполняется в форматах:

- *предобучение с последующим дообучением* – модели изначально обучались с использованием опосредованного обучения RuAttitudes, после которого следует *дообучение* контекстами коллекции RuSentRel;
- *объединенное обучение* – процесс обучения с объединенным набором данных коллекций RuAttitudes и RuSentRel (только нейронные сети);

Перед обучением применяется балансировка данных по числу контекстов классов тональности *методом дублирования* (Oversampling) для достижения объема, равного числу контекстов наибольшего класса[11].

Для объединенного обучения, алгоритм объединения зависит от формата оценки моделей. При *кросс-валидационном*, в каждом разбиении объединяется коллекция RuAttitudes с каждым обучающим блоком RuSentRel. При *фиксированном*, обучающий набор представляет

[10] Параметр, отвечающий за максимально допустимое расстояние в термах между участниками отношений контекста [26, 27] не рассматривается, так как такое ограничение оказывает влияние на результирующую разметку и отсутствие в ней некоторых контекстов.

[11] В случае объединенного обучения на коллекциях RuSentRel и RuAttitudes, балансировка применяется после объединения извлеченных контекстов обеих коллекций.

собой комбинацию RuAttitudes с фиксированным обучающим множеством документов RuSentRel.

Параметры обучения моделей. Для нейронных сетей измерение средних значений *точности* проводилось каждые 5 эпох. Оценка моделей производится на основе результатов последней эпохи обучения. Процесс обучения завершается в случае превышения лимита в 200 эпох. Для избежания проблем переобучения моделей предусмотрено использование механизма *dropout*. В качестве параметров нейронных сетей используются настройки работы [26]. Выбор коэффициента скорости обучения зависел от формата обучения: 0.1 (объединенное и предварительное обучение), 0.01 (дообучение). Предварительное обучение языковых моделей составляет 5 эпох. В качестве настроек обучения языковой модели используются параметры по-умолчанию [7], за исключением параметра прогрева модели (применение повышенной скорости обучения на начальном этапе). Значение такого коэффициента равно 1 на этапе предобучения модели, и 0.1 на этапе дообучения (по-умолчанию). Ограничение по длине входной последовательности выбрано в 128 токенов. Такое ограничение позволяет покрыть ≈95% примеров без проведения усечений длин контекстов.

5.3 Описание моделей и результаты их применения

Список моделей нейронных сетей, выбранных для экспериментов:

- **CNN**, **PCNN** – модели сверточных нейронных сетей [28];
- **AttCNN**$_e$, **AttPCNN**$_e$ – модели с кодировщиками на основе механизма внимания; e указывает на применения участников отношения ($\underline{E}_{obj}$, $\underline{E}_{subj}$) в качестве аспектов в механизме внимания [26];
- **LSTM**, **BiLSTM**, **Att-BLSTM** [26] – модели с кодировщиками на основе рекуррентных нейронных сетей LSTM [29].

Список используемых языковых моделей: **mBERT** [7], **RuBERT**, **SentRuBERT**.

Обучение с учителем и дообучение моделей исследовалось форматов {C, NLI, QA}, рассмотренных в п. 5.2. Предобучение моделей выполнялось только на контекстах, представленных в формате NLI (далее обозначено как NLI$_P$).

Результаты фиксировалась в следующих форматах:

- $F1^a_{cv}$ – усредненный показатель F^{PN}_{1-mean} в рамках 3-кратной кросс-валидационной проверки; разбиения проведены с точки зрения сохранения одинакового числа предложений в каждом из них;
- $F1_t$ – показатель F_{1-mean} на тестовом множестве (см. таблицу 6).

Для результатов моделей, обученных с применением опосредованного обучения ($F1_a$) и обучения с учителем ($F1_b$), оценка прироста качества опосредованного обучения подразумевает вычисление процентного соотношения по формуле:

$$\Delta(F1) = \left(\frac{F1_a}{F1_b} - 1\right) \cdot 100 \qquad (6)$$

Результаты нейронных сетей. В табл. 8 представлены результаты экспериментов для моделей нейронных сетей[12]. Средний результат по всем моделям при обучении с учителем приведен в последнем ряду таблицы. Результаты на фиксированном $F1_t$ выше, чем по метрике $F1^a_{cv}$ на 4% в случае двуклассового эксперимента на 10% при трехклассовой классификации. При дообучении моделей, прирост качества варьируется в диапазоне 2-5% и ≈0.4-7% для двух и трех-классовых форматов соответственно. При совместном обучении такой показатель увеличивается двое в случае двуклассовой классификации (5-9%) и более

[12] https://github.com/nicolay-r/neural-networks-for-attitude-extraction/tree/0.20.5

чем в 3 раза при трехклассовой классификации: ≈10.5% по $F1_{cv}$ и ≈23% по метрике $F1_t$. Наибольший прирост качества достигается при использовании RuAttitudes2017$_{\text{Large}}$.

Табл. 8: Результаты применения опосредованного обучения для моделей с кодировщиками на основе сверточных и рекуррентных нейронных сетей, а также моделей с механизмом внимания; результаты обучения с учителем отмечены прочерком в колонке «Версия RA»; наилучший результат по каждой модели выделен жирным шрифтом; результаты опосредованного обучения, превосходящие аналогичные при обучении с учителем отмечены подчеркиванием
Table 8. Results of the application of distant supervising for models with encoders based on convolutional and recurrent neural networks, as well as models with an attention mechanism; supervised learning outcomes are marked with a dash in the RA version column; the best result for each model is shown in bold; outcomes of distant supervising that are superior to those in supervised learning are underlined

		Дообучение				**Объединенное обучение**			
		Двуклассовая		**Трехклассовая**		**Двуклассовая**		**Трехклассовая**	
Модель	Версия RA	$F1^a_{cv}$	$F1_t$	$F1^a_{cv}$	$F1_t$	$F1^a_{cv}$	$F1_t$	$F1^a_{cv}$	$F1_t$
CNN	2017-Large	**68.2**	**69.8**	28.6	**36.1**	**70.0**	**74.3**	**32.8**	39.6
CNN	2017-Base	67.0	66.8	**29.8**	33.1	62.8	67.2	31.1	**40.3**
CNN	—	63.6	65.9	28.7	31.4	63.6	65.9	28.7	31.4
PCNN	2017-Large	66.1	**70.8**	29.8	32.1	**69.5**	70.5	31.6	**39.7**
PCNN	2017-Base	**66.9**	69.4	**30.5**	**33.6**	65.8	**71.2**	**31.9**	38.3
PCNN	—	64.4	63.3	29.6	32.5	64.4	63.3	29.6	32.5
LSTM	2017-Large	**69.9**	**70.4**	**30.5**	**33.7**	**68.0**	**75.4**	31.6	**39.5**
LSTM	2017-Base	66.1	64.6	27.6	32.7	65.2	69.9	31.5	37.2
LSTM	—	61.9	65.3	27.9	31.6	61.9	65.3	27.9	31.6
BiLSTM	2017-Large	62.1	64.0	28.4	**35.4**	**71.2**	68.4	32.0	38.8
BiLSTM	2017-Base	**65.6**	66.4	28.0	31.8	68.0	68.4	32.0	**39.5**
BiLSTM	—	62.3	**71.2**	**28.6**	32.4	62.3	**71.2**	28.6	32.4
AttCNN$_e$	2017-Large	**65.9**	**67.5**	28.0	**35.0**	66.8	72.7	30.9	**39.9**
AttCNN$_e$	2017-Base	62.6	65.7	**28.4**	33.5	**68.0**	69.2	**31.3**	37.6
AttCNN$_e$	—	65.0	66.2	27.6	29.7	65.0	66.2	27.6	29.7
AttPCNN$_e$	2017-Large	**66.8**	69.6	27.9	32.5	**70.2**	67.8	**32.2**	**39.9**
AttPCNN$_e$	2017-Base	63.3	**69.9**	**30.0**	34.8	68.2	68.9	31.8	38.9
AttPCNN$_e$	—	64.3	63.3	29.9	32.6	64.3	63.3	29.9	32.6
IAN$_e$	2017-Large	64.5	65.7	28.5	30.7	69.1	**72.6**	30.7	36.7
IAN$_e$	2017-Base	64.5	**66.9**	27.5	**33.5**	**69.8**	70.6	30.7	36.7
IAN$_e$	—	60.8	63.5	**30.8**	32.2	60.8	63.5	**30.8**	32.2
Att-BLSTM	2017-Large	**70.3**	67.0	**28.8**	33.3	66.2	**71.2**	31.0	37.3
Att-BLSTM	2017-Base	65.7	65.7	28.5	**33.7**	65.7	69.7	**31.8**	**40.1**
Att-BLSTM	—	65.7	68.2	27.5	32.3	65.7	68.2	27.5	32.3

Среднее $\Delta(F1)$	2017-Large	**+5.3%**	**+4.1%**	+0.4%	**+6.5%**	**+8.7%**	**+8.9%**	**+10.6%**	**+23.4%**
Среднее $\Delta(F1)$	2017-Base	+3.0%	+1.8%	+0.4%	+6.0%	+5.3%	+5.5%	+10.1%	+22.2%
Среднее	—	63.5	65.9	28.8	31.8	63.5	65.9	28.8	31.8

Языковые модели. Результаты экспериментов приведены в табл. 9. Средний результат по всем моделям при обучении с учителем приведен в последнем ряду таблицы. Сравнивая такие показатели с аналогичными результатами табл. 8, замена нейронных сетей на языковые модели повышает качество оценки на 7.8% ($F1^a_{cv}$) в двуклассовом, и на 9.7% ($F1^a_{cv}$) и 12% ($F1_t$) в трехклассовом. Смена формата обучения в языковых моделях на опосредованное оказывает прирост в ≈2-5% в двуклассовом, и 10-13% в случае трехклассового эксперимента. Преимущество в использовании русскоязычно-ориентированных моделей перед mBERT особенно наблюдается в трехклассовых экспериментах: дообученная версия RuBERT показывает наилучший результат при использовании формата NLI для TextB.

Табл. 9. Результаты применения опосредованного обучения для моделей BERT с различными представления контекста TextB; символ «P» указывает тип предобученной модели; результаты обучения с учителем отмечены прочерком в колонке «Версия RA»; наилучший результат по каждой модели выделен жирным шрифтом; результаты опосредованного обучения, превосходящие аналогичные при обучении с учителем отмечены подчеркиванием

Tab. 9. Results of applying distant supervising for BERT models with different TextB context representations; the "P" symbol indicates the type of pretrained model; supervised learning outcomes are marked with a dash in the RA version column; the best result for each model is shown in bold; outcomes of distant supervising that are superior to those in supervised learning are underlined

		Дообучение			
		Двухклассовая		**Трехклассовая**	
Модель	**Версия RA**	$\boldsymbol{F1^a_{cv}}$	$\boldsymbol{F1_t}$	$\boldsymbol{F1^a_{cv}}$	$\boldsymbol{F1_t}$
mBERT (NLIP + C)	2017-Large	68.9	67.7	**30.5**	31.1
mBERT (NLIP + C)	2017-Base	**72.9**	**71.5**	30.3	**37.6**
mBERT (C)	—	67.0	68.9	26.9	30.0
mBERT (NLIP + QA)	2017-Large	69.6	65.2	**30.1**	**35.5**
mBERT (NLIP + QA)	2017-Base	**74.4**	**71.4**	29.5	32.4
mBERT (QA)	—	66.5	65.4	28.6	33.8
mBERT (NLIP + NLI)	2017-Large	**69.4**	68.2	**33.6**	36.0
mBERT (NLIP + NLI)	2017-Base	69.2	**69.6**	31.1	**37.5**
mBERT (NLI)	—	67.8	58.4	29.2	37.0
RuBERT (NLIP + C)	2017-Large	**70.0**	**69.8**	35.6	35.4
RuBERT (NLIP + C)	2017-Base	68.2	68.4	34.9	35.6
RuBERT (C)	—	67.8	66.2	**36.8**	**37.6**
RuBERT (NLIP + QA)	2017-Large	**69.6**	68.2	34.8	37.0
RuBERT (NLIP + QA)	2017-Base	68.6	**68.5**	**38.0**	**39.1**
RuBERT (QA)	—	69.5	66.2	32.0	35.3
RuBERT (NLIP + NLI)	2017-Large	**71.0**	**68.6**	**36.8**	39.9
RuBERT (NLIP + NLI)	2017-Base	67.0	66.9	36.1	39.4
RuBERT (NLI)	—	68.9	66.4	29.4	39.6
SentRuBERT (NLIP + C)	2017-Large	70.0	69.8	37.9	**39.8**
SentRuBERT (NLIP + C)	2017-Base	**70.3**	68.1	**38.5**	39.0
SentRuBERT (C)	—	69.3	65.5	34.0	35.2
SentRuBERT (NLIP + QA)	2017-Large	69.6	64.2	**38.4**	**41.9**
SentRuBERT (NLIP + QA)	2017-Base	68.6	67.5	35.5	33.6
SentRuBERT (QA)	—	70.2	67.1	34.3	38.9

SentRuBERT (NLI$_P$ + NLI)	2017-Large	70.2	67.7	**39.0**	38.0
SentRuBERT (NLI$_P$ + NLI)	2017-Base	**70.6**	**69.0**	35.4	**40.6**
SentRuBERT (NLI)	—	69.8	67.6	33.4	32.7
Среднее-Δ(F1)	2017-Large	+1.8%	+3.7%	+**13.5%**	+10.0%
Среднее-Δ(F1)	2017-Base	+**2.8%**	+**4.6%**	+11.4%	+**11.7%**
Среднее	—	68.5	65.7	31.6	35.6

Модель SentRuBERT (NLI$_P$+ NLI) по качеству разметки сопоставима с качеством нейронных сетей объединенного формата обучения, при этом демонстрирует сохранение результата при переходе от фиксированного на кросс-валидационный формат тестирования (35.6-39.0). Такие результаты на 25% выше аналогичных результатов моделей нейронных сетей. Сохранение высоких оценок при разных форматах разбиения указывают на более высокую стабильность в результирующем состоянии в случае языковых моделей.

Оценка производительности языковых моделей. Обучение моделей проводилось на сервере с двумя процессорами Intel® Xeon® CPU E5-2670 v2 с частотой 2.50Ггц, 80 Гб ОЗУ (DDR-3), с двумя видеоускорителями Nvidia GeForce GTX 1080 Ti (11.2Гб); операционная система Ubuntu 18.0.4; обучение моделей выполнялось в контейнерах Docker версии 19.03.5. Применялись следующие параметры оценки: (1) общее время обучения модели; (2) общее число эпох. Оценка выполнялась при фиксированном формате разбиения документов[13].

Табл. 10. Оценка времени в трехклассовом эксперименте с фиксированным набором документов обучающей части коллекции RuSentRel при различных форматах обучения моделей; для языковых моделей приводится среднее время оценки по различным форматам представления входных данных
Table 10. Estimation of time in a three-class experiment with a fixed set of documents for the training part of the RuSentRel collection with various training formats for models; for language models, the average evaluation time is given for various formats of input data presentation

Модель	**Версия RA**	**с учителем Время|эпох**	**предобучение Время|эпох**	**дообучение Время|эпох**
Число используемых GPU		1	2	1
Контекстов в секунду		31	62	31
mBERT	2017-Large	—	8:40:14|04	00:10:32|14
mBERT	2017-Base	—	2:59:45|04	00:10:32|14
mBERT	—	00:10:32|35	—	—
Контекстов в секунду		54	62	54
RuBERT/SentRuBERT	2017-Large	—	6:30:11|03	00:06:10|7
RuBERT/SentRuBERT	2017-Base	—	2:14:47|03	00:06:10|7
RuBERT/SentRuBERT	—/1.0-Base	00:06:10|12	—	—

В табл. 10 приводится средняя оценка времени по каждому из форматов представления входных данных языковых моделей. Во всех форматах обучения на адаптацию русскоязычных моделей требуется меньше эпох при одинаковых настройках обучения: в 1.3 раза меньше на этапе дообучения, и в 2 раза меньше в остальных случаях. Замена mBERT на RuBERT или SentRuBERT сокращает время обучения в 3.5 раза.

5.5 Анализ влияния предварительного обучения на распределение весов механизма внимания в языковых моделях

Для анализа вклада различных элементов контекста в полученный результат часто производится сравнение весов механизма внимания. Для анализа были выбраны следующие состояния языковых моделей: mBERT, SentRuBERT и SentRuBERT-NLIP (предобученная версия SentRuBERT коллекцией RuAttitudes2017Large). Среди всего множества контекстов

[13] Временная оценка при проведении кросс-валидационного тестирования была опущена ввиду схожести оценок по каждому из разбиений.

рассматриваются только такие контексты, которые были извлечены дообученной моделью SentRuBERT (NLIP + NLI) из тестового множества коллекции RuSentRel. Таким образом, было проанализировано 1032 контекста. В контекстах дополнительно размечены вхождения лексикона оценочных слов русского языка RuSentiLex [30] (SENTIMENT) и вхождения фреймов (FRAMES).

Для каждого входного контекста длиной в s токенов, вектор весов внимания $a \in R^{l\times h\times s\times s}$ содержит значения каждого слоя, по каждой голове модели BERT (l – число слоев языковой модели; h – число голов). Для произвольного слоя l' и головы h', матрица $a_{l',h'} \in R^{s\times s}$ описывает веса связей токенов входных данных слоя l' с его выходными данными (токенами следующего слоя):

- [CLS] – класса;
- [SEP] – границ последовательностей;
- [S/O] – участников отношений (E_{subj}/E_{obj});
- группы FRAMES и внимание к ним остальных токенов контекста;
- группы SENTIMENT и внимание к ним остальных токенов контекста.

Табл. 11. Усредненная оценка вероятности внимания по головам языковой модели BERT по каждому из 12 слоев в отдельности для: токенов класса (CLS), разделителей (SEP), участникам отношения, всех сторонних токенов к FRAMES и SENTIMENT в отдельности; наибольшие значения в рядах отмечены жирным шрифтом

Table 11. The average estimate of the probability of attention by the heads of the BERT language model for each of the 12 layers separately for: class tokens (CLS), separators (SEP), relationship participants, all third-party tokens to FRAMES and SENTIMENT separately; the highest values in the rows are marked in bold

	номер слоя											
Группа термов	1	2	3	4	5	6	7	8	9	10	11	12
mBERT												
[CLS]	0.06	0.33	**0.36**	0.29	0.31	0.06	0.04	0.04	0.05	0.06	0.07	0.04
SEP	0.04	0.07	0.06	0.06	0.07	0.09	0.09	0.11	**0.12**	0.09	0.09	0.07
E*subj* / E*obj*	0.05	0.04	0.04	0.06	0.04	0.06	0.06	0.06	0.06	**0.07**	**0.07**	0.05
прочие→FRAMES	**0.07**	0.03	0.03	0.03	0.03	0.05	0.04	0.05	0.04	0.04	0.03	0.03
прочие→SENTIMENT	**0.08**	0.04	0.03	0.03	0.04	0.05	0.04	0.05	0.05	0.04	0.03	0.04
SentRuBERT												
[CLS]	0.03	0.27	0.33	0.30	**0.39**	0.09	0.02	0.03	0.03	0.05	0.04	0.02
SEP	0.05	0.06	0.03	0.04	0.04	0.15	0.22	**0.39**	0.28	0.29	0.07	0.04
E*subj* / E*obj*	0.10	0.06	0.07	0.07	0.05	0.06	0.08	0.04	0.06	0.05	0.11	**0.12**
прочие→FRAMES	0.05	0.03	0.03	0.03	0.03	0.04	0.04	0.03	0.05	0.05	**0.07**	0.06
прочие→SENTIMENT	0.06	0.03	0.03	0.03	0.03	0.04	0.04	0.05	0.06	0.06	**0.08**	**0.08**
SentRuBERT-NLI$_P$												
[CLS]	0.03	0.27	**0.36**	0.31	0.34	0.05	0.01	0.02	0.01	0.02	0.02	0.02
SEP	0.06	0.04	0.03	0.05	0.04	0.20	0.20	**0.28**	**0.28**	**0.28**	0.04	0.08
E*subj* / E*obj*	0.10	0.07	0.08	0.08	0.07	0.07	0.09	0.06	0.07	0.11	**0.28**	0.23
прочие→FRAMES	0.07	0.04	0.04	0.04	0.05	0.06	0.05	0.07	0.07	0.05	**0.10**	0.08
прочие→SENTIMENT	0.08	0.05	0.05	0.04	0.05	0.07	0.06	**0.09**	0.08	0.07	0.08	**0.09**

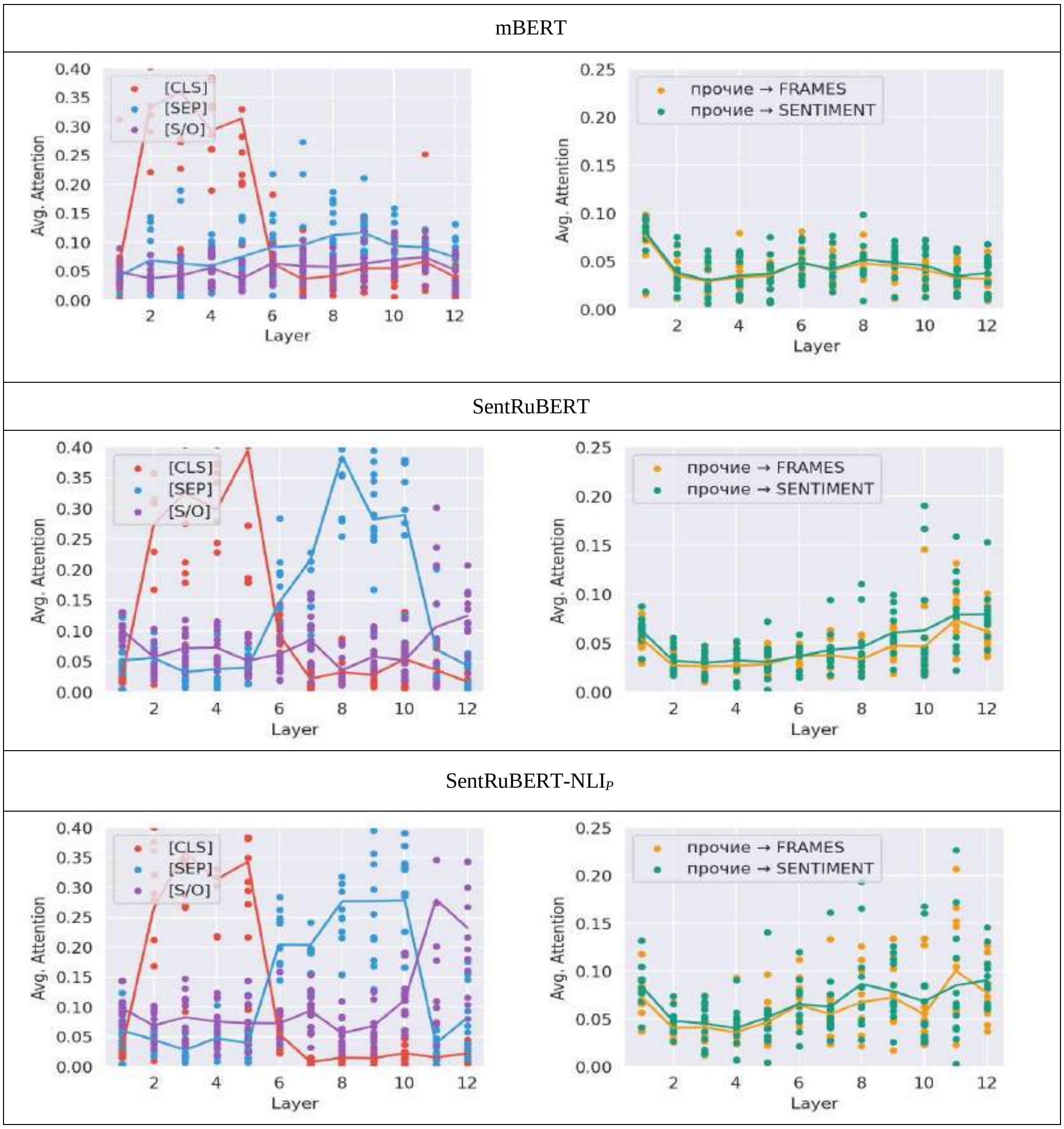


Рис. 4. Послойная оценка распределения внимания языковых моделей BERT к токенам [CLS], [SEP], объектам и субъектам отношения [S/O] (левая колонка) и фреймов и оценочных слов (правая колонка); линиями соединены средние значения весов каждого слоя модели [31]

Fig. 4. Layered assessment of the distribution of attention of the BERT language models to tokens [CLS], [SEP], objects and subjects of the relationship [S/O] (left column) and frames and evaluative words (right column); lines connect the average values of the weights of each layer of the model [31]

Рис. 4 иллюстрирует послойную оценку значений весов внимания к приведенным группам токенов. Средние значения по каждому слою указаны[14] в табл. 11.

[14] Для усредненных оценок к группам FRAMES и SENTIMENT учитываются только такие контексты, которые содержат хотя бы одно вхождение терма соответствующей группы. В результате 68% контекстов учитывалось при составлении оценки вероятности внимания для токенов «прочие→FRAMES», и 75% для «прочие→SENTIMENT»

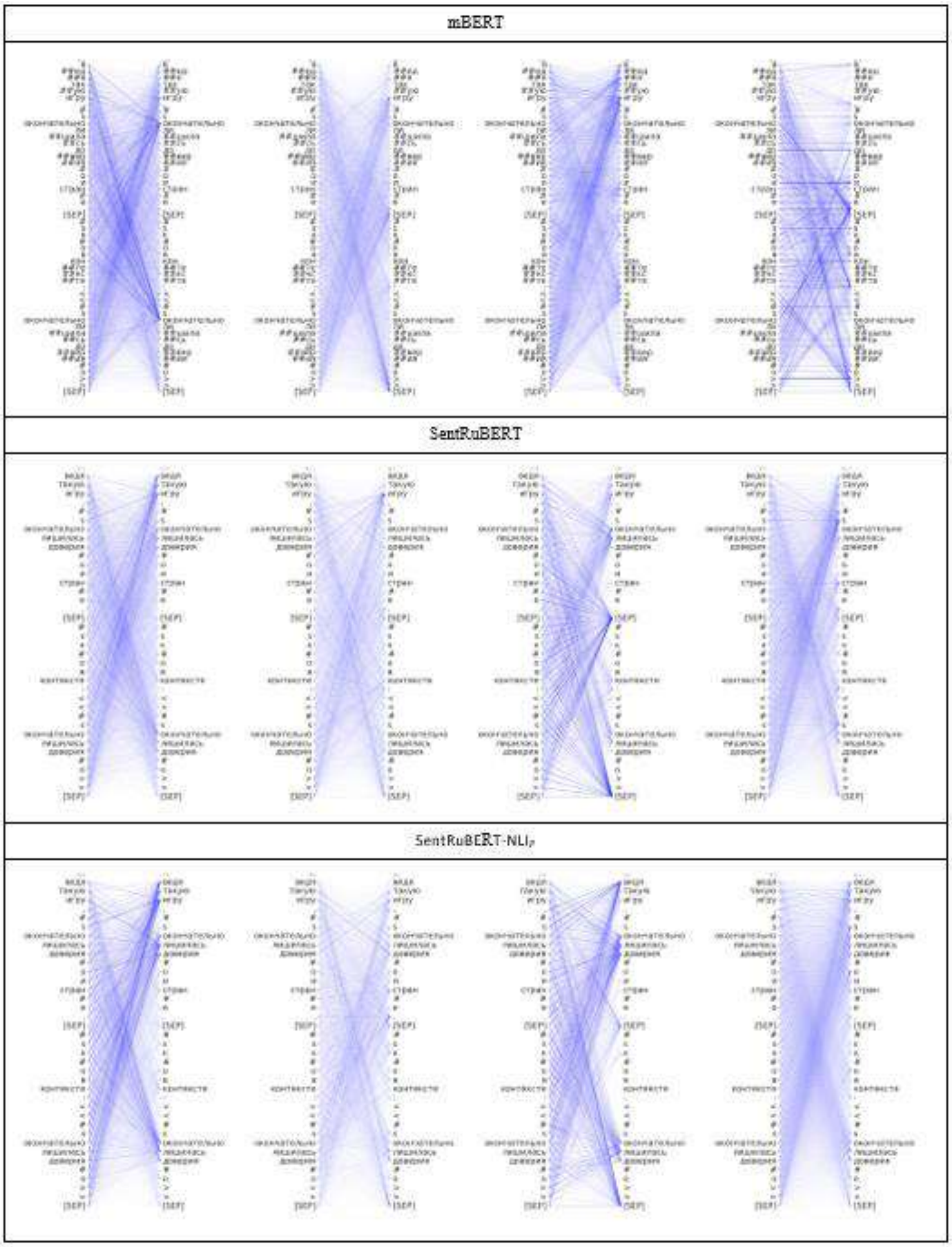


Рис. 5. Пример визуализации [31] весов головы №2 (слои слева-направо: 2,4,8,11) как эволюции внимания модели mBERT в процессе дообучения на примерах SentRuBERT и SentRuBERT-NLIP
Fig 5. An example of visualization [31] of head weights # 2 (layers from left to right: 2,4,8,11) as the evolution of attention of the mBERT model in the process of additional training using the examples of SentRuBERT and SentRuBERT-NLIP

Следует отметить высокие показатели внимания к токену класса [CLS] на слоях 2-5 до 35-40%. Для SentRuBERT наблюдается повышение внимания на токенах [SEP] (слои 7-10) и [S/O] (на конечных слоях). Также наблюдается повышение внимания к токенам FRAMES и SENTIMENT от прочих токенов на конечных слоях до 7-10%.

Применение опосредованного обучения (SentRuBERT-NLI*P*) повысило внимание к [S/O] на конечных слоях: весовые значения увеличились вдвое при сравнении с SentRuBERT. Отмечается также дополнительное повышение внимания к токенам SENTIMENT и FRAMES от прочих токенов на средних и конечных слоях.

В целях наглядной иллюстрации влияния дообучения на рис. 5 приведена визуализации весов головы №2 для каждой анализируемой модели BERT, по слоям (слева-направо) 2, 4, 8, 11 следующего примера: *Ведя такую игру,* $\underline{E}_{subj}$ *окончательно лишилась доверия* $\underline{E}_{obj}$ *и стран* $\underline{E}$*.* $\underline{E}_{subj}$ *к* $\underline{E}_{obj}$ *в контексте «*$\underline{E}_{subj}$ *окончательно лишилась доверия* $\underline{E}_{obj}$*»*. В модели SentRuBERT-NLI*P*, среди прочих, наиболее выражен фокус внимания ко вхождениям фреймов «окончательно» и «лишиться доверия» (слой 8).

6. Заключение

В данной работе предложен подход автоматического построения обучающей коллекции в задаче извлечения оценочных отношений из новостных текстов. Разметка основана на применении двух различных техник выделения оценочных отношений для взаимопроверки результатов. Первая подразумевает автоматическое порождение списка оценочных пар посредством предварительного анализа новостной коллекции. Вторая техника заключается в извлечении оценочных отношений из новостных заголовков на основе лексикона оценочных фреймов. В качестве дополнительного этапа предложен подход автоматической разметки нейтральных отношений.

Задача извлечения оценочных отношений рассматривалась как двуклассовая (положительные и отрицательные отношения) и трехклассовая (с введением нейтральных отношений) задачи классификации. Применение опосредованного обучения показало наибольший прирост качества в случае трех классовой классификации. Прирост качества при обучении языковых моделей BERT составил 10-13% по метрике F1 при сравнении с подходом без использования такой коллекции в обучении и на 25% при сравнении с аналогичными наилучшими результатами моделей сверточных и рекуррентных нейронных сетей.

Список литературы / References

[1]. N. Loukachevitch and N. Rusnachenko. Extracting sentiment attitudes from analytical texts. In Proc. of the International Conference on Computational Linguistics and Intellectual Technologies Dialogue-2018, 2018, pp. 459-468.

[2]. M. Mintz, S. Bills et al. Distant supervision for relation extraction without labeled data. In Proc. of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP, vol. 2, 2009, pp. 1003-1011.

[3]. R. Hoffmann, C. Zhang et al. Knowledge-based weak supervision for information ex traction of overlapping relations. In Proc. of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, 2011, pp. 541-550, 2011.

[4]. S. Vashishth, R. Joshi et al. RESIDE: improving distantly-supervised neural relation extraction using side information. In Proc. of the Conference on Empirical Methods in Natural Language Processing, 2018, pp. 1257-1266.

[5]. N. Rusnachenko, N. Loukachevitch, and E. Tutubalina. Distant supervision for sentiment attitude extraction. In Proc. of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019), 2019, pp. 1022-1030.

[6]. N. Loukachevitch and N. Rusnachenko. Sentiment frames for attitude extraction in russian. In Proc. of the International Conference on Computational Linguistics and Intellectual Technologies Dialogue-2020, 2020, pp. 541-552.

[7]. J. Devlin, M.-W. Chang et al. Bert: pre-training of deep bidirectional transformers for language understanding. arXiv preprint, arXiv:1810.04805, 2018.

[8]. A. Vaswani, N. Shazeer et al. Attention is all you need. In Proc. of the 1st Conference on Neural Information Processing Systems (NIPS 2017), 2017, pp. 6000-6010.

[9]. A. Radford, K. Narasimhan et al. Improving language understanding by generative pre-training, 2018. URL https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf.
[10]. T. B. Brown, B. Mann et al. Language models are few-shot learners. arXiv preprint, arXiv:2005.1416 5, 2020.
[11]. C. Alt, M. Hubner and L. Hennig. Improving relation ex traction by pre-trained language representations. arXiv preprint, arXiv:1906.03088, 2019.
[12]. C. Sun, L. Huang, and X. Qiu. Utilizing bert for aspect-based sentiment analysis via constructing auxiliary sentence. In Proc. of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, vol. 1, 2019, pp. 380-385.
[13]. Y. Kuratov and M. Arkhipov. Adaptation of deep bidirectional multilingual transformers for russian language. arXiv preprint arXiv:1905.07213, 2019.
[14]. S. R. Bowman, G. Angeli, C. Potts и C. D. Manning. A large annotated corpus for learning natural language inference. arXiv preprint, arXiv:1508.05326, 2015.
[15]. A. Conneau, G. Lample et al. Xnli: evaluating cross-lingual sentence representations. arXiv preprint, arXiv:1809.05053, 2018.
[16]. K. Clark, M.-T. Luong et al. Electra: pre-training text encoders as discriminators rather than generators. arXiv preprint, arXiv:2003.10555, 2020.
[17]. Y. Liu, M. Ott et al. Roberta: a robustly optimized bert pretraining approach. arXiv preprint, arXiv:1907.116 92, 2019.
[18]. A. Conneau, K. Khandelwal et al. Unsupervised cross-lingual representation learning at scale. arXiv preprint, arXiv:1911.02116, 2019.
[19]. M. Joshi, D. Chen et al. Spanbert: improving pre-training by representing and predicting spans. Transactions of the Association for Computational Linguistics, vol. 8, 2020, pp. 64-77.
[20]. I. Hendrickx, S. N. Kim et al. Semeval-2010 task 8: multi-way classification of semantic relations between pairs of nominals. In Proc. of the Workshop on Semantic Evaluations: Recent Achievements and Future Directions. 2009, pp. 94-99.
[21]. S. Wu and Y. He. Enriching pre-trained language model with entity information for relation classification. In Proc. of the 28th ACM International Conference on Information and Knowledge Management, 2019, pp. 2361-2364.
[22]. D. Dowty. Thematic proto-roles and argument selection. Language, vol. 67, no. 3, 1991, pp. 547-619.
[23]. R. Weischedel, M. Palmer et al. Ontonotes release 5.0. Linguistic Data Consortium, 2013. URL https://catalog.ldc.upenn.edu/LDC2013T19.
[24]. N. Loukachevitch, G. Lashevich, and B. Dobrov. Comparing two thesaurus representations for russian. в In Proc. of the Global WordNet Conference GWC, 2018, pp. 35-44.
[25]. N. Rusnachenko and N. Loukachevitch. Neural network approach for extracting aggregated opinions from analytical articles. Communications in Computer and Information Science, vol. 1003, 2018, pp. 167-179.
[26]. N. Rusnachenko and N. Loukachevitch. Attention-based neural networks for sentiment attitude ex traction using distant supervision. In Proc. of the 10th International Conference on Web Intelligence, Mining and Semantics (WIMS 2020), 2020, pp. 159-168.
[27]. N. Rusnachenko and N. Loukachevitch. Studying attention models in sentiment attitude extraction task. In Proc. of the 25th International Conference on Natural Language and Information Systems, 2020, pp. 157-169.
[28]. N. Rusnachenko and N. Loukachevitch. Using convolutional neural networks for sentiment attitude ex traction from analytical texts. EPiC Series in Language and Linguistics, vol. 4, 2019, pp. 1-10.
[29]. S. Hochreiter и J. Schmidhuber. Long short-term memory. Neural computation, vol. 9, no. 8, 1997. pp. 1735-1780.
[30]. N. Loukachevitch and A. Levchik. Creating a general russian sentiment lexicon. In Proc. of the Tenth International Conference on Language Resources and Evaluation (LREC'16), 2016, pp. 1171-1176.
[31]. K. Clark, U. Khandelwal et al. What does bert look at? an analysis of bert's attention. arXiv preprint, arXiv:1906.04341, 2019.

Информация об авторе / Information about the author

Николай Леонидович РУСНАЧЕНКО – аспирант кафедры «Теоретической информатики и компьютерных технологий» (ИУ-9) Московского государственного технического университета им. Н.Э Баумана. Область научных интересов: обработка естественного языка, анализ тональности сообщений, извлечение отношений.

Nicolay Leonidovich RUSNACHENKO – PhD student of «Theoretical Informatics and Computer Technologies» (IU-9), Bauman Moscow State Technical University (BMSTU) (Moscow, Russia). Graduated from BMSTU in 2016 (master degree). Scientific interests: computational linguistics, sentiment analysis, information retrieval.