Date of publication DEC-31, 2025, date of current version DEC-21, 2025.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.24.4.4338

A Method for Scaling Ontological Rule Reasoning for Adaptive Software

ILLIA LUTSYK, DMYTRO FEDASYUK
Department of Software, Lviv Polytechnic National University, Lviv, 79000, Ukraine

Corresponding author: Illia Lutsyk (e-mail: illia.i.lutsyk@lpnu.ua).

ABSTRACT A method for increasing the speed of the ontological rule reasoning process for adaptive software based on the proposed scaling method is presented. Modern research on the use of scaling approaches in the process of software design and development is analysed. In accordance with the analysis, it was found that the use of a combination of horizontal and vertical software scaling approaches provides better efficiency and speed of the software complex. Based on the considered software scaling approaches, a method of horizontal scaling of the reasoning process of processing rules for the software adaptation process is proposed. The designed method allows to distribute one large knowledge base into several according to a certain criterion (type of software component or system), which will optimize the process of designing adaptive software. The results of an experimental study of the proposed method are presented, demonstrating an increase in the speed of configuration determination: for an ontological model with the number of instances of 3300 and more, the speed of processing rules increased by 40%.

KEYWORDS adaptive software, software scaling, ontology, semantic reasoning, software architecture.

I. INTRODUCTION

Increasing demands on software and the growing computing capabilities of hardware and software systems are placing greater strain on software components. Complex software systems require constant monitoring of system resource usage, as a failure of a computing node can result in a complete or partial failure of the software application [1], [2]. As a result, there is a problem of effective use of methods of response and adaptation to changes in network traffic, taking into account the architecture of the software system [3].

Special attention should be paid to changes in server configuration and hardware/software settings during the design of adaptive software. Such software applications require not only an effective response to the growing number of adaptation requests, but also a corresponding increase in latency and the time required to generate a modified software application configuration [4].

To create intelligent mechanisms for software adaptation, researchers propose various approaches and methods [5]. In particular, to ensure the accuracy and flexibility of processing various data in the process of adapting information systems, the use of neural networks based on fuzzy logic is proposed [6].

The use of approaches based on machine learning methods and fuzzy logic allows for the adaptation of existing elements of the software system. However, when requirements change and new components are added, the problem of recompiling and static reconfiguration of both the system and the model arises. The solution to this problem is to use ontological models and rules that allow dynamic filling of information about changes in requirements in the subject area [4], [7].

One promising direction for optimizing the performance of complex systems and computations is resource scaling, which is based on the concept of classical control theory and allows for optimizing load distribution and reducing time delays [8]. In addition, the use of scaling approaches allows to control the costs required in the process of processing requests, since it allows not only to increase the number of resources but also to reduce computing power when the number of active users decreases.

Taking into account the indicated problems, the important task is to study methods for scaling the reasoning process of ontological rules of adaptive software to the growth of network traffic, in order to optimize the process of forming a dynamic configuration of a software application.

II. OVERVIEW OF THE SOFTWARE SCALING PROBLEM

The growth of network traffic requires an effective response to ensure the stability and operability of software, which consists in applying various approaches and methods for scaling it. Scalability is essentially defined as the property of an information system or software that allows it to handle an increasing volume of tasks by adding resources to the system according to predefined approaches [9]. The advantage of a software system being scalable is that it allows it to process all incoming requests without loss of performance and without creating additional delays.

It should be noted that the scalability of the software system must be taken into account during the design phase [10][11]. After all, if the software application architecture is designed incorrectly, scaling may require additional changes (which will cause delays in development or losses due to re-release and improvement of the software at the implementation stage) or may not be applicable at all if the system does not meet the requirements [12], [13].

The use of scaling is not uniform for all types of software. These methods require modification depending on the issues and requirements for increasing network traffic throughput. However, in general, methods for scaling software and hardware system resources can be divided into two types [14]:

– Horizontal scaling, which involves increasing the number of physical resources (Fig. 1) [15]. When using this scaling method, new computing nodes with identical or different computing capabilities are added to the existing system. However, it should be noted that in this case, it is necessary to use specialized load balancers to evenly distribute network traffic between the created nodes, [16].

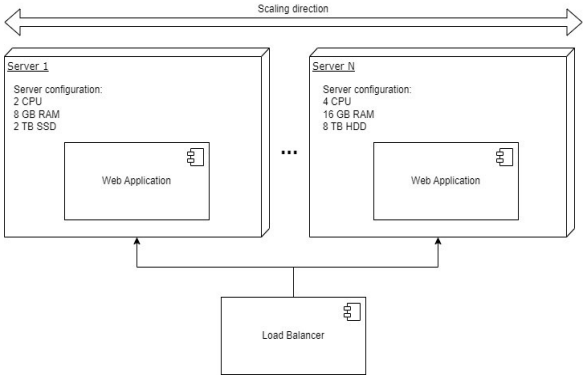

Figure 1. Horizontal scaling diagram of the software system

– Vertical scaling, which involves increasing or improving virtual and computing resources (Fig. 2) [17]. To implement this method, changes to the hardware properties of the corresponding server are required: upgrading the central processing unit, increasing the amount and size of RAM, and increasing the size of the system memory. This method is most often used when horizontal scaling is not possible, [18].

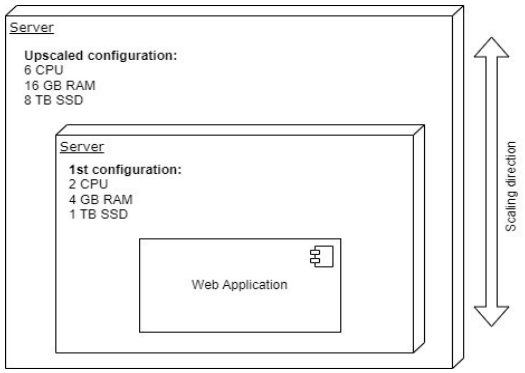

Figure 2. Vertical scaling diagram of the software system

Each of these methods is appropriate for use in different situations. If the architecture of the software system is modular or allows for the isolation and distribution of individual services, then horizontal scaling is appropriate. In addition to increasing throughput, this will also increase the availability of the software application, since in the event of a single node failure, the load balancer will systematically redirect all requests to the working nodes. However, if it is not possible to effectively separate functionality or there is a slight increase in network traffic, then it is advisable to use vertical scaling, since this method increases the resources of individual components [19].

Scaling methods are used for software in various application areas. In particular, [20] presents a framework for developing distributed software systems for bioinformatics. The authors presented a flexible platform that improves the creation and deployment of multi-stage workflows optimized for high-performance computing clusters and clouds. At the same time, thanks to the use of scaling approaches, the software implements the ability to modify internal processes to the needs of researchers.

Scaling and parallelization approaches are effectively used in high-performance cloud computing and systems. Researchers note that modern approaches to big data analysis require a transition to high-performance software systems [21]. However, traditional parallelization methods do not provide the necessary performance. Taking these problems into account, the article presents a new method for scaling software systems. The authors note that their proposed method not only reduces the use of hardware resources but also significantly speeds up the operation of the big data analysis system.

The advantages of using scaling approaches are also present in the process of reconfiguring distributed component-based software [22]. In the proposed review, the authors note that one of the effective methods for designing distributed and service-oriented systems is the use of component-oriented architecture. However, in addition to the correctness of reconfiguration, the time required to process the corresponding requests should also be taken into account during development.

Scaling techniques are widely used in various types of software. However, the question of applying these approaches when designing software systems based on ontological models remains open [23], [24].

In particular, the work [25] focuses on creating a framework for designing large-scale ontological models. The authors note that existing methods rely on manual work by experts in the subject area. This, in turn, makes the process labor-intensive, error-prone, and impractical for large, dynamic areas of knowledge. In addition, such solutions limit the adaptability and scalability of the designed models to new domains [26].

The paper [27] presents possible techniques for scaling ontological models. In particular, the authors identify two main approaches to scaling ontological models based on information agents: domain ontology agents and distributed domain ontology agents. This division allows ontological models to be controlled dynamically with the possibility of further expansion.

Considering the presented problems and prospects of using approaches and mechanisms for scaling software, the purpose of this article is to present a designed method for horizontal scaling of the ontological rule reasoning process. The proposed method improves the speed of the adaptation process and allows requests to be distributed by type and adaptation criteria.

III. ONTOLOGY REASONING HORIZONTAL SCALING METHOD FOR SOFTWARE ADAPTATION PROCESS

A. APPLICATION OF THE HORIZONTAL SCALING APPROACH FOR ONTOLOGICAL MODELS

In accordance with the software adaptation process, the determination of system characteristics is based on the classes, relationships, and properties specified in the ontological model [28]. Using the knowledge base of the designed ontological model, the user will receive the current software configuration depending on changing requirements or needs [29].

Integrating this process allows to dynamically change the content and functionality of the software, taking into account predefined conditions and ontology triggers. In the generalized case, the process of determining dynamic software characteristics based on SWRL rules can be divided into the following stages:

1. Defining the structure and content of a software ontology;
2. Formulation of adaptation requirements;
3. Creating SWRL adaptation rules;
4. Integrating rules into ontology;
5. Software validation, testing and support.

Using and following these steps for designing and implementing ontology rules will ensure that SWRL rules can be used in the software adaptation process. In addition, this process will ensure that the ontology remains compatible and consistent with changing requirements.

However, using and processing SWRL rules during software adaptation can be a resource-intensive process. The increase in the number of ontology elements such as concepts, relationships, and properties significantly affects the runtime of the semantic decision-making engine. In addition, most engines process semantic rules in a single-threaded and sequential (processing one statement at a time) mode. This implementation also contributes to an increase in the time required for ontological rule reasoning and system resource usage.

This problem can be solved by dividing one ontology into several parts. In this case, the number of elements in the ontological model will be significantly reduced, which will help improve the processing time of the rules. The division of the ontology processing can be carried out, in particular, using the following methods:

– vertical scaling, which involves adding resources to the system to keep system performance up to demand;
– horizontal scaling, which allows a single ontology file to be divided into parts for further processing in multithreaded and parallel modes.

According to the definition, vertical scaling involves increasing and improving system resources, which usually requires service overload. However, this is not an optimal solution, since such scaling will need to be performed each time the number of entities in the ontological model increases. An alternative solution is horizontal scaling and dividing a single ontological model into several submodels that meet certain criteria. Therefore, in our opinion, it is more expedient to use the horizontal scaling method, which allows solving the problem without unnecessary material costs already at the stage of software architecture design.

Therefore, in the case of adaptive software, division using the horizontal scaling method can be carried out in two approaches:

1. Based on component type – unique objects are selected by component type: functional and graphical. With this division, information about the system and users is duplicated, but a separate ontology file will contain only information about functional or graphical components.

2. Based on system type – unique system objects are selected using their type: mobile application, web application, or desktop application. With this division, each separate ontology file will contain only information about a specific application and its available components.

However, the specified methods of horizontal scaling have a number of disadvantages. In the case of dividing the ontology based on the type of components, there is a problem in data synchronization between parts, since user information is duplicated. In addition, if the system is aimed at adapting only functional or only graphical components, the problem of high system resource utilization and slow rule processing will remain.

The difficulties of synchronization and resource utilization are solved by dividing resources based on system type. This method preserves up-to-date information about users and system elements and allows rules to be processed on a compact set of elements. It should be noted that a disadvantage of this method may be the distribution of ontology into a large number of model files if the adaptive software supports several types of systems. This will complicate the process of determining the necessary part of ontology during adaptation.

B. METHOD OF AUTOMATED DIVISION OF AN ONTOLOGICAL MODEL

Considering the peculiarities of designing dynamic adaptive software systems, there is a need to apply scaling methods to improve performance. As noted in the previous section, in the case of systems based on an ontological model, horizontal scaling is more effective. In turn, this solution requires the definition of the correct criteria for the effective division of a single model into several submodels. For adaptive software systems, it is advisable to carry out such a division based on the type of adapted system. In this case, not only is the speed of adaptation optimized, but it also becomes possible to distribute the process of processing ontological rules across several services.

Classic approaches to horizontal and vertical scaling involve expanding an entire node or software service. In the case of ontological models, classic horizontal scaling does not solve the performance problem, since the size of the model remains unchanged. In addition, the execution of ontological rules and reasoning is a single-threaded process, which makes it impossible to expand correctly in accordance with classic approaches.

Therefore, according to the method we propose, which is based on a horizontal scaling approach, the ontological model is divided according to the type of system: web, mobile, and desktop application. Since the ontological model is single and indivisible in the initial version, entities can be distributed in two ways: manually and automatically. The automated approach not only allows to select the necessary instances of entities and properties of the subject area faster, but also reduces the complexity of this process, since it does not require the involvement of additional specialists to process the ontology.

Thus, the automated approach to distributing instances of entities and properties of the ontological model, the diagram of

which is shown in Fig. 3, is implemented according to the following sequence of steps:

1. Determination of the division criterion – in the case of adaptive software, it is advisable to select the type of software system as the division criterion;
2. Creation of submodels of the subject area ontology. Under these conditions, data processing occurs in parallel to reduce the processing time of ontological records:
   a. determining the records that are subject to the separation criterion for a specific type of software system;
   b. removing elements that are not included in the search results;
   c. saving the results obtained in the form of a submodel for a specific type of software system.

The division criterion in this case is the type of software system, which allows to reduce the duplication of information about the adaptive software system for each submodel. In addition, in the case of division by component type, there would be a problem of constant synchronization of submodels and ensuring the correct resolution of version conflicts. Such a division would not reduce the total size of the knowledge base, which would reduce the overall scaling efficiency.

Thus, the use of the proposed ontological model division provides the ability to perform automated ontological model division during the initial deployment or actual operation of the database and knowledge service. Accordingly, if the main model changes in structure or content, we will be able to reformat submodels according to the division criterion in real time without the need to stop the service.

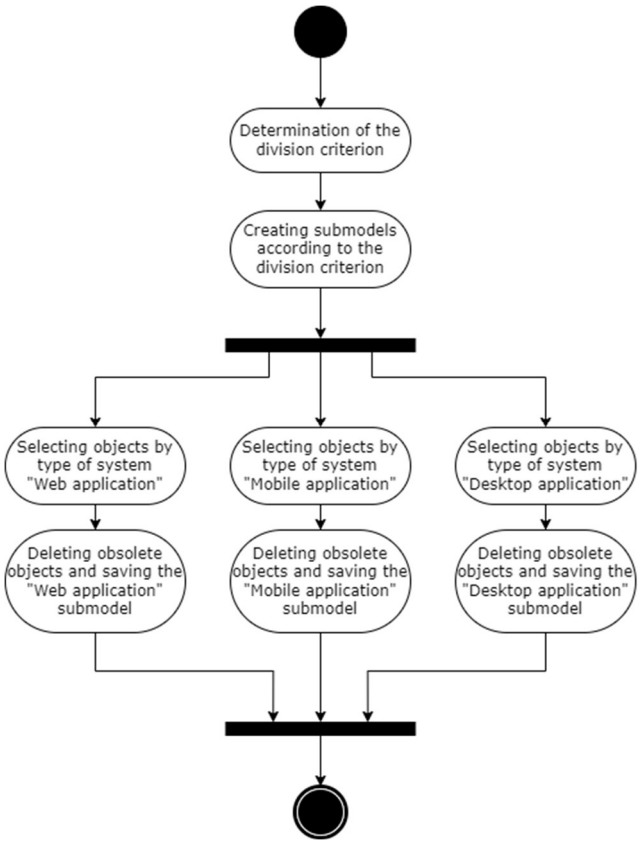

Figure 3. Scheme of the method of automated division of the ontological model by the criterion - "type of software system"

An example of automated ontology model partitioning using Python programming language tools and Owlready2 technology is shown in the code snippet:

```
from owlready2 import *
def scale ( ontology_path : str , scaling_type : str ):
    onto = get_ontology ( ontology_path ). load ()
    for indiv in onto [" Software_System "]. instances ():
        if scaling_type not in indiv.AssemblyVersion :
            destroy_entity ( indiv )
    onto . save ("./ Adaptive_system_v2_web . owl")
    return onto
onto_web = scale ("./ Ontology . owl" , "web")
onto_mobile = scale ("./ Ontology . owl" , "mobile ")
onto_desktop = scale ("./ Ontology . owl" , "desktop ")
```

IV. CASE STUDY

In previous works, we investigated the implementation of adaptive software systems based on the use of an abstract approach to the design of ontological models. [28], [29]. The approach we proposed made it possible to speed up the process of adapting and processing ontological rules by identifying abstractions of objects in the subject area. However, as noted earlier, the rule processing process is synchronous and single-threaded. This means that despite the relatively high speed of adaptation, the main thread of the adaptation process will be blocked each time a request to update the configuration is received. The solution to this problem is horizontal scaling.

In order to determine the indicators of the duration and speed of adaptation for generating software settings, the basic ontological model of a software system to assist people with cognitive impairments [28], presented in Figure 4, was used.

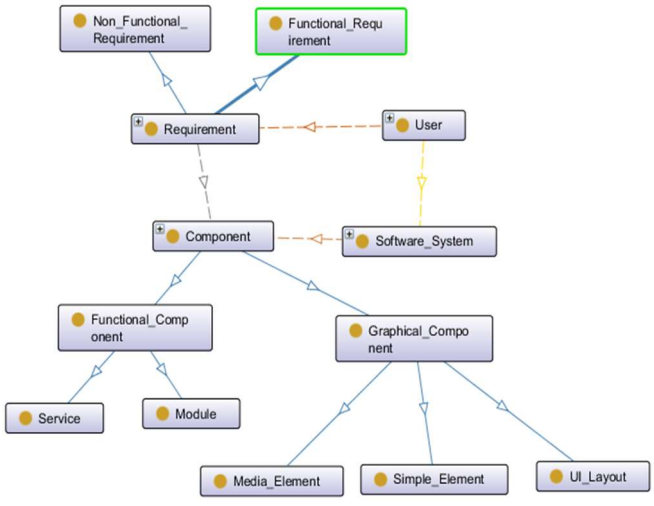

Figure 4. Ontological graph of a model designed based on an abstract approach.

To analyse the developed method for determining software system settings based on an abstract approach using horizontal scaling, a series of experiments was conducted using the developed software system prototype. Five devices were used to determine the adaptation duration, which allowed generating configurations for different execution environments. The

duration of adaptation was determined from the moment the request was sent to the web service to the final application of the obtained parameters. In the process of defining a new configuration, the ontological model generates, based on rules: parameters and settings for system components (font settings, styles), selection of new functionality based on registered modules, and resources for loading elements and settings for complex graphical interface elements. The system was tested on each device 5 times to determine the worst adaptation time depending on the number of entities in the ontological model.

The division of the model was carried out on the basis of the following types of software: web application, mobile application, desktop application. As a result of this division, configurations, components and requirements were formed in the submodels that relate only to the previously defined type of software. The results of the duration of processing and reasoning of ontological rules for the studied methods are presented in Tables 1 - 2.

Table 1. Duration (t, s) of determining software system settings using an abstract approach

Number of objects, n	100	150	250	450	850	1650	3300
time, t1 (c)	2,30	2,73	2,66	3,24	3,23	3,99	6,55
time, t2 (c)	2,20	2,21	2,82	3,05	3,47	4,39	6,96
time, t3 (c)	2,52	2,23	2,61	2,80	3,64	4,40	6,66
time, t4 (c)	2,55	2,85	2,75	2,94	3,59	4,32	6,59
time, t5 (c)	2,65	2,41	2,68	2,96	3,34	4,46	7,47
Average time, t_{avg} (c)	2,44	2,49	2,70	3,00	3,45	4,31	6,85

Table 2. Duration (t, s) of determining software system settings using an abstract approach with horizontal scaling

Number of objects, n	100	150	250	450	850	1650	3300
time, t1 (c)	2,12	2,82	2,61	2,50	2,80	2,90	4,37
time, t2 (c)	2,45	2,18	2,19	2,60	2,65	2,85	4,05
time, t3 (c)	2,16	2,25	2,25	2,45	2,64	2,82	3,90
time, t4 (c)	2,20	2,36	2,26	2,45	2,70	2,85	4,31
time, t5 (c)	2,35	2,16	2,40	2,55	2,65	3,10	4,21
Average time, t_{avg} (c)	2,26	2,35	2,34	2,51	2,69	2,90	4,37

A comparative analysis of the results of the duration of the method for determining the settings of the software system for the abstract approach, as well as for the abstract approach using horizontal scaling of the ontological model (Fig. 5) proves the higher efficiency of the method for determining the settings using the horizontal approach. At the same time, it was noted that this indicator increases with an increase in the number of ontology elements.

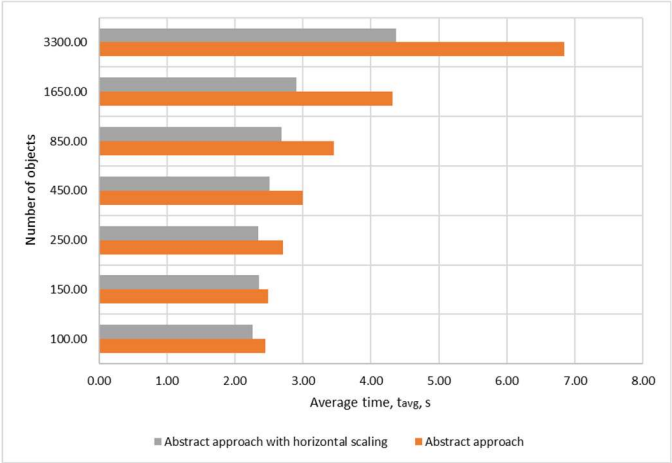

Figure 5. Comparison of approaches to determining optimal system settings

To track the noted trend, a comparative analysis of the results of the speed values of the process of determining software system settings based on ontological rules and relationships was additionally carried out (Table 3).

Table 3. Speed (instances/s) of determining software system settings for the proposed approaches

Number of objects, n	Adaptation speed (instances/s)	
	Abstract approach	Abstract approach with horizontal scaling
100	40.92	44.33
150	60.34	63.72
250	92.46	106.75
450	150.10	179.28
850	246.03	316.22
1650	382.65	568.18
3300	482.03	755.61

Thus, it has been established that both approaches are characterized by a tendency toward increased speed in determining software system settings with an increase in the number of elements and concepts in the ontological model (Fig. 6).

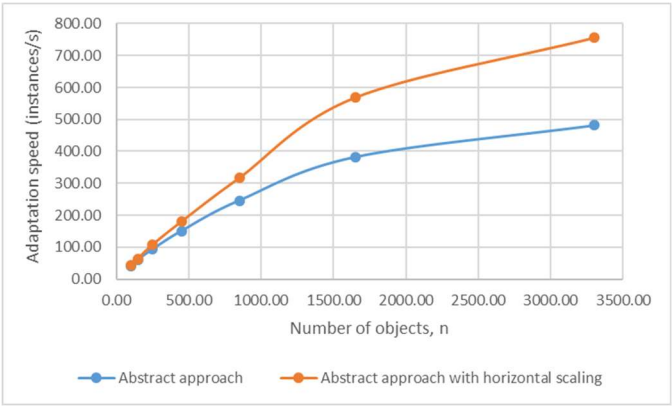

Figure 6. Speed of determining software system settings

The increase in speed and reduction in processing time of ontological rules for the abstract approach with horizontal scaling is explained by the reduction in the number of entity instances. The use of the method of automated division of ontological models based on the type of software system made

it possible to clearly define the context and discard instances that do not participate in the adaptation process.

It should be noted that removing instances does not affect the overall accuracy of rule processing and the final adaptation. The ontological approach assumes a clear definition of the context and a semantic description of the relationships between the concepts of the subject area. In this case, ontological rules will always contain correct information about the available adaptive components and provide the same qualitative result regardless of the size of the model.

V. DISCUSSION OF RESULTS

To identify the cause-and-effect relationship between the number of ontology elements (entity instances) and the duration of adaptation, the regression analysis method was used, which allows us to identify the influence of the factor feature (number of ontology elements) on the resulting feature (duration of adaptation). The basis was taken from experimental data based on the results of determining the duration of adaptation of the software system for different numbers of entity instances, which are given in Tables 1 and 2. A graphical representation of the regression dependencies is shown in Fig. 7.

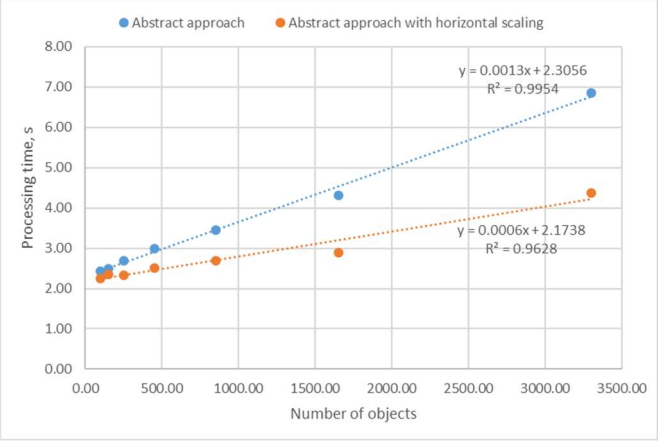

Figure 7. Results of regression analysis of the relationship between the duration of processing configuration requests and the number of ontological model elements

Having analysed using regression analysis the data on the duration of adaptation from five series of tests for the same set of entity instances according to the classical and abstract approaches, as well as the abstract approach using horizontal scaling, it was found that in all cases with a reliable approximation $R^2 = 0.96...0.99$ we obtain linear dependencies (1) – (2).

$$t_{abst\_apr}(x) = 0.0013N + 2.31,\ R^2 = 0.9954 \quad (1)$$

$$t_{abst\_scale\_apr}(x) = 0.0006N + 2.2,\ R^2 = 0.9628 \quad (2)$$

where: N – the number of instances;

t_{abst_apr} – duration of adaptation according to the abstract approach;

$t_{abst\_scale\_apr}$ – adaptation duration according to the abstract approach using horizontal scaling of the ontological model.

The determined values of the regression coefficient in this case indicate the intensity of the growth of the adaptation duration depending on the number of elements. Analyzing the obtained values, it was found that in the case of horizontal scaling, this coefficient is half as small. This confirms the feasibility of dividing one large knowledge base into several for faster processing of ontological rules.

Thus, the results of the experimental study confirm the feasibility of using horizontal scaling in the context of ontological rule processing. In addition, the use of horizontal scaling allows for increased performance in the configuration of adaptive software systems whose basic ontological model is characterized by a large number of connections.

It should be noted that despite the proven effectiveness of our proposed method of horizontal scaling of ontological rules, the problem of implementing multithreaded reasoning remains relevant. We see the use of a combination of ontological and large language models as one of the solutions to this problem. Therefore, the prospects for further research include the use of fine tuning or RAG methods to train LLM based on an ontological graph and knowledge base. In addition, the use of LLM, which uses an ontological model of an adaptive system as a basis for decision-making, will enable the dynamic generation of simple functional and graphic elements based on user requirements.

VI. CONCLUSIONS

An analysis of scientific research aimed at using methods and means of software scaling has been conducted. It has been established that the use of horizontal scaling provides the ability to distribute the load on software services. In addition, the use of the scaling process allows for better management and modification of software system configurations during software development and deployment.

A method for scaling the process of reasoning ontological rules for adaptive software is proposed. The designed method is based on horizontal scaling, which involves dividing the ontological model into several submodels according to a common criteri006Fn: the type of software system and the type of software components. In accordance with the division, the main ontological knowledge base is divided into several parts, which will improve the performance and efficiency of software adaptation.

An experimental study of the proposed method for scaling the software adaptation process based on ontology was conducted. Based on the results obtained, it was established that the use of horizontal scaling provides significant improvements in the processing speed of ontological rules. In addition, this indicator is higher for systems with a large number of instances. In particular, for systems with 3300 instances of ontological entities, the average adaptation time was reduced by 36% - from 6.85 seconds to 4.37 seconds. In addition, since the ontological rule processing mechanism is a blocking process, the use of scaling approaches will allow distributing the software adaptation process between different services depending on the selected partition.

Prospects for further research include combining the principles of the ontological approach, scaling system resources, and large language models. Combining ontology with LLM will reduce the load on the ontological adaptation service through fine-tuning or RAG methods based on the ontological knowledge base.

References

[1] O. Vyshnevskyy and L. Zhuravchak, "Semantic models for buildings energy management," *Proceedings of the 2023 IEEE 18th International Conference on Computer Science and Information Technologies (CSIT)*, Lviv, Ukraine, 2023, pp. 1-4. https://doi.org/10.1109/CSIT61576.2023.10324108.

[2] D. Balla, C. Simon, and M. Maliosz, "Adaptive scaling of Kubernetes pods," *Proceedings of the NOMS 2020 IEEE/IFIP Network Operations and Management Symposium*, Budapest, Hungary, 2020, pp. 1-5. https://doi.org/10.1109/NOMS47738.2020.9110428.

[3] T. Pan et al., "Sailfish: accelerating cloud-scale multi-tenant multi-service gateways with programmable switches," *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 194–206. https://doi.org/10.1145/3452296.3472889.

[4] D. Fedasyuk and I. Lutsyk, "Method of modification of self-adaptive software systems based on ontology," *Proceedings of the 2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, 2022, pp. 530–533. https://doi.org/10.1109/TCSET55632.2022.9766856.

[5] A. Angelis and G. Kousiouris, "A survey on the landscape of self-adaptive cloud design and operations patterns: Goals, strategies, tooling, evaluation and dataset perspectives," SSRN, 2025. https://doi.org/10.2139/ssrn.5253384.

[6] V. Mukhin et al., "A model for classifying information objects using neural networks and fuzzy logic," *Sci Rep*, vol. 15, no. 1, 2025, https://doi.org/10.1038/s41598-025-00897-4.

[7] I. Lutsyk and D. Fedasyuk, "Usage of the message broker technology in the adaptive software systems," *Communications in Computer and Information Science*, Springer Nature Switzerland, 2025, pp. 36–46. https://doi.org/10.1007/978-3-031-81372-6_3.

[8] V. Millnert and J. Eker, "HoloScale: horizontal and vertical scaling of cloud resources," *Proceedings of the 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, 2020, pp. 196–205. https://doi.org/10.1109/UCC48980.2020.00038.

[9] G. Blinowski, A. Ojdowska, and A. Przybylek, "Monolithic vs. microservice architecture: A performance and scalability evaluation," *IEEE Access*, vol. 10, pp. 20357–20374, 2022, https://doi.org/10.1109/ACCESS.2022.3152803.

[10] F. Rossi, M. Nardelli, and V. Cardellini, "Horizontal and vertical scaling of container-based applications using reinforcement learning," *Proceedings of the 2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, 2019. https://doi.org/10.1109/CLOUD.2019.00061.

[11] C.-Y. Liu, M.-R. Shie, Y.-F. Lee, Y.-C. Lin, and K.-C. Lai, "Vertical/horizontal resource scaling mechanism for federated clouds," *Proceedings of the 2014 IEEE International Conference on Information Science and Applications (ICISA)*, 2014, pp. 1–4. https://doi.org/10.1109/ICISA.2014.6847479.

[12] A. Kovalenko, H. Kuchuk, N. Kuchuk, and J. Kostolny, "Horizontal scaling method for a hyperconverged network," *Proceedings of the 2021 IEEE International Conference on Information and Digital Technologies (IDT)*, 22, 2021, pp. 331–336. https://doi.org/10.1109/IDT52577.2021.9497534.

[13] G. Blinowski, A. Ojdowska, and A. Przybylek, "Monolithic vs. Microservice architecture: A performance and scalability evaluation," *IEEE Access*, vol. 10, pp. 20357–20374, 2022, https://doi.org/10.1109/ACCESS.2022.3152803.

[14] V. Omelchenko and O. Rolik, "Hybrid method for horizontal and vertical computational resource scaling," *AIT*, no. 1 (3), pp. 49–58, 2024, https://doi.org/10.17721/AIT.2024.1.05.

[15] V. Lončarević, Ž. Jovanović, V. Luković, M. Milošević, S. Šućurović and A. Iričanin, "Horizontal scaling with session preservation of PHP applications with MVC architecture," *Proceedings of the 10th International Scientific Conference Technics, Informatic, and Education*, Čačak, 2024, pp. 34–41. https://doi.org/10.46793/TIE24.034L.

[16] B. Pashkovskyi, M. Slabinoha, and M. Romaniv, "Web application performance optimization with cqrs and horizontal scaling," *Visnyk of Kherson National Technical University*, no. 1(88), pp. 272–278, 2024, https://doi.org/10.35546/kntu2078-4481.2024.1.38.

[17] L. Yazdanov and C. Fetzer, "Vertical scaling for prioritized VMs provisioning," *Proceedings of the 2012 IEEE Second International Conference on Cloud and Green Computing*, 2012, pp. 118–125. https://doi.org/10.1109/CGC.2012.108.

[18] K. Rai, B. Sahana, A. N. Pai, S. Gautham, and U. Dhanush, "Vertical scaling of virtual machines in cloud environment," *Proceedings of the 2021 IEEE International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, 2021, pp. 458–462. https://doi.org/10.1109/RTEICT52294.2021.9573715.

[19] F. Magnanini, L. Ferretti, and M. Colajanni, "Scalable, confidential and survivable software updates," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 1, pp. 176–191, 2022, https://doi.org/10.1109/TPDS.2021.3090330.

[20] M. Bourgey et al., "GenPipes: an open-source framework for distributed and scalable genomic analyses," *GigaScience*, vol. 8, no. 6, 2019, https://doi.org/10.1093/gigascience/giz037.

[21] M. Mikailov et al., 'Scaling and parallelization of big data analysis on HPC and cloud systems," *Proceedings of the 2019 IEEE International Conference on Advances in Computing and Communication Engineering (ICACCE)*, 2019, pp. 1–8. https://doi.org/10.1109/ICACCE46606.2019.9079987.

[22] H. Coullon, L. Henrio, F. Loulergue, and S. Robillard, "Component-based distributed software reconfiguration: A verification-oriented survey," *ACM Comput. Surv.*, vol. 56, no. 1, pp. 1–37, 2023, https://doi.org/10.1145/3595376.

[23] P. Ochieng and S. Kyanda, "Large-scale ontology matching," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 1–35, 2018, https://doi.org/10.1145/3211871.

[24] M. McDaniel and V. C. Storey, "Evaluating domain ontologies," *ACM Comput. Surv.*, vol. 52, no. 4, pp. 1–44, 2019, https://doi.org/10.1145/3329124.

[25] Y. Tiwari, O. A. Lone, and M. Pal, "OntoRAG: Enhancing question-answering through automated ontology derivation from unstructured knowledge bases," 2025, arXiv. doi: 10.48550/ARXIV.2506.00664.

[26] V. K. Kommineni, B. König-Ries, and S. Samuel, "From human experts to machines: An LLM supported approach to ontology and knowledge graph construction," 2024, arXiv. doi: 10.48550/ARXIV.2403.08345.

[27] L. van Elst and A. Abecker, "Ontologies for information management: balancing formality, stability, and sharing scope," *Expert Systems with Applications*, vol. 23, no. 4, pp. 357–366, 2002, https://doi.org/10.1016/S0957-4174(02)00071-4.

[28] D. Fedasyuk and I. Lutsyk, "Approach to implementation of configuration process for adaptive software systems based on ontologies," *International Journal of Computing*, vol. 22, issue 3, pp. 381–388, 2023, https://doi.org/10.47839/ijc.22.3.3234.

[29] D. Fedasyuk and I. Lutsyk, "The use of ontology in the process of designing adaptive software systems," *Proceedings of the 2022 IEEE 17th International Conference on Computer Sciences and Information Technologies (CSIT)*, 2022, pp. 503–506. https://doi.org/10.1109/CSIT56902.2022.10000528.

***Illia LUTSYK** PhD in Information Technologies by Specialty of Software Engineering, Lecturer at Software Engineering Department Institute of Computer Sciences and Information Technologies, Lviv Polytechnic National University. Research interests: adaptive software, ontological models, software design*

***Dmytro FEDASYUK** Professor, Head of Software Engineering Department, Institute of Computer Sciences and Information Technologies, Lviv Polytechnic National University. Research interests: mathematical modeling and information technologies, modeling of thermal regimes in microelectronic systems, software design*