DOI: 10.15439/2024F3355
ISSN 2300-5963 ACSIS, Vol. 39

Optimization of the Cell-based Software Architecture by Applying the Community Detection Approach

Miloš Milić
0000-0002-2521-7607
University of Belgrade
Faculty of Organizational Sciences, Belgrade
Jove Ilića 154,
11000 Belgrade, Serbia
Email: milos.milic@fon.bg.ac.rs

Dragana Makajić-Nikolić
0000-0002-0790-6791
University of Belgrade
Faculty of Organizational Sciences, Belgrade
Jove Ilića 154,
11000 Belgrade, Serbia
Email: dragana.makajic-nikolic@fon.bg.ac.rs

***Abstract*—The aim of this research is to present the Cell-based software architecture and explore its optimization. Cell-based software architecture organizes a software system into interconnected cells, each containing multiple elements. This research focuses on optimizing cell-based architecture, particularly the number of cells and their internal organization. In this context, the Community Detection approach, which identifies closely connected elements, was applied. Additionally, the model incorporates the concept of functionality, defined as a set of capabilities allowable and actionable by the software system. We conducted a series of experiments based on the defined mathematical model to validate our approach, achieving optimal and near-optimal solutions within a given time limit. Considering that each cell can contain multiple elements realized in various architectural styles, the proposed model allows for the integration of different architectures within the same software system. This flexibility enhances the system's overall adaptability and efficiency.**

***Index Terms*—software architecture, cell-based architecture, community detection, architecture optimization.**

I. Introduction

IN TODAY'S digital age, the application of software systems spans across various domains. These systems enable seamless communication and data exchange within and across different industries. In the interconnected world, software systems can be utilized by a diverse range of clients, and it is essential to ensure they have capabilities to support them effectively.

In addition to functional requirements, these capabilities are related to non-functional requirements such as security, deployability, availability, scalability, reliability, resilience, maintainability, etc. [1]. However, achieving a high level of non-functional requirements can be a challenging task. Non-functional requirements are typically defined as quality attributes of a software system, and are closely related to software architecture [2].

Software architecture of a system can be defined as the set of elements needed to reason about the system [3], encompassing various software components, their relationships, as well as the properties of components and relationships [1]. Software architecture can be considered as a blueprint for further software design, based on which various components are created. In this context, software architects and engineers should consider software architecture from the earliest phases of development.

This research presents the Cell-based software architecture. Cell-based architecture considers the organization of a software system in the form of interconnected cells, while each cell can contain multiple elements [4]-[5]. The research observes cell-based software architecture optimization, specifically focusing on the number of cells and their internal organization, as community detection problem.

The rest of the paper is organized as follows. Section 2 introduces various software architectures that can be applied in the software development process. Additionally, the Cell-based software architecture is presented, as well as the Community Detection problem and its application in different fields. Section 3 introduces the problem and defines a mathematical model for Cell-based software architecture optimization. Evaluation and optimization results are presented in Section 4. Finally, the conclusion is presented in Section 5.

II. Background

This section introduces various software architectures, with a focus on Cell-based software architecture. Given that cells can be represented as a network of connected nodes, the section also covers the Community Detection problem and its application in various fields.

This work was supported by the University of Belgrade – Faculty of Organizational Sciences.

A. Software Architecture

When software design is concerned, various software architectures can be observed. For example, monolithic architecture represents a traditional software design approach. This architecture involves multiple modules that are executed together as a single unit at runtime, resulting in high coupling between the modules [6]. On the other hand, microservice architecture is an alternative to monolithic architecture. In microservice architecture, each element is implemented as a separate microservice, which operates independently as a single unit at runtime. This approach results in low coupling between microservices [6]. However, taking into account that each microservice is managed independently, microservice organization and communication must be carefully considered [7]. In addition, microservices typically require additional components for management, such as microservice orchestration and choreography [8], which can introduce additional complexity. Although monolith and microservice architecture can co-exist within the same system, researchers are exploring approaches decomposing and gradually transitioning from monolithic applications to microservices [9]-[12]. Both monolithic and microservice architectures require infrastructure services (e.g., application server, database server, etc.), which can be either on-premises or cloud-based.

Another alternative to monolithic and microservice architectures is serverless software architecture, an approach that focuses on designing services related to specific business capabilities [13]. In this context, Functions-as-a-Service (FaaS) can be coded and deployed, while the underlying infrastructure is managed by the cloud provider [14]. Although this approach allows software engineers to focus on business functions, it results in a high degree of coupling with the infrastructure services provisioned by the cloud provider.

Based on the previous discussion, it can be stated that each software architecture has its own pros and cons that should be carefully considered during the software design process.

B. Cell-based Software Architecture

Cell-based architecture can be defined as a software architecture that incorporates multiple units of workload, with each unit known as a cell [5]. Each cell is independent from other cells, does not share state with other cells, and can encapsulate multiple components of different types [4]-[5]. Additionally, each cell contains a cell gateway, serving as the central entry point for cell communication. In this context, intra-cell and inter-cell communication can be observed, which is realized with well-defined interfaces and protocols [4]. A specific set of functionalities or services can be incorporated within a cell, defining a cell boundary. In this context, cell-based architecture can be related with domain-driven software design [15].

Conceptual overview of the Cell-based software architecture is presented in Figure 1. The figure depicts two cells with multiple elements, with the cell boundaries outlined by octagons. Cell A incorporates three elements (e.g., one monolith and two microservices), while Cell B also includes three elements (e.g., three microservices). Additionally, element A2 from Cell A communicates with Cell B through the cell gateway. In this way inter-cell communication is realized [4]. On the other hand, element B2 communicates with element B3. This communication is performed inside Cell B and represents intra-cell communication [4]. Each cell is autonomous and can be managed independently of other cells. As a result, better encapsulation, isolation, and distribution of software architecture elements can be achieved, addressing some of the typical challenges in software architecture design [16].

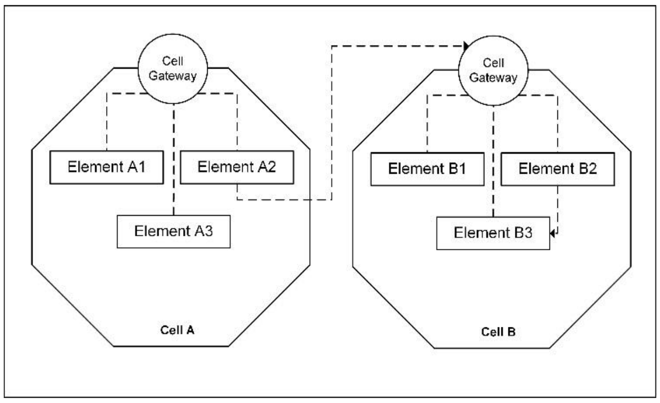

Fig 1. Conceptual overview of the Cell-based software architecture

Considering that a cell can incorporate multiple capabilities implemented in various architectures, the cell-based approach facilitates the introduction of multi-architecture software development. In this context, the benefits of each applied architecture can be utilized, while their cons can be managed. This approach allows each cell to be independent and iterate individually, resulting in decentralized software architecture [4].

C. Community Detection Problem

Community detection problem belongs to the field of Complex Network Analysis. Its most common areas of application are: social networks [17]-[18], neuroscience and biology [19], supply chain networks [20]-[21], politics, customer segmentation, smart advertising and targeted marketing [22], etc. Community detection approach were also applied in software engineering since the process-oriented and the object-oriented software architecture both can be presented as complex network [23] characterized by properties like those commonly observed in other complex networks [24]. Authors Pan, Jing, and Li used community detection approach for refactoring the package structures of object-oriented software in order to improve the maintenance process [25]. Software maintenance was also emphasized as the reason for using community detection in research conducted by Huang et al. [26]. Authors Hou, Yao, and Gong applied community detection approach to developer collaboration network in software ecosystem based on developer cooperation intensity [27].

Communities are groups of network's vertices with the common properties and/or role in the network [28]. The community detection problem is to find communities that maximize a given quality function. The solution of the problem is a set of communities such that the number of edges within the community is greater of the number of edges between the community's vertices and the rest of the network (Figure 2).

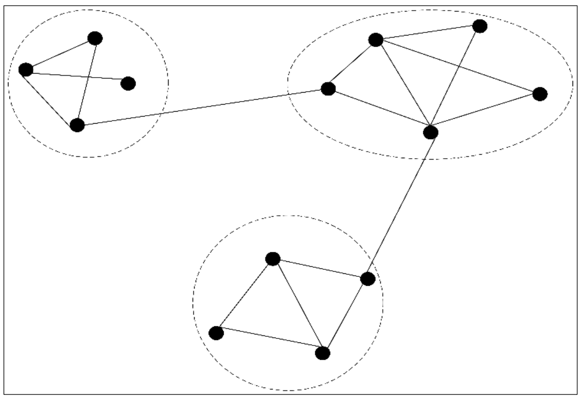

Fig 2. A simple graph with three communities

There are several quality measures intended to evaluate the structure of communities [29]. In this paper we use the standard and most used measure of quality, called Newman–Girvan modularity [30]. One of the formulations of Newman–Girvan modularity is:

$$Q = \sum_{m \in C} \left(\frac{L_m}{L} - \left(\frac{D_m}{2L} \right)^2 \right) \quad (1)$$

where C represents the set of communities, L_m is the sum of the weights of the edges within community m, L is the sum of weights of all edges in entire the network, and D_m is the degree of the vertices in community m.

The function (1) is nonlinear, and it can only be solved for small and medium-sized unweighted graphs [31]. Hence, several solving methods and linearizations of it can be found in the literature [32]-[33]. In this paper, we use the variant of modularity function (1) from [34] that enables further linearization:

$$Q = \sum_{k \in C} \left(\frac{1}{L} \sum_{(i,j) \in E_m} b_{ij} - \frac{1}{4L} \sum_{(i,j) \in E_m} d_i d_j \right) \quad (2)$$

where E_m represents the set of edges in community m of a given graph $G=(E,V)$, V is the set of the vertices. Parameter b_{ij} is the weight of the edge (i,j), $(i,j) \in E$, and d_i is the weight of the vertex, obtained as sum of weight of all input and output edges of vertex i:

$$d_i = \sum_{(i,j),(j,i) \in E} b_{ij}, \; i \in V \quad (3)$$

III. Mathematical Model for Optimizing Cell-Based Software Architecture

As previously discussed in Section 2, cell-based software architecture can be depicted as a network of interconnected cells and elements that communicate with each other. Given that the solution of Community Detection problem can identify closely connected items, this section presents a mathematical model for optimizing cell-based software architecture. Optimizing the cell-based architecture can potentially lead to better resource utilization through an optimal number of cells and their internal organization. Additionally, various software quality attributes can be enhanced.

When the software architecture whose elements should be grouped into cells based on community detection problem, the elements of the architecture are vertices of the graph $G=(E,V)$. The edges of the graph exist between the elements (vertices) which communicate, while the weight of the edge (i,j), b_{ij} represents the intensity of the communication.

Since the weights inside the parentheses in equation (2) should be calculated only for the edges and vertices belonging to the same community, the set of communities C and binary variables y_{ik} are introduced:

$$y_{ik} = \begin{cases} 1 & \text{if } i\text{-th vertice is in community } k \\ 0 & \text{otherwise} \end{cases}, \; i \in V, k \in C.$$

Equation (2) now became:

$$Q = \sum_{k \in C} \left(L_1 \sum_{(i,j) \in E} b_{ij} y_{ik} y_{jk} - L_2 \sum_{(i,j) \in E} d_i d_j y_{ik} y_{jk} \right) \quad (4)$$

where $L_1 = 1/L$, $L_2 = 1/4L$, and L is the sum of weights of all edges in entire the network.

The nonlinearity $y_{ik}y_{jk}$ could be replaced by auxiliary binary variables z_{ijk}:

$$z_{ijk} = \begin{cases} 1 & \text{if edge } (i,j) \text{ is in community } k \\ 0 & \text{otherwise} \end{cases}, \; (i,j) \in E, k \in C$$

and inequalities:

$$z_{ijk} \ge y_{ik} + y_{jk} - 1, \; k \in C, (i,j) \in E \quad (5)$$

$$\begin{aligned} z_{ijk} &\le y_{ik}, \; k \in C, (i,j) \in E \\ z_{ijk} &\le y_{jk}, \; k \in C, (i,j) \in E \end{aligned} \quad (6)$$

The condition (5) ensures that variable z_{ijk} get the value 1 if both y_{ik} and y_{ik} have the value 1, i.e. the edge (i,j) is inside the community k if both vertices i and j belong to the community k. Since based on condition (5), value of z_{ijk} can be 1 if y_{ik} and/or y_{ik} are equal to zero, the condition (6) is introduced to prevent such solutions. Furthermore, if for an edge (i,j) z_{ijk} equals zero for all $k \in C$, it indicates that edge (i,j) does not belong to any community; instead, it represents a link between two different communities.

Additionally, the model also incorporates the concept of functionality. Functionality can be defined as a set of capabilities allowable and actionable by the software system [35]. Each functionality contains elements focused on a specific domain and should not be mixed to maintain boundaries, reduce complexity, and ensure modularity. In a cell-based software architecture, a single functionality can be represented by one or more cells, forming the foundation for optimizing the software architecture. In addition, different functionalities should not be organized in the same cell, allowing better separation of concerns between cells. As a result, each cell can be independent and managed individually [4].

In addition to the already introduced parameters and variables, notation used for the mathematical model formulation is as follows.

Sets:

- FC - set of functionalities,
- F_l - set of l-th functionality,

$$F_l \subset FC, \bigcap_{l \in FC} F_l = \varnothing, \bigcup_{l \in FC} F_l = FC$$

Parameters:

- e – lower bound of the number of vertices in communities,

Variables:

$$x_k = \begin{cases} 1 & \text{if } k\text{-th community exist} \\ 0 & \text{otherwise} \end{cases}, k \in C$$

The proposed mathematical model is listed below.

$$\max f(z) = \sum_{k \in C} (L_1 \sum_{(i,j) \in E} b_{ij} z_{ijk} - L_2 \sum_{(i,j) \in E} d_i d_j z_{ijk}) \quad (7)$$

s.t.

$$z_{ijk} \ge y_{ik} + y_{jk} - 1, \; k \in C, (i,j) \in E \quad (8)$$

$$z_{ijk} \le y_{ik}, \; k \in C, (i,j) \in E \quad (9)$$

$$\sum_{k \in C} y_{ik} = 1, \; i \in V \quad (10)$$

$$y_{ik} \le x_k, \; k \in C, i \in V \quad (11)$$

$$\sum_{i \in V} y_{ik} \ge e \cdot x_k, \; k \in C \quad (12)$$

$$y_{ik} + y_{jk} \le 1, k \in C, i \in F(l), j \in F(p), l, p \in FC, l \ne p \quad (13)$$

$$x_k = \{0,1\}, \; k \in C \quad (14)$$

$$y_{ik} = \{0,1\}, \; i \in V, k \in C \quad (15)$$

$$z_{ijk} = \{0,1\}, \; (i,j) \in E, k \in C \quad (16)$$

The objective function (7) represents the modularity measure linearized by replacing $y_{ik}y_{jk}$ with z_{ijk} in (4). Since this function should be maximized, the first addend in parentheses will be as large as possible. Thus, the branches that have a greater weight will be within the same community, that is, the software elements with more frequent communication will be in the same cell. Constraints (8-9) are related to linearization. Constraint (10) ensures that each vertex is assigned exactly to one community. The constraint (11), the value 1 is set to x_k if some vertex is assigned to the k-th community. Constraint (12) is related to the minimal number of vertices assigned to the existing communities. Constraint (13) provides that vertices of different functionality cannot belong to the same community, i.e. only vertices of the same functionality can be in the same community. Constraints (14-16) are related to the binary restrictions on the variables.

If necessary, additional constraints can be introduced. For example, although the parameter e defines the lower bound for the number of vertices in communities, an additional constraint can be added to specify a different minimum number of vertices for a particular community. This allows for fine-grained definition of cell structure in specific circumstances.

For example, if some of the software element should be isolated in a cell, a set of such element $VI \subset V$ and additional constraint can be included into mathematical model:

$$y_{ik} + y_{jk} \le 1, \; k \in C, i \in VI, j \in V, j \ne i \quad (17)$$

Additionally, if some elements should be in the same cell, regardless the connections between them, the mathematical model can be extended as follows.

- G - set of predefined groups of elements,
- V_q - set of elements predetermined to be in the same cell, $q \in G$,

$$y_{ik} = y_{jk}, \; k \in C, i, j \in V_q, q \in G \quad (18)$$

$$y_{ik} + y_{jk} \le 1, \; k \in C, i \in V_q, j \in V \setminus V_q \quad (19)$$

The constraint (18) ensures that the predetermined elements are in the same cell but allows other elements to be assigned to that cell as well. If it is necessary to assign to the same cell only the elements from $q \in G$, constraint (19) should be included into mathematical model.

IV. Evaluation

The optimized structure of the solution can be graphically presented based on the results.

Figure 3 presents the results of the optimization of a manufacturing software system (e.g., mobile phone manufacturing). The input includes defined Production and Purchasing functionalities. The Production functionality comprises two monolithic applications (i.e., *production* and *legacy*) and two microservices (i.e., *inventory* and *product*), while the Purchasing functionality consists of three microservices (i.e., *order*, *payment*, and *notification*). Additionally, the communication between these elements is specified (the weights of the edges in Figure 3).

Based on the performed optimization, the resulted solution includes three communities (named Community A, Community B, and Community C), each containing different elements (see Figure 3). In the following text these communities will be referred to as the Production Cell (Community A), Purchasing Cell (Community B), and Legacy Cell (Community C).

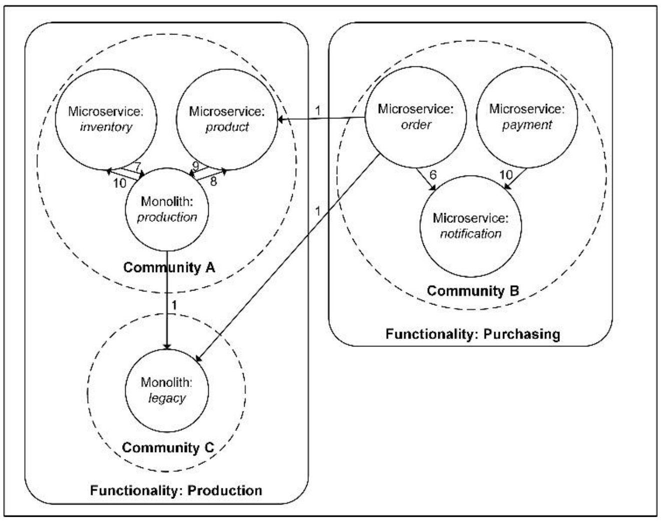

Fig 3. Results of the optimization of a manufacturing software system

An additional observation pertains to the Legacy Community, which contains only one element (i.e., the *legacy* monolith). A legacy software system is defined as a core system that has been functioning correctly in production for decades [36]. Considering the prevalence of legacy systems today, researchers are exploring approaches to migrate these systems to modern architectures [37]-[39]. In the context of cell-based software architecture, the legacy element is incorporated within a specific cell. From the optimization model perspective, this is represented as an additional constraint that restricts the particular cell structure:

$$y_{legk} + y_{jk} \le 1,\ k \in C,\ j \in V \setminus leg \qquad (20)$$

where *leg* is the index of the variable corresponding to the *legacy* monolith. This constraint ensures that no other element can be in the cell containing the *legacy* monolith.

Another interesting observation pertains to the Cell Gateway component. While this component is not explicitly represented in the mathematical model, it serves as the central entry point for cell communication [4]. Therefore, we have included one gateway per cell based on the optimal solution. The final software architecture is shown in Figure 4.

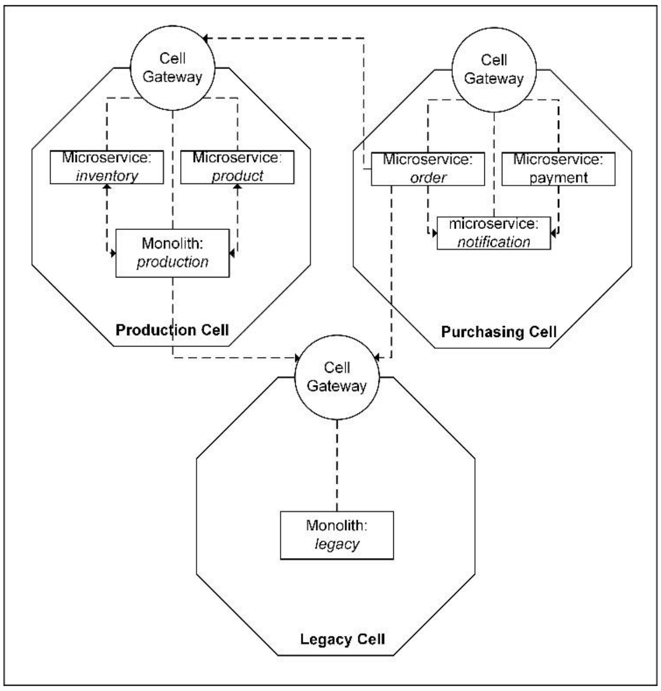

Fig 4. Cell-based software architecture based on the optimal solution

In order to validate the mathematical model and examine the dimensions of the problem that can be solved exactly, we conducted a series of experiments.

The evaluation considered three functionalities, with the number of elements varied. Each element represents an instance of software architecture encapsulating specific features (e.g., a monolith with multiple features, a microservice containing a single feature). Considering that these elements can vary in terms of their applied software architecture and size, a software system can encompass numerous elements. Within this context, the first functionality incorporated 40% of the elements, while the remaining elements were equally divided between the other two functionalities. Taking into account that elements cooperate with each other, each element has at least one connection with another element within the same functionality, while up to 40% of elements have double connections within the same functionality. Finally, considering that all functionalities are part of the same software system, two connections between elements from different functionalities are also established. The previously discussed elements, such as edges and their weights (the intensity of the communication between the elements - parameter b_{ij}) are randomly generated. The lower bound of the number of vertices in communities (parameter e) is set to 2.

The experiments were conducted for the graphs whose dimensions are given in Table 1. The columns named vertices and edges give the number of vertices and edges, respectively, while column L represents the total weight of all vertices in graphs.

TABLE I.
A SUMMARY OF DATA SETS USED IN THE EXPERIMENTS

Case	Number of vertices	Number of edges	L
1	30	38	163
2	40	52	214
3	50	64	252
4	60	77	307
5	70	90	351
6	80	103	393
7	90	116	462
8	100	128	519
9	150	193	746
10	200	257	991

Given that the model parameters are randomly generated, the mathematical model presented in section III was applied on ten instances of each graph from Table 1. All optimizations were performed solved using GLPK software on a laptop computer equipped with 11th Gen Intel(R) Core(TM) i5 and 16 GB of RAM. The solving method used in GLPK software was Branch and Cut, with Gomory's mixed integer cuts, MIR (mixed integer rounding) cuts, mixed cover cuts and clique cuts options. Execution time was limited, depending on graphs dimensions.

Time limitations were: 5 minutes for cases 1 to 3, 10 minutes for cases 4 to 7, and 15 minutes for cases 8 to 10. Table 2 gives the optimization time. The second column gives the number of instances (out of ten) for which the optimal solution was found within the given time limit. The three right columns give the minimum, maximum, and average duration (in seconds) of the optimization among ten generated instances.

TABLE II.
DURATIONS OF THE SUCCESSFUL OPTIMIZATIONS

Case	Number of optimal solutions	min	max	avg
1	8	0.2	12.9	1.9
2	9	0.8	8.1	1.8
3	9	1.7	86.7	12.0
4	8	6	13.9	8.7
5	8	11.2	181.7	39.9
6	8	20.4	61	32.5
7	6	30.4	58.1	44.6
8	7	30.2	44.2	39.2
9	7	87.3	218.8	117.3
10	5	142	552	423.3

TABLE III.
AVERAGE PERFORMANCE OF THE OPTIMAL SOLUTIONS

Case	Number of cells	Min cell size	Max cell size	Modularity value
1	8.63	2	5.13	4.25E-07
2	10.00	2	7.10	-2.78E-03
3	10.78	2	8.11	-5.88E-02
4	10	2	9.57	1.75E-07
5	11.63	2.25	11.5	1.89E-06
6	13	2.5	13.25	1.44E-06
7	11	2.2	16.67	-1.61E-02
8	10.14	2.43	17.43	8.86E-07
9	11.71	3.43	25.43	1.14E-06
10	15.80	2	32.00	0

As expected, the number of instances that can be solved within the time limit decreases and optimization time increases with increasing of graphs dimensions. However, even for lower dimensions, the graph topology and parameter values can prevent finding the optimal solution in a given time limit, as can be seen in the case of 30 nodes. Generally, in most instances of graphs of the same dimensions, the optimization times were similar. Figure 5 shows distribution of the optimization time for all solved instances.

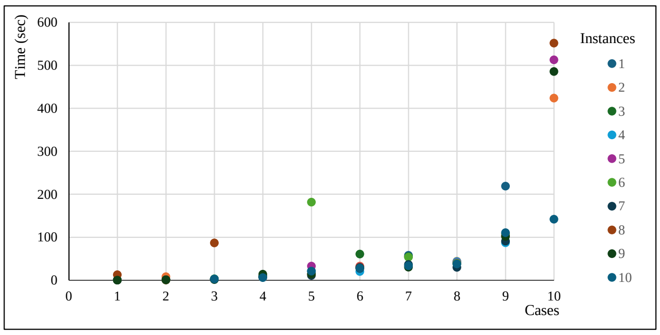

Fig 5. Duration of optimizations of all instances of all ten cases

In most cases, the optimization times are grouped, except for cases 3, 5, and 9 where the optimization of one instance takes significantly longer than the others and in case 10 where the optimization of one instance is significantly faster than the others. The reason for these deviations lies in the topology of the graphs and the values of the model parameters.

Table 3 shows the average performance of obtained solutions. The last column represents the optimal value of the objective function, i.e. linearized modularity function.

The number of cells increases slowly with the increasing of graphs dimensions even though the model does not contain its upper limit. Additionally, almost 20% of cells consist of two elements in the majority of instances. The only exception among 75 successfully solved instances is the two instances of the graphs with 70, 80, 100 and 150 vertices and one instance of the graph with 90 vertices. Figure 6 shows the number of cells for all solved instances.

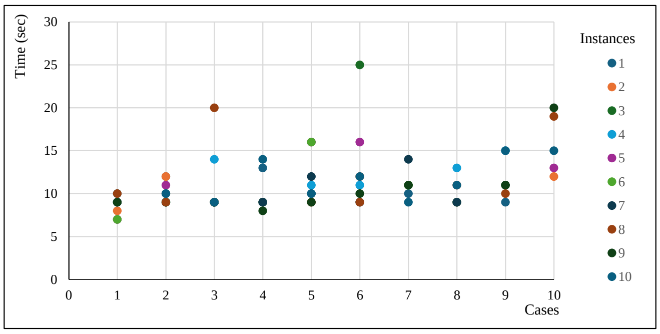

Fig 6. The number of cells in all instances of all ten cases

Based on Figure 6, it can be concluded that the number of cells is generally grouped for most instances within the same case. However, the number of cells slightly varies with different problem dimensions and typically ranges from 7 to 16, with a few exceptions. This indicates that the number of cells is more influenced by the topology of the correspondence graph and the parameters values than by the number of software elements.

The values of linearized modularity function in Table 3 are small, even negative in some cases. These negative values indicate that nodes are less connected within communities. However, the interpretation of the value of the modularity function in these experiments is not of great importance, given that hypothetical examples with randomly generated graph branches and mathematical model parameters were used.

V. Conclusion

The selection of software architecture guides the software design and development process, making it an important topic in the field of software engineering. Therefore, the chosen software architecture should be carefully selected to suit the specific needs of the software system being implemented. While this research focuses on optimizing cell-based software architecture, particularly the number of cells and their internal organization, additional research directions can be considered.

Further research could examine system workload in the context of additional non-functional attributes such as scalability, availability, and reliability. Additionally, the details of intra-cell and inter-cell communication could be further investigated. Regarding mathematical model, different linearization of modularity function can be examined as well as different quality measures. The main goal of this research was to investigate the validity of modeling cell-based software architecture as community detection problem. Since this problem is NP hard, the next step of the research will be to develop a heuristic for solving large scale problems.

Data Availability

Input data and optimization results from a series of experiments can be accessed at the following address: https://github.com/mmilicfon/fedcsis2024.

References

[1] P. Bourque and R. E. Fairley (Eds), *Guide to the Software Engineering Body of Knowledge (SWEBOK (R)): Version 3.0*, IEEE Computer Society Press, 2014.

[2] A. Chandrasekar, S. Rajesh, and P. Rajesh, "A research study on software quality attributes", *International Journal of Scientific and Research Publications*, Vol. 4, No. 1, pp. 14-19, 2014.

[3] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 4th Edition, Addison-Wesley Professional, 2021.

[4] A. Abeysinghe, "Cell-Based Architecture: A Decentralized Reference Architecture for Cloud-native Applications", 2024. Available online: https://github.com/wso2/reference-architecture/blob/master/reference-architecture-cell-based.md (Access Date: May 17, 2024).

[5] Amazon Web Services, "Reducing the Scope of Impact with Cell-Based Architecture: AWS Well-Architected", Amazon Corporation, 2024. Available online: https://docs.aws.amazon.com/wellarchitected/latest/reducing-scope-of-impact-with-cell-based-architecture/reducing-scope-of-impact-with-cell-based-architecture.html (Access Date: July 12, 2024).

[6] G. Blinowski, A. Ojdowska, and A. Przybyłek, "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.

[7] S. Hassan and R. Bahsoon, "Microservices and their design trade-offs: A self-adaptive roadmap", in *Proceedings of the 2016 IEEE International Conference on Services Computing (SCC)*, pp. 813-818, IEEE, 2016, https://doi.org/10.1109/SCC.2016.113.

[8] N. Singhal, U. Sakthivel, and P. Raj, "Selection mechanism of microservices orchestration vs. choreography". *International Journal of Web & Semantic Technology (IJWesT)*, Vol. 10, No. 1, pp. 1-13, 2019, https://doi.org/10.5121/ijwest.2019.10101.

[9] Y. Abgaz, A. McCarren, P. Elger, D. Solan, N. Lapuz, M. Bivol, G. Jackson, M. Yilmaz, J. Buckley, and P. Clarke, "Decomposition of monolith applications into microservices architectures: A systematic review", *IEEE Transactions on Software Engineering*, Vol. 49, No. 8, pp. 4213-4242, 2023, https://doi.org/10.1109/TSE.2023.3287297.

[10] R. Chen, S. Li, and Z. Li, "From monolith to microservices: A dataflow-driven approach", In J. Lv, H. Zhang, X. Liu, and M. Hinchey (Eds.), *Proceedings of the 2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 466-475, IEEE, 2017, https://doi.org/10.1109/APSEC.2017.53.

[11] G. Mazlami, J. Cito, and P. Leitner, "Extraction of microservices from monolithic software architectures", in *Proceedings of the 2017 IEEE International Conference on Web Services (ICWS)*, pp. 524-531, IEEE, 2017, https://doi.org/10.1109/ICWS.2017.61.

[12] K. Sellami, M. A. Saied, A. Ouni, and R. Abdalkareem, "Combining static and dynamic analysis to decompose monolithic application into microservices", in *International Conference on Service-Oriented Computing*, pp. 203-218, Cham: Springer Nature Switzerland, 2022, https://doi.org/10.1007/978-3-031-20984-0_14.

[13] M. Sewak and S. Singh, "Winning in the era of serverless computing and function as a service", in *Proceedings of the 2018 3rd International Conference for Convergence in Technology (I2CT)*, pp. 1-5, IEEE, 2018, https://doi.org/10.1109/I2CT.2018.8529465.

[14] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, (2020), "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider", in *Proceedings of the 2020 USENIX annual technical conference (USENIX ATC 20)*, pp. 205-218, 2020.

[15] S. Millett and N.Tune, *Patterns, principles, and practices of domain-driven design*, John Wiley & Sons, 2015.

[16] A. Bierska, B. Buhnova, and H. Bangui, "An Integrated Checklist for Architecture Design of Critical Software Systems", *Annals of Computer Science and Information Systems, Vol. 31, 17th Conference on Computer Science and Intelligence Systems (FedCSIS)*, pp. 133-140, 2022, IEEE, https://doi.org/10.15439/2022F287.

[17] C. C. Lin, J. R. Kang, and J. Y. Chen, "An integer programming approach and visual analysis for detecting hierarchical community structures in social networks", *Information Sciences*, 299, pp. 296-311, 2015, https://doi.org/10.1016/j.ins.2014.12.009.

[18] A. R. Costa, & C. G. Ralha, "AC2CD: An actor–critic architecture for community detection in dynamic social networks", *Knowledge-Based Systems*, 261, 110202, 2023, https://doi.org/10.1016/j.knosys.2022.110202.

[19] E. M. Mohamed, T. Agouti, A. Tikniouine, and M. El Adnani, "A comprehensive literature review on community detection: Approaches and applications", *Procedia Computer Science*, 151, pp. 295-302, 2019, https://doi.org/10.1016/j.procs.2019.04.042.

[20] N. M. Viljoen and J. W. Joubert, "Supply chain micro-communities in urban areas", *Journal of Transport Geography*, 74, 211-222, 2019, https://doi.org/10.1016/j.jtrangeo.2018.11.011.

[21] Z. Lu and Z. Dong, "A Gravitation-Based Hierarchical Community Detection Algorithm for Structuring Supply Chain Network", *International Journal of Computational Intelligence Systems*, Vol. 16, No. 1, 110, 2023, https://doi.org/10.1007/s44196-023-00290-x.

[22] A. Karataş, and S. Şahin, "Application areas of community detection: A review", in *Proceedings of the 2018 International congress on big data, deep learning and fighting cyber terrorism (IBIGDELFT)*, pp. 65-70, IEEE, 2018, https://doi.org/10.1109/IBIGDELFT.2018.8625349.

[23] D. Li, Y. Han, and J. Hu, "Complex network thinking in software engineering", in *Proceedings of the 2008 International Conference on Computer Science and Software Engineering*, pp. 264-268, IEEE, 2008, https://doi.org/10.1109/CSSE.2008.689.

[24] L. Šubelj and M. Bajec, "Community structure of complex software systems: Analysis and applications", *Physica A: Statistical Mechanics and its Applications*, Vol. 390, No. 16, pp. 2968-2975, 2011, https://doi.org/10.1016/j.physa.2011.03.036.

[25] W. F. Pan, B. Jiang, and B. Li, "Refactoring software packages via community detection in complex software networks", *International Journal of Automation and Computing*, Vol.10, No. 2, pp. 157-166, 2013, https://doi.org/10.1007/s11633-013-0708-y.

[26] G. Huang, P. Zhang, B. Zhang, T. Yin, and J. Ren, "The optimal community detection of software based on complex networks", *International Journal of Modern Physics C*, Vol. 27, No. 08, 1650085, 2016, https://doi.org/10.1142/S0129183116500856.

[27] T. Hou, X. Yao, and D. Gong, "Community detection in software ecosystem by comprehensively evaluating developer cooperation intensity", *Information and Software Technology*, 130, 106451, 2021, https://doi.org/10.1016/j.infsof.2020.106451.

[28] S. Fortunato, "Community detection in graphs", *Physics reports*, Vol. 486, No. 3-5, pp. 75-174, 2010, https://doi.org/10.1016/j.physrep.2009.11.002.

[29] V. L. Dao, C. Bothorel, and P. Lenca, "Community structure: A comparative evaluation of community detection methods", *Network Science* , Vol. 8, No. 1, pp. 1-41, 2020, https://doi.org/10.1017/nws.2019.59.

[30] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks", *Physical review E*, Vol. 69, No. 2, 026113, 2004, https://doi.org/10.1103/PhysRevE.69.026113.

[31] L. Bennett, S. Liu, L.G. Papageorgiou, and S. Tsoka, "A mathematical programming approach to community structure detection in complex networks", *Computer Aided Chemical Engineering*, Vol. 30, pp. 1387-1391, 2012, https://doi.org/10.1016/B978-0-444-59520-1.50136-6.

[32] B. Serrano and T. Vidal, "Community detection in the stochastic block model by mixed integer programming", *Pattern Recognition*, Vol. 152, 110487, 2024, https://doi.org/10.1016/j.patcog.2024.110487.

[33] A. Ferdowsi and M. D. Chenary, "Toward an Optimal Solution to the Network Partitioning Problem", *Annals of Computer Science and Information Systems, Vol. 35, 18th Conference on Computer Science and Intelligence Systems (FedCSIS)*, pp. 111-117, 2023, IEEE, https://doi.org/10.15439/2023F2832.

[34] E. Alinezhad, B. Teimourpour, M.M. Sepehri, and M. Kargari, "Community detection in attributed networks considering both structural and attribute similarities: two mathematical programming approaches", *Neural Computing and Applications*, Vol. 32, pp. 3203-3220, 2020, https://doi.org/10.1007/s00521-019-04064-5.

[35] ISO/IEC/IEEE 24765:2017 Systems and software engineering — Vocabulary, International Organization for Standardization, Available online: https://www.iso.org (Access Date: May 27, 2024).

[36] R. Khadka, B. V. Batlajery, A. M. Saeidi, S. Jansen, and J. Hage, "How do professionals perceive legacy systems and software modernization?", in *Proceedings of the ACM 36th International Conference on Software Engineering (ICSE 2014)*, ACM, pp. 36-47, 2014, http://dx.doi.org/10.1145/2568225.2568318.

[37] J. Kazanavičius and D. Mažeika, "Migrating legacy software to microservices architecture", in *Proceedings of the IEEE 2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*, pp. 1-5, 2019, IEEE, https://doi.org/10.1109/eStream.2019.8732170.

[38] A. Ahmad and M. A. Babar, "A framework for architecture-driven migration of legacy systems to cloud-enabled software", in *Proceedings of the WICSA 2014 Companion Volume*, pp. 1-8, 2014, http://dx.doi.org/10.1145/2578128.2578232.

[39] A. Menychtas, C. Santzaridou, G. Kousiouris, T. Varvarigou, L. Orue-Echevarria, J. Alonso, J. Gorronogoitia, H. Bruneliere, O. Strauss, T. Senkova, B. Pellens, and P. Stuer, "ARTIST Methodology and Framework: A novel approach for the migration of legacy software on the Cloud", in *Proceedings of the IEEE 2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 424-431, 2013, IEEE, https://doi.org/10.1109/SYNASC.2013.62.