Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Nikola Tanković

OPTIMIRANJE KONFIGURACIJE INFORMACIJSKOGA SUSTAVA U OBLAKU SUKLADNO SPORAZUMU O RAZINI USLUGE

DOKTORSKI RAD

Zagreb, 2017.

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Nikola Tanković

OPTIMIRANJE KONFIGURACIJE INFORMACIJSKOGA SUSTAVA U OBLAKU SUKLADNO SPORAZUMU O RAZINI USLUGE

DOKTORSKI RAD

Mentori:
Prof. dr. sc. Mario Žagar
Izv. prof. dr. sc. Tihana Galinac Grbac

Zagreb, 2017.

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Nikola Tanković

OPTIMIZING CLOUD INFORMATION SYSTEM CONFIGURATION COMPLIANT WITH SERVICE LEVEL AGREEMENT

DOCTORAL THESIS

Supervisors:
Professor Mario Žagar, PhD
Associate Professor Tihana Galinac Grbac, PhD

Zagreb, 2017

Doktorski rad izrađen je na Zavodu za automatiku i računalno inženjerstvo
Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu

Mentori:
Prof. dr. sc. Mario Žagar
Izv. prof. dr. sc. Tihana Galinac Grbac

Doktorski rad ima: 161 stranicu

Doktorski rad br.: _____

O mentorima

Mario Žagar rođen je u Kupjaku (Gorski kotar) 1952. godine. Diplomirao je, magistrirao i doktorirao u polju računarstva na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva (FER), 1975., 1978. odnosno 1985. godine. Od 1977. godine radi na Zavodu za automatiku i računalno inženjerstvo FER-a i od tada je uključen u različite znanstvene projekte, nastavne aktivnosti, poboljšanja nastavnih programa, laboratorija i opreme. Bio je gostujući istraživač na Sveučilištu u Rostocku (Njemačka, 1980. godine), nositelj je „British Council fellowship" (UMIST - Manchester, 1983. godine) i „Fulbright fellowship" (UCSB - Santa Barbara, 1983./1984. godine). Godine 2002. izabran je za redovitog profesora računarstva u trajnom zvanju. Sudjelovao je u brojnim domaćim i međunarodnim istraživačkim projektima. Bio je član i voditelj pet znanstvenih projekata financiranih od Ministarstva znanosti, obrazovanja i sporta Republike Hrvatske. Bio je također suvoditelj/voditelj projekata „Unity Through Knowledge (UKF)", „TEMPUS" (u suradnji sa Sveučilištem Paderborn, Njemačka) te više projekata sa Sveučilištem Mälardalen, Västerås, Švedska. M. Žagar autor je i koautor više od stotinu članaka i pet knjiga iz područja računalnih arhitektura, raspodijeljenog, sveprisutnog i prožimajućeg računarstva, Interneta stvari, otvorenog računarstva, e-učenja i drugih Internetu usmjerenih tehnologija. Prof. Žagar je član IEEE/CS, počasni predsjednik udruge HrOpen (Hrvatske udruge za Otvorene sustave i Internet) te redoviti član Akademije tehničkih znanosti Hrvatske (HATZ). Sudjelovao je u više programskih odbora međunarodnih konferencija, član je u dva uredničkа odbora i recenzent u više međunarodnih časopisa. M. Žagar primio je Brončanu plaketu Josip Lončar od FER-a 1975. godine, nagrade INFORMATIKA93 i INFORMATIKA96 od Hrvatske Informatičke Zajednice (HIZ), najviše priznanje Zlatnu plaketu Josip Lončar za unaprjeđenje nastave, znanstveno-istraživačkog rada i organizacije od FER-a 2002. godine, nagradu „Najbolji edukator" od IEEE/CS Hrvatska Sekcija, 2006. godine, nagradu Hrvatske akademije znanosti i umjetnosti (HAZU) „Josip Juraj Strossmayer" za najuspješniji znanstveni rad u Republici Hrvatskoj u polju informacijskih znanosti, 2007. godine (za e-izdanje knjige „UNIX i kako ga iskoristiti"). Također je dobio „Priznanje za vođenje razvoja dinamičke aplikacije FERWeb" koja je od 2001. godine do danas središnja stranica Weba FER-a, prvu nagradu za najbolji e-kolegij na Sveučilištu u Zagrebu za predmet Otvoreno računarstvo (akademska godina 2009./2010.) i još niz drugih nagrada i priznanja.

Mario Žagar was born in Kupjak (Gorski kotar) in 1952. He received Dipl.ing., M.Sc.CS and Ph.D.CS degrees, all from the University of Zagreb, Faculty of Electrical Engineering and Computing (FER) in 1975, 1978 and 1985 respectively. From January 1977 he is working at the Department of Control and Computer Engineering at FER and since then was involved in different scientific projects, educational activities, improvements of the educational programs, laboratories and computer equipment. He was visiting researcher at the University

of Rostock, Germany (1980), received British Council fellowship (UMIST - Manchester, 1983) and Fulbright fellowship (UCSB - Santa Barbara, 1983/84). In 2002 he was promoted to Tenure Professor of Computing. He participated in numerous domestic and international research projects. He was a member and project leader in five scientific projects financed by Ministry of Science, Education and Sports of the Republic of Croatia. He was also co-leader/leader in Unity Through Knowledge (UKF) project, TEMPUS project (in cooperation with the University of Paderborn, Germany), several projects in cooperation with the Mälardalen University, Västerås, Sweden. M. Žagar is author and co-author of more than hundred articles and five books in the area of computer architectures, distributed, ubiquitous and pervasive computing, Internet of Things (IoT), Open computing, e-learning and other Internet related technologies. Prof. Žagar is senior member of IEEE/CS, honorable president of Croatian Society for Open Systems, regular member of Croatian Academy of Technical Sciences. He participated in several international conference program committees, he is a member of two journal editorial boards and serves as a technical reviewer for various international journals. M. Žagar was awarded by FER with the Bronze plaque Josip Lončar (1975), awards INFORMATIKA93 (1993) and INFORMATIKA96 (1996) from the Croatian Informatics Society and the highest award, Golden plaque Josip Lončar for the improvement of education, scientific and research work and organization of the Faculty from FER (2002), «Best educator» award, IEEE/CS Croatia Section (2006), Croatian Academy of Sciences and Arts (HAZU) award "Josip Juraj Strossmayer" for the most successful scientific work in Croatian in the field of information science (2007), for e-edition of "Unix and how to utilize it" book. He also received the "Acknowledgment for leading the development of the dynamic FERWeb application" started in 2001 and currently the central FER Web site, first prize for the best e-course at the University of Zagreb (academic year 2009/2010), Open Computing course and several other awards and acknowledgments.

Tihana Galinac Grbac rođena je u Puli 29. travnja 1977. godine. Diplomirala je, magistrirala i doktorirala u polju elektrotehnike na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva (FER), 2000., 2005. odnosno 2009. godine. Od prosinca 2007. zaposlena je na Tehničkom fakultetu Sveučilišta u Rijeci. Uspostavila je brojne međunarodne suradnje na području računarstva u sklopu Erasmus, Ceepus i DAAD programa. Ustrojila je Laboratorij za programsko inženjerstvo i obradu informacija – SEIP Lab. Sudjelovala je na dva znanstvena projekata Ministarstva znanosti, obrazovanja i sporta Republike Hrvatske i dva EU COST projekta. Trenutno je voditelj uspostavnog istraživačkog projekta financiranog od Hrvatske zaklade za znanost pod nazivom *Evolving Software Systems: Analysis and Innovative Approaches for Smart Management (EVOSOFT).* U razdoblju od 2000. do 2007. godine radila je kao programski i sistemski inženjer u razvojno-istraživačkom centru Ericsson Nikole Tesle u Zagrebu. Kontinuirano se usavršavala i u drugim razvojno-istraživačkim centrima unutar Ericsson globalne korporacije u Švedskoj, Njemačkoj, Italiji, Grčkoj, Australiji i Kanadi. Suradnja s Ericsson Nikolom Teslom se nastavila kroz niz projekata u sklopu SEIP Laba. Aktivna je u projektima lokalnog razvoja e-Županija te Centru kompetencija za pametne gradove. Pomaže lokalnim gospodarstvenicima i zajednicama u europskim projektima i prijavama, iUrban, eGov4 te projekta TechLab Grada Rijeke. Objavila je više od 50 znanstvenih radova u međunarodno priznatim časopisima i međunarodnim konferencijama u području programskog inženjerstva, i upravljanja u programskom inženjerstvu. Organizirala je nekoliko međunarodnih znanstvenih skupova i ljetnih škola, član je uređivačkog tima Engineering Review časopisa, uređivala je zbornike znanstvenih skupova, član je programskih i znanstvenih odbora u nizu međunarodnih konferencija. Recenzent je u većem broju međunarodnih časopisa. Članica je udruga IEEE, ACM i MIPRO.

Tihana Galinac Grbac was born in Pula in 1977. She received B. Sc., M. Sc. and Ph. D. degrees in electrical engineering from the University of Zagreb, Faculty of Electrical Engineering and Computing (FER), Zagreb, Croatia, in 2000, 2005 and 2009, respectively. From December 2007 she is working at the Faculty of Engineering University of Rijeka. She established number of international collaborations in Computer Science within Erasmus, Ceepus and DAAD programs. She established Software Engineering and Information Processing Laboratory – SEIP Lab. She participated in 2 scientific projects financed by the Ministry of Science, Education and Sports of the Republic of Croatia and 2 EU COST projects. Currently she is a project leader of the installation research project: *Evolving Software Systems: Analysis and Innovative Approaches for Smart Management (EVOSOFT)* financed by the Croatian Science Foundation. In period from 2000. until 2007. She was working as software and system engineer in research and development center of Ericsson Nikola Tesla in Zagreb. She was professionally specialized in other research and development centers within Ericsson global corporation: Sweden, Germany, Italy, Greece, Australia, Canada. Collaboration with Ericsson Nikola Tesla is conti-

nued through number of projects realized within SEIP Lab. She is active in local development projects e-Županija, i Competence Center for Smart Cities and supports local bodies and communities in European projects and applications, iUrban i eGov4, and TechLab, projects of City of Rijeka. She published more than 50 papers in journals and conference proceedings in the area of software engineering and software engineering management. Assist. Prof. Galinac Grbac organized several international scientific conferences and summer schools. She is associate editor in Engineering Review journal and was editing SQAMIA conference proceedings, and is a member program and scientific boards in number of international conferences. She serves as a technical reviewer for various international journals. She is a member of IEEE, ACM and MIPRO.

Voljenoj supruzi i djeci.

Nikola Tanković

Zahvala

Neizmjerno hvala mentorima prof. dr. sc. Mariju Žagaru na svoj mudrosti koju je nesebično dijelio kroz sve ove godine doktorskog studija i mentorici izv. prof. dr. sc. Tihani Galinac Grbac na brojnim usmjerenjima, a posebno što je probudila u meni novi žar za znanstvenim istraživanjem. Hvala članovima povjerenstva za ocjenu i obranu doktorskog rada: prof. dr. sc. Krešimiru Fertalju, izv. prof. dr. sc. Željki Car i izv. prof. dr. sc. Mladenu Tomiću koji su detaljno proučili disertaciju te ju unaprijedili vrijednim savjetima i ispravcima. Veliko hvala Mirjani Grubiši i Jadranki Lisek na pomoći tijekom studija. Zahvaljujem tvrtki Superius d.o.o. iz Pule na čelu s direktorom Nikolom Rovisom na financiranju doktorskog studija.

Hvala roditeljima što su u mene usadili nepresušnu znatiželju i čvrste radne navike. Vaša ljubav rodila je vjerom da mogu hrabro koračati nepoznatim putovima. Hvala braći i sestrama što su uvijek blizu.

Najveće hvala supruzi Ani. Hvala joj za ljubav koja je živa, koja hrabri, motivira, tješi, prašta i ispunjava. I na kraju, hvala dragom Bogu što me uvijek u svemu prati, što uz Njega uvijek imam sve jer je On moje sve.

Sažetak

Organizacije sve više ovise o kvaliteti informacijskih sustava kao okosnici stabilnog poslovanja. Za razliku od zatvorenih sustava koje koristi unaprijed određeni broj korisnika, informacijski sustavi okrenuti vanjskom poslovanju moraju biti skalabilni i otporni na promjene u radnom opterećenju. Zbog osjetljivosti i na najmanje ispade, garancija kvalitete usluge pojedinog sustava ostvarena je strogim sporazumima o razini usluge. U ostvarenju takvih sustava, infrastruktura računarstva u oblaku pruža dinamičan zakup sklopovlja razmjeran trenutnom radnom opterećenju te ostvaruje elastičnost usluge.

Ova disertacija predstavlja model arhitekture informacijskoga sustava koji odlikuje mogućnost rekonfiguracije u sklopu prilagodbe promjenjivom radnom opterećenju te mogućnost optimiranja njegove konfiguracije. Cilj optimiranja je ostvarivanje željene kvalitete usluge definirane sporazumom uz minimalni trošak zakupa infrastrukture u oblaku.

Za ocjenu atributa kvalitete pojedinih konfiguracija predložena je metoda koja omogućuje automatsku konfiguraciju sustava, simuliranje promjenjivog radnog opterećenja te analizu prikupljenih podataka. Kako prostor konfiguracija pojedinog sustava naglo raste zavisno broju komponenti, za efikasno pretraživanje i određivanja optimalnih kandidata predlaže se genetski algoritam koji koristi metodu simulacije u određivanju atributa kvalitete. Predloženi simulacijski model zasnovan je na mreži redova čekanja. Zbog elastičnosti modelirane aplikacije predlaže se simulator koji raspolaže mogućnošću prilagodbe modela sustava tijekom izvođenja.

Naposljetku, predložen je model razvojne okoline ElaClo, skraćeno od *Elastic Cloud*, koja provodi optimizaciju koristeći računalni oblak. Realiziran je prototip okoline ElaClo kako bi se provela validacija optimiranja aplikacije iz domene maloprodaje koju svakodnevno koristi nekoliko tisuća korisnika. Validacija simulacijskog modela provedena je na podacima prikupljenim tijekom provedbe 40 uzastopnih optimizacija gdje se pomoću neparametrijskih statističkih testova pokazalo da su ostvarene zadane pretpostavke o preciznosti i efikasnosti postupka. Predstavljena automatska optimizacija konfiguracije omogućuje davatelju usluge značajnu vremensku uštedu u ostvarenju optimalne konfiguracije te informirano odlučivanje prilikom oblikovanja sporazuma o razini usluge u skladu s očekivanim troškovima izvođenja u oblaku.

Ključne riječi: dinamička rekonfiguracija informacijskoga sustava, elastičnost, računarstvo u oblaku, optimiranje konfiguracije informacijskoga sustava, sporazum o razini usluge

Extended Abstract

Optimizing Cloud Information System Configuration Compliant with Service Level Agreement

Applications in the cloud are implemented with elasticity attribute to achieve lower operating costs without sacrificing quality. Software architects strive to provide efficient services by deciding on a software configuration: a set of structural architectural decisions. For a given application, there can be numerous software configuration alternatives creating the need for automated optimization methods. Current optimization approaches rely on experts providing application performance models built upfront, based on their experience and the requirements provided. While such techniques are effective and valuable, they require additional maintenance effort as the software evolves.

This dissertation introduces ElaClo, a framework for optimizing application configuration in a cloud environment. ElaClo's main contribution is in providing optimization in the software assembly phase from automatically extracted application models. ElaClo provides workload generation, monitoring, configuration management, elasticity mechanisms, and algorithms to support the optimization process. ElaClo was implemented as an expert tool and evaluated on a real-life cloud application from the retailing business domain where ElaClo was applied to select optimal configurations with regards to service response time objectives and infrastructure costs. The efficiency of the optimization process and the quality of optimization results were validated quantitatively on a set of optimization runs. Results demonstrate the effectiveness of the suggested framework in yielding optimal configurations.

Theoretical background and previous research

The first part of the thesis introduces all the necessary concepts. It defines the software architecture and software configuration and their relationship where architecture restricts possible configurations. It defines a software component and different architectural styles as existing patterns for solving repeating requirements. It then introduces component-based systems and the role of component model as a specification for developing software components. An explanation of software component life-cycle is also provided covering all phases from releasing to retiring components. Software quality is also defined together with quality attributes, metrics, and criteria that are commonly used in the service-level agreements.

The context of this thesis is optimizing the software configuration in the cloud so the concept of cloud computing is also explained with special attention given to cost models. Application workload is defined as the sum of all individual user requests, and system scalability as the base requirement for addressing volatile workloads. In order to scale effectively, elasti-

city attribute of the system is introduced as its ability to adapt the system processing capacity to closely follow the current workload demands.

The final part of this chapter is the dissemination of the process of systematic literature review — SLR. First, the methodology for SLR is given together with the plan for conducting the literature search. Literature search process resulted in 4648 publications which are narrowed down to 48 final publication included in SLR. A taxonomy is given which involves all the different aspects of configuration optimization by which the selected publications have been classified. Finally, a critical analysis of previous work is given highlighting the novelties in this thesis. These are primarily the optimization of the configuration under the dynamic workload, quality criteria expressed with quantiles, and introducing a cloud development environment for optimizing the configuration after the initial development phase. This contrasts the previous work where optimization relies on the provided performance models in the design phase.

Elastic component-based information system architecture

This chapter enumerates the applied architectural styles in the specification of an elastic architecture with dynamic reconfiguration capabilities. The proposed architecture is described from different architectural views together with a model for the system configuration that allocates components to elastic groups. Each elastic group can be dynamically reconfigured during runtime for implementing elastic behavior. A meta-model of the architecture is also given in the form of a UML profile.

Configuration optimization problem is formulated based on the proposed elastic component-based architecture. The configuration search-space size is also defined based on all the permutations of allocating components to elastic groups. Finally, an extension of component-based software process model is given that places the optimization step between existing implementation and component assembly phases.

A method for evaluating configurations compliant with service-level agreements

In order to assess individual quality attributes of each configuration under the dynamic workload, the method introduced in this chapter first defines two models: (1) application workload model, and (2) SLA model. Workload model is defined as the non-homogeneous Poisson Processes belonging to multiple request classes. In order to generate such volatile workload, an algorithm is also provided. Finally, methods for evaluation each configuration is provided by a model of components that coordinate the optimization, generate the synthetic workload and monitor the key metrics of the targeted application.

In order to apply different configurations, an algorithm is introduced to generate all possible configurations of the given application model. For the purpose of collecting the key

metrics, an algorithm is also defined that approximates the response time distribution.

An evolution algorithm for determining optimal configurations

An evolution algorithm is defined in this chapter for the efficient exploration of the space of possible application configurations. The rationale for selecting the genetic algorithm in evolutionary approach is primarily based on the ability to work with populations thus yielding Pareto optimal sets of configurations. All steps of the genetic algorithm are disseminated together with the candidate representation method. For evaluating each candidate a multi-class queue network is used together with a method for conducting the simulation. The simulation process is able to simulate elasticity by dynamically changing the queueing network to represent different states of the system expressed by the number of cloud resources used.

An execution environment for optimizing configurations

This chapter gives a model of the execution environment ElaClo (*Ela*stic *clo*ud). ElaClo is a distributed system that can be divided into central ElaClo optimization environment and application framework integrated into a targeted application that supports the optimization process. Component models are given for both the central ElaClo components and application components. The process of adapting the configuration is described together with possible communication alternatives between components of the targeted application. Finally, an ElaClo user interface is presented in the form of web application.

Validation

In this chapter, the validation the optimization process is given. It describes the validation goals, research questions, and the methodology for answering them. ElaClo was implemented as a prototype tool and it was then evaluated on a real-life system. Case study introduces the CashRegister application used in the Croatian, Slovenian, and Czech market by several thousand active users. The dynamic workload has been modeled by analyzing the usage of the real system.

The optimization process was statistically proven to converge based on the data gathered from 40 optimization runs. The quality of results was also shown to be satisfactory by evaluating them in the real cloud system using ElaClo. Validation concludes by enumerating all limitations of this research and threats to validity.

Conclusion

This dissertation introduced the ElaClo framework for optimizing application configuration according to response-time based service-level objectives and cloud infrastructure ope-

ration costs. ElaClo provides these capabilities by providing a development framework to be used in the targeted application together with a cloud-based development environment with an integrated simulator to accelerate optimization process. By using ElaClo, developers are empowered with meaningful information about configuration evaluation and optimization processes and so they can target and offer feasible service-level objectives.

ElaClo was evaluated on a real-life case study and indicated non-trivial optimal topologies. Since the space of all possible topologies is large, ElaClo determines optimal topology candidates using a simulation environment by automatically extracting simulation models based on measurement from the real application components. Suggested topologies were re-evaluated on a real infrastructure under same volatile workload model, exploiting ElaClo to automatically manage the cloud topology, generate workload, monitor and enable service elasticity. Several statistical tests were applied to evaluate the accuracy and efficiency of the optimization process in the simulated environment. It was demonstrated and statistically supported that applying genetic algorithms on simple queue network performance models can differentiate topologies that tend to be optimal when tested in the real environment.

The largest challenge in applying ElaClo is the process of integrating ElaClo's application framework into applications that need optimization. This requires additional effort from service providers and limits ElaClo applicability in different component framework implementations. For that reason, future work will involve integrating application containers (e.g. Docker) to facilitate managing incremental changes to component artifacts. This will enable evaluation of different configurations across application releases throughout the development process and enable optimization of systems with heterogeneous components.

Keywords: dynamic information system reconfiguration; elasticity; cloud computing; optimizing information system configuration; service-level agreement; genetic algorithm

Sadržaj

1. Uvod 1

1.1. Područje istraživanja i motivacija 1

1.2. Ciljevi istraživanja 3

1.3. Metodologija istraživanja 4

1.4. Znanstveni doprinosi istraživanja 5

1.5. Struktura rada 6

2. Teoretske postavke i prethodna istraživanja 9

2.1. Arhitektura, konfiguracija i kvaliteta informacijskoga sustava 9

2.1.1. Definicije 9

2.1.2. Sustav zasnovan na komponentama 13

2.1.3. Proces postavljanja komponente u izvedbenu okolinu 13

2.1.4. Model razvojnog procesa sustava zasnovanih na komponentama . . . 16

2.1.5. Atributi kvalitete 18

2.2. Sporazum o razini usluge 22

2.2.1. Metrike sporazuma o razini usluge 23

2.3. Računarstvo u oblaku 24

2.3.1. Definicije 24

2.3.2. Modeli naplate 25

2.4. Radno opterećenje i kapacitet informacijskoga sustava 25

2.4.1. Radno opterećenje 26

2.4.2. Skalabilnost sustava 26

2.4.3. Elastičnost sustava 29

2.4.4. Postupak adaptacije kapaciteta 30

2.4.5. Ostvarivanje elastičnosti 31

2.5. Sustavni pregled literature 33

2.5.1. Metodologija 34

2.5.2. Proces prikupljanja literature 35

2.5.3. Oblikovanje taksonomije 38

2.5.4. Kategorizacija dosadašnjih istraživanja . . . 40
2.5.5. Diskusija prethodnih istraživanja . . . 41
2.5.6. Optimiranje konfiguracije modeliranjem sustava . . . 42
2.5.7. Zaključak . . . 43

3. Arhitektura elastičnog informacijskoga sustava zasnovana na komponentama 45
3.1. Zahtjevi prema modelu arhitekture . . . 46
3.2. Primijenjeni arhitekturalni stilovi . . . 46
3.3. Pogledi na predloženu arhitekturu . . . 48
3.3.1. Pogled na module . . . 48
3.3.2. Pogled na komponente i spojke . . . 48
3.3.3. Pogled na razmještaj . . . 52
3.4. Meta-model arhitekture . . . 54
3.5. Optimiranje konfiguracije . . . 56
3.5.1. Optimizacijski problem . . . 56
3.5.2. Model troškova infrastrukture . . . 57
3.5.3. Formulacija optimizacijskog problema . . . 58
3.6. Model razvojnog procesa zasnovan na predloženoj arhitekturi . . . 60

4. Metoda ocjene i odabira konfiguracije sustava u skladu sa sporazumom o razini usluge . . . 63
4.1. Simuliranje radnoga opterećenja . . . 65
4.1.1. Model radnoga opterećenja . . . 65
4.1.2. Generiranje radnog opterećenja . . . 67
4.2. Model ugovora SLA . . . 68
4.2.1. Metrike sporazuma . . . 68
4.3. Metoda generiranja i ocjene konfiguracija . . . 70
4.3.1. Generiranje mogućih konfiguracija . . . 71
4.3.2. Metoda praćenja relevantnih metrika . . . 74
4.3.3. Metoda redefiniranja ugovora SLA . . . 77
4.4. Zaključak . . . 77

5. Evolucijski algoritam za odabir optimalnih konfiguracija . . . 79
5.1. Zahtjevi pri određivanju optimizacijskog algoritma . . . 79
5.2. Tehnika optimizacije . . . 80
5.3. Postupak provođenja evolucijskog algoritma . . . 81
5.4. Reprezentacija kandidata . . . 83
5.5. Evaluacija kandidata . . . 83

5.5.1. Provođenje simulacije mrežom redova čekanja . . . 85
5.6. Reprodukcija kandidata . . . 89
5.7. Kriterij zaustavljanja . . . 90
5.8. Zaključak . . . 91

6. Izvedbena okolina za optimiranje konfiguracije . . . 93
6.1. Aplikacijski okvir za ostvarivanje elastične konfiguracije . . . 94
6.2. Razvojno okruženje za optimiranje konfiguracije u skladu sa sporazumom o razini usluge . . . 95
6.3. Izvođenje programske usluge u sklopu postupka optimizacije . . . 98
6.3.1. Postavljanje ciljane programske usluge na infrastrukturu oblaka . . . 98
6.3.2. Komunikacija između komponenti . . . 100
6.4. Korisničko sučelje . . . 103

7. Validacija . . . 107
7.1. Ciljevi validacije . . . 107
7.2. Metodologija validacije . . . 109
7.3. Implementacija prototipnog alata ElaClo . . . 111
7.4. Studija slučaja . . . 111
7.4.1. Definiranje sporazuma SLA . . . 113
7.4.2. Radno opterećenje . . . 113
7.4.3. Cilj optimizacije . . . 115
7.5. Postupak optimizacije . . . 115
7.5.1. Primjena evolucijskog algoritma u reduciranju broja kandidata . . . 117
7.5.2. Validacija rezultata evolucijskog algoritma . . . 119
7.5.3. Evaluacija odabranih kandidata pomoću alata ElaClo . . . 122
7.5.4. Ograničenja istraživanja . . . 124

8. Zaključak . . . 127
8.1. Implikacije istraživanja . . . 128
8.2. Budući pravci razvoja . . . 129

Literatura . . . 131

Životopis . . . 159

Biography . . . 161

Poglavlje 1

Uvod

Efikasno upravljanje kompleksnošću jedan je od bitnih ciljeva programskog inženjerstva kao discipline koja teži primjeni sustavnog, disciplinirano i praktičnog razvoja programskih sustava [1]. Pritom je tehnika apstrakcije sredstvo koje omogućuje pažljivo usmjeravanje koncentracije na rješavanje izdvojenih problema koristeći prilagođenu razinu detalja. Apstrakcija u ostvarenju računalnih sustava zasniva se na postavljanju procesa izgradnje složenih sustava kao niza apstraktnih slojeva koji koriste jednostavnije gradivne elemente. Svaki sljedeći sloj pruža apstraktni model korištenja nižih konstrukata čime omogućuje njihovu lakšu uporabu. Time je omogućena praktična izgradnja složenih sustava koji se u konačnici oslanjaju na svega nekoliko nužnih teorema postavljenih kroz otkrića velikana računarske znanosti*.

Oblikovanje jednostavnijih dijelova u složene sustave odvija se u fazi dizajna programske podrške [2]. Pritom je definiranje arhitekture sustava glavni čimbenik za ostvarenje njegove kvalitete koja se izražava kroz različite atribute kvalitete (engl. *quality attributes*) [3, 4]: mjerljive pokazatelje različitih poželjnih karakteristika programskih sustava. Te se karakteristike primjerice mogu odnositi na ocjenu različitih aspekata kod razvoja sustava, pouzdanosti u radu, ili radne karakteristike sustava (engl. *software performance*) koje ono ostvaruje prilikom izvršavanja [5].

1.1 Područje istraživanja i motivacija

Jedna od suvremenijih računarskih apstrakcija kao podloga za izgradnju većih raspodijeljenih programskih sustava jest računarstvo u oblaku. Njegova je zadaća pojednostaviti izgradnju složenih sustava tako da se razdvoji poimanje apstraktnih računalnih resursa od konkretnog fizičkog sklopovlja. Računarstvo u oblaku (skraćeno oblak) zasnovano je nad

*Bacon, Leibnitz, Boole, Turing, Shannon i Mors otkrivaju kako se svaka informacija može predstaviti koristeći pomoću dva ili više osnovna stanja, zatim slijedi otkriće Turingovog stroja i formulacije Turingove kompletnosti te naposljetku Boehmovo i Jacopinijevo definiranje strukturnog programiranja kao postulata o nužnosti tri gramatička pravila za definiranje bilo kojeg niza složenih instrukcija

tehnologijom virtualizacije sklopovlja i dijeljenja resursa između više korisnika [6] pri čemu upravljanje postavljanjem i održavanjem infrastrukture sklopovlja preuzimaju *davatelji usluga* u oblaku. *Korisnici oblaka* smatraju se pojedinci ili organizacije koji ostvaruju programske aplikacije ili usluge koristeći usluge oblaka u obliku najma infrastrukture. Ubrzanim razvojem tržišta usluga u oblaku [7], posebice uključivanjem organizacija poput *Googlea* ili *Amazona*, ali i brojnih drugih organizacija, računalni resursi postaju dostupni na zahtjev kao što je to slučaj kod električne energije ili vode. Sukladno time, podmiruju se razmjerno potrošnji [8]. Takav model naplate dodatno omogućuje i jednostavnije ostvarivanje razmjernog rasta ukupnog kapaciteta obrade informacija nekog sustava. Razlog tomu jest visoka razina automatizacije pri zakupu infrastrukture koju omogućuju usluge oblaka [9].

U složene programske sustave ubrajaju se i *informacijski sustavi*: sustavi koji služe za prikupljanje, organizaciju, pohranu i komunikaciju informacija između većeg broja korisnika [10]. Informacijski sustavi grade se od jedne ili više korisničkih aplikacija koje zaokružuju određenu užu funkcionalnost. Proučavanje informacijskih sustava obuhvaća široko područje uključujući osobe i računalne sustave koji sudjeluju u procesu obrade informacija [11]. Ovaj rad stavlja naglasak na područje razvoja raspodijeljenih aplikacija kao sastavnica informacijskih sustava.

Sve više današnjih organizacija posluje na globalnoj razini elektroničkim putem. Globalni razmjeri donose velike promjene u opterećenju (engl. *workload*) informacijskih sustava koji pružaju elektroničke usluge. Promjenljivo opterećenje posebno vrijedi za aplikacije u oblaku koje su pružene kao usluge u oblaku (engl. *Software as a Service*) [12] gdje ukupan broj korisnika sustava nije poznat tijekom dizajna. Budući da svaki korisnički zahtjev upućen prema sustavu predstavlja pojedinačno korištenje sustava, niz korisničkih zahtjeva predstavlja ukupno opterećenje sustava [13]. Arhitektura takvog sustava mora biti ostvarena u vidu unaprijed nepoznatog ukupnog opterećenja. Klasični pristup koji unaprijed određuje kapacitet sustava nije moguć. Neadekvatna arhitektura sustava uzrokuje pojavu povećanog vremena odaziva ili ispada usluge uslijed velike i nagle potražnje. Takva degradacija usluge ne samo da uzrokuje oportunitetne troškove organizacije koja pruža usluge, već narušava cjelokupni imidž što povlači dalekosežnije posljedice.

Elektroničko poslovanje putem mrežnih usluga zahtjeva osmišljavanje informacijskih sustava koji su u stanju kontinuirano prilagođavati ukupni kapacitet obrade prema trenutnom opterećenju, što definiramo kao prilagodljivost ili elastičnost sustava [14, 15]. Najčešći oblik prilagodljivosti kapaciteta sustava ostvaruje se korištenjem promjenjive količine infrastrukturnih komponenti u obliku broja poslužitelja na kojima se sustav izvodi [16]. Dinamično korištenje resursa omogućuje upravo usluga računarstva u oblaku omogućivši naplatu samo trenutno korištenih resursa u dogovorenoj obračunskoj jedinici [8].

Način na koji je ostvarena elastičnost ima direktan utjecaj na radne karakteristike cje-

lokupnog sustava. Izgradnja elastičnog sustava uvodi novu razinu složenosti te je potrebno osmisliti novi sloj apstrakcije i pripadajuću metodu optimizacije koja efikasno povezuje ljudski definirane koncepte iz višeg sloja konceptima nižeg sloja, poput infrastrukture oblaka. Takva optimizacija mora biti vođena kriterijima kvalitete koji su usklađeni s očekivanjima korisnika.

Kvaliteta usluge kao ukupni odraz svih kriterija kvalitete propisuje se putem sporazuma o razini usluge (engl. *Service Level Agreement*, skraćeno SLA) [17]. SLA je ugovor koji obvezuje usluge da kontinuirano održava kriterije kvalitete unutar zadanih granica uz definirane penale. Time SLA uključuje i pravne norme [18]. Ovo istraživanje usmjereno je prema tehničkim aspektima SLA-ova koji definiraju kvalitetu usluge.

Analizom dosadašnjeg znanstvenog dosega, utvrđeno je da nije dana dovoljna pažnja definiciji i optimizaciji arhitekture sustava promjenljivog opterećenja gdje regulacija putem sporazuma postavlja dodatni izazov.

1.2 Ciljevi istraživanja

Glavni cilj ovog istraživanja je definiranje modela elastične arhitekture kao sloja apstrakcije za lakše upravljanje elastičnošću te pripadnog postupka i alata za optimiranje konfiguracije sustava u ostvarivanju unaprijed definiranih radnih karakteristika. Optimizacija konfiguracije ostvarena je kroz odabir optimalne konfiguracije programskih komponenti koristeći raspodijeljenu infrastrukturu računarstva u oblaku. Razmještaj komponenti na raspodijeljenu infrastrukturu važan je čimbenik arhitekture aplikacije [19], gdje svaki jedinstveni razmještaj podrazumijeva različita ostvarenja komunikacijskih kanala između komponenti te različitu topologiju mrežne infrastrukture [20]. Cilj optimizacije konfiguracije u ovom istraživanju jest ostvariti željene radne karakteristike uz optimalne troškove zakupa infrastrukture u oblaku pri promjenjivom opterećenju.

Prethodna istraživanja utvrdila su da razmještaj komponenti na sklopovlje značajno utječe na radne karakteristike aplikacije [21, 22]. Doprinos ovog istraživanja je u optimizaciji elastičnih sustava s kriterijima kvalitete zadanih prema SLA-u. Pri takvim uvjetima dizajn arhitekture mora omogućiti prilagodljivost različitim opterećenjima automatskim zakupom ili otpuštanjem poslužitelja na kojima se aplikacija izvodi [15]. Upravo takva automatizacija omogućuje višu razinu apstrakcije izgrađenu nad modelima računarstva u oblaku.

U svrhu ostvarenja konkretnog alata za provođenje optimizacije sustava, ovo istraživanje definira model izvedbene okoline za optimizaciju konfiguracije raspodijeljene aplikacije u oblaku. Predložena okolina prilagođena je aplikaciji koja implementira predloženi model arhitekture čime okolina dobiva uvid u korištene komponente aplikacije i njihove zavisnosti. Taj uvid omogućuje automatsku izgradnju simulacijskih modela aplikacije čime se može ubrzati postupak optimiranja njezine konfiguracije.

Izvedbena okolina u stanju je autonomno ostvariti različite konfiguracije željene aplikacije te ocijeniti pripadne radne karakteristike i troškove izvođenja. Za potrebe takve ocjene nad aplikacijom generira se korisnički definirano promjenljivo radno opterećenje u skladu s očekivanim korištenjem aplikacije.

Cilj postupka optimizacije jest automatsko otkrivanje različitih konfiguracija unutar postojeće arhitekture aplikacije koje zadovoljavaju željenu razinu radnih karakteristika uz minimalne troškove izvođenja na infrastrukturi računarstva u oblaku.

1.3 Metodologija istraživanja

U ovom radu primijenjeni su elementi *empirijske metode* i *inženjerske metode* [23] kao dvije prilagodbe znanstvene metode u sklopu inženjerske prakse [24, 25]. Primijenjena metoda se sastoji od četiri koraka koji se mogu ponavljati dok ne postoji novih izglednih poboljšanja sustava:

1. analiza postojećih rješenja
2. prijedlog novog rješenja
3. izrada novog rješenja
4. definiranje statističke metode za validaciju
5. validacija rješenja.

Kao dio analize postojećih rješenja proveden je sustavni pregled literature proveden prema smjernicama prilagođenim području programskog inženjerstva [26]. Na temelju takve analize osmišljeno je novo rješenje u obliku izvedbene okoline za optimiranje konfiguracije aplikacije.

Sve komponente predloženog rješenja ostvarene su u obliku prototipnog alata, dok su za potrebe validacije unaprijed definirane metrike i statistički testovi koji moraju ukazati na zadovoljavanje željene kvalitete samog rješenja. Pri provođenju validacije predloženog modela izvedbene okoline koristi se studija slučaja nad odabranim tipičnim primjerom populacije ciljanih aplikacija [27].

Studije slučaja ograničene su u slobodi provođenja istraživanja zbog nepredvidivih utjecaja vanjskih čimbenika [28]. Kako bi se osiguralo nesmetano prikupljanje svih mogućih podataka tijekom eksperimenta, za potrebe ovog rada izrađena je replika sustava iz studije. Replika ostvaruje osnovni skup funkcionalnosti stvarnog sustava u skladu s arhitekturom predstavljenom u poglavlju 3. Replika zadržava ključne funkcionalnosti odabranog tipičnog primjera uz potpunu kontrolu nad izvođenjem istraživanja čime poprima odlike eksperimenta [28]. Negativna posljedica ovakvog pristupa jest da tijekom izrade replike može biti promijenjen neki važan čimbenik, zbog čega je važno pokazati da replika zadržava svojstva tipične aplikacije. U ovom istraživanju, posebna pažnja usmjerena je u niz provjera da replika ostvari sličan odaziv na promjenljivo radno opterećenje kao aplikacija iz studije slučaja.

1.4 Znanstveni doprinosi istraživanja

Znanstveni doprinosi ove disertacije uključuju skup modela, metoda i algoritama potrebnih za provođenja postupka optimiranja konfiguracije informacijskoga sustava sukladno sporazumu o razini usluge:

1. **Model izvedbene okoline za optimiranje konfiguracije informacijskoga sustava u oblaku radi postizanja ciljnog radnog opterećenja i troškova**
   U sklopu ovog doprinosa ostvaruje se model izvedbene okoline za automatizirano provođenje optimizacije konfiguracije raspodijeljene aplikacije informacijskoga sustava. Model izvedbene okoline definira sve potrebne komponente: (1) komponente za modeliranje i generiranje promjenljivog radnog opterećenja aplikacije, (2) komponente za automatsku prilagodbu kapaciteta aplikacije radnom opterećenju te (3) komponente za upravljanje konfiguracijom informacijskoga sustava nad infrastrukturom računarstva u oblaku. Navedena okolina ostvarena je kao prototip pomoću kojeg je provedena validacija studijom slučaja tipičnoga informacijskoga sustava iz poslovne domene.
2. **Arhitektura informacijskoga sustava zasnovanoga na komponentama u svrhu dinamičke prilagodbe konfiguracije**
   Kako bi se omogućila dinamička promjena konfiguracije informacijskoga sustava definiran je model arhitekture za ostvarenje raspodijeljenog sustava prilagodljivog kapaciteta. Modelom se definiraju zavisnosti između komponenata aplikacije te konfiguracija razmještaja komponenti na infrastrukturu računarstva u oblaku. Konfiguracija aplikacija izvor je diferencijacije u ostvarenim kriterijima kvalitete i troškovima najma infrastrukture oblaka koje aplikacija ostvaruje prilikom prilagođavanja radnom opterećenju. Navedeni model koristi se kao strojna reprezentacija ciljane aplikacije u procesu optimiranja konfiguracije. Naposljetku, predložena je i nadogradnja procesa razvoja takvog informacijskih sustava koja uključuje proces optimiranja.
3. **Metoda ocjene značajki i troškova te odabira prikladne konfiguracije u skladu sa sporazumom o razini usluge**
   Za potrebe provođenja optimiranja razvijena je metoda koja omogućuje kvantitativnu analizu i usporedbu različitih konfiguracija. Metoda se sastoji od nekoliko dijelova:
   - Automatsko ostvarivanje pojedine konfiguracije aplikacije pomoću infrastrukture računarstva u oblaku koja prati prethodno predložen model arhitekture. Sukladno konfiguraciji, metoda osigurava odgovarajuće komunikacijske kanale između komponenti aplikacije koristeći mrežne usluge
   - Simuliranje promjenljivog opterećenja aplikacije te ocjenu radnih značajki i troškova izvođenja aplikacije statističkom analizom prikupljenih metrika. Simulacija opterećenja ostvarene je pomoću generatora prometa sukladno zadanom modelu opterećenja zasnovanog na nehomogenom Poissonovom procesu

- Usporedba troškova i značajki prilikom izvođenja aplikacije u različitim konfiguracijama ističući najbolja rješenja koja zadovoljavaju kriterije definirane sporazumom o razini usluga
- Postupak redefinicije sporazuma o razini usluge sukladno prikupljenim podacima tijekom optimizacije.

4. **Evolucijski algoritam za određivanje optimalnih konfiguracija pomoću računalne simulacije**
   U svrhu reduciranja velikog broja mogućih konfiguracija ostvaren je genetski algoritam za efikasniju pretragu i evaluaciju kriterija kvalitete pojedine konfiguracije. Definirana je reprezentacija populacije i potrebni operatori genetskog algoritma te metoda simulacije pomoću simulacijskog modela mreže redova čekanja. Kao dio simulacije ostvarena je i mogućnost prilagodbe kapaciteta aplikacije te upotreba nehomogenih resursa u modeliranju različitih vrsta poslužitelja infrastrukture oblaka.

1.5 Struktura rada

Disertacija je podijeljena na tri cjeline. Prvu cjelinu sačinjava drugo poglavlje koje opisuje i definira korištene termine te daje pregled dosadašnjeg znanstvenog dosega pomoću sustavnog pregleda literature. Drugi dio, koji čine treće, četvrto, peto i šesto poglavlje, pokriva glavne doprinose disertacije, dok treći dio (sedmo i osmo poglavlje) opisuje postupak validacije uz diskusiju rezultata i zaključak. U nastavku slijedi detaljniji sadržaj svakog poglavlja.

Drugo poglavlje opisuje temelje koji se dotiču arhitekture i konfiguracije informacijskoga sustava, skalabilnosti sustava i dinamičke prilagodbe kapaciteta. Pojašnjeni su sporazumi o razini usluge koji se koriste kao okvir za provođenje postupka optimiranja. Naposljetku, opisuje se postojeći proces razvoja sustava zasnovanog na komponentama te daje sustavni pregled literature.

Treće poglavlje definira arhitekturu informacijskoga sustava zasnovanog na komponentama koji ostvaruje prilagodbu radnom opterećenju. Opisani su zahtjevi prema takvoj arhitekturi, primijenjeni postojeći arhitekturalni stilovi te definicija modela arhitekture. Prema predloženom modelu arhitekture opisan je optimizacijski problem koji se rješava ovim radom.

U četvrtom poglavlju definirane su metode za ocjenu i odabir konfiguracije informacijskoga sustava koji slijedi model iz trećeg poglavlja. Postupak ocjene uključuje simuliranje radnog opterećenja aplikacije te je opisan model opterećenja i postupak njegovog generiranja. Definiran je korišteni model ugovora SLA korišten pri optimiranju te algoritam za generiranje mogućih konfiguracija koje sustav može ostvariti. Opisana je korištena metoda za praćenje ključnih metrika kriterija kvalitete te metoda za redefiniranje ugovora SLA u skladu s kriterijima kvalitete koje sustav može postići.

Peto poglavlje iznosi evolucijski algoritam koji koristi simulacijski model aplikacije te samim time brže evaluira pojedinačne konfiguracije informacijskoga sustava. Opisan je postupak provođenja simulacija pomoću modela iz teorije redova čekanja te pojedine sastavnice evolucijskog algoritma.

Šesto poglavlje predstavlja izvedbenu okolinu za optimiranje konfiguracije aplikacija koje slijede model arhitekture predstavljen u trećem poglavlju. Predložena izvedbena okolina koristi metode ocjene konfiguracija iznesene u četvrtom poglavlju te evolucijski algoritam za prethodnu selekciju pogodnih kandidata predstavljenu u petom poglavlju. Predstavljena je struktura izvedbene okoline, opis svih sadržanih komponenti te korisničko sučelje za korištenje predložene izvedbene okoline.

U sedmom poglavlju opisan je postupak provođenja validacije metode optimiranja. Definirani su ciljevi validacije uz primijenjenu metodologiju i korištene metrike. Slijedi opis provođenja studije slučaja za potrebe validacije i rezultati validacije uz identificirana ograničenja istraživanja.

Konačno, osmo poglavlje zaključuje rad dajući sažeti pregled predstavljenog istraživanja te analizira značaj ostvarenih znanstvenih doprinosa. Naposljetku, pružaju se daljnje smjernice budućih istraživanja koje je moguće ostvariti temeljem rezultata iz okvira ove disertacije.

Poglavlje 2

Teoretske postavke i prethodna istraživanja

Ovo poglavlje iznosi definicije osnovnih termina vezanih uz područje disertacije. Definirana je arhitektura sustava te njegova konfiguracija, koja je predmet optimiranja. Ukazano je na specifičnosti sustava zasnovanih na komponentama i računarstva u oblaku nad kojima se zasniva prijedlog arhitekture u poglavlju 3. Naposljetku, iznosi se pregled dosadašnjih istraživanja sustavnim pregledom literature koja se dotiče optimizacije konfiguracija informacijskih sustava.

2.1 Arhitektura, konfiguracija i kvaliteta informacijskoga sustava

U ovom poglavlju definira se arhitektura informacijskoga sustava. Poseban naglasak dan je na komponentu kao gradivni element sustava. Iako su komponente u nekom od postojećih oblika dio svakog sustava, sustavi zasnovani na komponentama daju naglasak na unaprijed definirani oblik komponenti kao nositelja dekompozicije prilikom oblikovanja konačnog sustava. Oblikovanje sustava svodi se na definiranje strukture zasebnih dijelova, definiranja ovlasti svakog dijela i određivanje metode komunikacije u ostvarenju konačnog sustava.

2.1.1 Definicije

Arhitektura

Postoji niz definicija za pojam programske arhitekture, no većina daje naglasak na ulogu koju arhitektura nosi pri odlučivanju i zaključivanju o ključnim svojstvima sustava. Arhitekturom se određuju različite strukture — podjele sustava na manje dijelove koje određuju ključne karakteristike sustava.

Definicija 2.1 Arhitektura

Arhitektura programskog sustava je niz struktura potrebnih za zaključivanje o sustavu koje se sastoje od programskih elemenata, poveznica između njih te njihovih pripadnih svojstava [29].

Dekompozicija sustava na jasno razdijeljene komponente omogućuje da u izgradnji sustava sudjeluje veći broj ljudi što omogućava sustavni pristup većim projektima. Općenito, između različitih dijelova sustava postoji dogovorena interakcija u obliku sučelja (engl. *interface*) koja o svakom djelu sustava otkrivaju minimalno potrebnu količinu informacija [30]. Razlog tome je sakrivanje složenosti svakog dijela sustava, kroz enkapsulaciju, apstrakciju i modularnost. Tako je omogućen distribuirani rad na sustavu i postojanje specijaliziranih timova usko vezanih za svaki dio sustava. Upravo su enkapsulacija, apstrakcija i modularnost temelj dekompozicije složenih sustava na jednostavnije dijelove te omogućuje jednostavnije rukovođenje složenim sustavima.

Pojam *model arhitekture* označava prikaz određene arhitekture kao niza odluka prilikom njezina oblikovanja. Primjerice, ako je arhitektura dokumentirana kao struktura komponenti i njihovih veza, model arhitekture može predstavljati određeni komponenti model jezika UML. Često to nije dovoljno da se odredi cjelovita arhitektura, ali to zavisi o arhitektu sustava i procesima unutar organizacije koja razvija sustav.

Za model arhitekture kažemo da je formalan, ako njegova notacija slijedi definirani meta-model. Primjer formalne notacije za modeliranje arhitekture jest jezik AADL [31]. Pomoću meta-modeliranja ili proširenja jezika UML može se također ostvariti formalna notacija modela arhitekture.

Definicija 2.2 Model arhitekture

Model arhitekture je artefakt koji prikazuje dio ili sve odluke donesene tijekom oblikovanja arhitekture sustava. Modeliranje arhitekture predstavlja operacionalizaciju i dokumentaciju takvih odluka [32].

Komponenta

Programska komponenta je jedinica dekompozicije sustava koja: (1) enkapsulira podskup funkcionalnosti ili podatke u sustavu, (2) ograničava pristup tom podskupu putem jasno definiranog sučelja te (3) sadrži eksplicitno definirane zavisnosti u odnosu na okolinu u kojoj se izvodi [32].

Definicija 2.3 Programska komponenta (engl. *software component*)

Programska komponenta je jedinica kompozicije s definiranim sučeljem i kontekstualnim zavisnostima. Programska komponenta može se nezavisno postaviti u rad i biti subjekt kompozicije od treće strane [33].

Pomoću definiranih sučelja komponenti izražavaju se usluge koje komponenta pruža u obliku skupa operacija. To može biti dohvat, obrada ili pohrana podataka u skladu s definiranim poslovnim pravilima i domeni primjene komponente. Kako bi se komponenta mogla koristiti u različitim sustavima potrebno je da njezine zavisnosti budu jasno definirane. Postoje dva oblika zavisnosti koje komponenta može posjedovati. Prvi tip zavisnosti izražen je kao definicija zavisnih sučelja koja druge komponente moraju realizirati. Komponenta obavlja svoju funkcionalnosti pomoću funkcionalnosti tih zavisnih sučelja. Drugo, komponenta posjeduje zavisnosti prema izvedbenoj okolini: ona može biti dio platforme koja podržava njezin rad ili zahtijevati integraciju s određenim programskim ili sklopovskih resursima. Konačno, ako su ispunjeni svi zahtjevi koje komponenta nalaže, ona može biti nezavisno postavljena u sustav te isporučivati usluge koje definira. Budući da je komponenta cjelovita jedinica koja se postavlja u sustav, ona ne može biti djelomično postavljena, i nisu poznati unutrašnji detalji njezina rada, već se njezino korištenje odvija isključivo pomoću njezinoga sučelja [33].

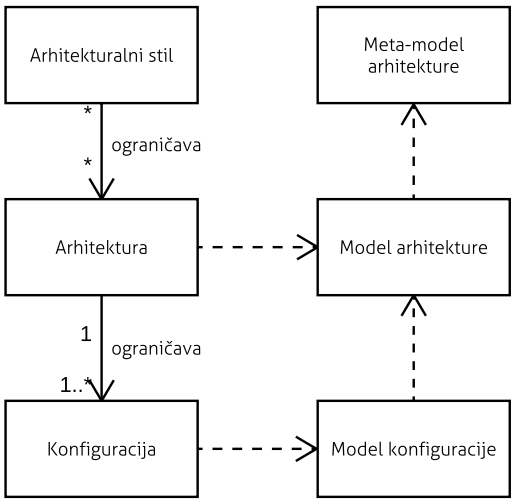
Arhitekturalni stil

Proučavanjem različitih problema za koje se pri njihovom rješavanju primjenjuju računalni algoritmi i sustavi, stvorio se niz obrazaca u obliku rješenja za pojedine ponavljajuće probleme. Poznata skupina takvih obrazaca postoji na razini objektno-orijentiranog programiranja [4, 34]. Na razini arhitekture sustava, gdje je razina apstrakcije još veća, također postoje uzorci koji rješavaju često prisutne probleme u obliku *arhitekturalnih stilova* (engl. *architectural style*).

Definicija 2.4 Arhitekturalni stil (engl. *architectural style*)

Arhitekturalni stil je specijalizacija elemenata i različitih vrsta njihovih međusobnih veza koja uključuje i skupinu ograničenja pri njihovoj uporabi [19].

Arhitekturalni stilovi mogu se podijeliti s obzirom na primjenu, poput stilova za oblikovanje raspodijeljenih sustava (npr. klijentsko-poslužiteljska arhitektura, višeredna arhitektura) ili općenitih stilova (npr. slojevita arhitektura). Arhitektura može biti ostvarena i pomoću više arhitekturalnih stilova.



Slika 2.1: Odnosi između termina *arhitektura*, *arhitekturalni stil* i *konfiguracija*

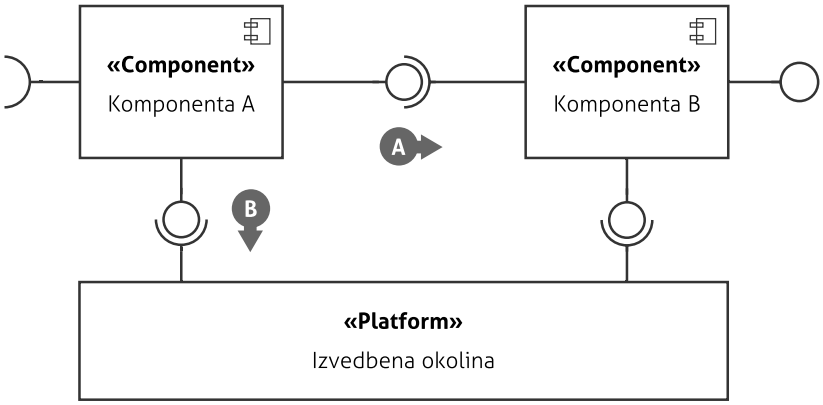
Konfiguracija

U konačnici, dolazimo do pojma konfiguracije pojedinog sustava. Općeniti pojam *konfiguracija* određen je normom ISO/IEC/IEEE 24765:2010 kao uređenje sustava definirano kroz količinu, svojstva i veze između njegovih sastavnica [35]. U kontekstu komponenata i njihove interakcije, konfiguracija označava odabir komponenti, uređenje između komponenti te njihove međusobne veze i veze s okolinom (primjerice njihova raspodjela po računalnim resursima).

Slika 2.1 prikazuje odnos između pojmova *arhitektura*, *arhitekturalni stil* i *konfiguracija*. Arhitektura sustava označava model po kojem pojedine komponente sustava međusobno ostvaruju ukupnu funkcionalnost. Samim time, arhitektura je skup ograničenja različitih mogućih konfiguracija sustava. Drugim riječima, konfiguracija sustava određena je arhitekturom, a pritom arhitektura može slijediti više arhitekturalnih stilova. U ovom se radu također koriste termini: *model arhitekture* kao formalni opis pojedine arhitekture koji slijedi određeni *meta-model*. *Model konfiguracije* je formalni opis koji proizlazi iz konkretizacije pojedine konfiguracije arhitekture.

Definicija 2.5 Konfiguracija (engl. *configuration*)

Konfiguracija sustava definira način na koji su komponente postavljene u sustavu i međusobno povezane [32]. Konfiguracija definira konkretan odabir svake komponente sustava, njezino pridruživanje s računalnim resursima te poveznice između zavisnih komponenti.



Slika 2.2: Dijelovi sustava zasnovanoga na komponentama

Izvor: Prilagođeno prema [36, 37]

2.1.2 Sustav zasnovan na komponentama

Sustav zasnovan na komponentama (engl. *component-based system*) jest sustav čija je arhitektura definirana strukturom komponenata i njihovom kompozicijom. Sustavi zasnovani na komponentama sačinjeni su od pojedinih komponenti koje za razliku od ostalih sustava podliježu unaprijed definiranim pravilima propisanim od strane komponentnog modela [36].

Definicija 2.6 Komponentni model

Komponentni model definira skup normi za implementaciju, imenovanje, prilagodbu, kompoziciju, evoluciju i postavljenje komponenti [36].

Komponentni model usmjeruje veći dio razvoja propisujući pravila za oblikovanje pojedinih komponenti te njihovo sastavljanje u konačan sustav. Trenutno postoji mnogo komponentnih modela za različite primjene, a mogu se klasificirati sukladno domeni primjene na komponentne modele općenite te komponentne modele specijalizirane primjene [37]. Potonji se još mogu podijeliti na komponentne modele za primjenu kod uklopljenih sustava te komponentne modele za izgradnju raspodijeljenih informacijskih sustava [37].

Slika 2.2 prikazuje glavne sastavnice sustava zasnovanoga na komponentama: (1) komponente, (2) platforma (engl. *underlying platform*) u obliku izvedbene okoline te (3) definicija vezivanja (engl. *binding*) između komponenti (A) te komponenti i platforme (B).

2.1.3 Proces postavljanja komponente u izvedbenu okolinu

Ovo poglavlje definira termine koji se dotiču postavljanja (engl. *deployment*) komponente ili sustava u izvedbenu okolinu koji se koriste u nastavku rada. Iako specifične tehnologije uvode vlastite procese, postoji opće prihvaćen definirani proces nastanka i postavljanja sus-

tava u rad [4, 38] prikazan slikom 2.3. U nastavku se definira svaka aktivnost zasebno. Iste aktivnosti vrijede za svaku komponentu zbog njezine nezavisnosti koja proizlazi iz same definicije komponente. Aktivnosti su definirane na razini komponente, iako su one iste i za cijeli sustav. Na slici su naznačeni mogući prijelazi između pojedinih aktivnosti opisanih u nastavku.

Izdanje (engl. *release*) komponente jest proces koji je poveznica između procesa razvoja (engl. *development*) i postavljanja komponente u rad. Izdavanje podrazumijeva prisutnost sve potrebne dokumentacije koja je potrebna kako bi komponenta ispravno obavljala svoju funkciju u odredišnoj okolini. Kod komponenti to posebice vrijedi za ranije spomenuta sučelja i zavisnosti prema platformi. Aktivnosti uključene u izdavanje su *pakiranje* i *oglašavanje*. Pakiranje komponente obuhvaća sastavljanje svih potrebnih dijelova u jedinstveni artefakt koji se prenosi u izvedbenu okolinu, a oglašavanje podrazumijeva provođenje aktivnosti koje su potrebne kako bi zainteresirane strane dobile informaciju i karakteristike komponente koja se izdaje.

Instalacija (engl. *installation*) pokriva aktivnosti **prijenosa** komponente na odredišnu okolinu te *podešavanje* (engl. *configuration*) komponente u skladu s potrebama primjene ili izvedbene okoline.

Aktivacija (engl. *activation*) uključuje pokretanje izvođenja komponente u izvedbenoj okolini. Aktivacija može uvjetovati aktivaciju ostalih sustava ili komponenti koji su definirani kao zavisnosti.

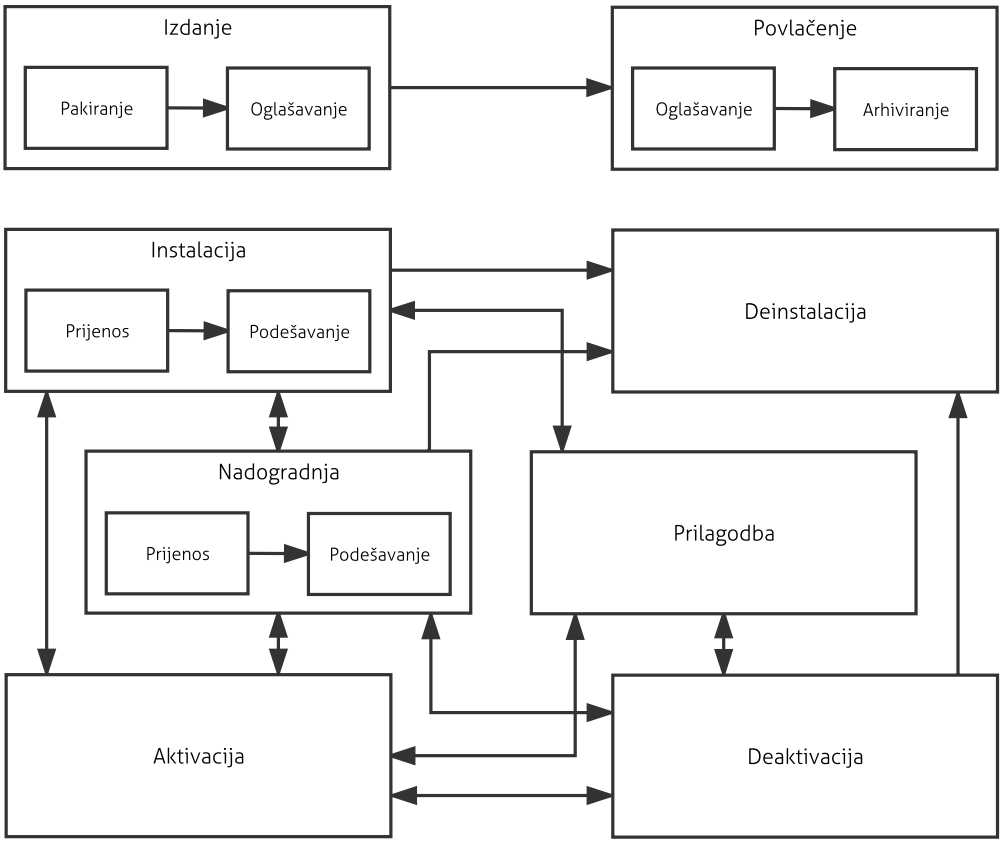
Deaktivacija (engl. *deactivation*) jest operacija suprotna aktivaciji te podrazumijeva zaustavljanje izvođenja komponente.

Nadogradnja (engl. *update*) jest specijalni slučaj instalacije gdje se podrazumijeva da su mnogi preduvjeti već ostvareni zbog činjenice da prethodna inačica komponente već postoji. Zavisno o konkretnoj implementacijskoj tehnologiji, nadogradnja komponente može uključivati aktivnosti prethodne deaktivacije ili deinstalacije te ponovne instalacije i aktivacije.

Prilagodba (engl. *adaptation*) jest slična operacija kao nadogradnja s razlikom da označava promjenu komponente koja je inicirana lokalnim izmjenama direktno na izvedbenoj okolini. Prilagodba je potrebna ako se promjena u izvedbenoj okolini odražava na nesmetani rad komponente.

Deinstalacija (engl. *deinstallation*) jest aktivnost suprotna instalaciji te podrazumijeva prethodnu deaktivaciju komponente. Deinstalacija označava uklanjanje komponente iz izvedbene okoline.

Arhiviranje (engl. *retire*) komponente uključuje njezin prestanak razvoja i pružanja podrške za njezin rad. Takvu je radnju potrebno objaviti svim korisnicima komponente kako bi mogli poduzeti potrebne korektivne mjere.



Slika 2.3: Aktivnosti vezane uz postavljanje komponente u rad

Izvor: Prilagođeno prema [38]

2.1.4 Model razvojnog procesa sustava zasnovanih na komponentama

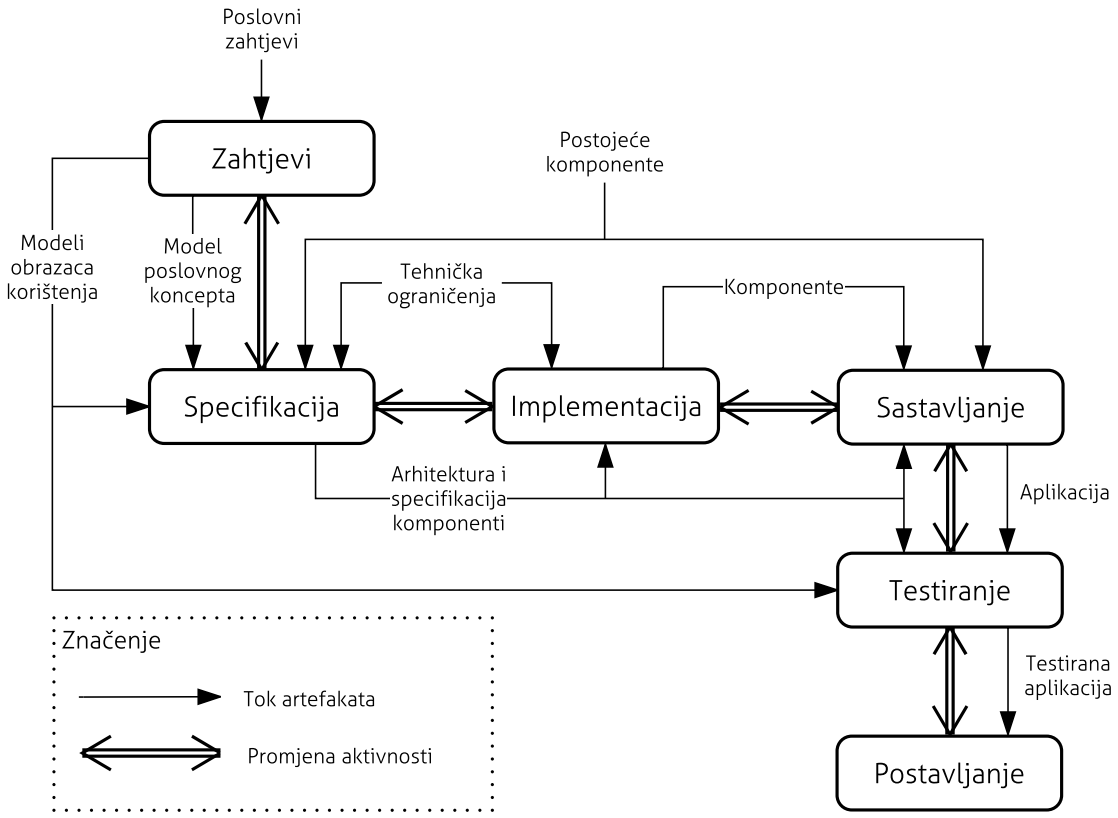
Sustavi zasnovani na komponentama veoma su specifični po pitanju razvojnog procesa zbog toga što se bilo koji od postojećih procesa može primijeniti na razini samo jedne komponente. Kako je svaka komponenta odijeljena od ostalih, može se promatrati kao zasebna jedinica razvoja. Ovakva odlika sustava zasnovanih na komponentama zahtijeva jedinstvenu kombinaciju pristupa koji kreću istovremeno *odozgo prema dolje* (engl. *top-down*) i pristupa od *odozdo prema gore* (engl. *bottom-up*) [33]. Postoji nekoliko procesa za razvoj sustava zasnovanih na komponentama poput pristupa autora Cheesemana i Danielsa [39], Atkinsona [40] ili Hasselbringa [41]. Svi navedeni procesi kombiniraju pristupe *odozdo prema gore* s pristupima *odozgo prema dolje*. Kao primjer modela procesa za razvoj sustava zasnovanog na komponentama iznosi se proces autora Cheesmana i Danielsa [39] koji je zasnovan na procesu Rational Unified Process (RUP) [42]. Taj isti proces se u nastavku rada proširuje korakom koji uključuje predstavljenu metodu optimizacije. Sam postupak optimizacije je općenit i može se prilagoditi i ostalim procesima razvoja.

Slika 2.4 prikazuje spomenuti proces razvoja. Predstavljeni proces ne uključuje aktivnosti vezane uz proces rukovođenja razvojem poput procjene rizika, planiranja, organizacije timova, ili praćenja, zbog njegove neovisnosti od procesa rukovođenja. Želja je autora Chessmana i Danielsa da predstavljeni proces bude moguće provesti s više različitih metodologija upravljanja. Drugi razlog je i njegovo lakše shvaćanje time što se predstavlja izolirano od čimbenika rukovođenja.

Proces se može podijeliti u pet setova aktivnosti (engl. *workflow*), gdje svaki set aktivnosti rezultira određenim *artefaktima* koji prenose informacije između aktivnosti. Artefakte predstavljaju različiti modeli nastali korištenjem jezika UML, gotove implementirane komponente ili cjelovita aplikacija. Proces je iterativne prirode te se provodi tako da arhitekt sustava usmjeruje svoj angažman sukladno prijelazima između različitih aktivnosti naznačenih dvostrukim strelicama na slici 2.4.

Proces uključuje sljedeće aktivnosti:

- **Utvrđivanje zahtjeva** - prvi korak koji uključuje prikupljanje zahtjeva od strane klijenata i izradu modela obrazaca korištenja (engl. *use-case model*) koji prikazuju željenu interakciju korisnika sa sustavom u obliku skupa funkcionalnosti koje sustav pruža. Uz funkcionalne zahtjeve, mogu se odrediti i nefunkcionalni zahtjevi u vidu količine korisnika, ili određenih vremena odaziva koje sustav mora ostvariti pri pojedinim zahtjevima korisnika. Važan dio prikupljanja zahtjeva uključuje i model poslovnog koncepta koji opisuje ključne informacije koje sustav mora obraditi zavisno o okolini i konteksta uporabe.
- **Specifikacija** - u ovoj fazi oblikuje se arhitektura sustava pomoću modela poslovnog koncepta i modela obrazaca korištenja. Određuju se sučelja pojedinih komponenti, gdje



Slika 2.4: Model procesa razvoja sustava zasnovanih na komponentama

se u obzir se uzimaju i postojeće komponente, koje su također važan čimbenik. Gledaju se i tehnička ograničenja postojećih komponenti ili korištenih alata kao i ograničenja proizašla iz kasnijih faza razvoja. Artefakti proizašli iz ove fazu uključuju specifikaciju komponenti koje treba razviti ili iskoristiti, njihovim sučeljima te arhitekturom koja prikazuje na koji način komponente ostvaruju komunikaciju.

- **Implementacija** - temeljem informacija proizašlih iz prethodne faze, u ovoj se fazi razvijaju komponente koje nisu unaprijed dostupne. Ako je to moguće, određene komponente se mogu i dobaviti te po potrebi modificirati. Ova faza također uključuje i testiranje svake komponente zasebno (engl. *unit testing*).
- **Sastavljanje** - ova faza podrazumijeva sastavljanje svih komponenti u cjeloviti sustav sa svim ostalim mogućim dijelovima konačne aplikacije poput artefakata koji definiraju način na koji se provodi komunikacija između komponenti te njihove postavke, kao i postavke cijelog sustava.
- **Testiranje** - provodi se testiranje cjelokupne aplikacije u testnoj okolini provjeravajući ponašanje sustava prema modelima obrazaca korištenja.
- **Postavljanje** - u ovom završnom koraku, sustav se postavlja u konačnu izvedbenu okolinu prema procesu opisanom u poglavlju 2.1.3.

Aktivnosti utvrđivanja zahtjeva, testiranja i postavljanja preslikane su iz procesa RUP,

dok su aktivnosti analize (engl. *analysis*) i oblikovanja (engl. *design*) iz RUP-a zamijenjene aktivnostima specifičnim za sustave zasnovane na komponentama, a to su izrada specifikacije, implementiranje i sastavljanje komponenti.

2.1.5 Atributi kvalitete

Arhitektura sustava je ključna za ostvarivanje kvalitete [43] što čini kvalitetu važnim čimbenikom prilikom oblikovanja arhitekture. Atributi kvalitete pokazuju koliko dobro su ostvarene želje korisnika sustava. Funkcionalni zahtjevi sustava iskazuju namjenu i očekivano ponašanje sustava, dok ih zahtjevi u obliku atributa kvalitete nadopunjuju metrikama poput brzine odaziva određenih operacija ili razinu otpornosti na pogreške.

Definicija 2.7 Atribut kvalitete (engl. *quality attribute*)

Atribut kvalitete je svojstvo sustava koje je moguće mjeriti ili testirati, a ukazuje na razinu zadovoljavanja potreba dionika (engl. *stakeholders*) - kupaca, naručitelja ili korisnika [43].

Arhitektura sustava određuje funkcionalne zahtjeve i kriterije kvalitete kao konkretizaciju pojedinih atributa. Funkcionalne zahtjeve definira u vidu dodjele potrebnih odgovornosti tijekom dizajna, a atribute kvalitete određuje različitim strukturama kao dijelovima arhitekture te karakteristikama elemenata koji sačinjavaju tu strukturu.

S druge strane, funkcionalnosti sustava ne određuju arhitekturu. Kada bi tome bilo tako, sustav ne bi bilo potrebno strukturirati na manje dijelove. Strukturiranje se obavlja zbog zadovoljavanja atributa kvalitete. Primjerice, tijekom implementacije sustav se ostvaruje kroz više modula kako bi timovi mogli paralelno raditi na izgradnji sustava ili zbog ponovne iskoristivosti dijela sustava. Tijekom izvođenja, sustav se strukturira u više dijelova kako bi se primjerice ostvarile radne karakteristike u vidu brzine odaziva ili zbog redundancije koja utječe na pouzdanost.

Postoje dvije grupe atributa kvalitete. Prva skupina odnosi se na svojstva sustava u tijeku izvođenja poput raspoloživosti (engl. *availability*), radnih značajki (engl. *performance*) ili upotrebljivosti (engl. *usability*), dok se druga skupina dotiče implementacijskih osobina poput mogućnosti izmjene (engl. *modifiability*) ili testiranja (engl. *testability*). Organizacija ISO izdala je model kvalitete programskog proizvoda kao normu ISO 25010:2011 [44].

Atributi kvalitete programske podrške koje je važno sagledati prilikom oblikovanja arhitekture jesu pouzdanost (eng. *reliability*), mogućnost izmjene, radne značajke, sigurnost (engl. *security*), mogućnost testiranja i upotrebljivost [43].

Radne značajke vezane su uz vremensku dimenziju izvođenja sustava te učinkovitost raspolaganja pripadnim resursima. Najvažnije metrike radnih značajki su vrijeme odaziva različitih usluga koje sustav ostvaruje, iskoristivost (engl. *utilization*) resursa i propusnost

(engl. *throughput*) [45].

Pouzdanost je sposobnost sustava da ostvaruje predviđene funkcionalnosti ispravno tijekom određenog vremena u zadanoj okolini. Često se iskazuje kao vjerojatnost pogreške. Za razliku od pouzdanosti, raspoloživost sustava određuje na udio u vremenu u kojem je sustav dostupan da odgovori na zahtjeve [43]. Često se iskazuje kao postotak u određenom intervalu. Primjerice, za sustav koji ostvaruje 99,9-postotnu raspoloživost unutar 30 dana podrazumijeva se da ukupna nedostupnost sustava u tom periodu bude manja od 4,32 minute.

Mogućnost promjene izražava trošak promjene programskog sustava, gdje se promjena može manifestirati u obliku dodavanja nove funkcionalnosti ili izmjene postojećih funkcionalnosti.

Zaštićenost označava mogućnost sustava da spriječi neovlaštenu uporabu sustava.

Mogućnost testiranja opisuje u kojoj mjeri i s kojom lakoćom je sustav moguće testirati na pojavu greški [43]. Ovaj atribut kvalitete pospješuje kvalitetnu i sustavnu provedbu verifikacije [46].

Upotrebljivost opisuje razinu jednostavnosti interakcije korisnika sa sustavom kod obavljanja različitih zadaća [43].

Zadovoljavanje pojedinog kriterija kvalitete ne može obaviti bez utjecaja na ostale atribute. Česta je pojava da ostvarenje pojedinih kriterija kvalitete može biti u međusobnoj sprezi. Primjerice, uvođenje dodatne zaštićenosti utječe negativno na radne značajke, kao što centralizacija zbog više razine sigurnosti može uvesti jedinstvenu točku nepouzdanosti (engl. *single point of failure*). Općenito vrijedi da pospješivanje gotovo svakog atributa kvalitete negativno utječe na radne značajke [43].

Postoje i *poslovni atributi kvalitete* [43] koji uključuju ekonomske pokazatelje kod razvoja programske podrške. To mogu biti različiti troškovi i potencijali pomoću kojih se izračunava povrat investicija, ili vrijeme stavljanja proizvoda na tržište (engl. *time-to-market*). Troškovi se dodatno mogu podijeliti na troškove razvoja, održavanja, nabavke sklopovlja, izvođenja te na troškove licenciranja potrebnih dijelova programske podrške. Javni oblak podrazumijeva veći trošak izvođenja zbog potrebnog najma infrastrukture, dok privatni oblak podrazumijeva veći trošak nabavke sklopovlja i utrošene energije.

Mjere atributa kvalitete

Sa stanovišta mjerenja različitih atributa kvalitete, oni se dijele na one koje je moguće izraziti kvantitativno i one koji su kvalitativne prirode. U nastavku su definirane mjere pomoću kojih je moguće izraziti atribute kvalitete.

Svakom atributu kvalitete a iz skupa atributa $a \in A$ moguće je dodijeliti element V_a skupa V koji označava skup svih vrijednosti koje određeni atribut može poprimiti. Te vrijednosti mogu biti kvalitativne ili kvantitativne, a takvo preslikavanje nazivamo *metrikom kvalitete*

(M_a):

$$M_a : A \rightarrow V_a$$

.

Definicija 2.8 Metrika kvalitete M_a

Metrika kvalitete M_a je preslikavanje atributa kvalitete a iz skupa atributa A u vrijednost iz skupa V_a.

Drugim riječima, metrika kvalitete M_a pridjeljuje element skupa V_a koji označava kvalitetu nekog sustava S. Ista metrika M_a može biti izražena kroz više slučajeva za isti sustav S. Primjerice, u slučaju odabire metrike vremena odaziva, ono može biti pridruženo različitim operacijama sustava S zavisno o scenariju (na primjer drukčije opterećenje sustava). Svako korištenje metrike M_a smatra se odvojenim iskazom kvalitete sustava. U tu svrhu, važno je definirati sustavni *scenarij* koji pobliže definira sve okolnosti pod kojima se određena metrika M_a mjeri. Pojedini iskaz određene metrike naziva se *kriterijem kvalitete. Kriterij kvalitete* definira mjerenje neke metrike M_a na točno određenom mjestu sustava S pod scenarijom s. Kriterij kvalitete možemo dodijeliti svim mogućim konfiguracijama sustava (K) jer svaka konfiguracija sustava može ostvariti različitu kvalitetu. Kriterij kvalitete q je dakle preslikavanje:

$$q : K \xrightarrow{s} V_a$$

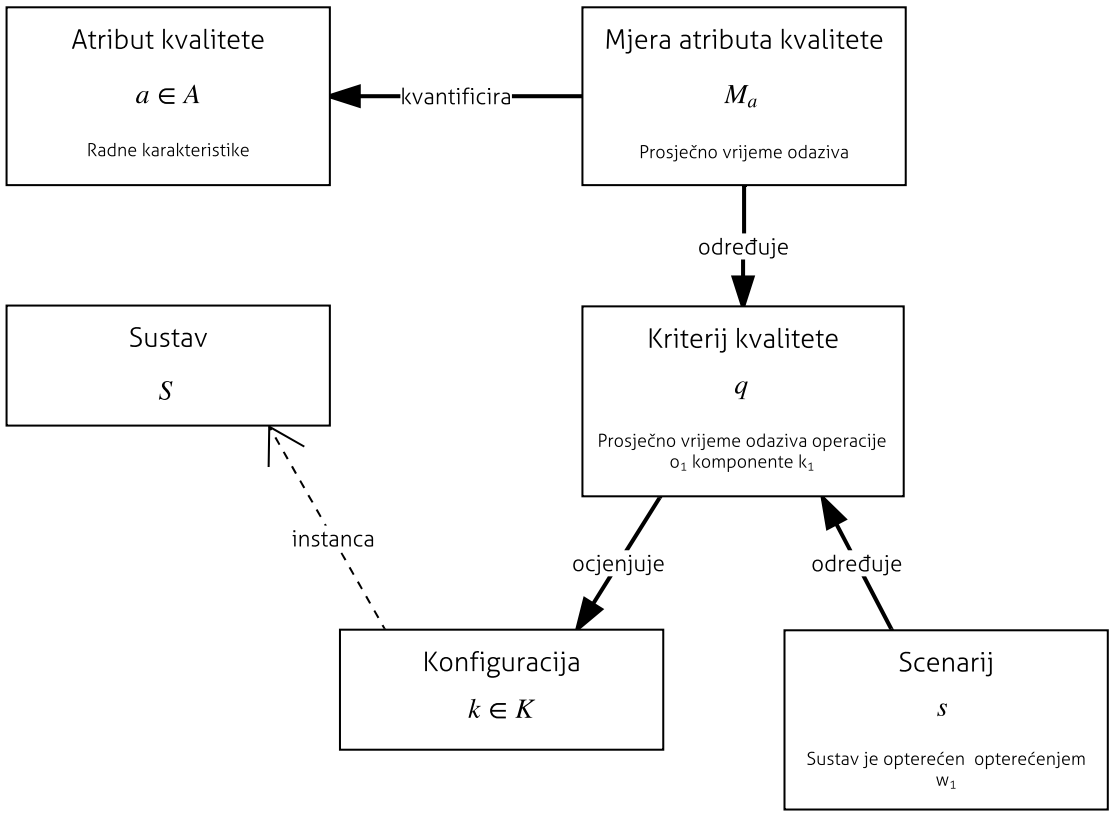
.

Odnosi između iznesenih pojmova atributa, metrike i kriterija predstavljeni su slikom 2.5.

Vrijeme odaziva

Vrijeme odaziva pripada među atribute kvalitete koji se dotiču radnih značajki i jedan je od važnih čimbenika kvalitete sustava, osobito kod sustava koju su u neposrednoj interakciji s korisnicima. Vrijeme odaziva kod sustava može se mjeriti nad različitim operacijama: internim, poput vremena pristupa memoriji ili diskovlju, ili vanjskim, poput vremena izvršavanja pojedine operacije. Za sustav zasnovan na komponentama često se vrijeme odaziva mjeri na razini pojedinih operacija koje komponente vrše.

Na vrijeme odaziva utječu brojni parametri, no postoje određene zakonitosti pri određivanju očekivanog reda veličine pojedinih često korištenih operacija određenih samom konfiguracijom računala. Poznato je da je vrijeme pristupa glavnoj memoriji nekoliko redova veličine brže od pristupa diskovlju, pogotovo ako se radi o rotacijskim tvrdim diskovima. Okvirni redovi veličina latencija koje se mogu očekivati prilikom implementacije informacijskoga



Slika 2.5: Odnosi između termina koji definiraju kvalitetu programske podrške

sustava prikazani su tablicom 2.1 te su veoma važan čimbenik u projektiranju sustava.

Vrijeme odaziva možemo podijeliti na sljedeće čimbenike:

1. **Latencija** predstavlja vrijeme koje je potrebno da informacija stigne od odredišta do cilja gdje se vrši daljnja obrada. Latencija se često iskazuje kod informacijskih sustava koji su fizički udaljeni od korisnika ili čije su komponente raspodijeljene, a međusobna komunikacija je ostvarena računalnom mrežom, primjerice Internetom. Postoje mnogi faktori koji utječu na latenciju mreže, a njezinu donju granicu postavljaju fizička ograničenja poput brzine svjetlosti. To bi značilo da latencija ne može biti manja od 3.33 µs po svakom kilometru udaljenosti. Nesavršenosti komunikacijskog kanala, šumovi, procesne jedinice, usmjerivači, pojačivači signala i ostalo mrežno sklopovlje dodatno povećavaju latenciju komunikacije. Latencija sustava ovisi i o količini informacije koju je potrebno prenijeti, što je pak uvelike određeno propusnošću i pouzdanošću komunikacijskog kanala [47].
2. **Vrijeme čekanja na obradu** predstavlja vrijeme čekanja na dijeljene resurse poput procesora ili diskovlja. Veća čekanja često su uzrokovana zagušenjima: trenucima kada količina posla koja određeni resurs mora obaviti prelazi njegov ukupni kapacitet obrade u jedinici vremena. Tada novi poslovi koji dolaze u sustav moraju čekati na obradu. Takvo vrijeme čekanja često se analitički opisuje teorijom redova čekanja (engl. *Queue*

Theory) [48] te je moguće provesti adekvatno kapacitiranje pojedinih resursa ili čitavog sustava putem modela redova čekanja. Redovi čekanja najčešći su oblik modeliranja radnih karakteristika sustava gdje se pomoću jednostavnijih modela radi predviđanje ponašanja ili kapacitiranje na razini sustava.

3. **Vrijeme obrade u sklopu komponente** predstavlja vrijeme koje je potrebno da se informacija obradi u željenom elementu obrade poput programske komponente ili sklopovske komponente poput tvrdog diska ili središnjeg procesora zavisno o razini apstrakcije promatranja.

Ukupno vrijeme čekanja (engl. *end-to-end latency*) može biti sastavljeno od niza spomenutih čimbenika na različitim komponentama sustava. Komponente su međusobno povezane i vrijeme obrade pojedine komponente ovisi o vremenu obrade zavisnih komponenti te naposljetku samog mrežnog i ostalog računalnog sklopovlja.

Tablica 2.1: Primjer različitih latencija kod različitih aspekata informacijskih sustava

Događaj	**Trajanje**
Ciklus procesora (takta 2GHz)	$\sim 0.5\ ns$
Pristup memoriji računala (DRAM) s procesora	$\sim 100\ ns$
Pristup povratnom sučelju za protokol TCP na istom računalu	$\sim 10\ \mu s$
Nasumično čitanje 4KB podataka sa SSD diska	$\sim 150\ \mu s$
Latencija mreže u podatkovnom centru (između dva računala)	$\sim 500\ \mu s$
Latencija LAN mreže	$\sim 1\ ms$
Latencija WAN mreže (San Francisco - New York)	$\sim 40\ ms$
Latencija WAN mreže (San Francisco - Australia)	$\sim 180\ ms$
Dodavanje i pokretanje novog virtualnog stroja	$\sim 2\ m$

Izvor: Prilagođeno prema [49]

2.2 Sporazum o razini usluge

Sporazum o razini usluge služi kao ugovorni sporazum između davatelja usluge i klijenata [50, 51]. SLA kao garancija kvalitete usluge (engl. *Quality of Service*) predstavlja most između poslovnih i tehničkih aspekata. Pružanje strožeg ugovora znači veću kompetitivnu prednost davatelja usluge, kao i veću razinu povjerenja kod klijenta. Prodornost mrežnih usluga (engl. *web service*) i uslužno orijentiranih arhitektura (engl. *service-oriented architecture*) među organizacije uvjetovala je pojavu ovakvih sporazuma. Razlog tome jest značajna količina usluga

koje organizacija koristi, a nisu pod njezinim direktnim utjecajem čime se rizik poslovanja prenosi na takve usluge. Današnji sustavi teže labavoj sprezi između komponenti (engl. *loose coupling*) što ostvaruju koristeći mrežne usluge u obliku uslužno orijentirane arhitekture. Pri takvim arhitekturama od sve veće važnosti je da svaka komponenta koja nije pod direktnim utjecajem vlasnika sustava pruža zadovoljavajuće radne karakteristike uz visoku razinu pouzdanosti zbog utjecaja na ostatak sustava.

Svrha sporazuma jest formalna definicija parametara usluge, najčešće pojedinih kriterija kvalitete, koje davatelj usluge mora ostvariti. Jednom definirani parametri, kontinuirano se prate te se zabilježava svaka devijacija od ugovorenih vrijednosti. Zavisno o dogovoru, često se uspostavljaju određeni penali ako usluga u određenom vremenskom intervalu nije poštovala vrijednosti parametara zadane sporazumom. Rekapitulacija svih penala izvršava se na kraju platnog razdoblja. Važno je da vremenski interval praćenja bude što manji kako bi se zabilježilo što više incidenata. Davatelju usluge također je važno uspostaviti sustavno praćenje svih parametara sustava te unaprijed djelovati na svaki mogući problem koji bi mogao uzrokovati nepoštovanje sporazuma.

2.2.1 Metrike sporazuma o razini usluge

Metrike koje su definirane sporazumima najčešće se dotiču jednostavne provjere dostupnosti usluge (engl. *availability*). Dostupnost usluge osnovni je zahtjev klijenata koji se ostvaruje periodičnom provjerom dostupnosti usluge unaprijed zadanim procedurama. Primjerice, mrežne usluge često pružaju posebnu metodu za provjeru koja odgovara pozitivno ukoliko je usluga dostupna.

S druge strane, metrike koje definiraju radne karakteristike usluge, poput vremena odaziva ili propusnosti usluge teže su za definirati i ostvariti jer se zasnivaju na tome da to praćenje ostvaruje klijent ili davatelj usluge, što dovodi u pitanje vjerodostojnost mjerenja. Mjerenje se tada prepušta trećoj nezavisnoj strani koju davatelj i korisnik usluge zajednički dogovore. Zbog toga je važno da davatelj usluga osigura mogućnost mjerenja parametara usluge koje pruža tako da se definirani parametri mogu provjeriti od treće strane.

Nadalje, važno je da parametri koji se definiraju budu relevantni korisniku usluge te da su povezani s njegovom percepcijom kvalitete usluge. Veza između korisničke percepcije usluge i mjerljive kvalitete usluge tema je trenutnih istraživanja [52, 53]. Kao česti parametar u sporazumima definira se vrijeme odaziva pojedinih operacija koje ostvaruje davatelj usluga zbog njegovog direktnog utjecaja na korisničko iskustvo [54]. U slučaju da se pod uslugom podrazumijeva mrežna usluga, prati se vrijeme odaziva pojedinih operacija koje usluga pruža.

2.3 Računarstvo u oblaku

Jedan od velikih umova računarske znanosti John McCarthy je još davne 1961. javno iznio ideju o uslužnom računarstvu, tehnologiji koja će omogućiti računalnu infrastrukturu i gotove aplikacije koje će se naplaćivati po potrošnji poput struje ili vode. Ta je ideja dugo sazrijevala kroz prve poslužitelje — načina računanja u dijeljenim vremenskim intervalima (engl. *time sharing*) sve do velikih umreženih infrastruktura u podatkovnim centrima. Ono što razlikuje računarstvo u oblaku od ostalih inačica raspodijeljenog računarstva jest upravo ekonomski model kakvog je zamislio McCarthy. Pojam računarstvo u oblaku (engl. *cloud computing*) prvi je definirao Ramnath Chellappa 1997. godine [55] kao oblik računarstva čije granice određuju ponajviše ekonomski čimbenici. Drugim riječima, računarstvo u oblaku za potrošača pruža prividno neiscrpan skup računalne infrastrukture koja se naplaćuje po uporabi. Podršku tome daju istraživanja na području virtualizacije sklopovlja (engl. *virtualization*), pametnog upravljanja podatkovnim centrima i vremenskog dijeljenja resursa [9, 56].

2.3.1 Definicije

Računarstvo u oblaku uključuje pružanje gotovih aplikacija korištenih preko Interneta što nazivamo *programska podrška kao usluga* (engl. *Software as a Service*, skraćeno SaaS), kao i pripadajuće *infrastrukture kao usluge* (engl. *Infrastructure as a Service*, skraćeno IaaS). Treći oblik računarstva u oblaku jest *programska platforma kao usluga* (engl. *Platform as a Service*, skraćeno PaaS) kao oblik usluge koji sporije gradi svoj tržišni udio zbog onemogućene prenosivosti aplikacija između platformi.

Područje interesa ovoga istraživanja je korištenje infrastrukture kao usluge koja omogućuje najam virtualiziranog sklopovlja u obliku *virtualnih strojeva* (engl. *Virtual Machine*, skraćeno VM). VM-ovi omogućuju izolaciju korisničkih aplikacija od sklopovske infrastrukture pomoću koje se izvode, kao i izolaciju od ostalih VM-ova s kojima dijeli tu istu infrastrukturu. Takav model dijeljenja otvorio je istraživanja usmjerena upravljanju smještaja VM-ova na fizičke resurse [57, 58, 59].

Pojam *oblak* podrazumijeva sklopovlje i programsku podršku lociranu u podatkovnim centrima. Ako je *oblak* dostupan javnosti kao usluga nazivamo ga *javni oblak* (engl. *public cloud*), dok samu uslugu nazivamo *uslužno računarstvo* (engl. *utility computing*) [9]. S druge strane, ako se govori o podatkovnom centru koji je u vlasništvu korisnika tada to definiramo kao *privatni oblak* (engl. *private cloud*). Prisutna su i mješovita rješenja u obliku *hibridnih oblaka* (engl. *hybrid cloud*).

Razlika između oblaka i prethodnih paradigmi poput računarstva zasnovanog na spletu računala (engl. *grid computing*) ili grozdovu računala (engl. *cluster*) jest u tome što se kod oblaka uvodi tehnologija virtualizacije i naplate razmjerne uporabi. Novost su i automatizirani

pregovori između korisnika i davatelja usluge te postojanje SLA-ova. Oblak dakle predstavljaju podatkovni centri čije je sklopovlje virtualizirano, dinamički pripremljeno za korisnika na zahtjev (engl. *provisioned*) kao skup računalne infrastrukture koja zadovoljava ugovoreni SLA gdje je cijeli postupak moguće automatizirati putem mrežnih usluga.

2.3.2 Modeli naplate

Postupak određivanja cijene za usluge oblaka jedan je od kritičnih čimbenika koji davatelj usluge mora odrediti. Cijena usluge regulira ponudu i potražnju računalnih usluga te tako utječe i na ponuđača i na potrošača [60]. Postoji više oblika naplate kojima se koristi davatelj usluge. U modelu pretplate, korisnici u određenim intervalima plaćaju unaprijed zakupljene usluge. Najčešće su ti intervali definirani na mjesečnoj ili godišnjoj razini [61]. S druge strane model naplate *pay-per-use* podrazumijeva naknadnu naplatu samo korištenih resursa, gdje najčešći minimalni interval korištenja pojedinog resursa iznosi jedan sat [62]. Model naplate razmjeran korištenju najčešće je prisutan u IaaS-u i PaaS-u, dok je model pretplate karakterističan za IaaS i SaaS [62]. Postoje i ostali napredniji oblici naplate poput aukcija [63, 64]. Aukcija se koristi kako bi se povećala iskoristivost infrastrukture davatelja usluge jer se cijene usluga dinamički prilagođavaju trenutnom broju korisnika. Primjerice, u razdoblju niže potražnje resursi su jeftiniji. *Amazon Spot* * je jedan od primjera takve usluge.

Modeli naplate često su razlikuju zavisno o vrsti resursa. Ako je ponuda ostvarena kroz zakup VM-ova, kod većine javnih IaaS oblaka, poput Amazona ili DigitalOceana, nude se tipizirani VM-ovi naplaćeni po satu korištenja. Svaki tip VM-a ima pripadnu cijenu koja je razmjerna karakteristikama pripadnih resursa izraženim u količini procesora, memorijskog prostora i prostora na diskovlju ili snazi procesora [65]. U slučaju mrežnih usluga ili usluga pohrane podataka, koriste se i druge metrike korištenja, poput ukupno ostvarene količine podataka.

2.4 Radno opterećenje i kapacitet informacijskoga sustava

Ovo poglavlje definira pojmove vezane uz opterećenje sustava te mogućnost utjecanja na ukupni kapacitet obrade sustava. Pojašnjen je pojam skalabilnosti sustava, kao nužnoga preduvjeta elastičnosti koju je moguće ostvariti koristeći infrastrukturu oblaka. Poglavlje 2.4.1 iznosi potrebne termine vezane uz opterećenje i kapacitet sustava, a poglavlje 2.4.2 pojašnjava pojam skalabilnost sustava. Pojam elastičnosti definiran je u poglavlju 2.4.3.

*Usluga *Amazon Spot* dostupna je na *https://aws.amazon.com/ec2/spot/*

2.4.1 Radno opterećenje

U nastavku se iznose termini vezani za opterećenje sustava definirani prethodnim istraživanjima [13, 45, 66, 67] prikazani na slici 2.6. Kao subjekt opterećenja uvodimo pojam *programske usluge* (engl. *software service*) kao jednu ili više programskih komponenti postavljenih u izvedbenu okolinu s ciljem obavljanja određenih operacija za korisnika ili druge programske usluge [68]. Pojam programske usluge koristi se kao poopćenje pojma *mrežna usluga* (engl. *web service*) s obzirom da komponente ne komuniciraju samo putem mrežnih protokola. Termin *aplikacija* označava skupinu srodnih programskih usluga ostvarenih ili korištenih u krajnjem korisničkom programskom rješenju. Konačno, već uvedeni pojam *informacijskoga sustava* obuhvaća jednu ili više aplikacija koje određena organizacija upotrebljava.

Zahtjev (engl. *request*) označuje jedno nedjeljivo korištenje programske usluge od strane korisnika ili druge programske usluge [69]. Primjerice, to može biti jedan poziv prema željenoj operaciji mrežne usluge. Ako se radi o komponenti čije se sučelje zasniva na protočnoj obradi podataka tada poziv može označiti paket ili događaj koji je pristigao na sučelje.

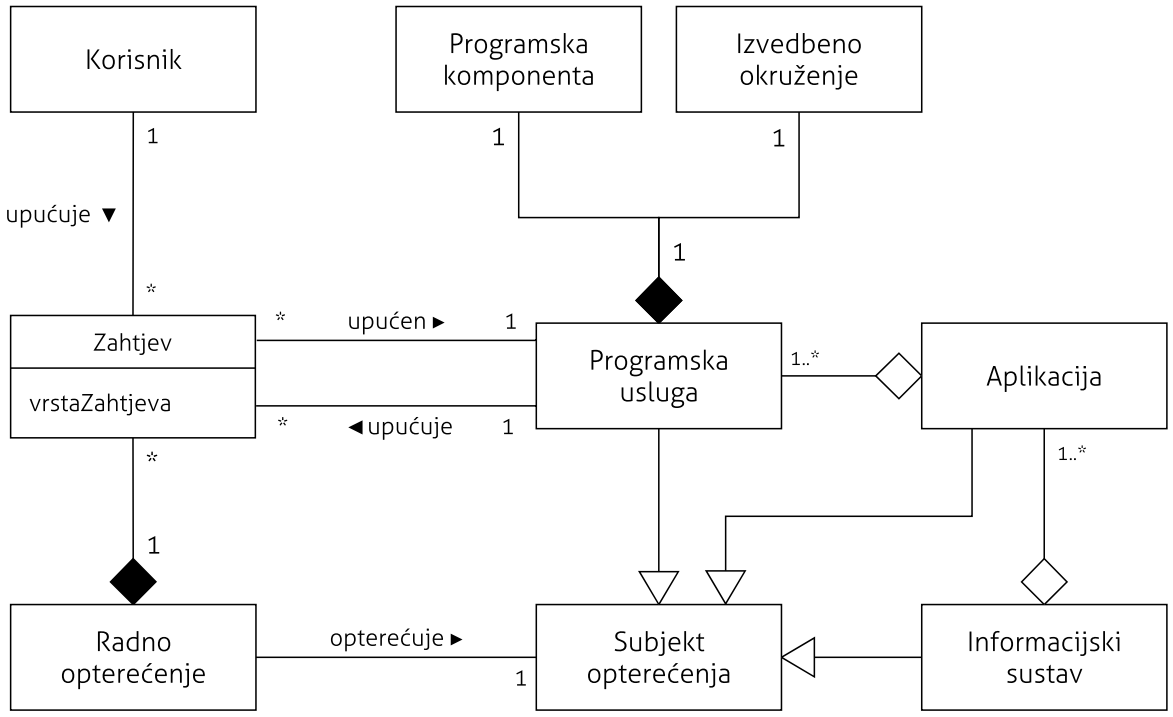
Vrsta zahtjeva (engl. *request class*) podrazumijeva kategorizaciju svih zahtjeva kod kojih ne postoji statistički značajna razlika u potrebi za resursima [69]. Primjerice, jedna klasa zahtjeva mogu biti svi zahtjevi prema određenoj operaciji programske usluge poput zahtjeva za izvršavanjem određene transakcije.

Radno opterećenje (engl. *workload*) predstavlja fizičko korištenje infrastrukture tijekom vremena sačinjeno od niza različitih zahtjeva. Radno opterećenje može sačinjavati različite ponavljajuće uzorke u vremenu pomoću kojih se može prognozirati buduće opterećenje ili obavljati kapacitiranje sustava [13].

Kapacitet (engl. *capacity*) programske usluge predstavlja najveći intenzitet radnog opterećenja u jedinici vremena pod kojim usluga može zadržati ciljane radne karakteristike. Kapacitet primjerice može biti određen gornjom granicom dopuštenoga vremena odaziva ili maksimalnom propusnošću usluge [70].

2.4.2 Skalabilnost sustava

Skalabilnost (engl. *scalability*) je važan atribut kvalitete informacijskoga sustava koji postaje od sve veće važnosti u suvremenim sustavima. Jedinstvena definicija pojma skalabilnosti nije jednostavna zbog široke primjene tog pojma [71]. Skalabilnost sustava nužno je osigurati već u ranijim fazama evolucije programske podrške zbog toga što resursi računalne infrastrukture posjeduju ograničen kapacitet. Kapaciteti resursa razlikuju se zavisno o vrsti resursa. Na primjeru centralnih procesnih jedinica kapacitet je ograničen maksimalnom količinom instrukcija u jedinici vremena; kod komunikacijskih resursa kapacitet predstavlja maksimalna propusnost, dok resurse za pohranu podataka ograničava prostor pohrane. U svakom od na-



Slika 2.6: Odnosi između korištenih termina u definiranju radnog opterećenja

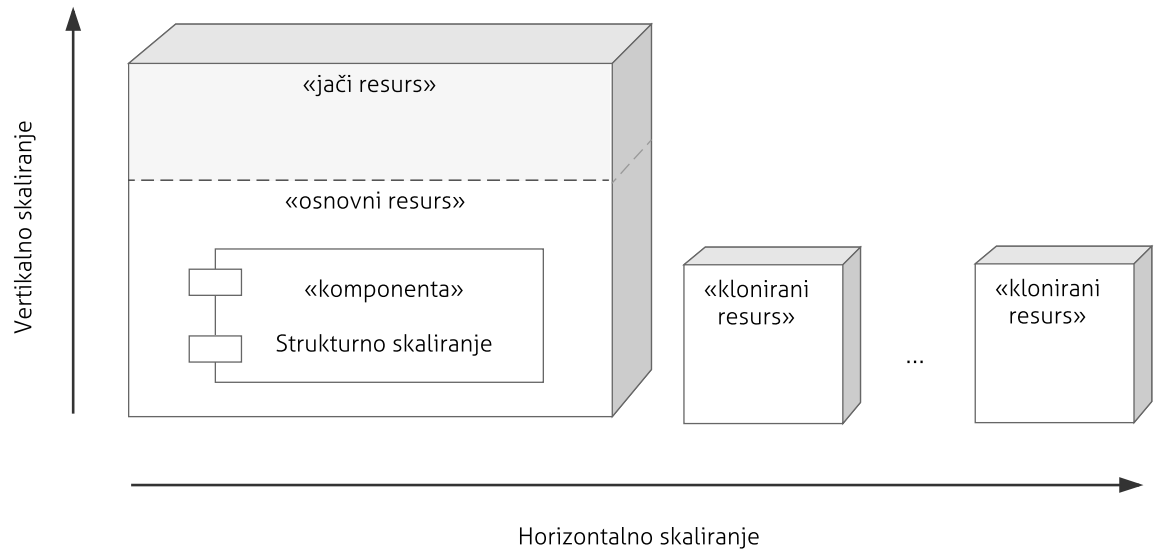
vedenih slučaja, skalabilnost sustava označuje mogućnost sustava da dodavanjem dodatnih resursa poveća kapacitet i pritom zadrži radne karakteristike [71].

Savršenu skalabilnost često je nemoguće ostvariti zbog činjenice da je mnogo algoritama po svojoj prirodi teško razdijeliti u istovremeni rad kroz više resursa bez dodatne sinkronizacije i prijenosa podataka, što značajno narušava početnu efikasnost. Ispravna definicija skalabilnosti još uvijek ne postoji već se problem sagledava s praktičnog aspekta tako da se promatra ekonomska isplativost prilikom rasta sustava [72].

Postoje određeni postupci pomoću kojih se skalabilnost može mjeriti [73], a u slučajevima gdje je potrebna dubinska analiza veće količine podataka koriste se specijalizirani sustavi [74]. Problem narušene skalabilnosti najčešće se javlja prilikom oblikovanja informacijskoga sustava zbog korištenja određenih krivih obrazaca (engl. *scalability anti-patterns*) [75].

Općenito, poželjne karakteristike skalabilnog sustava mogu se podijeliti u tri skupine [76]:

- **skalabilnosti opterećenja** podrazumijeva sustav koji na različitim razinama radnog opterećenja ne koristi više resursa nego što je to potrebno (izbjegava nepotrebna kašnjenja ili zagušenja prema određenim resursima),
- **skalabilnost prostora** podrazumijeva uporabu resursa ograničenih prostorom (radna memorija, širina komunikacijskog kanala, prostor na diskovlju) između dozvoljenih granica,
- **strukturalna skalabilnost** označava korištenje implementacijskih normi koje uzimaju u obzir predviđeni rast sustava (u najjednostavnijem primjeru: tipovi podataka kod po-



Slika 2.7: Dvije dimenzije skalabilnosti: vertikalna i horizontalna

hrane informacija određenih veličina)

Sama skalabilnost kod informacijskih sustava dotiče dvije dimenzije prikazane slikom 2.7. Vertikalna skalabilnost označava povećanje kapaciteta pojedinih resursa infrastrukture. Primjerice, ako se radi o poslužiteljima to može biti povećanje količine radne memorije ili broja procesora, dok u slučaju komunikacijskih kanala to može označavati povećanje njihove propusnosti. Druga dimenzija označava horizontalnu skalabilnost u obliku dodavanja novih jedinica infrastrukture, primjerice dodatnih poslužitelja ili komunikacijskih kanala [70].

Skalabilnost sustava u oblaku

Najam infrastrukture u javnom oblaku omogućuje naplatu resursa razmjernu korištenju. Korisniku se omogućuje najam gotovo neograničene količine resursa. Javni oblaci omogućuju veliki potencijal za realizaciju skalabilnosti programskih sustava. Virtualizacijske tehnologije pružaju mogućnosti veoma brzih iskorištavanja dodatnih poslužitelja, mrežnih jedinica i prostora za pohranu. Dokup dodatnih resursa je automatiziran i cijeli proces se mjeri u minutama, dok su prethodno za to trebali dani. To omogućuje brze reakcije u slučajevima kada opterećenje sustava dosiže ukupni kapacitet te je potrebno uvesti nove jedinice.

Vertikalno skaliranje u oblaku omogućeno je kroz promjenu dostupnih resursa po svakom VM-u ili zamjenom jednog VM-a drugim. Međutim, većina operacijskih sustava ne podržava promjenu količine resursa bez ponovnog pokretanja [77], a postoje i mnogi izazovi u brzini migracije VM-ova tijekom rada [78]. Samim time, vertikalno skaliranje veoma je vremenski skupo, a zbog ponovnog pokretanja VM-a može doći i do dodatnih usporenja zbog ponovnog punjenja priručne memorije (engl. *cache*). Za razliku od vertikalnog skaliranja, horizontalno skaliranje je u oblaku dobro podržano. Većina davatelja javnih oblaka omogućuje dodavanje

novog VM-a unutar samo jedne minute. Iako se međuspremnici novog VM-a i dalje trebaju popuniti za optimalan rad, ne gube se podaci u prethodno korištenim VM-ovima s obzirom na to da ne dolazi do njihovog ponovnog pokretanja. Jednostavno se dio radnog opterećenja usmjeruje na novonastale VM-ove pomoću raspoređivača opterećenja (engl. *load balancer*).

2.4.3 Elastičnost sustava

Za ostvarenje elastičnosti sustava potrebno je da svi sklopovski, virtualizacijski i programski slojevi koji s njime graniče budu skalabilni. Bez adekvatne razine skalabilnosti koja omogućuje povećanje kapaciteta sustava elastičnost neće dovesti do poboljšanja radnih karakteristika. Drugi zahtjev za ostvarenje elastičnosti jest postojanje procesa za prilagodbu kapaciteta koji koriste mrežne usluge infrastrukture. Provođenje horizontalnog ili vertikalnog skaliranja u oblaku označava upravljanje infrastrukturom oblaka kroz specijalizirano programsko sučelje (engl. *Application Programming Interface*, skraćeno API) ili korisničko sučelje. Poželjno je da proces elastičnosti bude automatiziran koristeći API oblaka, no to nije nužno [15].

U literaturi postoji dvadesetak različitih definicija elastičnosti [70], koje se razlikuju po kriterijima ostvarivanja elastičnosti, ovaj rad koristi definiciju prema [15].

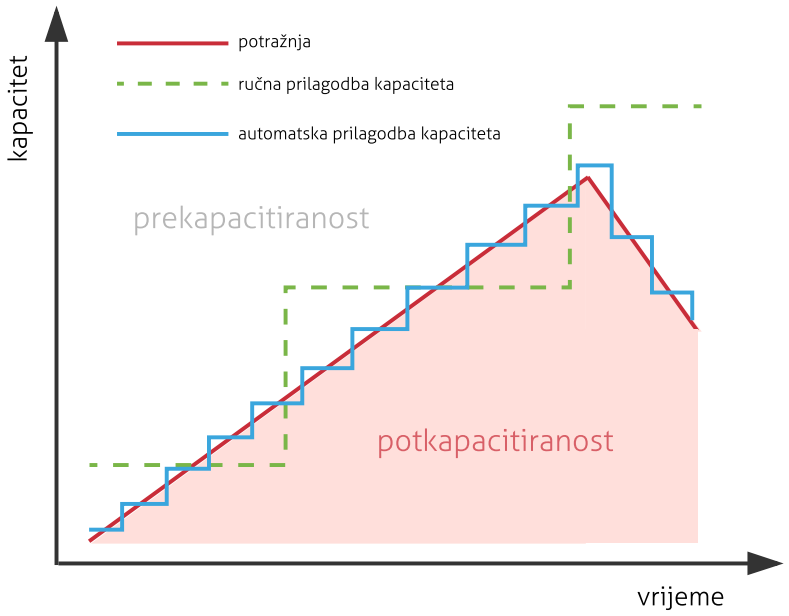
Definicija 2.9 Elastičnost

Elastičnost je stupanj u kojem sustav ostvaruje adaptaciju uslijed promjena radnoga opterećenja autonomnim zakupom i otpustom korištenih resursa tako da kapacitet sustava u svakom trenutku odgovara trenutnim potrebama [15].

U evaluaciji elastičnosti, potrebno je odgovoriti na nekoliko pitanja [15]:

- **Autonomno skaliranje** - je li proces prilagodbe kapaciteta automatiziran te na koji način?
- **Dimenzije elastičnosti** - promjenom koje vrste resursa se utječe na kapacitet sustava?
- **Zrnatost skaliranja** - koja se zrnatost koristi kod promjene količine resursa (u kojim se pomacima mijenja količina korištenih poslužitelja, procesora ili radne memorije) po svakoj vrsti resursa?
- **Ograničenje skalabilnosti** - koja je gornja granica u broju dopuštenih resursa (primjerice, količina radne memorije po poslužitelju ili ukupan broj poslužitelja) po svakoj vrsti resursa?

Najvažniji čimbenici kod ostvarivanja elastičnosti su **brzina adaptacije** i **preciznost adaptacije**. Brzina adaptacije definira se u dvije varijante: brzina povećanja kapaciteta i brzina smanjenja kapaciteta. Kritična za dobru elastičnost je naravno brzina povećanja kapaciteta koja se definira kao vrijeme koje je potrebno da sustav iz stanja preopterećenosti (engl. *underprovisioning*) dođe u stanje zadovoljavajućega kapaciteta. Na slici 2.8 prikazana



Slika 2.8: Usporedba klasičnih metoda elastičnosti s metodama koje omogućuje virtualizacija resursa u oblaku

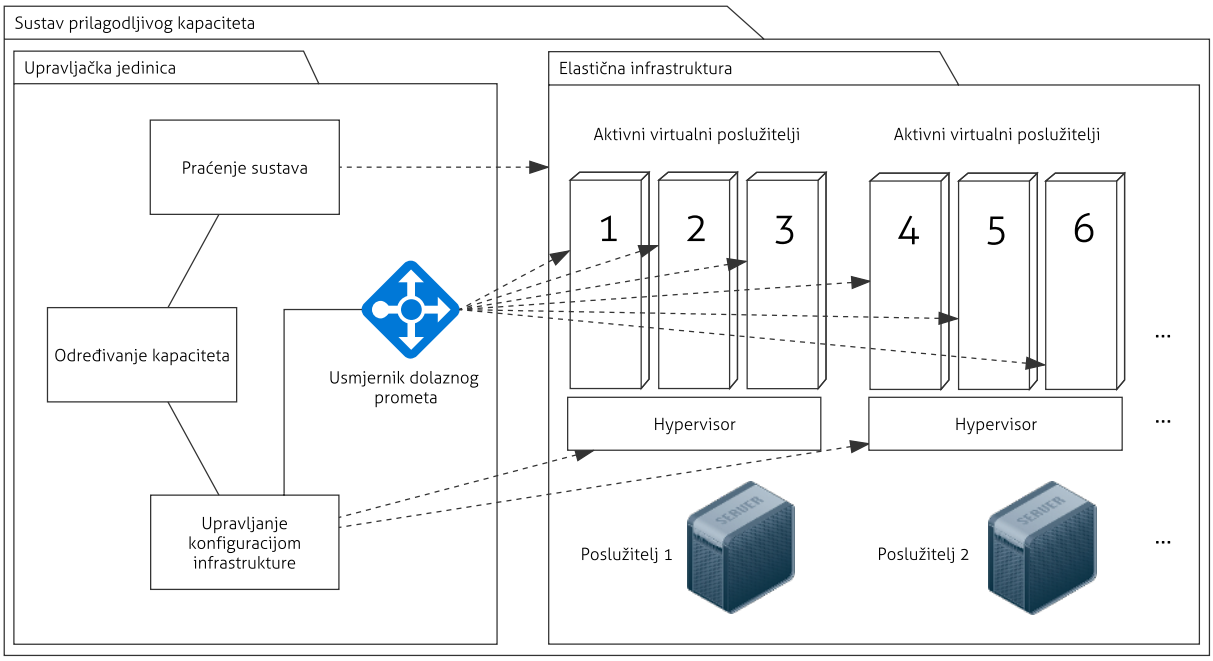
je usporedba između ručne i automatske prilagodbe kapaciteta te je vidljivo kako automatizacija omoguće veću brzinu reakcije. Brzina povećanja kapaciteta uključuje brzinu zakupa dodatnih resursa, njihovo postavljanje i rekonfiguraciju sustava za što je najmanje potrebno nekoliko minuta. Ono što može odužiti taj proces jest prijenos veće količine podataka ako nova instalacija komponente ili sustava to zahtijeva prije puštanja u rad.

2.4.4 Postupak adaptacije kapaciteta

Slika 2.9 prikazuje referentni primjer ostvarenja elastičnosti horizontalnim skaliranjem infrastrukturnih resursa [79]. Cjelokupna se arhitektura sustava sastoji od dva dijela: infrastrukture na kojoj se pruža programska usluga i sustava za upravljanje kapacitetom. Davatelj programske usluge unajmljuje VM-ove od davatelja infrastrukture oblaka nad kojima postavlja programske usluge, ali dodatno postavlja i autonomne algoritme za upravljanje kapacitetom. Pojedini davatelji infrastrukture poput Amazona nude uslugu upravljanja kapacitetom [†] pružajući mogućnost definiranja kriterija elastičnosti poput metrika koje se želi pratiti i pripadnih graničnih vrijednosti koje iniciraju promjenu kapaciteta.

Sustav za upravljanje kapacitetom u pravilu se sastoji od komponenti za ostvarivanje praćenja (engl. *monitoring*) koje prate relevantne metrike, komponenti za analizu prikupljenih podataka i donošenje odluka te komponenti za provedbu potrebne rekonfiguracije programske usluge. Rekonfiguracija se odvija u dva dijela: postavljanjem ili otpustom resursa te po-

[†]Amazon AutoScale



Slika 2.9: Referentna arhitektura elastičnog sustava

dešavanjem raspoređivača opterećenja.

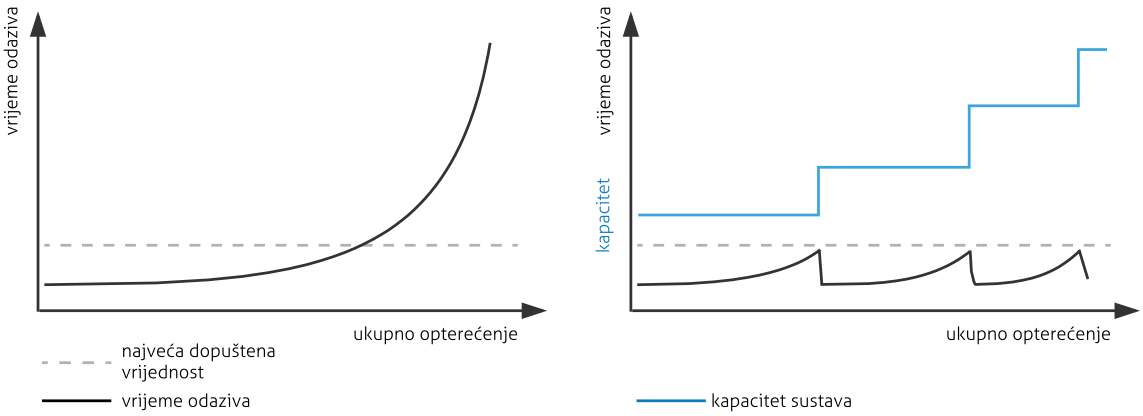
Sustavi ili programske usluge koje ostvaruju elastičnost u odnosu na radno opterećenje promjenom ukupnog kapaciteta obrade još se nazivaju *elastični sustavi*, *elastične programske usluge* ili skraćeno *elastične usluge*.

Slika 2.10 prikazuje pojednostavljeni primjer usporedbe dviju usluga kod kojih druga ostvaruje elastičnost horizontalnim skaliranjem za razliku od prve usluge koje ne posjeduje takvu mogućnost. Metrika pomoću koje je u ovom slučaju ostvarena elastičnost jest *vrijeme odaziva* tako da je zadano najveće dopušteno vrijeme odaziva. U slučaju izmjere stvarnoga vremena koje premašuje zadanu vrijednost dolazi do rekonfiguracije sustava zakupom dodatnih resursa.

2.4.5 Ostvarivanje elastičnosti

U ostvarivanju elastičnosti upotrebljavaju se poznati koncepti iz teorije upravljanja (engl. *control theory*). Najvažniji dio elastične usluge jest upravljač elastičnošću (engl. *elasticity controller*) koji prati relevantne metrike usluge i provodi potrebne korekcije. Proces koji vrši navedene radnje poznat je kao regulacijski krug MAPE-K [80]. U nastavku je pojašnjena svaka faza postupka MAPE-K.

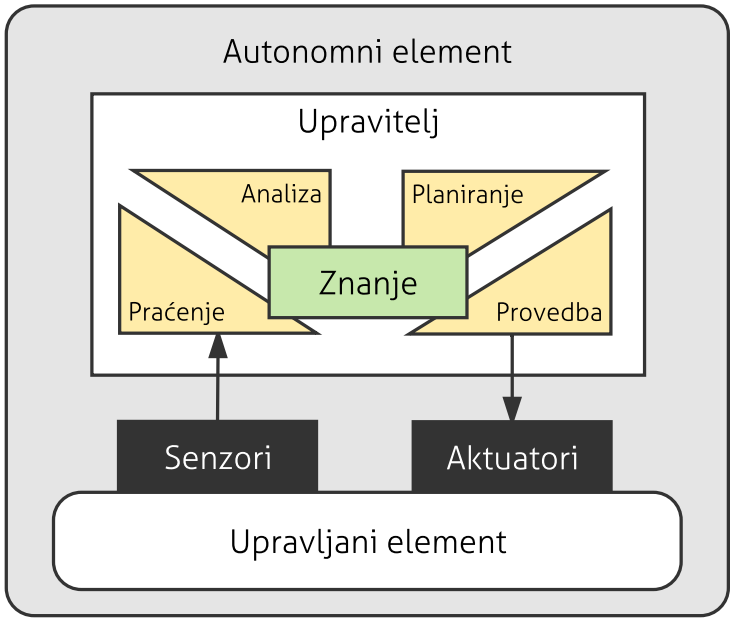
Praćenje podrazumijeva prikupljanje podataka o radnim karakteristikama usluge preko odabranih metrika za praćenje. Praćenje usluge može biti izvanjsko (praćenje metrika koje pruža njezina okolina: operacijski sustav ili infrastruktura oblaka) ili unutarnje implementacijom u sklopu usluge ako željene metrike nisu dostupne iz okoline. U fazi **analize** obrađuju



(a) Bez prilagodbe kapaciteta (b) Sustav s prilagodljivim kapacitetom

Slika 2.10: Usporedba vremena odaziva običnog i elastičnog sustava

se prikupljeni podaci i pretvaraju u znanje, najčešće parametrizacijom određenog modela sustava poput modela redova čekanja. Analizom se može utvrditi da određeni kriterij više nije zadovoljen (reaktivno praćenje) ili pomoću metoda predikcije (engl. *forecasting*) predvidjeti da uskoro više neće biti zadovoljen (proaktivno praćenje). Ako dođe do takve situacije, prelazi se u stanje *planiranja* gdje se pomoću prikupljenog znanja planiraju aktivnosti koje praćene kriterije ponovno dovode u zadovoljavajuće stanje. U konačnici, u stanju *provedbe* izvršavaju se planirane aktivnosti. U slučaju infrastrukture u oblaku, u provedbi se koriste programske usluge davatelja infrastrukture za zakup ili otpuštanje resursa.



Slika 2.11: Referentni model upravljačkog kruga MAPE-K

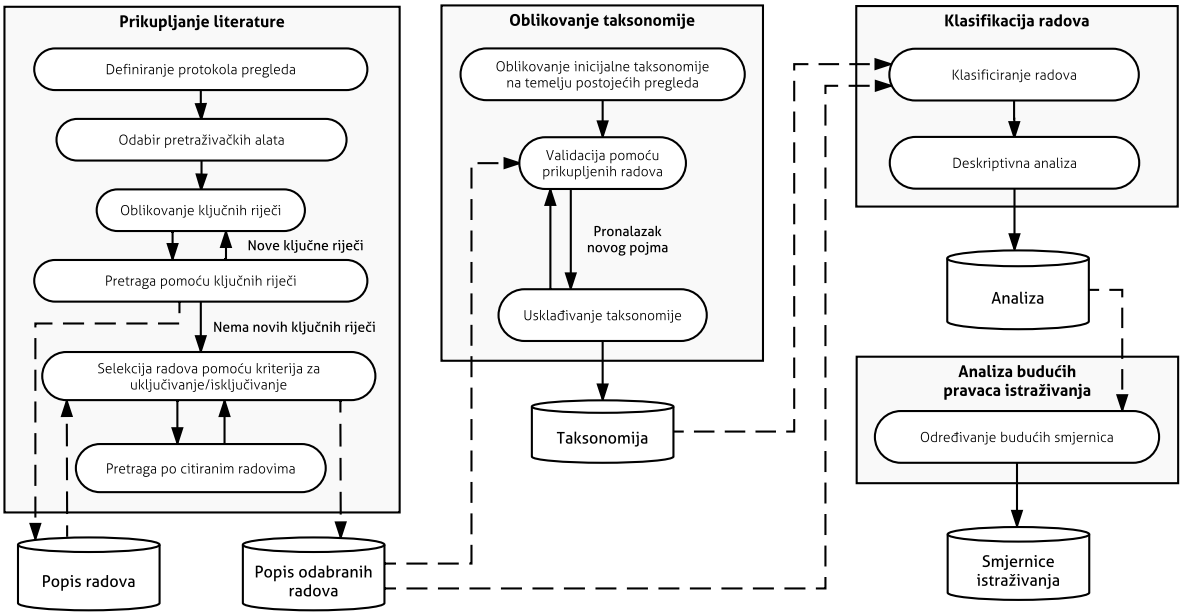
Izvor: Izvješće IBM-a o autonomnom računarstvu [81], uz grafičku obradu autora

2.5 Sustavni pregled literature

U ovom poglavlju iznosi se opsežan pregled postojećih pristupa optimizacije konfiguracije informacijskih sustava. Cilj pregleda literature jest utvrditi dosadašnji znanstveni doseg koji je podloga za znanstvene doprinose iz ovog rada. Provedeni postupak otkrivanja relevantne literature uključio je 48 odabranih radova iz više znanstvenih područja objavljenih u međunarodnim znanstvenim časopisima i zbornicima s međunarodnih skupova.

Pregled literature zasniva se na postojećem sustavnom pregledu objavljenom 2013. godine [82] koji uključuje literaturu iz područja optimizacije arhitekture informacijskih, ali i ukloplje-nih sustava. Metodologija provedbe pregleda izvedena je po smjernicama danim u [26] koje su posebno prilagođene znanstvenoj grani programskog inženjerstva. Smjernice predlažu podjelu procesa provođenja na tri dijela: priprema, prikupljanje radova i diseminacija rezultata. Prema ciljevima pregleda oblikovana su tri istraživačka pitanja koja usmjeravaju provođenje sustavnog pregleda:

- **P1**: Na koji se način može klasificirati trenutni znanstveni doseg iz područja optimizacije strukture informacijskih sustava?
- **P2**: Što predstavlja trenutni znanstveni doseg u skladu s predloženom klasifikacijom?
- **P3**: Što se može zaključiti iz analize trenutnog dosega kao smjernica daljnjeg istraživanja?



Slika 2.12: Model procesa koji se koristio pri provođenju sustavnog pregleda literature

Izvor: Prilagođeno prema [82]

2.5.1 Metodologija

S ciljem odgovaranja na postavljena istraživačka pitanja provedene su četiri aktivnosti: aktivnost prikupljanja relevantne literature te po jedna aktivnost za svako pitanje **P1-3**. Model procesa provedbe navedenih aktivnosti izveden je iz procesa korištenog u [82] te je prikazan na slici 2.12.

U sklopu prve aktivnosti razrađen je protokol koji definira istraživačka pitanja, proces provedbe te kriterije uključivanja i isključivanja radova [26]. Zatim su odabrane relevantne baze podataka za provedbu pretrage za koje su zasebno oblikovani upiti za pretraživanje temeljeni na definiranim ključnim pojmovima. Inicijalna lista ključnih pojmova oblikovana je prema ručnoj pretrazi relevantne literature te je kontinuirano nadopunjavana tijekom probne pretrage s novootkrivenim izvedenicama željenih termina. Finalna lista ključnih pojma prikazana je na slici 2.13 uz dodatne kriterijima pretrage: vrijeme, polje i jezik objave. Nakon pretrage po navedenim ključnim pojmovima provedena je selekcija relevantnih radova u tri faze:

1. postupak isključivanja literature temeljem naslova
2. postupak isključivanja literature temeljem sažetka
3. postupak isključivanja i uključivanja literature temeljem cjelokupnog sadržaja.

Nakon odabira konačnog skupa literature pridodane su publikacije koje odabrana literatura navodi, a nisu sadržane u skupu. Nakon toga je ponovno proveden postupak uključivanja i isključivanja publikacija. Prilikom odlučivanja korišteni su sljedeći kriteriji isključivanja publikacije:

- publikacija ne predstavlja metodu optimizacije ili opisuje optimizaciju koja se ne pro-

vodi na razini strukture ili arhitekture programske podrške, što uključe više komponenti i njihove međusobne odnose
- publikacija ne predstavlja optimizaciju s aspekta razvoja programske podrške, već neki drugi oblik optimizacije (npr. optimizacija izvedbene platforme)
- publikacija ne dotiče optimizaciju strukture općenitih informacijskih sustava već je specijaliziran za užu tehnologiju ili arhitekturu (npr. MapReduce arhitektura [83, 84])

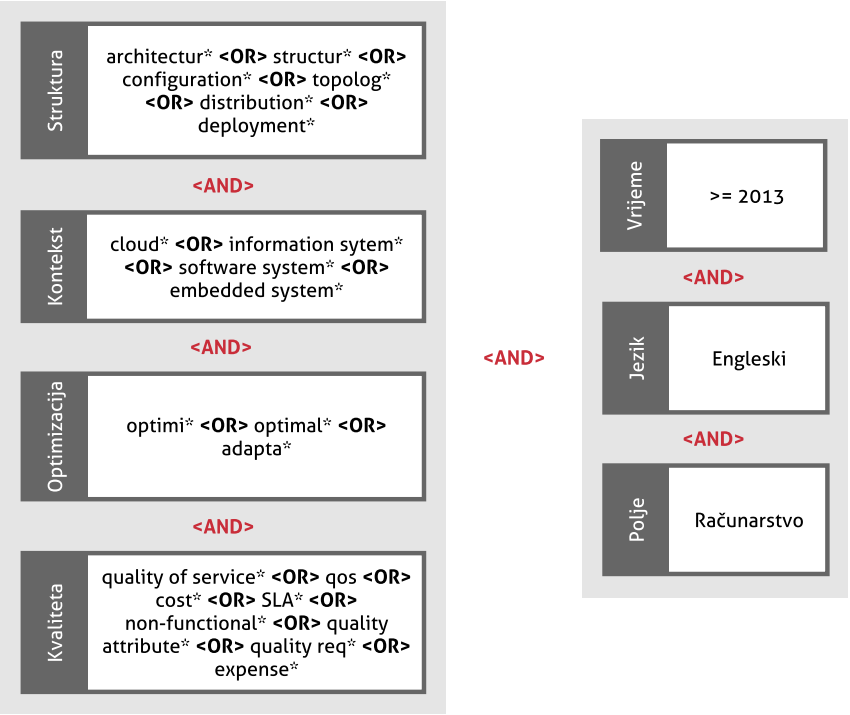
te sljedeći kriteriji uključivanja publikacije:
- optimira se struktura informacijskoga sustava koji se izvodi s unaprijed definiranim skupom funkcionalnosti čime se isključeni specijalizirani sustavi za provođenje unaprijed nepoznatih izračuna (npr. radna opterećenja na principu *Bag-of-Tasks* ili *Workflow* gdje korisnici definiraju procese koji se imaju odviti)
- postupak optimizacije mora biti automatiziran što nužno znači da moraju biti zadovoljeni sljedeći uvjeti: postoji strojno čitljiva reprezentacija strukture sustava koji se optimira, postoji postupak evaluacije različitih kriterija kvalitete i definicija prostora kojeg čine različite inačice strukture
- proces optimiranja se provodi uzimajući u obzir barem jedan od kriterija kvalitete programske podrške, bilo kao cilj optimizacije ili kao njezino ograničenje.

U svrhu odgovaranja na pitanje **P1** provedena je analiza sadržaja [85] odabranih radova. Zatim je prema rezultatima iz [82] određena prvotna inačica taksonomije koja je kontinuirano dorađivana u sklopu rezultata analize sadržaja. Rezultat ove aktivnosti jest završna taksonomija svih važnih koncepata pomoću koje su klasificirani svi radovi s ciljem odgovaranja na **P2**. Naposljetku, pomoću klasificiranih radova provodi se određivanje budućih pravaca razvoja prema pitanju **P3**.

2.5.2 Proces prikupljanja literature

Kao alati za provođenje pretrage odabrane su dvije najveće baze znanstvenih radova [86]: *Clarivate Analytics Web Of Science (WOS)* i *Elsevier SCOPUS*, a dodatno i *IEEE Explore* kao specijalizirana baza za područje tehničkih znanosti i inženjerstva. Tablica 2.2 prikazuje broj radova nakon svakog provedenog koraka selekcije, dok tablica 2.3 prikazuje konačan popis odabranih radova. Duplikati su izbačeni na temelju identifikatora DOI[‡] nakon čega je provedeno isključivanje i uključivanje definirano iznesenom metodologijom. Prikupljanje literature provedeno je u ožujku 2017. godine.

[‡]Digitalni identifikator objekata, specifikacija dostupna na *https://www.iso.org/obp/ui/*



Slika 2.13: Korištene ključne riječi i uvjeti prilikom pretrage

Tablica 2.2: Broj prikupljenih radova po fazama selekcije

	Scopus	IEEE	WOS	**Ukupno**
Pretraga po ključnim riječima	1951	605	2092	**4648**
Izbacivanje duplikata	1675	138	752	**2565**
Isključivanje temeljem naslova	227	11	57	**298**
Isključivanje temeljem sažetka	156	7	30	**193**
Uključivanje temeljem cjelokupnog rada	43	2	3	**48**

Tablica 2.3: Konačni popis uključene literature

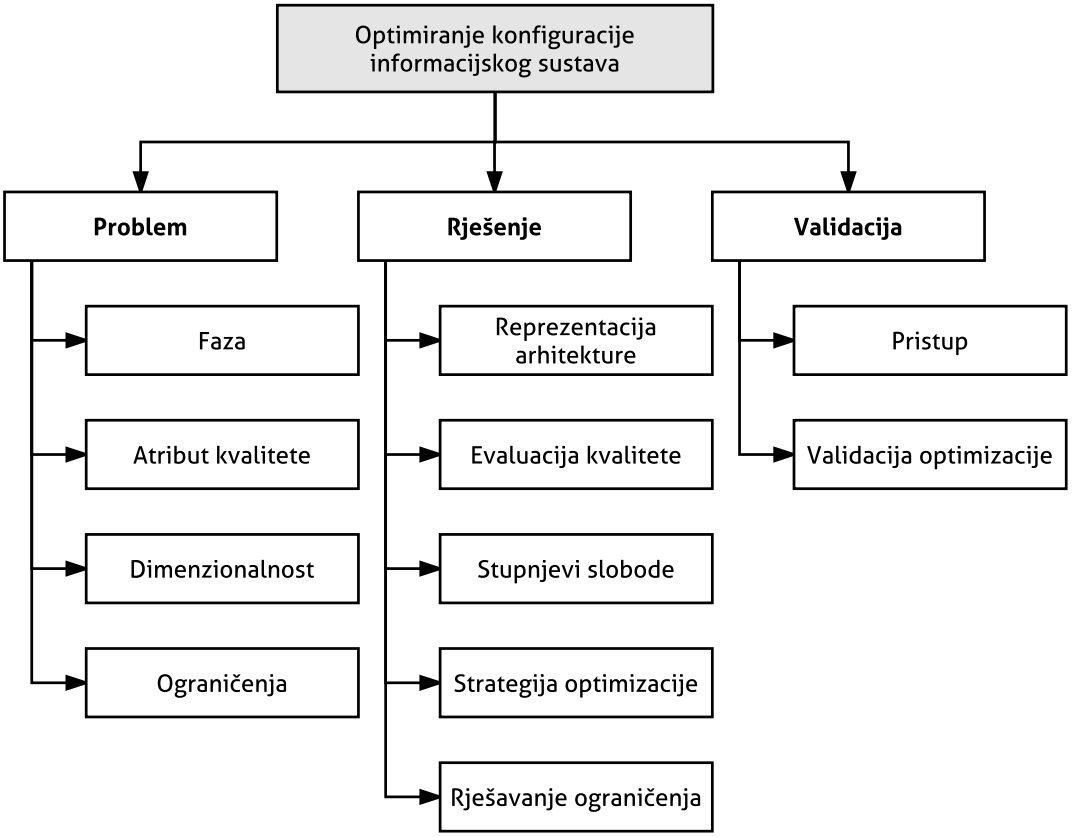
Naziv rada	God.	Ref.
A genetic-based approach to web service composition in geo-distributed cloud environment	2015	[87]
A mixed integer linear programming optimization approach for multi-cloud capacity allocation	2017	[88]
A multi-agent reinforcement learning approach to dynamic service composition	2016	[89]
A Multi-objective ACS Algorithm to Optimize Cost, Performance, and Reliability in the Cloud	2015	[90]
A New Placement Optimization Approach in Hybrid Cloud Based on Genetic Algorithm	2016	[91]
A Particle Swarm Optimization approach for cost effective SaaS placement on cloud	2015	[92]
Adaptive cloud deployment using persistence strategies and application awareness	2015	[93]
An exact placement approach for optimizing cost and recovery time under faulty multi-cloud environments	2013	[94]
Architecture optimization with sysML modeling: A case study using variability	2015	[95]
Automated QoS-oriented cloud resource optimization using containers	2017	[96]
Automatic optimisation of system architectures using EAST-ADL	2013	[97]
Business-driven optimization of component placement for complex services in federated Clouds	2014	[98]
CloudPick: A framework for QoS-aware and ontology-based service deployment across clouds	2015	[99]
Cost and utilization optimization of Amazon EC2 instances	2013	[100]
Cost minimization of service deployment in a public cloud environment	2013	[65]
Cost optimization in multi-site multi-cloud environments with multiple pricing schemes	2014	[101]
Cost optimization of virtual infrastructures in dynamic multi-cloud scenarios	2015	[102]
Cost-optimal cloud service placement under dynamic pricing schemes	2013	[103]
Cross-tier application and data partitioning of web applications for hybrid cloud deployment	2013	[104]
Decision support tool for IoT service providers for utilization of multi clouds	2016	[105]
Dynamic Deployment and Auto-scaling Enterprise Applications on the Heterogeneous Cloud	2016	[106]
Efficient Heuristics for Placing Large-Scale Distributed Applications on Multiple Clouds	2016	[107]
Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures	2016	[108]
Heterogeneity-aware adaptive auto-scaling heuristic for improved QoS and resource usage in cloud environments	2017	[109]
Hybrid cloud placement algorithm	2015	[110]
Hybrid multi-attribute QoS optimization in component based software systems	2013	[111]
Improving availability of cloud-based applications through deployment choices	2013	[112]
Infra: SLO aware elastic auto-scaling in the cloud for cost reduction	2016	[113]
Mechanisms for SLA provisioning in cloud-based service providers	2013	[114]
Minimizing deployment cost of cloud-based web application with guaranteed QoS	2015	[115]
Model-driven optimal resource scaling in cloud	2017	[116]
Modeling run-time adaptation at the system architecture level in dynamic service-oriented environments	2014	[117]
Multi-cloud Application Design through Cloud Service Composition	2015	[118]
Optimal distribution of applications in the cloud	2014	[119]
Optimal resource provisioning for scaling enterprise applications on the cloud	2015	[120]
Optimal service selection based on business for cloud computing	2013	[121]
Optimization of adaptation plans for a service-oriented architecture with cost, reliability, availability and performance tradeoff	2013	[122]
OptiSpot: minimizing application deployment cost using spot cloud resources	2016	[123]
Partitioning of web applications for hybrid cloud deployment	2014	[124]
Predictive model for dynamically provisioning resources in multi-tier web applications	2017	[125]
QoS-aware provider selection in e-services supply chain	2016	[126]
QoS-aware service VM provisioning in clouds: Experiences, models, and cost analysis	2013	[127]
Reliable Web service composition based on QoS dynamic prediction	2015	[128]
ROAR: A QoS-oriented modeling framework for automated cloud resource allocation and optimization	2016	[129]
Search-based genetic optimization for deployment and reconfiguration of software in the cloud	2013	[130]
Symbolic performance adaptation	2016	[131]
Towards an efficient service provisioning in cloud of things (CoT)	2016	[132]
Towards distributed service allocation in Fog-to-Cloud (F2C) scenarios	2016	[133]

2.5.3 Oblikovanje taksonomije

Nakon provedene analize sadržaja nisu utvrđene veće izmjene u odnosu na taksonomiju određenu u istraživanju [82]. Prve dvije razine upotrijebljene taksonomije prikazane su na slici 2.14. Prva razina hijerarhije dijeli prikupljene radove s obzirom na to kako udovoljavaju tri osnovna kriterija:

1. način na koji je problem optimiranja postavljen
2. upotrijebljene tehnike u rješavanju postavljenoga problema
3. način na koji je ispitana valjanost postupka.

U nastavku su opisani pojmovi druge razine taksonomije koji spadaju pod navedene kriterije.



Slika 2.14: Taksonomija pristupa optimiranju konfiguracije informacijskih sustava

Izvor: Prilagođeno prema [82]

Kategorija problema

Ova kategorija razlikuje radove po tome na koji su način postavili problem optimizacije. *Faza* predstavlja u kojem trenutku se odvije optimizacije gdje su moguće vrijednosti: tijekom oblikovanja sustava - DT (engl. *design-time*) ili tijekom izvedbe - RT (engl. *Runtime*). Kategorija *Atribut kvalitete* iznosi sve atribute kvalitete koji su uključeni kao cilj optimizacije,

primjerice cilj može biti minimalni trošak izvođenja uz što bolje vrijeme odaziva. Postupak optimizacije može istovremeno optimirati jedan kriterij - SOO (engl. *Single objective optimization*) ili više kriterija. Ako se optimira više kriterija kvalitete to se može ostvariti tako da se funkcija cilja oblikuje kao algebarski izraz više kriterija - MTS (engl. *Multiple-objectives To Single-objective*), primjerice linearnom kombinacijom, čime se efektivno svodi na korištenje tehnika kao za SOO. Ova tehnike reduciranja više kriterija na jedinstveni kriterij poznata je još kao metoda optimizacije *a priori* [134]. No moguće je koristiti i tehnike koje polučuju skup rješenja Pareto [135] koji udovoljava više kriterija istovremeno (metoda optimizacije *a posteriori* [134]). *Ograničenja* predstavljaju kriterije kvalitete ili ostala svojstva sustava koja nisu dio funkcije cilja već ograničavaju skup mogućih rješenja na ona koja udovoljavaju svako postavljeno ograničenje. Primjer ograničenja može biti prosječno vrijeme odaziva koje mora biti ispod određene vrijednosti.

Kategorija rješenja

Kategorija rješenja odjeljuje pristupe po načinu na koji ostvaruju postavljene ciljeve optimizacije. *Reprezentacija arhitekture* predstavlja strukturu podataka pomoću koje je predstavljena arhitektura sustava koju je potrebno optimirati. U analiziranim radovima koriste se različite reprezentacije: formulacija u obliku optimizacijskog problema (OPT), arhitekture (ARC) u obliku grafa koji prikazuje komponente i spojke, UML-a, modela Palladio Component Model (PCM) [136], modela redova čekanja (QM) ili Petrijeve mreže. *Stupnjevi slobode* [137] predstavljaju dimenzije varijabilnosti u formiranju strukture sustava. Skup stupnjeva sloboda definira ukupni prostor pretrage optimizacijskog problema. Primjer stupnja slobode može biti odabir između različitih komponenti koje implementiraju isto sučelje, količina u replikaciji komponenata kod ostvarivanje elastičnosti ili odabir vrste poslužitelja. *Strategija optimizacije* označava način na koje je utvrđeno rješenje optimizacijskog problema. Dijeli se na aproksimaciju pomoću genetskih algoritama, heuristika, strojnog učenja ili pronalazak egzaktnih rješenja pomoću različitih oblika cjelobrojnog programiranja ili algoritama nad grafom [138]. Naposljetku, kriterij *Rješavanje ograničenja* predstavlja način na koji algoritam pronalazi rješenja koje zadovoljavaju postavljena ograničenja što može biti: (1) uvođenjem penala u funkciju cilja ili (2) isključivanjem rješenja koje krše ograničenja.

Kategorija validacije

Kategorija validacije predstavljena je pomoću dvije potkategorije: (1) korišteni *pristup* pri validacije te (2) način na koji je ostvarena *validacija postupka optimizacije*.

Pristup validacije opisuje na koji način je validirana primjenjivost i praktičnost postupka optimizacije. Pritom se mogu koristiti postupci koji simuliraju sustav čiju konfiguraciju treba

optimizirati nasumičnim parametrima ili stvarnim prikupljenim parametrima, ili pak stvarni sustav iz prakse pomoću studije slučaja [27].

2.5.4 Kategorizacija dosadašnjih istraživanja

Tablice 2.4, 2.5 i 2.6 redom prikazuju podjelu uključene literature prema dimenzijama postavljanja, rješavanja problema te pripadne validacije. Pojedinačno gledano, najčešće je problem optimizacije strukture postavljen kao optimizacija troška korištenja infrastrukture (90% slučajeva) te najviše u fazi dizajna sustava (60% slučajeva). U 83% slučajeva promatra se optimizacija samo jednog kriterija ili linearne kombinacije više kriterija, dok samo u 17% istraživanja promatra istovremenu optimizaciju više kriterija gdje se konačno vaganje između oprečnih kriterija prepušta korisnicima nakon što se sagledaju odnosi između pojedinih kriterija. Razlog tome je što višekriterijska optimizacija isključuje korištenje lako dostupnih egzaktnih metoda optimizacije kroz različite dostupne pakete cjelobrojnog programiranja (engl. *integer programming*).

Stupnjevi slobode koji čine mogući prostor konfiguracija informacijskoga sustava najčešće se dotiču odabirom adekvatne vrste i količine poslužitelja prilikom skaliranja sustava u 56%, odnosno 46% radova. Kriteriji kvalitete pojedinih konfiguracija najčešće se određuju jednostavnim agregacijskim funkcijama koje predviđaju kvalitetu (58% slučajeva). Razlog tome je što je zbog velikog prostora pretrage postoji snažna sklonost u korištenju brzih i jednostavnih modela predikcije kriterija kvalitete niske prediktivne moći. Kvalitetnije evaluacije u obliku simulacije ili mjerenja nad stvarnim sustavom zastupljene su u svega 13% slučajeva.

Prilikom rješavanja optimizacijskog problema najčešće se koriste aproksimacijske metode (60% radova) u obliku genetskih algoritama (19%) ili heuristika pohlepnog tipa (engl. *greedy heuristic*) (8%), gdje se iterativno odabiru bolja rješenja na lokalnoj razini. Egzaktna rješenja (35% slučajeva) uglavnom se oslanjaju na različite oblike cjelobrojnog programiranja.

Tablica 2.4: Podjela literature po dimenzijama s obzirom na formulaciju problema

Faza		
DT	29	60,4%
RT	18	37,5%
DT+RT	1	2,1%

Dimenzionalnost		
SOO	30	62,5%
MTS	10	20,8%
MOO	8	16,7%

Atribut kvalitete		
Trošak	43	89,6%
Vrijeme odziva	9	18,8%
Dostupnost	6	12,5%
Pouzdanost	4	8,3%
Iskoristivost	4	8,3%
Latencija	2	4,2%
Propusnost	2	4,2%
Brzina procesuiranja (MIPS)	1	2,1%
Oslonjivost	1	2,1%
Energija	1	2,1%
Općenito	1	2,1%
Reputacija	1	2,1%
Sigurnost	1	2,1%

Ograničenje		
Ne postoji	21	43,8%
Ograničenja prema resursima	14	29,2%
Vrijeme odziva	11	22,9%
Dostupnost	4	8,3%
Trošak	3	6,3%
Propusnost	3	6,3%
Latencija	2	4,2%
Pouzdanost	2	4,2%
Percentil vremena odziva	2	4,2%
Iskoristivost	2	4,2%
Općenito ograničenje	1	2,1%
Reputacija	1	2,1%
Sigurnost	1	2,1%

Tablica 2.5: Podjela literature po dimenzijama s obzirom na formulaciju rješenja

Reprezentacija strukture		
Ne postoji	**12**	**25%**
Model arhitekture	**20**	**41,7%**
- Graf	12	25%
- Palladio Component Model	2	4,2%
- ADL	1	2,1%
- CloudML	1	2,1%
- DSL	1	2,1%
- KDM	1	2,1%
- SysML	1	2,1%
Optimizacijski problem	**14**	**29,2%**
Model redova čekanja	**3**	**6,3%**

Evaluacija kriterija kvalitete		
Agregacijske funkcije	28	58,3%
Nepoznato	6	12,5%
Mjerenje nad stvarnim sustavom	5	10,4%
LQN	3	6,3%
MILP	3	6,3%
QM	3	6,3%
DTMC	2	4,2%
Simulacija	1	2,1%

Stupnjevi slobode		
Vrsta poslužitelja	27	56,3%
Horizontalno skaliranje	22	45,8%
Odabir pružatelja infrastrukture	8	16,7%
Odabir regije poslužitelja	8	16,7%
Odabir programske usluge	8	16,7%
Odabir komponente	3	6,3%
Odabir pravila skaliranja	1	2,1%
Veličina medija za pohranu	1	2,1%
Određivanje cijene	1	2,1%

Udovoljavanje ograničenja		
Nije predstavljeno	29	60,4%
Iskjučivanjem	19	39,6%
Penalizacijom	3	6,3%

Strategija optimizacije		
Nepoznato	**2**	**4,2%**
Aproksimacija	**29**	**60,4%**
- GA	9	18,8%
- Pohlepna heuristika	4	8,3%
- Ostale heuristike	3	6,3%
- Bin-packing	2	4,2%
- ML	2	4,2%
- Q-Learning	2	4,2%
- Jednostavna pravila	2	4,2%
- Ant-colony	1	2,1%
- Branch-and-Bound	1	2,1%
- CT	1	2,1%
- EMOA	1	2,1%
- PSO	1	2,1%
- min-k-cut	1	2,1%
Egzaktno	**17**	**35,4%**
- ILP	5	10,4%
- BIP	3	6,3%
- CSP	2	4,2%
- BIS	1	2,1%
- Formula	1	2,1%
- GP	1	2,1%
- MINLP	1	2,1%
- MIP	1	2,1%
- SMT	1	2,1%

Tablica 2.6: Podjela literature po dimenzijama s obzirom na način validacije

Način validacije		
Studija slučaja	20	41,7%
Simulacija primjene optimizacije	16	33,3%
Simulacija pomoću izmjerenih podataka	10	20,8%
Nije predstavljeno	1	2,1%
Eksperiment	1	2,1%

Validacija optimizacije		
Simulacija	36	75%
Eksperiment	10	20,8%
Nije predstavljeno	2	4,2%

2.5.5 Diskusija prethodnih istraživanja

Prethodnim istraživanjima pokazano je da razmještaj komponenti na sklopovlje najčešći predmet optimizacije te da ima veliki utjecaj na radne karakteristike aplikacije [21, 22]. Naglasak ove disertacije je promjenjivost radnog opterećenja sustava te korištenje metrika atributa kvalitete korištenih u sporazumima o razini usluge. Pri takvim uvjetima arhitektura sustava mora omogućiti prilagodljivost različitim opterećenjima automatskim zakupom ili otpuštanjem poslužitelja na kojima se aplikacija izvodi [15].

Huang i Shen [139] proveli su istraživanje koje uzima u obzir strukturu razmještaja komponenti na sklopovlje u reduciranju vremena odaziva pojedinih usluga, ali pri uvjetima konstantnog opterećenja. Hadded [140] predlaže algoritam koji određuje količinu potrebnih regulatora (engl. controller) za ostvarenje prilagodljivosti usluge, ali ne razmatra modeliranje željenog promjenjivog opterećenja i ocjenu uspješnosti u ostvarivanju željenih radnih karak-

teristika. Okvir ROAR [141] omogućuje modeliranje statičnog opterećenja te osigurava željenu razinu radnih karakteristika odabirom odgovarajuće strukture razmještaja komponenti na sklopovlje infrastrukture računarstva u oblaku. Međutim, ROAR ne omogućava modeliranje promjenjivog opterećenja i samim time ne evaluira prilagodljivost sustava. Andrikopolous [142] istražuje optimalne topologije koristeći infrastrukturu računarstva u oblaku, ali također u uvjetima statičkog opterećenja. Optimizaciju ostvaruje definiranjem potrebne količine resursa za ostvarenje aplikacije te troškova zakupa infrastrukture. Sustav tada rangira različite strukture razmještaja komponenti ovisno o ukupnom trošku izvođenja pomoću jednostavnijih analitičkih modela.

Metode optimizacije konfiguracije mogu se također podijeliti na: (1) metode optimizacije modeliranjem programske potpore u ranijim fazama razvoja te (2) metode optimizacije realiziranog sustava mjerenjem tijekom izvođenja u kasnijim fazama razvoja programske potpore [143]. Metode optimizacije zasnovane na modeliranju programske potpore podrazumijevaju modeliranje sustava na temelju iskustva arhitekta i poznatih zahtjeva koje sustav treba ostvariti. Sustav se modelira Markovljevim modelima i modelima teorije redova čekanja (engl. queueing theory) [48] te se pritom često koriste evolucijski algoritmi koji istražuju veliki broj mogućih instanci arhitektura. Rezultati su ograničeni preciznošću i kvalitetom odabranog modela sustava zbog čega se takve metode najčešće upotrebljavaju u ranijim fazama. Primjer takvih metoda dan je u istraživanjima [90, 144, 145].

2.5.6 Optimiranje konfiguracije modeliranjem sustava

Postoje alati koji se bave optimizacijom arhitekture ili konfiguracije sustava u svrhu ostvarivanja željenih kriterija kvalitete. U nastavku se iznose relevantni alati. Novina ovog rada u odnosu na predstavljene alate jest podrška za informacijske sustave u oblaku koje karakterizira elastičnost kapaciteta obrade u odnosu na promjene u radnom opterećenju. Zbog takvih promjenjivih uvjeta potrebno je razviti metodu optimizacije konfiguracije koja pruža mogućnost definiranja i simulacije promjenjivosti u radnom opterećenju uz autonomno horizontalno skaliranje sustava. Druga novina jest korištenje hibridne tehnike optimizacije pri kojoj se veliki broj početnih kandidata smanjuje koristeći evolucijske algoritme evaluacijom ekvivalentnih simulacijskih modela, dok se najbolja rješenja dobivena tim putem u konačnici evaluiraju na stvarnom sustavu. Postupak simulacije korišten u ovom radu razvijen je specifično za evaluaciju sustava elastičnog kapaciteta u oblaku.

ArcheOpterix [146] je alat koji je razvijen za optimizaciju konfiguracije uklopljenih sustava. Zasniva se na specifikaciji AADL za opisivanje arhitekture sustava za koju pruža optimiranje u skladu s ograničenjima i kriterijima kvalitete. Trenutna inačica sustava ArcheOpterix optimira razmještaj komponenti na jedinice sklopovlja s različitim karakteristikama s ciljem poboljšanja pouzdanosti u prijenosu podataka između komponenti i količine prenese-

nih podataka. ArcheOpterix koristi evolucijske algoritme u evaluaciji velikog broja mogućih kandidata. ArcheOpterix je ostvaren kao dodatak za razvojnu okolinu Eclipse.

PerOpteryx [137] se također koristi za optimizaciju sustava zasnovanih na komponentama, ali to ne moraju nužno biti uklopljeni sustavi. PerOpterix se temelji na definiranju stupnjeva slobode koje konfiguracija sustava može ostvariti poput odabir između alternativnih komponenti istog sučelja, odabir vrste poslužitelja i rasporeda komponenti na poslužitelje. U optimizaciji također koristi evolucijske algoritme gdje se kao evaluacijska funkcija koristi slojevita mreža redova čekanja (engl. *Layered Queue Network*, skraćeno LQN). Moguće je definirati više oprečnih ciljeva optimizacije pomoću metode za selekciju kandidata NSGA-II [147]. Slično kao i ArcheOpterix, PerOpterix je ostvaren kao dodatak za razvojnu okolinu Eclipse.

SPACE4CLOUD [148] proširuje model Palladio Component Model (PCM) [136] za evaluaciju informacijskih sustava u oblaku. Cilj alata jest predikcija troškova i performansi sustava. SPACE4CLOUD definira meta-model informacijskoga sustava u oblaku koji omogućuje korištenje više davatelja usluge. SPACE4CLOUD omogućuje definiranje promjenljivog radnog opterećenja te analizu modela konkretnog sustava u određenoj konfiguracije, no nije moguća pretraga prostora različitih konfiguracija s ciljem pronalaska optimalnih rješenja. Također, radne karakteristike nije moguće stohastički modelirati što je važan element za sustave u oblaku zbog visokog stupa promjenljivosti.

U razvoju je također novi alat koji je kombinacija SPACE4CLOUD i PerOpteryxa [88, 149] koja omogućuje pretragu različitih konfiguracija sustava u oblaku. Cjelokupni sustav je ostvaren u dvije faze: PerOpteryx sustav pronalazi optimalnu raspodjelu komponenti na poslužitelje, a SPACE4CLOUD pronalazi optimalnu konfiguraciju u vrsti i broju poslužitelja za rješenja iz prvog koraka. Ovakav je alat logičan slijed stapanja prethodna dva alata, međutim optimalna rješenja prvog koraka ne moraju nužno biti dobar set kandidata za provođenje optimizacije u drugom koraku te je potrebno ostvariti jedinstvenu metodu optimizacije koja obje dimenzije rješenja sagledava zajedno. Rješenje predstavljeno ovim radom provodi upravo takvu optimizaciju gdje se ta dva koraka zajedno evaluiraju zbog njihove međusobne zavisnosti.

2.5.7 Zaključak

Iz analize dosadašnjeg znanstvenog dosega, uvidjelo se da nije dana dovoljna pažnja modeliranju promjenjivog opterećenja u višekriterijskoj optimizaciji strukture sustava. Iz provedene sustavne analize, 35% radova pretpostavlja elastičnost sustava, od čega se 24% radova dotiče višekriterijske optimizacije što čini ukupno 4 istraživanja [129, 130, 131, 150].

Ovaj rad spada u skupinu hibridne metode optimizacije konfiguracije koje se zasnivaju na mjerenju stvarnih karakterističnih vrijednosti poput vremena odaziva sustava, dok se simulacijski modeli koriste za bržu evaluaciju kriterija kvalitete prilikom pretrage prostora kandi-

data.

Za razliku od prethodnih istraživanja, ovaj rad istražuje modeliranje promjenjivog opterećenja sustava zbog utjecaja koje ono odražava na kriterije kvalitete tako da se umjesto srednjih vrijednosti metrika koje predstavljaju kriterije kvalitete prate percentili korišteni u ugovorima SLA. Nadalje, ostvarena je izvedbena okolina za optimizaciju arhitekture aplikacije koja prilagodljivost promjenjivom opterećenju ostvaruje korištenjem infrastrukture računarstva u oblaku. Razina prilagodljivosti evaluira se statističkom analizom kriterija kvalitete koje aplikacija ostvaruje pri različitim mogućim strukturama i uvjetima modeliranog promjenjivog opterećenja. Konačan cilj izvedbene okoline jest omogućiti davateljima usluge olakšani način pronalaska optimalnih arhitektura koja zadovoljavaju željene kriterije kvalitete definirane SLA-ovima.

Poglavlje 3

Arhitektura elastičnog informacijskoga sustava zasnovana na komponentama

Primarni cilj razvoja programske podrške jest ispunjavanje utvrđenih korisničkih zahtjeva. Pritom je arhitektura sustava najvažniji čimbenik u ostvarenju kako funkcionalnih tako i nefunkcionalnih zahtjeva [43]. Postoji mnogo arhitekturalnih stilova koji služe kao uzorci pri izgradnji konačne arhitekture. Svaki stil sadrži pripadne karakteristike namijenjene rješavanju određene vrste problema [151]. Takve je karakteristike potrebno sagledati i ocijeniti njihovu primjenu u realizaciji konkretne arhitekture. Arhitektura sustava ostvaruje se kombiniranjem različitih arhitekturalnih stilova koji se odabiru tako da najbolje udovolje postavljenim zahtjevima [29].

U ovom poglavlju iznosi se meta-model arhitekture informacijskoga sustava koja omogućuje optimalnu prilagodbu konfiguracije zavisno o trenutnom radnom opterećenju. Važno je naglasiti kako se ovdje ne radi o arhitekturi ili modelu arhitekture određene instance sustava, već o meta-modelu arhitekture koji postavlja ograničenja u izgradnji konkretnog modela arhitekture. Kao alat za definiciju predloženog meta-modela arhitekture korišten je *profil* jezika UML [152]. Pomoću predloženog profila UML-a moguće je modelirati sustave koji ostvaruju karakteristike predložene arhitekture. Meta-model arhitekture ostvaren je da zadrži općenitost primjene. Meta-model kao takav ne dotiče implementacijske detalje vezane uz odabranu platformu razvoja ili izvođenja programske podrške, već opisuje problem na višoj razini, nadogradnjom postojećih arhitekturalnih stilova [153].

U nastavku ovog poglavlja prvo se definiraju glavni zahtjevi koje meta-model arhitekture mora ostvariti, prema čemu se daje poveznica na postojeće arhitekturalne stilove koji su uključeni u predloženi meta-model. Ograničenja koja predložena arhitektura postavlja opisana su kroz različite poglede (engl. *architectural views*) iz čega će se deducirati meta-model za ostvarivanje konkretne arhitekture u obliku profila UML-a. Naposljetku, iznosi se model procesa izgradnje sustava pomoću arhitekture koji uključuje korak optimizacije.

3.1 Zahtjevi prema modelu arhitekture

Ovo potpoglavlje opisuje glavne zahtjeve koje je potrebno ostvariti kroz model arhitekture koji udovoljava potrebama određenih kontekstom ovog istraživanja.

Primarni zahtjev predložene arhitekture jest omogućiti elastičnost sustava u skladu s promjenljivim radnim opterećenjem koji proizlazi iz promjenljivog broja korisnika sustava ili promjenljivog intenziteta korištenja. Elastičnost ukupnog kapaciteta obrade sustava bitan je čimbenik kvalitete usluge u takvim uvjetima. Ostvarivanju elastičnosti uvjetuje postojanje skalabilnosti — mogućnosti da se kapacitet sustava poveća jednostavnim dodavanjem infrastrukturnih resursa.

Iz toga proizlazi i sekundarni zahtjev: omogućiti dinamičku rekonfiguraciju sustava u korištenju infrastrukturnih resursa tijekom izvođenja. Rekonfiguracija mora omogućiti automatiziranu promjenu konfiguracije koja može značiti jednostavnije horizontalno skaliranje pojedinih elemenata sustava, ili promjenu u razmještaju komponenti na infrastrukturne resurse. Mogućnost rekonfiguracije nužan je preduvjet za automatizirano provođenje optimizacije konfiguracije. U kontekstu ovog istraživanja svrha optimizacije konfiguracije jest odabir kandidata koji ostvaruje optimalne kriterije kvalitete u odnosu na trošak izvođenja.

3.2 Primijenjeni arhitekturalni stilovi

Svaki arhitekturalni stil izriče određena ograničenja u strukturi komponenata sustava. Odabrani stilovi potkrijepljeni su dodatnim pojašnjenjem u čemu doprinose postavljenim zahtjevima.

Predložena arhitektura slijedi nekoliko poznatih arhitekturalnih stilova. To su:

1. arhitektura zasnovana na komponentama [36]
2. uslužno orijentirana arhitektura [68]
3. slojevita arhitektura [2].

Od arhitekturalnih uzoraka uključena je i replikacija [154] te klijentsko-poslužiteljski stil [43]. U nastavku je pojašnjen svaki od odabira.

Arhitekturalni stil zasnovan na komponentama podrazumijeva dekompoziciju sustava na komponente od kojih svaka pruža definirano sučelje. Komponente su međusobno u interakciji te podliježu pravilima definiranim kroz odabrani komponentni model (npr. CORBA, EJB, .NET). Razmatranje sustava podijeljenoga na više komponenata omogućuje lakšu evoluciju jer naglašava nezavisnost svake pojedine komponente. Dodirnu točku između komponenti predstavlja unaprijed definirano sučelje pomoću kojega komponente ostvaruju međusobnu interakciju. Dokle god se sučelje ne mijenja, moguće je unaprijediti implementaciju određene komponenti ili ju zamijeniti drugom s boljim radnim karakteristikama. Funkcionalnosti

sustava određene su dekompozicijom sustava na manje module, dizajnom sučelja komponenti te radnim karakteristikama svake komponente. Cilj predložene arhitekture je uputiti na strukturiranje sustava pružajući model arhitekture koji naglašava specifičnosti raspodijeljenog sustava podložnog promjenjivom radnom opterećenju. Korištenjem predloženog modela, korisnik je primoran donositi odluke u skladu s karakteristikama i ograničenjima raspodijeljenog elastičnog sustava pri čemu će u odlučivanju koristiti informacije predložene izvedbene okoline za optimiranje konfiguracije.

Uslužno orijentirani arhitekturni stil omogućuje višu razinu nezavisnosti između komponenti jer se svaka usluga može ostvariti u nezavisnoj tehnologiji. Često su usluge i geografski dislocirane i održavane od strane različitih davatelja usluga. Jedina dodirna točka između usluga je komunikacijski protokol gdje je potrebno odabrati obostrano podržane protokole (npr. SOAP, REST, XML-RPC). Za razliku od arhitekture zasnovane na komponentama, uslužno orijentirana arhitektura stavlja veći naglasak na sustav koji je raspodijeljen putem mreže, dajući prijedloge za rješavanje čestih problema prisutnih na mreži: pouzdanosti u prijenosu informacije putem mreže te lociranja željenih usluga. Uslužno orijentirana arhitektura predlaže protokole koji su kompatibilni s postojećim nižim protokolima razmjene informacija u organizacijama, zbog čega su mrežne usluge često jednostavnije za implementaciju u postojećim infrastrukturama koje koriste vatrozide i razdjelnike prometa za razliku od raspodijeljenih komponentnih modela poput CORBA-e. Predložena arhitektura primjenjuje prednosti uslužno orijentirane arhitekture ostvarujući komunikaciju između komponenti pomoću mrežnih usluga. Razlog tomu jest ostvarivanje horizontalne skalabilnosti replikacijom pojedinih usluga pri čemu je omogućeno korištenje razdjelnika prometa na mrežnom sloju (engl. *load balancer*).

Također, u predloženoj arhitekturi sadržan je i *slojeviti arhitekturalni stil* koji komponente sustava odjeljuje na komponente za ostvarenje prezentacije, logike, i trajnosti podataka. Komponente se dijele po slojevima zbog različitih postupaka u ostvarenju elastičnosti specifičnih za svaki sloj. Klijentsko-poslužiteljski arhitekturni stil prisutan je u većini programskih aplikacija unutar informacijskih sustava [155, 156], a označava odijeljenost sustava na klijentske i poslužiteljske komponente koje međusobno komuniciraju putem računalne mreže. Time je omogućeno olakšano dodavanje novih klijenata u sustav [19]. Ovaj stil neizostavan je u ostvarivanju informacijskih sustava koji podliježu promjenama u broju korisnika. Primjerice, mobilne aplikacije često koriste udaljene usluge centralnog sustava.

Konačni arhitekturalni stil, koji očituje navedene arhitekturne stilove, odražava sljedeća svojstva:

- sustav je sastavljen od komponenti u obliku mrežnih usluga koje koriste klijenti informacijskoga sustava
- komponente mrežnih usluga ostvaruju horizontalnu skalabilnost

- broj klijenata i intenzitet korištenja je promjenjiv što rezultira promjenjivim radnim opterećenjem čitavog sustava.

3.3 Pogledi na predloženu arhitekturu

Autori iz [19] predlažu tri osnovna pogleda koja se koriste prilikom definiranja arhitekture programske podrške: pogled na module (engl. *module view*) koji daje odgovore kako je sustav strukturiran s implementacijskog aspekta, pogled na komponente i spojke (engl. *component-and-connector view*) pokazuje elemente sustava i njihove radne značajke tijekom izvođenja te pogled na razmještaj (engl. *allocation view*) koji definira na koji su način elementi povezani sa strukturama iz okoline poput računalne infrastrukture. Prilikom vizualizacije različitih pogleda na arhitekturu koristi se notacija jezika UML.

3.3.1 Pogled na module

Modul je jedinica implementacije koja pruža koherentan skup odgovornosti. To može biti razred ili skup razreda u objektno-orijentiranoj paradigmi, skup srodnih funkcija iz proceduralnih i funkcijskih jezika, aspekt iz aspektno-orijentirane paradigme ili bilo koja dekompozicijska jedinica u fazi implementacije. Relacije zavise o odabranoj paradigmi prilikom implementacije, no većinom se mogu svesti na ove tri vrste: *sastoji se od*, *zavisi od*, i *je*.

Predložena arhitektura ne stavlja posebna arhitekturalna ograničenja na razini modula niti općenito prilikom oblikovanja arhitekture pogledom na module, već su sva potrebna ograničenja izvedena prilikom promatranja sustava kao skupine komponenti i spojki te se ta ograničenja onda prenose i na same module koji čine različite komponente sustava. Prilikom odabira paradigme za izgradnju različitih modula također ne postoje nikakva ograničenja, iako se često u izgradnji informacijskih sustava koristi objektno-orijentirana paradigma.

3.3.2 Pogled na komponente i spojke

Model predložene arhitekture definira komponentu kao skupinu modula koji međusobno komuniciraju pomoću sitnozrnatih (engl. *fine-grained*) sučelja te počivaju u istom adresnom prostoru, što znači da su i implementirane istom tehnologijom (Java, Python, ...). Ograničenje koje se postavlja pred komponentu u ovoj arhitekturi jest da njezino sučelje mora zadovoljiti uvjete koji su potrebni da se komunikacija odvije pomoću mrežnih usluga. To znači sljedeće:

1. svi parametri se predaju po vrijednosti (engl. *by-value*)
2. podrazumijeva se da je sučelje krupnozrnato (engl. *coarse-grained*)
3. nije moguće sačuvati stanje između dva različita poziva metoda sučelja
4. metode sučelja posjeduju svojstvo idempotentnosti operacija

Dodatne poželjne karakteristika komponente koje utječu na njezinu razinu elastičnosti jesu brzina njezine aktivacije. Brza aktivacija komponente osigurava kvalitetnije horizontalno skaliranje zbog bržeg stabiliziranja kapaciteta obrade cijeloga sustava.

Komponente sustava mogu biti dio različitih slojeva, poput prezentacijskog, aplikacijskog ili podatkovnog sloja [2]. Predložena arhitektura stavlja nekoliko važnih ograničenja na komponente iz aplikacijskog sloja te se u nastavku zasebno opisuju implikacije svakog od tih ograničenja.

Nepostojanje stanja u komponentama

Stanje (engl. *state*) komponente podrazumijeva pohranu informacija između dva ili više zahtjeva pojedinog klijenta. Primjerice, to može biti stanje rezerviranih artikala kod mrežne trgovine ili podaci o korisniku. Takvo privremeno zadržavanje informacija u adresnom prostoru komponente narušava skalabilnost sustava jer prilikom povećanja kapaciteta sustava paralelnim izvršavanjem više istih komponenti, potrebno je voditi računa o tome da se zahtjevi istih korisnika uvijek upućuju prema modulima koji sadrže njihovo vezano stanje. Samim time narušava se i elastičnost sustava, jer se stanje unutar komponente ne smije izgubiti dokle god postoji korisnička sesija prema toj komponenti. Posljedično, nije moguće ostvariti dinamičnu deinstalaciju komponente i oslobađanje resursa bez dodatnog pohranjivanja stanja.

Kako su korisničke sesije često potrebne, zaobilazno rješenje je koristiti gotove komponente za pohranu stanja poput specijaliziranih memorijskih baza podataka tipa *ključ-vrijednost* koje posjeduju svojstvo skalabilnosti (Redis, Riak, ...). Negativna posljedica ovog zaobilaznog rješenja jest uvođenje dodatne latencije u komunikaciji s komponentom za pohranu stanja što predstavlja dodatno ograničenje prilikom određivanja konfiguracije konačnog sustava.

Komunikacija između komponenti

Komunikacija između komponenti predložene arhitekture odvija se pomoću krupnozrnatih sučelja te može biti ostvarena na dva načina zavisno o razmještaju komponenti na sklopovlje. Ako komponente počivaju u istom adresnom prostoru, što ujedno podrazumijeva da su dio isto komponentnog modela i izvedbene okoline, tada se komunikacija odvija prema načelima definiranim komponentnim modelom. Kada su komponente dio udaljenih okolina, tada se komunikacija odvija putem mrežnih protokola tehnologijom mrežnih usluga. Komunikacija u prvom scenariju nije podložna degradaciji, dok je udaljena komunikacija podložna degradaciji i potrebno je razviti dodatne sigurnosne provjere jednom kada se odredi optimalna konfiguracija.

Cilj ove arhitekture jest omogućiti dinamičnu rekonfiguraciju u distribuiranom okruženju. To znači da se ne može unaprijed odrediti način komunikacije između komponenti, što nadalje

povlači da se mora pretpostaviti da će interakcija između komponenti biti ostvarena mrežnim protokolima ako se komponente ne nalaze u istoj izvedbenoj okolini.

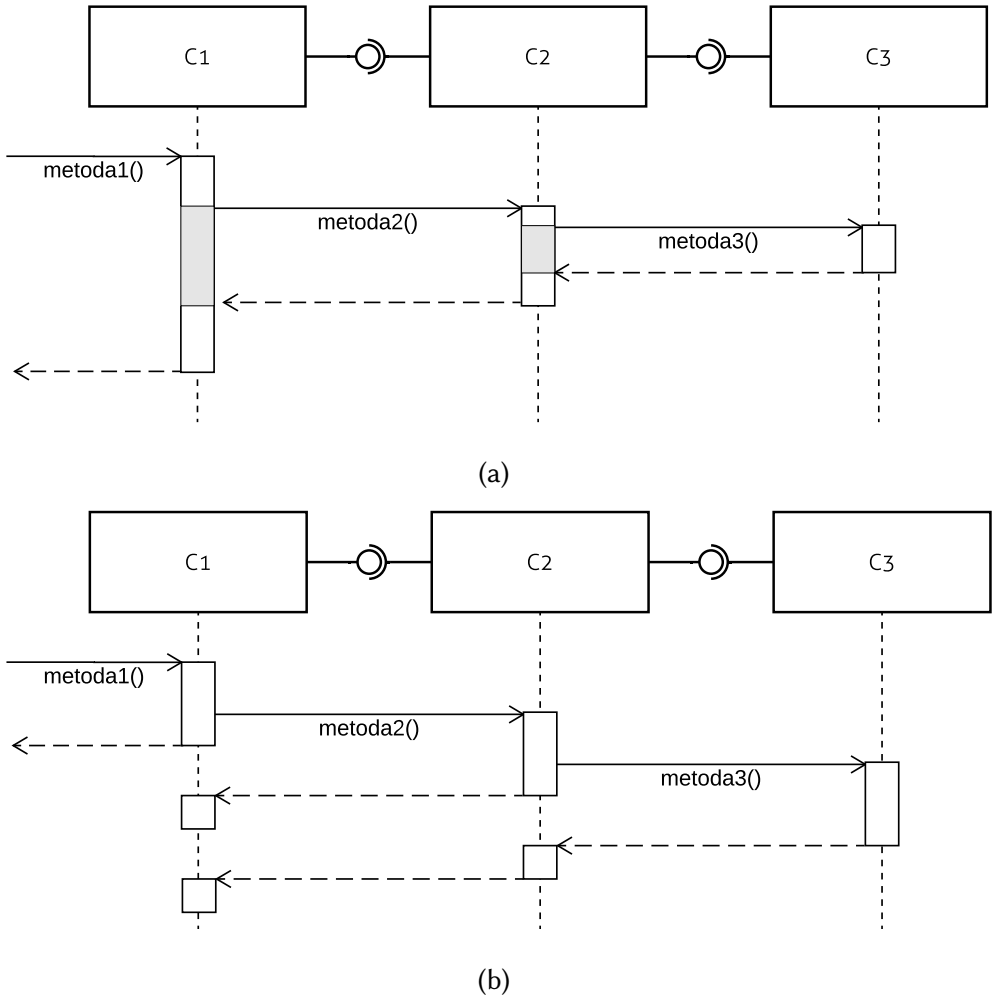
Veoma je važno sagledati implikacije koje nosi interakcija između komponenti u distribuiranom sustavu. Zanemarivanje takvih implikacija jedan je od glavnih razloga zbog čega komponentni model CORBA nije zaživio u potpunosti te se prešlo na korištenje mrežnih usluga [157]. Te implikacije mogu se podijeli na četiri osnovna problema: (1) latencija u komunikaciji, (2) pristup memorijskom prostoru, (3) parcijalni ispadi, (4) konkurentnost u pristupu. Svaku navedenu implikaciju potrebno je razmotriti prilikom dizajna sučelja komponenti i njezinih radnih karakteristika.

Latencija distribuirane komunikacije naspram lokalne jest 3-4 redova veličine veća što upućuje na potrebu za pažljivom organizacijom komponenti i međusobnih sučelja. Potrebno je strukturirati komponente u protuteži s oprečnim ciljevima: zadržati labavu spregu (engl. *loose-coupling*) između komponenti te istovremeno ostvariti koheziju između modula u sklopu iste komponente.

Pozivanje udaljenih metoda mora zadržavati proceduru oporavka ako komunikacija nije uspješno provedena. Sama procedura oporavka neće narušiti latenciju u komunikaciji jer se izvršava samo u slučajevima degradacije kvalitete komunikacijskog kanala. Negativan utjecaj koji ovo ograničenje nosi jest vrijeme prilikom dizajna sustava koje je potrebno odvojiti u osmišljavanju procedure oporavka i sagledavanju implikacija parcijalnih mrežnih ispada i ispada ostalih komponenti sustava.

Kako bi se pouzdano ponovila komunikacija između komponenti nakon ispada, potrebno je osigurati idempotentnost operacija [158]. Idempotentnost operacije označava njezinu sposobnost da, iako je na identičan način pozvana više puta, ne uzrokuje neželjene efekte. Primjerice, ako funkcija ostvaruje pohranu podataka u bazu podataka, višestruko pozivanje te funkcije za neki objekt mora rezultirati najviše jednom pohranom. Time konzistentnost sustava nije narušena kada je zbog problema u komunikacijskom kanalu potrebno operaciju izvršiti više puta.

Idempotentnost metoda koje sačinjavaju sučelje komponenti postavlja se kao uvjet u predloženoj arhitekturi, a dodatno poboljšanje u komunikaciji između komponenti može se ostvariti u vidu asinkrone komunikacije između komponenti. Asinkrona komunikacija doprinosi smanjenju vremena odaziva komponenti koje se oslanjaju na komunikaciju s drugim komponentama. Razlog tomu je što vrijeme obrade prilikom komunikacije s drugom komponentom se ne pribraja vremenu obrade pozivajuće komponente jer nije potrebno čekati odgovor što je ilustrirano na slici 3.1. Kod sinkrone komunikacije između komponenti (3.1a) ukupno vrijeme odaziva pribraja vremena odaziva svih operacija komponenti koje sudjeluju u obradi podataka, dok kod asinkrone komunikacije (3.1b) vrijeme odaziva pozivajućih operacija ne utječe na ukupno vrijeme odaziva. Asinkronu komunikaciju između komponenti nije moguće ostva-



Slika 3.1: Usporedba vremena odaziva kod sinkrone i asinkrone komunikacije između komponenti.

riti ako je informacija koja je isporučena pozivanjem daljnjih operacija potrebna u odgovoru na prvotni zahtjev te je često potreban redizajn operacija.

U ostvarenju asinkrone komunikacije mogu se koristiti komponente koje temelje komunikaciju na redovima čekanja [159].

Komponente za pohranu stanja i podataka

Komponente za pohranu podataka, najčešće sustav za upravljanje bazom podataka, nisu trivijalno skalabilne replikacijom. Zbog teorema CAP [160] nije moguće ispuniti sva potrebna svojstva pomoću raspodijeljenih komponenti za pohranu podataka. U skladu s navedenim teoremom postoje različite implementacije raspodijeljenih baza podataka koje rješavaju problem skalabilnosti različitim pristupima zavisno o željenom ishodu. Relacijske baze podataka ostvaruju konzistentnost podataka nauštrb dostupnosti. Baze podataka NoSQL najčešće rješavaju problem skalabilnosti i dostupnosti unutar mrežnih ispada, ali privremeno žrtvuju konzistentnost podataka. Pri većoj količini podataka, ostvarenje elastičnosti na podatkovnom

sloju nije efikasno. Razlog tomu jest potreba za migracijom veće količine podataka prilikom skaliranja na dodatni poslužitelj. Komponente na podatkovnom sloju kapacitirane prema najvećem očekivanom opterećenju što čini njihovu konfiguraciju statičnom. Samim time, kako ne postoji promjena konfiguracije uslijed promjene radnog opterećenja, optimizacija konfiguracije na podatkovnom sloju nije razmatrana.

3.3.3 Pogled na razmještaj

Razmještaj komponenti pridružuje programske komponente na poslužitelje infrastrukture u oblaku. To se pridruživanje definira pomoću tri preslikavanja:

1. Grupiranje programskih komponenti u skupinu koja ostvaruje elastičnost kapaciteta.
2. Odabir odgovarajućih vrsta poslužitelja za pojedinu elastičnu skupinu.
3. Dinamički razmještaj elastičnih skupina na poslužitelje u oblaku tijekom izvođenja zavisna o trenutnom opterećenju.

Konfiguracija, kao subjekt optimizacije u ovom istraživanju, formalno se definira kroz navedena tri pridruživanja. Samim time, na jedinstveni je način određen prostor svih mogućih konfiguracija kao prostor koji treba pretražiti prilikom optimizacije.

Konfiguracija sustava se definira kao usmjereni graf K sastavljen od tri podgrafa:

$$\mathrm{K} = G_C \cup G_{ES} \cup G_{TR} \cup G_A$$

gdje je G_C graf komponenti i spojki, G_{ES} graf podjele komponenti na elastične skupine, G_{TR} graf pridruživanja tipa resursa pojedinim skupinama, a G_A graf razmještaja elastičnih skupina na poslužitelje.

Graf komponenti i spojki se definira kao

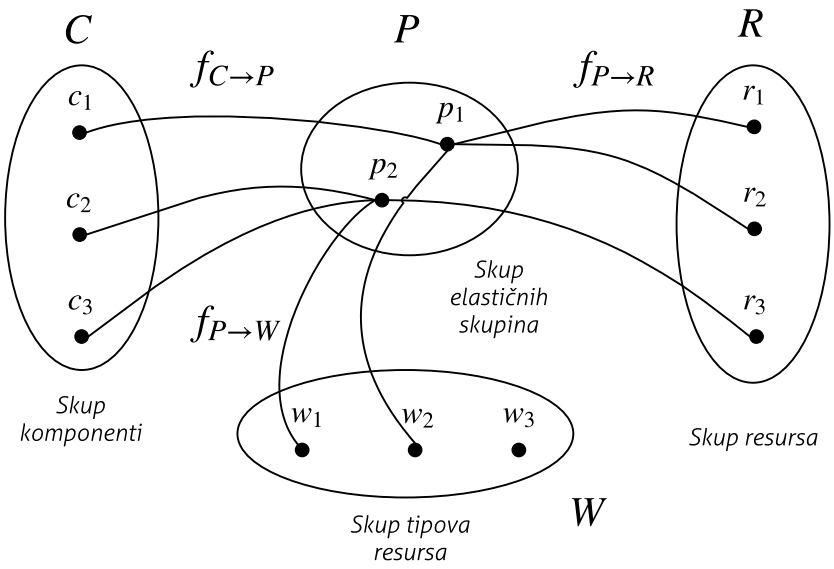
$$G_C = (C, E_C)$$

gdje C predstavlja skup dostupnih komponenti, a $E_C = \{(c_i, c_j) | i \neq j, c_i, c_j \in C\}$ predstavlja zavisnosti — spojke između komponenti.

Graf podjele komponenti na elastične skupine G_{ES} je bipartitni graf koji povezuje komponente s elastičnom skupinom kojoj pripadaju:

$$G_{ES} = (C, P, E_{CP})$$

gdje svaka komponenta $c \in C$ pripada nekom skupu $p \in P$, gdje je P particija skupa K. E_{CP} predstavlja pridruživanje komponenti iz C elementima iz P: $E_{CP} = \{(k, p) | \forall c \in C, \exists p \in P, p = f_{C \to P}(k)\}$, gdje je funkcija $f_{C \to P} : C \to P$ surjektivna funkcija koja označava da niti jedna komponenta ne može biti u više skupina $p \in P$.



Slika 3.2: Grafička reprezentacija funkcija $f_{C\to P}$, $f_{P\to R}$ i $f_{P\to W}$

Graf tipa resursa G_{TR} je bipartitni graf koji pridružuje elastičnim skupinama određeni tip resursa infrastrukture u oblaku:

$$G_{TR} = (P, W, E_{PW})$$

gdje je svakoj skupini $p \in P$ dodijeljen tip resursa iz W pomoću preslikavanja $E_{PW} = \{(p_i, w_j) | p_i \in P, w_j \in W, w_j = f_{P\to W}(p_i)\}$. Tipovi resursa zavise o ponudi davatelja usluge infrastrukture u oblaku u obliku različitih virtualnih strojeva. Tipično se razlikuju po količini procesorskih jedinica, radne memorije ili kapaciteta diskovlja. Elastična grupa kod autonomnog skaliranja koristi navedenu vrstu virtualnog stroja tako da je u svakom trenutku aktivna barem jedna instanca.

Graf razmještaja je također bipartitni graf koji povezuje elastične skupine s poslužiteljima infrastrukture u oblaku:

$$G_A = (P, R, E_{PR})$$

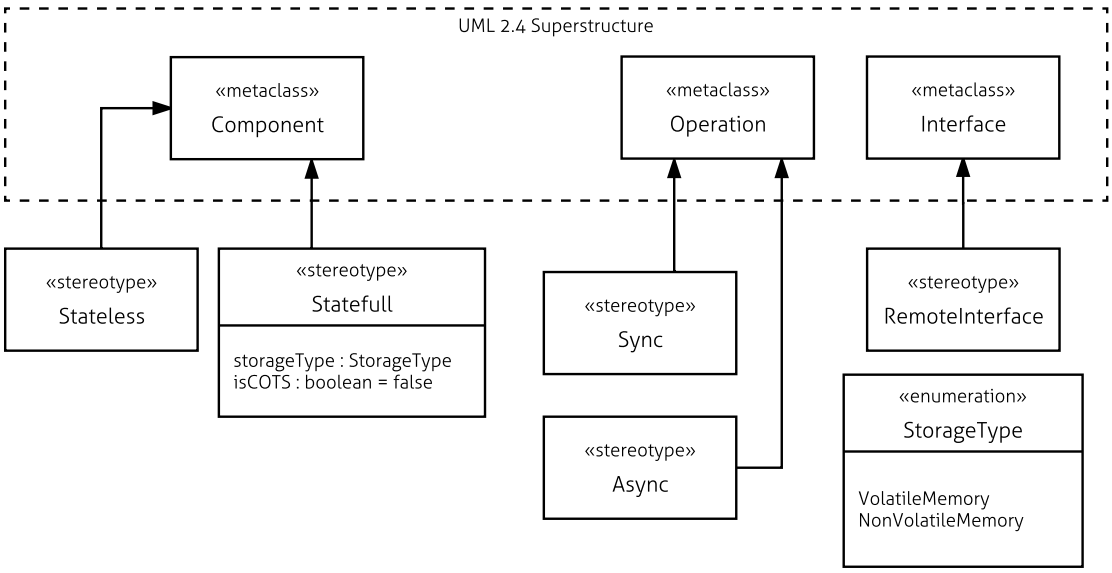
gdje je $\forall p \in P$ povezan sa $r \in R$ vezom $E_{PR} = \{(p, r) | \forall p \in P, \exists r \in R, p = f_{P\to R}(r)\}$. Funkcija $f_{P\to R}$ također je surjektivna. Funkcije $f_{K\to P}$ i $f_{P\to R}$ grafički su prikazane slikom 3.2.

Broj poslužitelja u svakoj skupini izveden funkcijom $f_{P\to R}$ dinamički se određuje tijekom izvođenja adaptacijom kapaciteta (poglavlje 2.4.3). Svakoj elastičnoj grupi potreban je zasebni upravljač koji brine o adekvatnoj kapacitiranosti skupine (poglavlje 2.4.4). Neovisno o broju dodijeljenih poslužitelja svakoj elastičnoj skupini, svaki pridodani poslužitelj sadrži sve komponente unutar elastične grupe.

3.4 Meta-model arhitekture

Arhitekturu sustava koji ostvaruje navedene zahtjeve definirat ćemo oblikovanjem meta-modela ostvarenog kao proširenje jezika UML putem profiliranja. Profiliranje kao metoda proširivanja jezika UML predviđeno je samom specifikacijom jezika [161]. Profiliranje služi kako bi se općeniti koncepti jezika UML koji su široko primjenjivi specificirali za korištenje u određenom kontekstu modeliranja. Instancijacija navedenog profila polučuje model arhitekture konkretnog sustava.

Slika 3.3 iznosi specifikaciju profila UML za modeliranje pogleda na komponente predložene arhitekture informacijskoga sustava. Specifikacija proširuje značenje meta klase *Component* te uvodi koncepte *Stateless* i *Stateful*. Komponente *Stateless* ne pohranjuju nikakvo stanje, a komponente *Stateful* omogućuju pohranu stanja u obliku podataka koji se pohranjuju trajno. Ako se radi o komponentama *Stateful*, moguće je eksplicitno navesti koji se oblik pohrane koristi zbog utjecaja na radne karakteristike aplikacije te daljnjeg modeliranja modela za predviđanje radnih karakteristika. Komponenta može ostvariti pohranu informacija koristeći sklopovlje koje zahtijevaju električnu energiju za pohranu podataka (engl. *volatile memory*) ili sklopovlje koje trajno pohranjuju podatke (engl. *non-volatile memory*). Razlika u brzini pristupa i propusnosti prema podacima između takvih sklopovlja može doseći i nekoliko redova veličine.



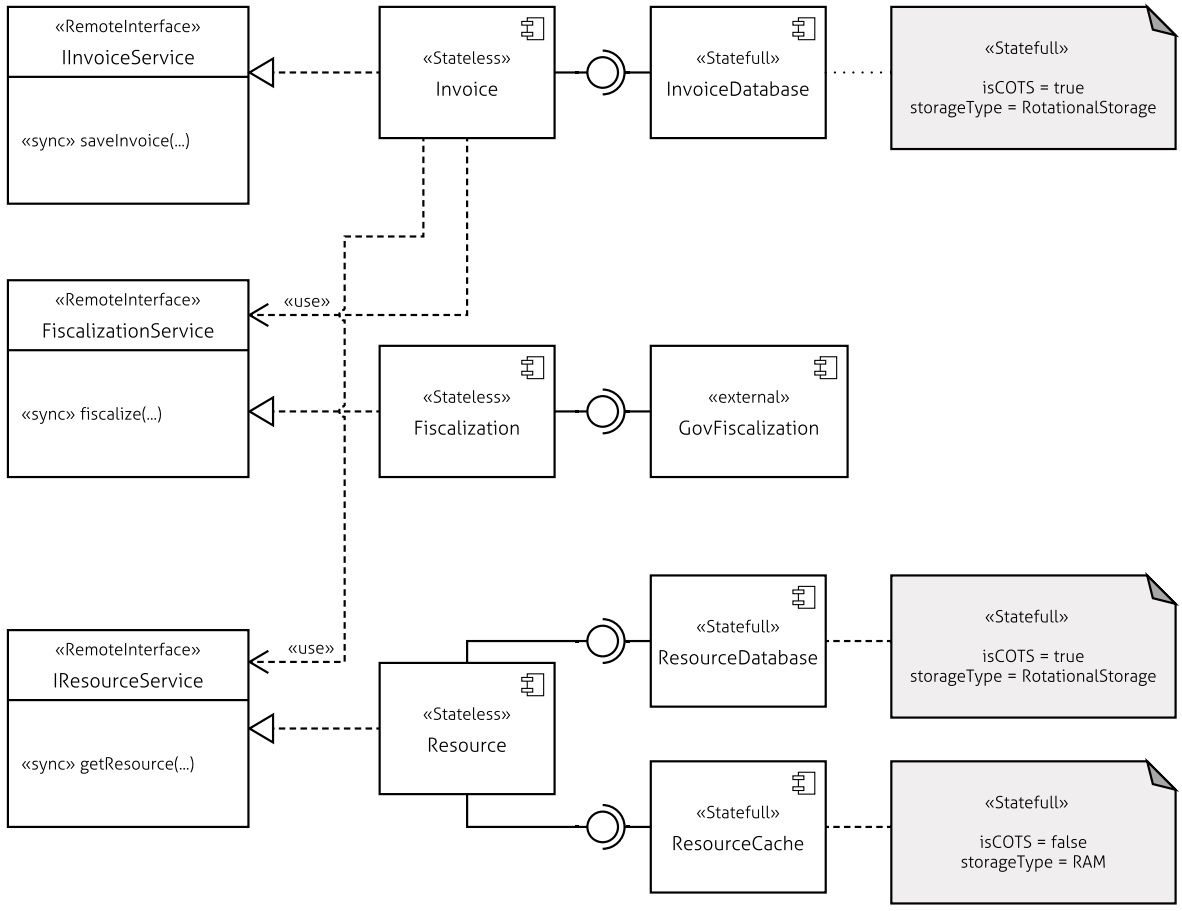
Slika 3.3: Profil jezika UML za modeliranje predložene arhitekture

Treća vrsta komponente pri ostvarenju sustava može biti mrežna usluga koja nije dio sustava, ali sudjeluje u ostvarivanju funkcionalnih zahtjeva sustava. Takve je mrežne usluge također potrebno navesti prilikom modeliranja zbog ulazno-izlaznog prometa koji se očekuje te zbog činjenica da za takvu uslugu nije potrebno osigurati infrastrukturu. Pritom se koristi

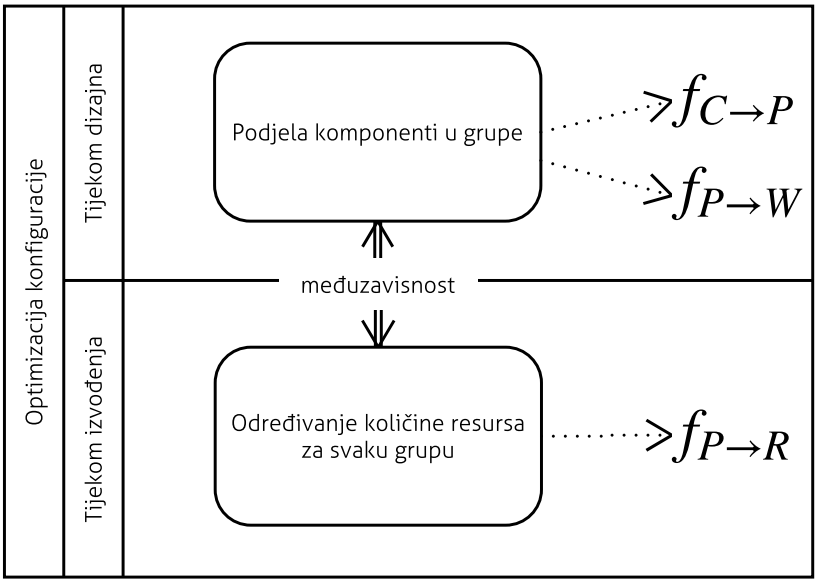
već dostupan stereotip *external* u sklopu definicije UML-a u inačici 2.4 te ga nije potrebno definirati profilom.

Za sučelje pojedinih komponente u sustavu potrebno je specificirati radi li se o sučelju namijenjenom udaljenom pristupu (*RemoteInterface*) zbog utjecaja na implementaciju same komponente i oporavka u slučaju ispada komunikacijskog kanala. Dodatno, za svaku operaciju koju komponenta ostvaruje pomoću takvog sučelja dodatno je potrebno navesti je li oblik komunikacije asinkron (*Async*) ili sinkron (*Sync*).

Slika 3.4 prikazuje primjer jedne instance predloženog profila jezika UML na aplikaciji korištenoj u studiji slučaja u poglavlju 7.4. Slika prikazuje tri ključne komponente aplikacije: *Invoice*, *Fiscalization* i *Resource* te njihove međusobne zavisnosti i način kako je ostvarena komunikacija.



Slika 3.4: Primjer modela prema predloženom profilu jezika UML koji prikazuje komponente iz studije slučaja



Slika 3.5: Određivanje optimalnih funkcija $f^*_{C\to P}$, $f^*_{P\to W}$ i $f^*_{P\to R}$ u dvije međusobno zavisne faze

3.5 Optimiranje konfiguracije

Različite konfiguracije predložene arhitekture ostvaruju različite radne karakteristike. Ovo poglavlje iznosi predloženu metodu optimizacije konfiguracije koja slijedi predloženu arhitekturu. Definira se optimizacijski problem i prostor svih mogućih kandidata. Kao dio funkcije cilja optimizacije oblikovan je i model troškova infrastrukture u oblaku.

3.5.1 Optimizacijski problem

Optimizacijski problem definira se po konkretnom modelu arhitekture M aplikacije A_M nastalog instanciranjem predloženog meta-modela MM iz poglavlja 3.4. Konkretni sustav moguće je ostvariti različitim konfiguracijama koje slijede iz modela M, a zajednički ćemo ih prikazati skupom $\mathrm{K^M}$. Cilj optimizacije je pretražiti prostor kandidata iz $\mathrm{K^M}$ koristeći jednu ili više željenih funkcija cilja $\Phi_q : \mathrm{K^M} \to V_q$ gdje je V_q uređeni skup mogućih vrijednosti određenog kriterija kvalitete q. Skup V_q mora biti strogo uređen skup s antisimetričnom, tranzitivnom i potpunom binarnom relacijom definiranom nad V_q tako da je moguća direktna usporedba između svih vrijednosti $v \in V_q$.

Optimizacija konfiguracije dijeli se u dva međuzavisna koraka (Slika 3.5): optimalna podjela komponenti u elastične skupine (gdje postoji više optimalnih funkcija $f^*_{C\to P}$) koje je ujedno uvjetovana kontinuiranom optimizacijom potrebnog broja poslužitelja po svakoj skupini, određena funkcijom $f^*_{P\to R}$, što karakterizira elastično ponašanje svake skupine. U ovome se odražava doprinos predložene metode zbog toga što se provodi optimizacija nad sustavom čije sastavnice posjeduju mogućnost elastičnosti u zavisnosti od opterećenja.

Optimizacija konfiguracije koja se provodi prilikom sastavljanja sustava svodi se na pro-

blem odabira optimalnog rasporeda komponenti u elastične skupine određivanjem preslikavanja $f_{C\to P}$ i $f_{P\to W}$ te se može formulirati kao optimizacijski problem. Problem određivanja optimalnog broja poslužitelja po svakoj skupini svodi se na problem dinamičkog kapacitiranja sustava koji zavisi o ukupnom opterećenju svih komponenti u elastičnoj skupini i njihovim radnim karakteristikama.

Prostor svih mogućih konfiguracija koje sustav može ostvariti sačinjavaju sve moguće kombinacije grupiranja komponenti tako da svaka komponenta može biti najviše u jednoj skupini, dok broj komponenti po svakoj skupini nije ograničen, ali skupina ne može biti prazna. Broj mogućih konfiguracija $\mathrm{K^M}$ za skup svih komponenti K može iskazati pomoću *Strilingovog broja druge vrste* [162] $S(n,k)$ koji određuje na koliko je načina moguće podijeliti n elemenata u k skupina.

$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^j\binom{k}{j}(k-j)^n \tag{3.1}$$

Prostor pretrage predstavljaju sve moguće konfiguracije gdje broj skupina može varirati od jedne, ako su sve komponente u istoj skupini, do $n = |K|$ skupina, gdje je svaka komponenta u zasebnoj skupini. Samim time, broj mogućih konfiguracija $\mathrm{K^M}$ određen je sa:

$$\left|\mathrm{K^M}\right| = \sum_{k=1}^{n} S(n,k)\cdot k^{|W|} \quad , \quad n = |K|\,. \tag{3.2}$$

gdje je W skup svih vrsta poslužitelja koje je moguće koristiti u konfiguracijama.

Već i za manji broj komponenti, radi se o velikom broju mogućih rješenja zbog čega je potrebno koristiti metode optimizacije koje mogu pretražiti prostor rješenja na efikasan način [163]. U tu svrhu koristi se pretrage prostora rješenja koji ostvaruju rezultate blizu optimalnih, no u puno kraćem vremenu. Dodatan izazov predstavlja računska složenost funkcije cilja koja je ostvarena u dvije inačice: (1) pomoću računalne simulacije i (2) konkretnog mjerenja kriterija kvalitete nad stvarnim sustavom.

3.5.2 Model troškova infrastrukture

Važan dio funkcije cilja kod optimizacija predstavlja model troškova. Model troškova (engl. *cost model*) koristi se tijekom ocjene kandidata i zasniva se na modelu kojeg koristi većina davatelja infrastrukture javnog oblaka. Model pretpostavlja virtualne poslužitelje — resurse r od kojih je svaki određenog tipa $type(r) = w \in W$, a svaki tip resursa w ima pripadnu jediničnu cijenu $U_{COST}(w)$. Naplata se vrši po duljini najma u definiranim intervalima korištenja T_{cost}. Ukupno trošak zakupa resursa r u trajanju od t vremenskih jedinica određen je sa

$$COST(r,t) = \lceil \frac{t}{T_{cost}} \rceil * U_{COST}(w) \quad .$$

Različiti tipovi $w \in W$ pretpostavljaju poslužitelje s unaprijed određenom količinom procesorskih jezgri i radne memorije što ovisi o odabiru davatelja usluge.

3.5.3 Formulacija optimizacijskog problema

Optimizacijska metoda koju ćemo koristiti mora posjedovati određena svojstva specifična za problem iz konteksta ovog istraživanja. Potrebno je odabrati metodu koja omogućuje optimiranje po više kriterija kvalitete koji su često oprečni. U zavisnosti kako korisnik vrednuje različite kriterije postoje tri strategije kako ostvariti takvu optimizaciju [164]:

- *a priori* metoda gdje korisnik unaprijed definira preferencije prema svakom kriteriju te je samim time moguće objediniti ih u jedinstvenu funkciju cilja; time je moguće primijeniti klasične metode za optimizaciju s jednom funkcijom cilja;
- *a posteriori* metoda koja pronalazi skup Pareto optimalnih rješenja [135] gdje za svako rješenje u skupu vrijedi da ne postoji nijedno drugo rješenje koje je po jednom kriteriju bolje, a da pritom po nekom drugom kriteriju nije lošije; korisnik zatim sam odabire željeno rješenje iz Pareto skupa.

Nedostatak *a priori* metoda je česta nedovoljna informiranost korisnika koji donosi odluku oko odnosa između različitih kriterija koje je moguće ostvariti, zbog čega se *a posteriori* metoda smatra poželjnijom. Upravo se ova metoda primjenjuje u ovom istraživanju.

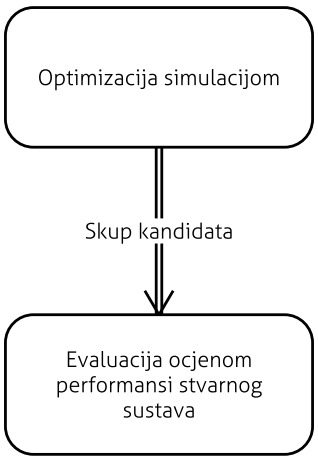
Određivanje vrijednosti za pojedine funkcije cilja u kontekstu ovog istraživanja znači odrediti sve promatrane kriterije kvalitete mogućih konfiguracija sustava. Prilikom određivanja kriterija kvalitete određenih radnim karakteristikama i troškovima izvođenja, moguće su dvije različite metode u njihovom određivanju [165]:

- korištenje prediktivnih modela [166] gdje se radne karakteristike određuju postupcima simulacijskim ili analitičkim postupcima
- mjerenjem nad stvarnim sustavom [167]

Mjerenje nad stvarnim sustavom polučuje realne rezultate, ali je vremenski zahtjevno. Kod velikih sustava u oblaku, upitna je i efikasnost zbog velikih troškova infrastrukture tijekom izvođenja testova. S druge strane, korištenje prediktivnih modela polučuje rezultate koji su određeni prediktivnom snagom i kvalitetom korištenog modela gdje je u slučaju modeliranja radnih karakteristika teško postići visoku preciznost zbog toga što na radne karakteristike utječe gotovo svaki faktor tijekom razvoja i izvođenja programske podrške.

Zbog velikog prostora pretrage i spore evaluacije kriterija metodom mjerenja nad stvarnim sustavom, korišteno je hibridno rješenje. U prvoj fazi optimizacije koristi se brža evaluacijska funkcija pomoću modela sustava manje preciznosti, pomoću koje se određuje prijedlog rješenja u obliku Pareto skupa konfiguracija. Potom se u drugoj fazi predloženi kandidati evaluiraju pomoću sporije i preciznije evaluacijske funkcije. Te dvije etape optimizacije prikazane su slikom 3.6:

- optimizacija gdje se za evaluacijsku funkciju koristi metoda simulacije
- optimizacija ocjenom performansi nad stvarnim sustavom.



Slika 3.6: Određivanje optimalne funkcije f^*_{KP} u dvije etape

Optimizacijski problem koji treba riješiti definiramo kao:

$$\begin{aligned} &\text{minimize} && fq_1(k_i^M), fq_2(k_i^M), ..., fq_n(k_i^M) \\ &\text{subject to} && k_i \in K^M \end{aligned}$$

gdje za funkcije fq_j vrijedi:

$$fq_j(k_i^M) = \begin{cases} -1 * q_j(k_i^M) & \text{ako za kriterij } q_j \text{ vrijedi da su veće vrijednosti poželjne,} \\ q_j(k_i^M) & \text{u suprotnom} \end{cases}$$

Tablica **??** pojašnjava korištenu notaciju koja se koristi u ostatku rada.

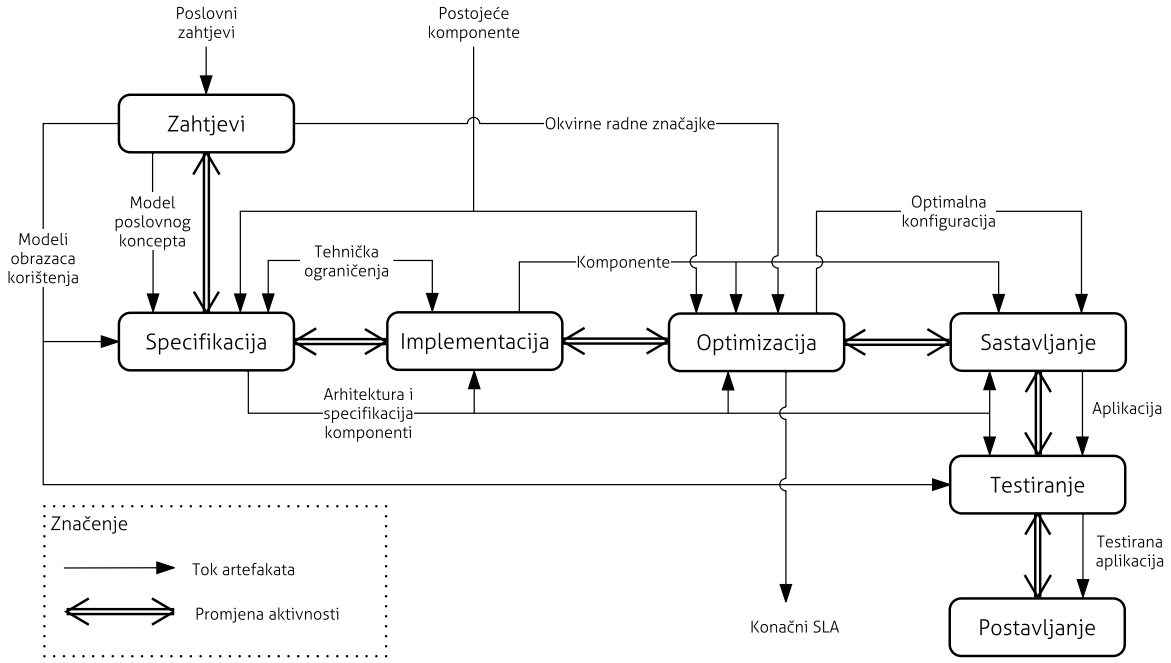
Tablica 3.1: Korištena notacija

MM	$\stackrel{\text{def}}{=}$	Komponentni meta-model predstavljen u poglavlju 3.4
M	$\stackrel{\text{def}}{=}$	Komponentni model određene aplikacije, instanca modela MM
C^M	$\stackrel{\text{def}}{=}$	Skup komponenti u modelu M, koje je potrebno grupirati
c_i^M	$\stackrel{\text{def}}{=}$	Određena komponenta iz skupa C^M
K^M	$\stackrel{\text{def}}{=}$	Skup svih konfiguracija određenih modelom M
k_i^M	$\stackrel{\text{def}}{=}$	Određena konfiguracija iz K^M
Q	$\stackrel{\text{def}}{=}$	Skup odabranih kriterija kvalitete
q_i	$\stackrel{\text{def}}{=}$	Određeni kriterij kvalitete
$q_i(k_j^M)$	$\stackrel{\text{def}}{=}$	Vrijednost određenog kriterija kvalitete za konfiguraciju $k_j^M \in K^M$
$\mathrm{K_Q^{M*}}$	$\stackrel{\text{def}}{=}$	Skup Pareto optimalnih konfiguracija po kriterijima Q modela M
$\mathrm{k_{q_i}^{M*}}$	$\stackrel{\text{def}}{=}$	Optimalna konfiguracija po kriteriju q_i modela M

3.6 Model razvojnog procesa zasnovan na predloženoj arhitekturi

Ovo poglavlje iznosi prijedlog modela razvojnog procesa koji se može primijeniti u implementaciji sustava koji slijede prethodno predstavljenu arhitekturu i metodu optimizacije. Predloženi model je nadogradnja modela predstavljenog u poglavlju 2.1.4. Odlika predložene arhitekture jest odgađanje odluka o konkretnoj konfiguraciji za kasnije faze implementacije, gdje sama arhitektura podržava dinamičku rekonfiguraciju, čime se ne narušuje funkcionalnost sustava, već se utječe na kriterije kvalitete. Time je postignuta bolja odijeljenost između različitih faza procesa koji se bave realizacijom funkcionalnih i nefunkcionalnih zahtjeva. Prilikom dizajna sustava naglasak je na ostvarenju funkcionalnosti i specifičnosti raspodijeljenih sustava dok je prilikom optimizacije naglasak na udovoljavanju sporazuma o razini usluge kroz optimalnu konfiguraciju.

Za razliku od većine postojećih procesa optimizacije koji se koriste u ranoj fazi implementacije [166], predstavljeni proces optimizaciju provodi se u kasnijoj fazi, nakon što su komponente sustava već implementirane. Razlog tome jest automatsko izlučivanje modela sustava prema stvarnim mjerenjima vremena odaziva kod različitih operacija koje komponente pružaju te generiranje simulacijskih modela sukladno takvim mjerenjima. Prethodna istraživanja ukazuju na potrebu ovakvog pristupa [165].



Slika 3.7: Model procesa razvoja sustava zasnovanog na komponentama uz optimizaciju konfiguracije

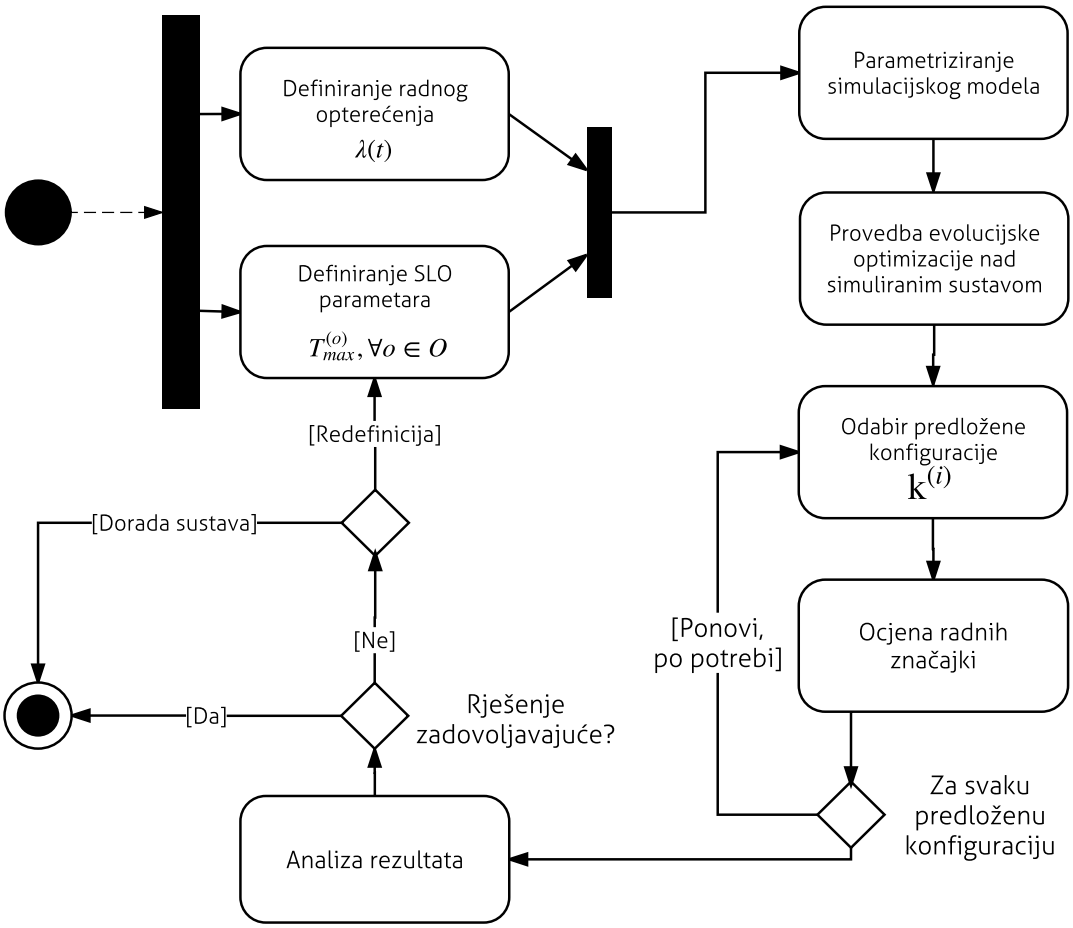
Nadogradnja procesa iz poglavlja 2.1.4 predstavlja proces optimizacije smješten između procesa implementacije i sastavljanja. Proces je prikazan na slici 3.7. Proces podrazumijeva već implementirane komponente jer se nad komponentama mora provesti inicijalno mjerenje potrebno za konstruiranje modela korištenog u procesu optimizacije metodom simulacije. Rezultat procesa optimizacije jest prijedlog ugovora SLA uz odabranu optimalnu konfiguraciju. Prema optimalnoj konfiguraciji prelazi se u proces sastavljanja gdje se navedena konfiguracija realizira u produkcijskom okruženju. Sam proces optimizacije zahtijeva implementirane i postojeće komponente sustava te model radnog opterećenja. Prilikom daljnje evolucije sustava, poželjno je model opterećenja izvesti prema stvarnom korištenju sustava.

Poglavlje 4

Metoda ocjene i odabira konfiguracije sustava u skladu sa sporazumom o razini usluge

Ovo poglavlje iznosi metodu za optimiranje konfiguracije aplikacije koja slijedi arhitekturu predloženu u poglavlju 3. Metoda naglašava razumijevanje utjecaja konfiguracije na radne karakteristike koji se naposljetku očituju u zadovoljavanju sporazuma o razini usluge. Metoda se primjenjuje od strane davatelja programske usluge u zadovoljavanju ili određivanju ugovora SLA.

Slika 4.1 prikazuje glavne korake predložene metode. Metoda uključuje definiranje radnog opterećenja po modelu predstavljenom u poglavlju 4.1. Pomoću navedenog modela ostvarena je komponenta za generiranje sintetičkog opterećenja. Drugi važan korak metode jest definiranje parametara ugovora SLA po modelu predstavljenom u poglavlju 4.2. Određivanje prostora mogućih konfiguracija i ocjena radnih značajki za odabranu predloženu konfiguraciju opisana je u poglavlju 4.3. Parametriziranje simulacijskog modela i provedba evolucijske optimizacije služe pri ubrzanju postupka optimizacije te su opisani u poglavlju 5.



Slika 4.1: Metoda određivanja prikladne konfiguracije

4.1 Simuliranje radnoga opterećenja

Postojeće metode ocjene radnih karakteristika sustava ne pružaju dovoljnu razinu podrške u generiranju promjenjivog radnog opterećenja. Alati za generiranje radnog opterećenja pružaju ograničene mogućnosti statičnog određivanja intenziteta opterećenja. Ovo poglavlje opisuje predloženi postupak za modeliranje i generiranje promjenjivog radnog opterećenja.

4.1.1 Model radnoga opterećenja

Cilj modela opterećenja je pojednostavniti specificiranje i generiranje sintetičkog radnog opterećenja u svrhu evaluacije pojedinih konfiguracija. Model mora omogućiti reprezentaciju radnog opterećenja koje udovoljava sljedećim zahtjevima:

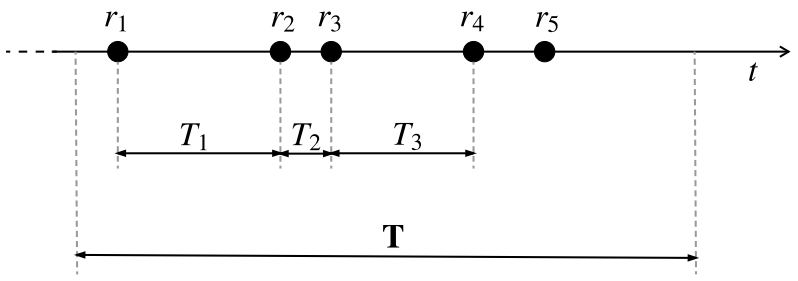
1. (Z1) promjenjivost u intenzitetu radnog opterećenja,
2. (Z2) nasumičnost pojedinih trenutaka dolazaka zahtjeva i
3. (Z3) postojanje više vrsta zahtjeva — ostvarivanje mješovitog radnog opterećenja (engl. *workload mix*) [69].

Slika 4.2 prikazuje primjer pojedinih zahtjeva r_i prikazanih na vremenskom pravcu t. Sa T_i je označeno vrijeme između dva zahtjeva (vrijeme međudolaska). Prema zahtjevu (Z2) mora vrijediti da je vrijeme između pojedinih dolazaka zahtjeva T_i slučajna varijabla (engl. *random variable*), dok prema (Z1) mora vrijediti da očekivani broj zahtjeva u određenom intervalu $\mathbf{T} = [t, t+\Delta t)$ bude određen funkcijom intenziteta međudolaska $f_I(t)$:

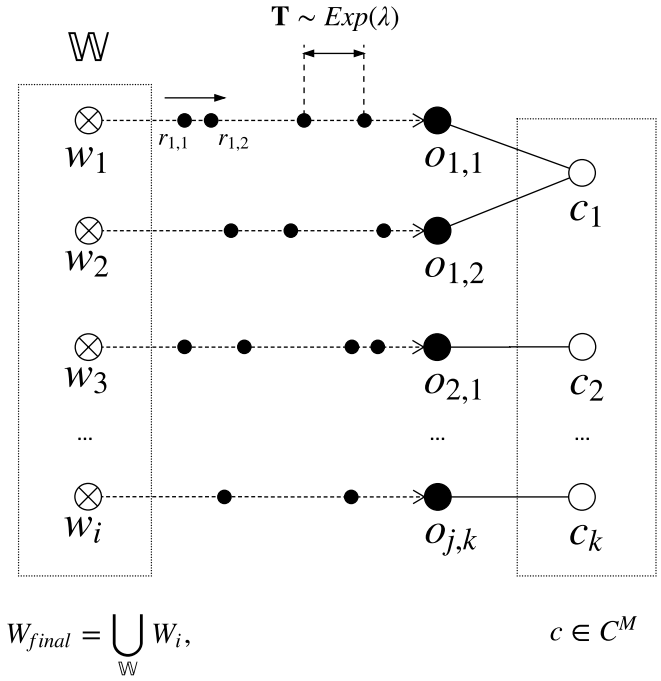
$$\mathbb{E}[N(t+\Delta t) - N(t)] = f_I(t)$$

gdje je $N(t)$ ukupni broj pristiglih zahtjeva do trenutka t. Tako definirani nasumični proces označit ćemo sa w.

Model predloženog radnog opterećenja razlikuje više vrsta zahtjeva gdje svaka vrsta zahtjeva označuje poziv određene operacije o koju je ostvarena pripadnom komponentom c. Sukladno tome, definirat ćemo n nasumičnih procesa $w_1, w_2, \ldots, w_n$ za n vrsti zahtjeva upućenim komponentama $c \in C^M$. Zajednički nasumični proces W_{final} predstavlja uniju procesa w_i čime



Slika 4.2: Trenuci dolazaka zahtjeva prema usluzi



Slika 4.3: Model radnog opterećenja koji se sastoji od više nasumičnih procesa, svaki proces označuje pozive prema jednoj operaciji komponente

je ispunjen (Z3):

$$W_{final} = \bigcup_{\mathbb{W}} w_i,$$

U najjednostavnijem slučaju možemo procese $w \in \mathbb{W}$ modelirati kao Poissonove procese [168] što znači da vremena između međudolaska T slijede eksponencijalnu distribuciju s parametrom λ,

$$T \sim Exp(\lambda),$$

gdje vrijedi $\mathbb{E}[N(t)] = \lambda t$ što znači da je broj zahtjeva u nekom intervalu t proporcionalan parametru λ.

Ako procese $w \in \mathbb{W}$ modeliramo kao Poissonove procese, tada je i konačni model radnog opterećenja također Poissonov proces po principu superpozicije Poissonovih procesa [169]. Ta činjenica olakšava implementaciju generatora opterećenja jer nije potrebna sinkronizacija između razdijeljenih komponenti.

Poissonov proces s parametrom λ zadovoljava zahtjev (Z1). Kada konstantu λ zamijenimo funkcijom $f_I(t)$ dobivamo nehomogeni Poissonov proces s parametrom intenziteta koji je određen vremenskom komponentom t. Tada za broj očekivanih zahtjeva u nekom vremenskom intervalu $\mathbf{T} = [t, t + \Delta t]$ vrijedi:

$$\mathbb{E}[N(t+\Delta t) - N(t)] = \int_t^{t+\Delta t} f_I(t)dt,$$

dakle određen je funkcijom $f_I(t)$ što ispunjava zahtjev (Z2). Radi konzistencije s literaturom koja opisuje nehomogeni Poissonov proces, u nastavku se funkcija $f_I(t)$ označava kao $\lambda(t)$.

4.1.2 Generiranje radnog opterećenja

U nastavku se iznosi algoritam za generiranje zahtjeva pomoću modela predstavljenog u poglavlju 4.2 i željenog intervala T. Varijabla U predstavlja slučajnu varijablu kontinuirane uniformne distribucije na intervalu $(0,1)$. Pretpostavka algoritma jest da postoji implementacija slučajne varijable U, što je ispunjeno za većinu programskih okruženja tehnikama generiranja pseudoslučajnih brojeva [170]. Vektor $P = (p_1, p_2, ..., p_n)$ predstavlja vjerojatnosti pojedinih tipova zahtjeva $r_1, r_2, r_3, ..., r_n$ upućenih prema programskoj usluzi, n označava ukupni broj tipova zahtjeva, $\lambda(t)$ predstavlja funkciju promjenjivog intenziteta radnog opterećenja, $S(1), ..., S(I)$ predstavlja trenutke kada treba uputiti zahtjeve, a $R(1), ..., R(I)$ predstavlja tip zahtjeva koji treba uputiti u odgovarajućim trenucima S. Broj zahtjeva u intervalu T označen je sa I.

Algoritam 1 Algoritam za generiranje zahtjeva prema programskoj usluzi

```
1:  procedure GenerirajZahtjeve(P)
2:      t ← 0, I ← 0, n ← |P|
3:      u_1 ← U                                  ▷ Generiraj nasumični broj
4:      λ_1 ← λt
5:      t ← t + (−1/λ) log(u_1)
6:      S(I) ← t
7:      u_2 ← U                                  ▷ Generiraj nasumični broj
8:      s ← 0
9:      for i ← (0, n − 1) do                    ▷ Određivanje tipa zahtjeva
10:         s ← s + p_i
11:         if u_2 ≤ s then
12:             R(I) ← i
13:             break
14:         end if
15:     end for
16:     return (S, R)
17: end procedure
```

4.2 Model ugovora SLA

Sukladno ciljevima ovog istraživanja, predložena metoda istražuje odabir prikladne konfiguracije programske usluge u postizanju dva oprečna atributa kvalitete: zadovoljavajućih radnih značajki i troškova izvođenja. Radne značajke poput propusnosti usluge ili vremena odaziva često su obrnuto proporcionalne količini korištene infrastrukture, u skladu s čime su definirani troškovi izvođenja usluge. No to ne mora nužno biti tako. Primjerice, ako se poveća količina resursa za komponente koji ne sačinjavaju usko grlo usluge, ne mora nužno doći do povećanja kvalitete u vidu radnih značajki. Glavna prednost predložene metode u odnosu na dosadašnje metode optimiranja jest uključivanje promjenjivoga intenziteta radnoga opterećenja te postojanje elastičnosti komponenata programske usluge kao odgovora promjenjivom opterećenju.

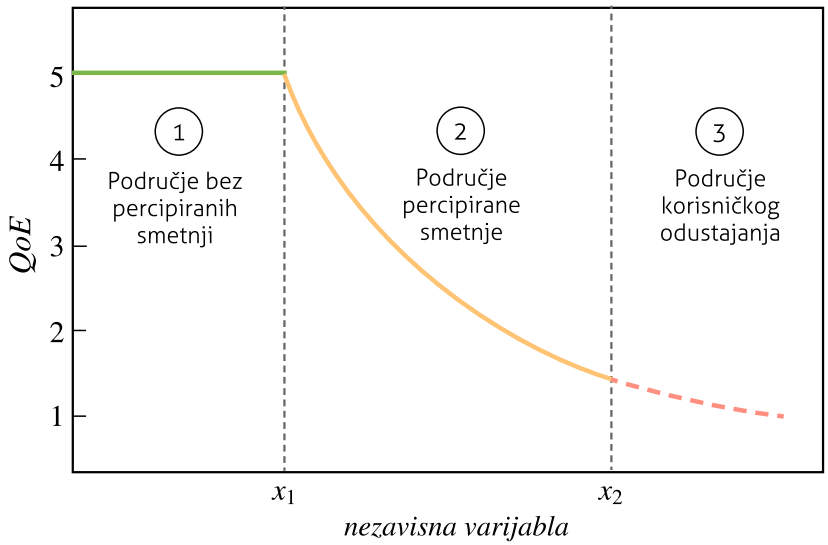
4.2.1 Metrike sporazuma

Gledano sa stanovišta korisnika usluge prema kojem se sklapa SLA, dva najčešća kriterija kvalitete koji se vežu uz radne značajke jesu vrijeme odaziva i propusnost usluge. Vremenska komponenta radnih značajki predstavlja glavnu karakteristiku koja utječe na percepciju korisnika prema usluzi. Usluga koja brže odgovara na podražaje krajnjem se korisniku čini prirodnijom i kvalitetnijom [171, 172, 173]. Također, korisnik koncipira kvalitetu usluge subjektivnim mjerilima te postoji nelinearna veza između korisnički percipiranog i objektivnog kriterija kvalitete. Slika 4.4 prikazuje općeniti primjer povezanosti između izmjerenog kriterija kvalitete i korisničke percepcije [172]. Važno je prepoznati tri različita područja:

- *područje konstantne korisničke percepcije kvalitete* gdje postoji gornja granica kriterija kvalitete nakon koje korisnik više ne raspoznaje daljnje poboljšanje
- *područje eksponencijalnog opadanja korisničke percepcije kvalitete*
- *područje odustajanja* gdje korisnik odustaje od daljnje uporabe usluge ako se pojedini kriterij kvalitete usluge spusti ispod donje dopuštene granice.

Sukladno takvoj nelinearnoj vezi, važno je sagledati ukupno vrijeme odaziva sustava koje sačinjavaju pojedina vremena odaziva programskih usluga te osigurati odgovarajuća vremena koja neće narušiti korisničku percepciju. Mjerenje poželjnih vremena odaziva nije cilj ovog istraživanja, već se polazi od početne pretpostavke da su vremena odaziva unaprijed zadana.

U suvremenim sustavima zasnovanim na mrežnim uslugama (engl. *service-based system*) pojedina usluga može se komponirati od niza drugih usluga čime vrijeme odaziva pojedine usluge u lancu usluga postaje još važnije [171]. Najčešća metrika kojom se definira vrijeme odaziva usluge jest *prosječno vrijeme odaziva*. Međutim, prosječno vrijeme odaziva ne odražava dovoljno dobro općenito vezu korisničke percepcije prikazanu slikom 4.4. Na slici 4.5 prikazana je distribucija vremena odaziva za odabranu operaciju iz studije slučaja gdje se



Slika 4.4: Nelinearna veza između objektivnog i subjektivnog kriterija kvalitete

očituje velika razlika između prosječnih vrijednosti i pojedinih percentila. Potrebna je metrika koja garantira da kriterij kvalitete ne padne ispod poželjne gornje granice na dovoljno visokom postotku svih korisničkih zahtjeva.

Metrika koja udovoljava tako postavljenim zahtjevima jest *P-percentil* vremena odaziva pojedinih operacija koje pruža programska usluga.

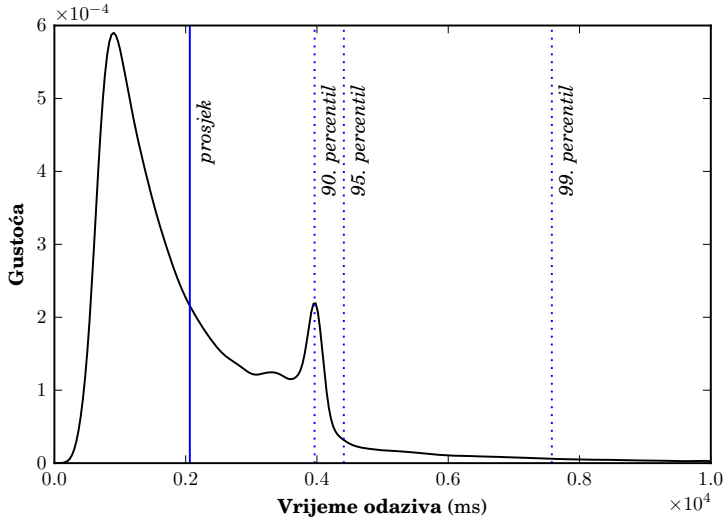
Definicija 4.1 P-percentil vremena odaziva

P-percentil vremena odaziva jest vrijednost vremena odaziva unutar koje sustav isporučuje P % svih upućenih zahtjeva prema usluzi.

P-percentil metrika često je korištena u mrežnim tehnologijama za kvantificiranje količine kašnjenja paketa koji prometuju mrežom, a definirana je normama MEF 10.3 [174] i IETF RFC 7679 [175]. P-percentil je metrika koja je relevantnija od prosječne vrijednosti zbog toga što je razdioba vremena odaziva ukošena udesno (engl. *right-skewed distribution*) često s *teškim repovima* (engl. *heavy-tailed distribution*) [176]. P-percentil omogućuje projektiranje sustava uzimajući u obzir statističku vjerojatnost scenarija u kojima je vrijeme odziva prihvatljivo, što je posebno bitno u kompoziciji mrežnih usluga [177].

P-percentil kao sastavnica ugovora SLA

Ako za određenu operaciju programske usluge označimo vrijeme odaziva sa T, pripadnu razdiobu vremena odaziva sa $f_T(t)$, a željeni P-percentil ($\tau_{\%}$) za ciljano vrijeme odaziva sa



Slika 4.5: Primjer odnosa između prosječnog vremena odaziva i različitih P-percentila vremena odaziva. Vrijednosti su prikupljene na stvarnoj programskoj usluzi iz studije slučaja iz poglavlja 7.4

t_{P-max}, tada možemo izračunati postotak zahtjeva čije vrijeme odaziva premašuje t_{P-max}:

$$viol\% = \int_{t_{P-max}}^{\infty} f_T(t)\mathrm{d}t \cdot 100\% \quad , \tag{4.1}$$

gdje radi udovoljavanja sporazuma SLA mora vrijediti:

$$viol\% \leq \tau_{\%} \quad . \tag{4.2}$$

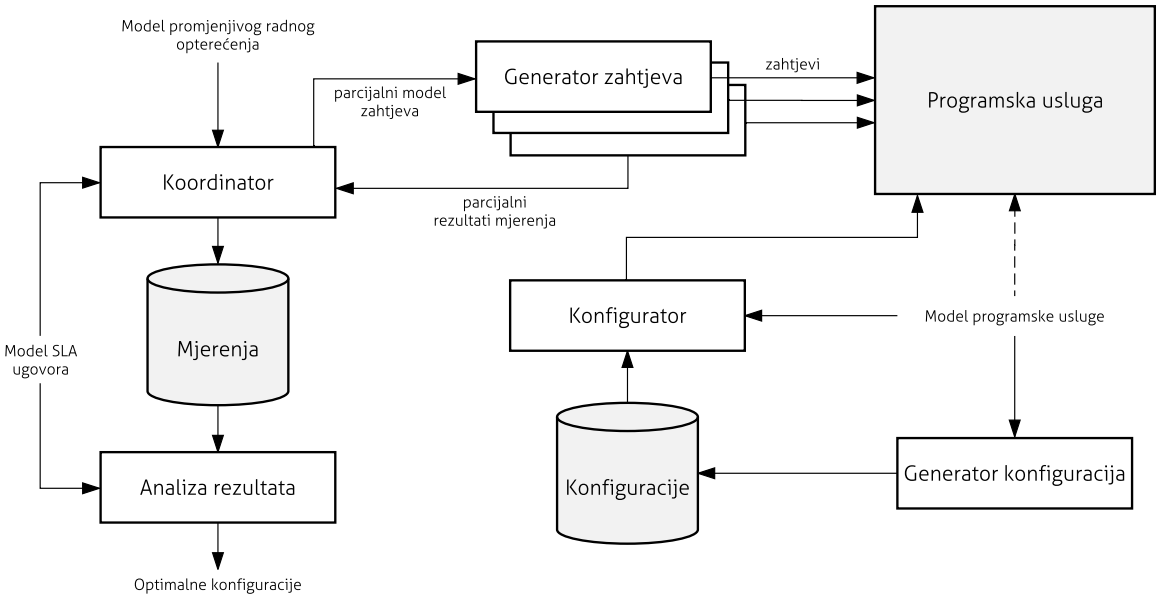
P-percentil vremena odaziva može se definirati nad svakom operacijom o programske usluge na određenom mjernom intervalu T^m. Primjerice, za slučaj gdje je T^m postavljen na sat vremena, svaki sat za koji nije zadovoljena relacija (4.2) smatra se prekršajem ugovora te se primjenjuju adekvatni penali.

4.3 Metoda generiranja i ocjene konfiguracija

U ovom poglavlju predlaže se metoda za generiranje mogućih konfiguracija pojedinog modela arhitekture M te ocjena pojedine konfiguracije u ostvarivanju željene razine usluge zadane SLA-om. Iznose se potrebni dijelovi metode kao i detaljniji opis svakog dijela.

Glavne komponente za provedbu ocjenu konfiguracije prikazane na slici 4.6 jesu:

1. *Koordinator* u izvršavanju testova koji upravlja instanciranjem odgovarajućeg broja generatora sintetičkog radnog opterećenja te upravljanje konfiguracijama programske usluge. Dodatno, kombinira i pohranjuje rezultate testiranja sakupljene od strane više generatora zahtjeva.
2. *Konfigurator* postavlja programsku uslugu u jednu od mogućih konfiguracija.



Slika 4.6: Postupak optimizacije konfiguracije

3. *Generator konfiguracija* određuje moguće konfiguracije programske usluge prema njezinom modelu.
4. *Generator zahtjeva* upućuje zahtjeva prema usluzi i mjeri vrijeme odaziva.

4.3.1 Generiranje mogućih konfiguracija

Ovo poglavlje iznosi postupak kojim se određuju sve moguće konfiguracije K^M pomoću seta komponenti $c \in C^M$ određenoga modela informacijskoga sustava M koji slijedi metamodel MM definiran u poglavlju 3.4.

Postupak generiranja mogućih konfiguracija odvija se u dvije faze.

U prvoj fazi određuju se moguće grupe komponenti i njihove kardinalnosti u obliku skupa $\widehat{K}$ koji sadrži višekratne skupove (engl. *multi-set*) $\widehat{G}$ koji pak sadrže kardinalnosti grupa u završnim konfiguracijama. Problem se može predstaviti kao particioniranje cijelih brojeva [178] gdje je cijeli broj koji se particionira jednak ukupnom broju dostupnih komponenti. Primjerice, u slučaju sustava s osam komponenti particija $\widehat{G} = \{4,4\}$ označuje one završne konfiguracija koje imaju dvije grupe po četiri komponente, dok $\widehat{G} = \{3,4,1\}$ označuje konfiguracije s tri grupe koje imaju tri, četiri, i jednu komponentu. Algoritam 2 prikazuje pseudo-kod za generiranje particija. Postupak započinje s inicijalnom trivijalnom particijom s jednom grupom u kojoj su sve komponente (linija 3) te se nastavlja rekurzivnim rastavljanjem te grupe na manje grupe.

U drugoj fazi vrši se popunjavanje različitih kombinacija u količini i veličini grupa iz prve faze gdje se za svaki $\widehat{G} \in \widehat{K}$ raspoređuju komponente u svim mogućim kombinacijama. Primjerice, za $\widehat{G} = \{2,1\}$ i dostupne komponente $C^M = \{a,b,c\}$ određene su tri konfiguracije

$K^M = \{K_0^M, K_1^M, K_2^M\}$, gdje je $K_0^M = \{\{a,b\},\{c\}\}$, $K_1^M = \{\{a,c\},\{b\}\}$, $K_2^M = \{\{b,c\},\{a\}\}$. Algoritam 3 prikazuje pseudo-kod za navedeno popunjavanje. Procedura *Kombiniraj* vrši klasično kombiniranje bez ponavljanja što nije opisano.

Algoritam 2 Prva faza određivanja konfiguracija

```
1: procedure PARTICIONIRAJ(n)                    ▷ n = |C^M|, broj komponenti u modelu M
2:     Ĝ ← {n}
3:     K̂ ← {Ĝ}                                   ▷ Inicijalizacija skupa rješenja početnom konfiguracijom
4:     for all i ∈ [1,n⟩ do
5:         a ← n − i
6:         R ← PARTICIONIRAJ(i)
7:         for all r ∈ R do
8:             if r_0 ≤ a then
9:                 K̂ ← K̂ ∪ (a ∪ r)
10:            end if
11:        end for
12:    end for
13:    return K̂
14: end procedure
```

Algoritam 3 Druga faza određivanja konfiguracija: smještanje komponenti u grupe određene prvom fazom

```
1: procedure GenerirajKonfiguracije(C^M)                ▷ C^M, komponente modela M
2:     K^M ← ∅
3:     K̂ ← Particioniraj(|C^M|)
4:     for all Ĝ ∈ K̂ do
5:         K^M ← K^M ∪ Smjesti(C^M, Ĝ)
6:     end for
7:     return K^M
8: end procedure

9: procedure Smjesti(C, Ĝ)
10:    K ← ∅
11:    a ← DohvatiElement(Ĝ)          ▷ Nasumično izdvajanje jednog elementa skupa
12:    P* ← Kombiniraj(C, a)
13:    for all P ∈ P* do
14:        G ← {P}
15:        C' ← C \ P
16:        Ĝ' ← Ĝ \ {a}
17:        if |C'| > 0 then
18:            K' ← Smjesti(C', Ĝ')
19:            for all k' ∈ K' do
20:                K ← K ∪ G ∪ k'
21:            end for
22:        else
23:            K ← K ∪ G
24:        end if
25:    end for
26:    return K
27: end procedure

28: procedure Kombiniraj(X, a)
                ▷ Kombinacije bez ponavljanja a-tog razreda nad elementima skupa X
29:    ...
30: end procedure

31: procedure DohvatiElement(X)                ▷ Dohvaćanje jednog elementa skupa X
32:    ...
33: end procedure
```

4.3.2 Metoda praćenja relevantnih metrika

Kako bi se ocijenila razina udovoljavanja ugovora SLA potrebno je odrediti razdiobu $f_T(t)$ za svaku operaciju. Ovo poglavlje iznosi postupak kojim se određuje razdioba $f_T(t)$ iz prikupljenih podataka pomoću čega se naknadno ocjenjuje svaka konfiguracija pri različitim inačicama ugovora.

Praćenje programske usluge osmišljava se sukladno definiranom modelu ugovora SLA. U nastavku se opisuje postupak praćenja prema modelu predstavljenom u poglavlju 4.2. Točnije, mjeri se vrijeme odaziva za svaki zahtjev upućen prema usluzi. Kako se radi o velikoj količini zahtjeva koji mogu biti generirani tijekom testiranja sustava, potrebno je ostvariti algoritam za aproksimaciju kvantila (engl. *quantile*) poput srednje vrijednosti ili percentila uz prihvatljivu brzinu izračuna i količinu prostora za pohranu mjerenih vremena odaziva [179]. Predloženi algoritam može se iskoristiti i za aproksimaciju razdiobe $f_T(t)$.

U skladu s predloženom metodom potrebno je ostvariti algoritam koji ispunjava sljedeće zahtjeve:

1. Algoritam ne može imati istovremeni uvid u sve podatke, s obzirom na to da se oni prikupljaju vremenito, nakon svakog upućenog zahtjeva
2. Mora biti efikasan po pitanju ukupne količine podataka koje pohranjuje
3. Mora omogućiti agregaciju više skupova podataka zbog raspodijeljene implementacije generatora zahtjeva
4. Mora pružiti zadovoljavajuću preciznost za potrebe formiranja ugovora SLA

Također, možemo definirati sljedeće pretpostavke koje se dotiču postupka praćenja vremena odaziva:

- Vrijednosti vremena odaziva koje pojedine operacije mogu ostvariti spadaju u red veličine 10^0 do 10^3 sekundi uz razdiobu koja je slična razdiobama *Pareto* ili *Log-normal* [180];
- Kvantili za koje se traži procijenjena vrijednost nalaze se unutar intervala 0,8 – 1. Uobičajeno je za ugovore SLA da određuju vrijednosti kvantila većih od 0.9 (90%).

U skladu sa zahtjevima i pretpostavkama oblikovan je algoritam za aproksimacije kvantila 4. Algoritam se sastoji od sljedećih metoda:

- Metoda DodajVrijednost(t) zaokružuje vrijednost na najbližu vrijednost zadane preciznosti te pohranjuje koristeći postupak zasnovan na B-stablu za statističke izračune OST+ koji je prilagodba osnovnog algoritma OST predstavljenoga u [181]. Metode koje OST+ podržava popisane su tablicom 4.1.Prosječna složenost operacije je određena složenošću operacije OST-SET() te iznosi $O(log\ n)$ gdje je n količina zapisa.
- Metoda DohvatiKvantil(t) izračunava pripadajući kvantil za vrijeme odaziva t. Složenost operacije je određena složenošću operacije OST-SELECT() te iznosi $O(log\ n)$.
- Metoda OdrediKvantil(p) vraća vrijeme odaziva koje odgovara kvantilu p. Složenost

Tablica 4.1: Operacije nad OST stablom

Metoda	**Složenost**	**Opis**
OST-SET($\mathscr{T}, k, v$)	$O(log\ n)$	Dodaje vrijednost v uz ključ k u stablo $\mathscr{T}$
OST-GET($\mathscr{T}, k$)	$O(log\ n)$	Dohvaća vrijednost pod ključem k iz stabla $\mathscr{T}$
OST-INCREMENT($\mathscr{T}, k, i$)	$O(log\ n)$	Uvećava vrijednost pod ključem k za inkrement i unutar stabla $\mathscr{T}$, a ako ključ ne postoji, postavlja se na i
OST-SELECT($\mathscr{T}, i$)	$O(log\ n)$	Dohvaća zapis i-tog ranga u stablu $\mathscr{T}$
OST-RANK($\mathscr{T}, k$)	$O(log\ n)$	Vraća rank za ključ k unutar stabla $\mathscr{T}$
OST-SIZE($\mathscr{T}$)	$O(1)$	Vraća broj zapisa u stablu $\mathscr{T}$

operacije je određena složenošću operacije OST-RANK() te iznosi $O(log\ n)$.

Procedure algoritma upotrebljavaju se tako da se za svaku vrstu zahtjeva oblikuje novo stablo $\mathscr{T}_\rangle$ te pozivaju pripadajuće metode. Za svaki zahtjev upućen prema programskoj usluzi, nakon izmjere vremena odaziva t poziva se metoda DodajVrijednost($\mathscr{T}_\rangle, t$). Metode DohvatiKvantil($\mathscr{T}_\rangle, p$) i OdrediKvantil($\mathscr{T}_\rangle, t$) pozivaju se za izračun percentila.

Algoritam 4 Procedure za učinkovitu aproksimaciju percentila iz skupa vremena odaziva

```
1: 𝒯 ← ∅
2: h ← 10^−3                                   ▷ Željena preciznost zaokruživanja

3: procedure DodajVrijednost(𝒯, t)              ▷ Dodaj novi zapis (u sekundama)
4:     t′ ← ⌈t ∗ h − 0.5⌉
5:     OST-INCREMENT(𝒯, t′, 1)
6: end procedure

7: procedure DohvatiKvantil(𝒯, p)
8:     n ← OST-SIZE(𝒯)
9:     r ← ⌈p ∗ n − 0.5⌉
10:    t ← OST-SELECT(𝒯, r)
11:    return t
12: end procedure

13: procedure OdrediKvantil(𝒯, t)
14:    t′ ← ⌈t ∗ h − 0.5⌉
15:    r ← OST-RANK(𝒯, t′)
16:    n ← OST-SIZE(𝒯)
17:    return r/n
18: end procedure
```

4.3.3 Metoda redefiniranja ugovora SLA

Neka je $\mathscr{T}_{o_1}$ OST stablo koje sadrži razdiobu vremena odaziva popunjeno pozivanjem metode DodajVrijednost(t) iz algoritma 4 kod provođenja testiranja sintetičkim opterećenjem za metodu o_1 programske usluge. Također, neka je T_{o_1} vrijednost $\tau\%$ percentila vremena odaziva zadana SLA ugovorom.

Možemo reći da operacija o_1 zadovoljava ugovor ako vrijedi

$$\text{DohvatiKvantil}(\mathscr{T}_{o_1}, \tau\%) \leq T_{o_1} \ .$$

Ako operacija o_1 ne udovoljava SLA, može se pristupiti parametrizaciji ugovora u skladu s izmjerenim vremenima odaziva koja može ostvariti usluga o_1 tako da se podesi T_{o_1} na vrijednost:

$$T_{o_1} = \text{DohvatiKvantil}(\mathscr{T}_{o_1}, \tau\%) \ .$$

Ako se želi zadržati T_{o_1} može se promijeniti vrijednost promatranog percentila $\tau\%$:

$$\tau\% = \text{OdrediKvantil}(\mathscr{T}_{o_1}, T_{o_1}) \ .$$

Pomoću navedenih operacija može se definirati ugovor koji je utvrđen temeljem stvarnih izmjera radnih karakteristika koje programska usluga može ostvariti.

4.4 Zaključak

U ovom je poglavlju iznesena metoda za ocjenu pojedinih konfiguracija programske usluge koristeći se modeliranim sintetičkim radnim opterećenjem. Sukladno tomu prikazan je model radnoga opterećenja zasnovan na nehomogenom Poissonovom procesu te pripadni algoritam za generiranje zahtjeva u skladu s modelom. Opisan je i model ugovora SLA koji se temelji na percentilima vremena odaziva operacija koje programska usluga nudi. Zatim je opisan algoritam za generiranje različitih konfiguracija programske usluge koja slijedi predstavljeni model iz poglavlja 3. Zaključno, prikazana je metoda za izračun kvantila vremena odaziva koja efikasno pohranjuje i određuje vremena odaziva prema modelu ugovora SLA. Pomoću navedenih modela i algoritama ostvarena je razvojna okolina za optimiranje konfiguracije sustava predstavljena u nastavku rada.

Poglavlje 5

Evolucijski algoritam za odabir optimalnih konfiguracija

Ovo poglavlje iznosi algoritam koji pretražuje prostor svih mogućih konfiguracija u potrazi za optimalnim kandidatima. Navedeni algoritam koristi metodu simulacije pomoću mreža redova čekanja (engl. *queueing network*, skraćeno QN) za bržu evaluaciju svake konfiguracije. Time je omogućena brža evaluacija kandidata i pronalazak prihvatljive količine kandidata za optimiranje stvarnim testiranjem pomoću metode iznesene u poglavlju 4.

5.1 Zahtjevi pri određivanju optimizacijskog algoritma

Postoji velika količina optimizacijskih metoda i odabir prave tehnike zavisi o vrsti optimizacijskog problema i željenim ishodima optimizacije. Prvi zahtjev prilikom optimiranja konfiguracije jest odabir optimizacijske tehnike nad diskretnim skupom kandidata (kombinatorna optimizacija [138]). Pojedine kandidate predstavljaju konfiguracije proizašle iz modela sustava M.

Sljedeći zahtjev jest postojanje više ciljeva optimizacije (engl. *multi-objective optimization* [182]) gdje postoji više od jedne funkcije cilja poput optimizacije troškova izvođenja i radnih karakteristika. Funkcije cilja mogu također biti u međusobnoj sprezi gdje poboljšanje jednog kriterija poput troška izvođenja dovodi do lošijih rezultata pri drugom kriteriju poput vremena odaziva sustava. Zbog oprečnih ciljeva nije moguće odrediti uređenje koje bi odredilo optimalnog kandidata zbog parcijalnoga uređenja gdje nije moguća direktna usporedba svih kandidata. Određivanje dodatnoga kriterija ostavlja se korisniku optimizacijske tehnike nakon što dobije uvid u optimalne kandidate. Rezultat takve optimizacije jest skup *efikasnih rješenja* ili *Pareto optimalnih rješenja* [183].

Optimizacijski algoritmi dodatno se mogu podijeliti na egzaktne i aproksimacijske. Egzaktne metode pronalaze najkvalitetnija rješenja dok aproksimacijski algoritmi pronalaze pri-

bližna rješenja. Predloženi algoritam koristi aproksimacijsku metodu iz dva razloga: (1) evaluacijska funkcija svakog kandidata ionako aproksimira ponašanje stvarnog sustava, (2) zbog složenosti i stohastičnih elemenata u računalnim sustavima ne postoji egzaktna evaluacijska funkcija.

Metoda simulacije značajno je sporija od analitičkih metoda. Razlog zbog kojega nije moguće upotrijebiti analitičke metode jest nepostojanje analitičkih metoda u evaluaciji redova čekanja s mogućnošću izračuna percentila vremena odaziva [5]. Zbog toga je poželjna mogućnost paraleliziranja postupka optimizacija tako da se u svakom trenutku provodi evaluacija više kandidata.

Prema svemu navedenome određena su sljedeća svojstva optimizacijskog problema:

- diskretni konačan skup kandidata - koji proizlazi iz svih mogućih konfiguracija sustava
- višekriterijski problem - koji zahtijeva optimizaciju po više oprečnih kriterija
- primjena računski zahtjevne funkcije cilja - koristi se metoda simulacije pri evaluaciji pojedinih kandidata.

5.2 Tehnika optimizacije

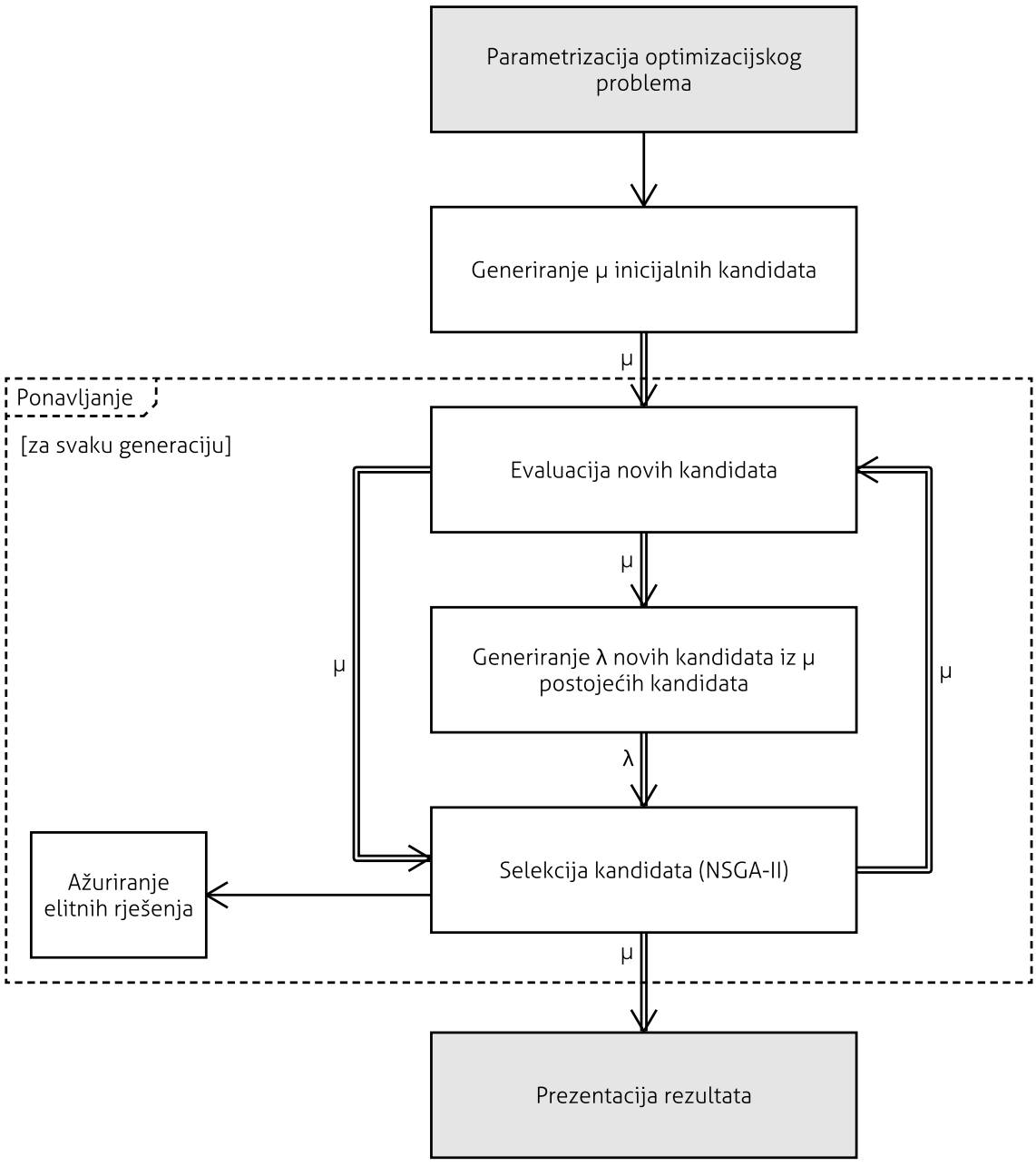
Prema zahtjevima optimizacijskog problema odabrana je tehnika optimizacije u obliku *metaheuristike* (engl. *metaheuristics*) [163]. Metaheuristika je procedura osmišljena za optimizaciju određenog problema kojeg karakterizira velik broj mogućih kandidata te neizvjesnost i računska zahtjevnost pri evaluaciji, zbog čega nije moguće iterativno proći kroz svako rješenje i ocijeniti ga. Postoji više vrsta metaheuristika. Među najprimjenjenije spadaju *optimizacije mravljom kolonijom* (engl. *ant-colony optimization*), evolucijski algoritmi (engl. *evolutionary algorithms*) te postupak simuliranog kaljenja (engl. *simulated annealing*) [163].

U ovom radu koriste se evolucijski algoritmi zbog mogućnosti rada nad populacijom i određivanja Pareto optimalnosti [184]. Pojam populacije predstavlja postojanje više kandidata koji se mogu paralelno ocijeniti što pogoduje ubrzanju provođenje postupka optimizacije. Točnije, koristi se višekriterijski evolucijski algoritam i metoda elitizma NSGA-II (engl. *nondominated sorting genetic algorithm*) [147] za koju je pokazano da polučuje kvalitetnije rezultate u kraćem vremenu od ostalih metoda, poput SPEA-2 (engl. *strength Pareto evolutionary algorithm*) ili PAES (engl. *Pareto-archived evolution strategy*). Tehnika NSGA-II se sastoji u uvođenju elitizma u obliku čuvanja Pareto optimalnih rješenja kroz svaku iteraciju evolucijske optimizacije što ubrzava cjelokupni postupak i osigurava da se najkvalitetnija rješenja ne izgube.

5.3 Postupak provođenja evolucijskog algoritma

Slika 5.1 prikazuje korake optimizacije evolucijskog algoritma zasnovanog na tehnici NSGA-II. Postupak započinje postavljanjem optimizacijskog problema definiranjem reprezentacije svakog kandidata (konfiguracije) i evaluacijskih funkcija. Zadaje se cilj optimizacije određivanjem kriterija koje treba maksimizirati ili minimizirati. Evaluacijska funkcija se oblikuje tako da vraća ocjenu kandidata po svakom odabranom kriteriju zasebno. U nastavku se iznose detalji o načinu na koji su reprezentirani kandidati (poglavlje 5.4) te na koji se način oni evaluiraju (poglavlje 5.5). Nakon postavljanja optimizacijskog problema on se parametrizira svim potrebnim vrijednostima koje su potrebne u provođenju evaluacije što uključuje definiranje komponenti sustava, razdiobu vremena odaziva za sve operacije koju komponente pružaju, dolazno radno opterećenje sustava, strategiju za postizanje elastičnosti te zavisnosti između komponenti. Svi potrebni parametri izneseni su u poglavlju 5.5.

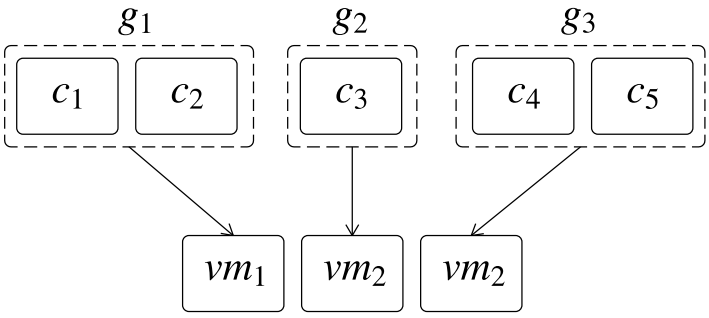
Nakon što je optimizacijski problem postavljen i izvršena parametrizacija svih potrebnih modela za evaluaciju, daljnji postupak slijedi $(\mu + \lambda)$ postupak [185] uz iznimku da se selekcija vrši po NSGA-II principu. $(\mu + \lambda)$ postupak uključuje nasumično generiranje μ početnih kandidata nakon čega slijedi njihova evaluacija. Nakon toga, iterativno se ponavlja postupak generiranja λ novih kandidata te selekcije μ kandidata između μ početnih i λ novogeneriranih kandidata. Postupak se ponavlja dok se ne ostvari željeni kriterij zaustavljanja. Jednu takvu iteraciju nazivamo *generacijom*. Postupak se može zaustaviti nakon željenog broja generacija ili ako se ponavljanjem ne dobivaju bolja rješenja i dolazi do konvergiranja populacije prema Pareto optimalnom setu rješenja.



Slika 5.1: Postupak optimizacije elitističkim evolucijskim algoritmom NSGA-II

5.4 Reprezentacija kandidata

Kandidata predstavlja svaka moguća konfiguracija sustava po modelu iznesenom u poglavlju 3. Svaki kandidat je predstavljen s komponentama i elastičnim grupama kojima komponente pripadaju te s vrstom poslužitelja na koji se elastična grupa postavlja. Inicijalni set kandidata određen je nasumičnim odabirom. Slika 5.2 prikazuje primjer kandidate gdje je pet komponenti raspoređeno na tri elastične grupe te je za svaku definirana različita vrsta poslužitelja.

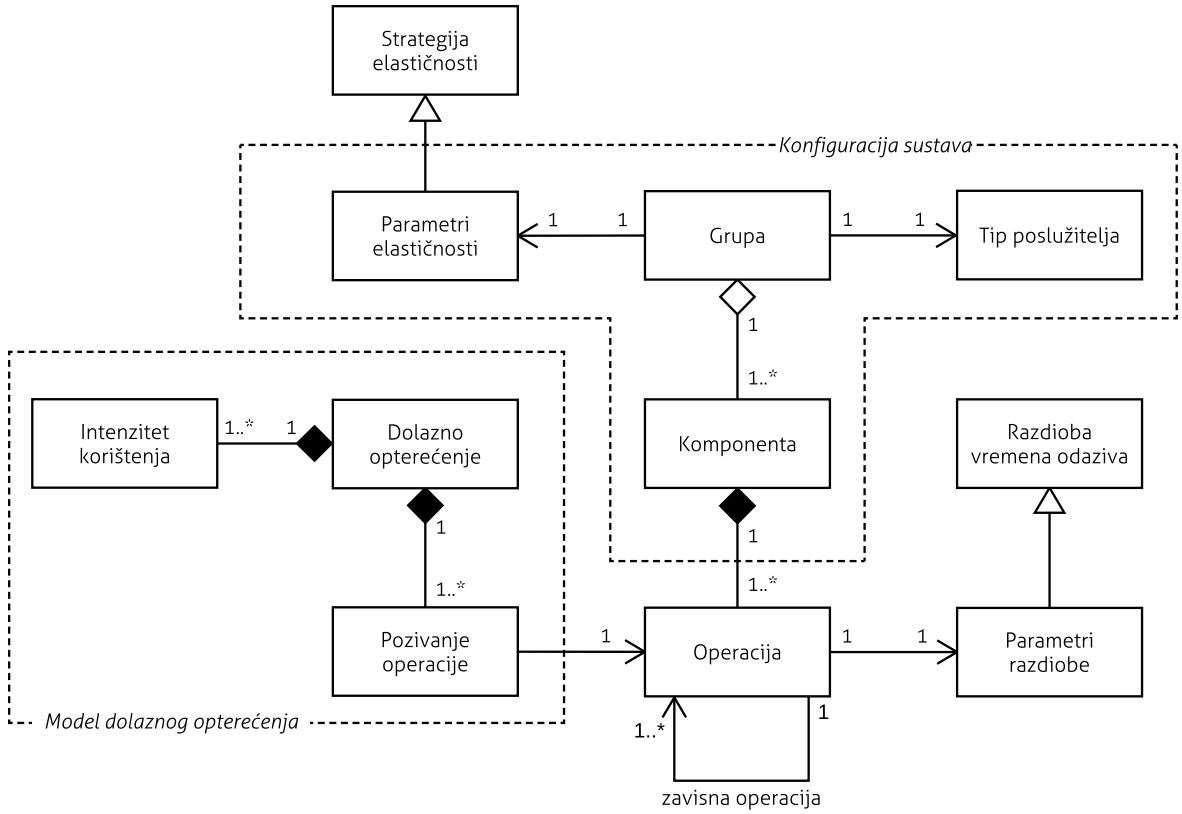


Slika 5.2: Primjer jednog kandidata (konfiguracije)

5.5 Evaluacija kandidata

Kako ne postoji odgovarajući analitički model za određivanje karakteristika pojedinih konfiguracija, koristi se metoda simulacije pomoću QN-a. Slika 5.3 prikazuje klasni dijagrama jezika UML s odnosima između svih relevantnih pojmova pomoću kojih se oblikuje simulacijski model. Pojedina konfiguracija sustava predstavljena je rasporedom komponenti u elastične grupe i dodjeljivanjem željenog tipa poslužitelja svakog grupi. Također, za svaku elastičnu grupu određena je strategija i parametri elastičnosti (v. poglavlje 2.4.3). Svaka komponenta predstavljena je s operacijama koje vrši gdje za svaku operaciju postoji statistička razdioba vremena odaziva. Realizacija međuzavisnosti između operacija pojedinih komponenti zavisi o tome nalaze li se komponente u istoj elastičnoj grupi. Operacije koje se pozivaju izvana uključene su u model dolaznog opterećenja koje posjeduje funkciju intenziteta (v. poglavlje 4.1).

Proces simulacije provodi se na način kako je to prikazano slikom 5.4. Model opterećenja zadan od strane korisnika preslikava se u simulacijski model opterećenja. Pomoću komponentnog modela sustava (v. poglavlje 3.4) i trenutne konfiguracije (v. poglavlje 6.2) gradi se QN prema poglavlju 5.5.1. Parametri QN-a određuju se prema mjerenim ili predviđenim razdiobama vremena odaziva za svaku operaciju komponenti koje čine sustav te ulaze u postupak optimizacije. Rezultirajući simulacijski model koristi se u postupku simulacije s modelom naplate (v. poglavlje 3.5.2) i modelom SLA-a (v. poglavlje 4.2).

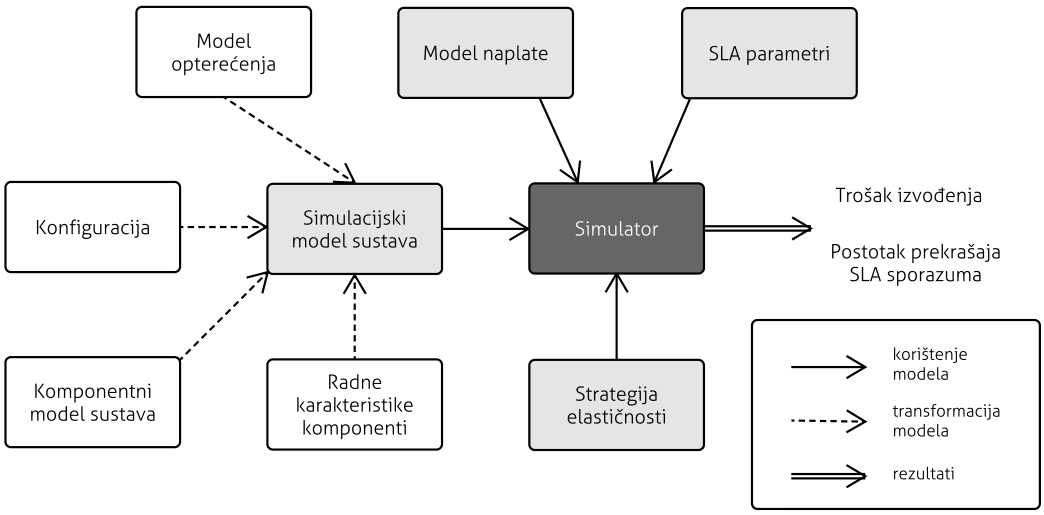


Slika 5.3: Meta-model sustava za provođenje simulacije

Rezultat postupka simulacije jesu željeni kriteriji kvalitete postavljeni u skladu s ciljevima višekriterijske optimizacije. Postupak simulacije u ovom radu omogućuje mjerenje sljedećih kriterija kvalitete:

- trošak najma infrastrukture prema modelu naplate
- razdioba ukupnog vremena izvođenja svake operacije prema kojoj se mogu odrediti željeni kvantili ili srednje vrijednosti
- razdioba vremena čekanja na izvođenje svake operacije
- razdioba iskoristivosti pojedinih resursa u željenim diskretnim vremenskim intervalima

U skladu sa željenim ciljevima optimizacije mogu se postaviti dodatni analitički ili simulacijski modeli za određivanje ostalih kriterija kvalitete (zaštićenost, raspoloživost, pouzdanost, ...). Postupak optimizacije evolucijskim algoritmom ne postavlja nikakva ograničenja na realizaciju funkcije cilja. U ovom radu odabran je simulacijski model prema ciljevima rada: zadovoljavanje SLA-ova unutar promjenjivog radnog opterećenja sustava uz optimalan trošak izvođenja.



Slika 5.4: Postupak provođenja simulacije

5.5.1 Provođenje simulacije mrežom redova čekanja

Za potrebe optimizacije evolucijskim algoritmom u skladu s ciljevima ovog istraživanja oblikovan je postupak simulacije QN-a s promjenjivim radnim opterećenjem i svojstvom elastičnosti pojedinih čvora QN-a. Servisni čvorovi QN-a su tipa $G/G/k$ po Kendallovoj notaciji [186] što označava da je moguće odrediti općenitu razdiobu vremena dolazaka i posluživanja s više kanala (k) posluživanja. Kanali posluživanja koriste se za modeliranje broj procesora virtualnih poslužitelja. Način funkcioniranja simulacije pojašnjen je na osnovnom primjeru bez gubitka općenitosti.

Za primjer ćemo razmotriti jednostavan sustav M od tri komponente $C^M = \{A, B\}$. Komponente su redom realizirane kroz nekoliko operacija: $oper(A) = \{o_{A,1}, o_{A,2}\}$, $oper(B) = \{o_{B,1}, o_{B,2}\}$. Model radnog opterećenja sustava sastoji se od funkcije 5.1 sa zahtjevima prema tablici 5.1 gdje je $\tau\%$ željeni percentil vremena odaziva ispod vrijednosti t_{SLA} (formula 4.2). U primjeru se koriste dvije vrste virtualnih poslužitelja $VM = \{VM_1, VM_2\}$ gdje vm_1 ima jednu procesorsku jedinicu, a vm_2 dvije. Koristi se model troškova infrastrukture iz poglavlja 3.5.2 gdje je $cost(vm_1) < cost(vm_2)$.

$$\lambda(t_{sim}) = \begin{cases} 5 & \text{if } t_{sim} \leq 5 \\ t_{sim} & \text{if } 5 < t_{sim} < 20 \\ -2t_{sim} + 60 & \text{if } 20 \leq t_{sim} < 50 \end{cases} \tag{5.1}$$

Komponente postavljene na virtualne poslužitelje modelirane su pomoću otvorene mreže redova čekanja s višestrukim klasama zahtjeva (engl. *multi-class open queue network*, skraćeno MC-OQN) [187] poslužitelja iz teorije redova čekanja. Svaka vrsta zahtjeva ($R = \{r_1, r_2, r_3\}$) prikazana je kao jedna klasa MC-OQN-a. Za vektor promjenjivog intenziteta dolazaka vri-

Tablica 5.1: Radno opterećenje za primjer simulacije

Vrsta zahtjeva	**Komponenta**	**Operacija**	**Zavisne operacije**	**Udio** p (%)	t_{SLA}(ms)	$\tau_{\%}$
r_1	A	$o_{A,1}$	-	30	500	0.95
r_2	B	$o_{B,1}$	-	30	500	0.95
r_3	A	$o_{A,2}$	$o_{B,2}$	40	2000	0.95

jedi $\vec{\lambda}(t_{sim}) = \lambda(t_{sim}) \cdot [p_1, p_2, p_3]^\top$ gdje su $p_i, i \in [1,3]$ udjeli za svaku vrstu zahtjeva (tablica 5.1) čime je u svakom trenutku simulacije (t_{sim}) određen intenzitet dolazaka po svakoj klasi zahtjeva. Svaka vrsta zahtjeva poziva određenu operaciju na komponenti na kojoj se nalazi, a dodatno vrijedi da svaka operacija može imati zavisnih operacija po modelu 5.3.

Svaki servisni čvor MC-OQN-a koji predstavlja virtualni poslužitelj *vm* s komponentama sustava ima dodijeljenu razdiobu vremena obrade zasebno po svakoj klasi zahtjeva:

$$D_{vm,r_i} \sim \sum_{o_{c,local}} D_{o_c} + \sum_{o_{c,remote}} D_{comm} \tag{5.2}$$

gdje su $o_{c,local}$ sve operacije koje r_j poziva pod uvjetom da se komponenta $o_{c,local}$ nalazi na istom poslužitelju *vm*. Ako se operacija nalazi na komponenti koja je na nekom drugom poslužitelju, u vrijeme obrade trenutnog poslužitelja uračunava se i vrijeme komunikacije pripadajućom razdiobom $f_{D_{comm}}$.

Drugim riječima, servisni čvorovi QN-a predstavljaju obradu pojedinih zahtjeva pomoću pripadajućih komponenata koje su postavljene na poslužitelje. Zavisno o konfiguraciji, ako se više komponenta nalazi na jednom poslužitelju zbrajaju se vremena obrade potrebnih operacija svake komponente koja sudjeluje u obradi zahtjeva, a ako komponente komuniciraju s komponentama na drugim poslužiteljima uračunava se vrijeme komunikacije. Različite konfiguracije određuju smještaj komponenti na poslužitelje te samim time utječu na razdiobe vezane uz svaki poslužitelj.

Finalno vrijeme obrade svakog zahtjeva određuje se zbrajanjem svih vremena obrade i čekanja na sudjelujućim servisnim čvorovima QN-a.

Pomoću komponenti iz primjera $C = \{A, B\}$ moguće je ostvariti podjelu na elastične grupe na dva načina. Prvi način je da svaka komponenta bude u zasebnoj elastičnoj grupi, a drugi način je da budu zajedno u jednoj elastičnoj grupi. Slika 5.5a prikazuje QN za konfiguraciju iz primjera gdje je svaka komponenta na svom poslužitelju te time postoje dvije elastične grupe. Prema tablici 5.1 klasa zahtjeva r_1 predstavljena dolaznim intenzitetom λ_1 obrađuje se pomoću operacije $o_{A,1}$ na komponenti A i servisnom čvoru $s_{1,1}$ koji predstavlja jedan virtualni

poslužitelj u prvoj elastičnoj grupi. Klasa zahtjeva r_2 predstavljena intenzitetom λ_2 obrađuje se pomoću operacije $o_{B,1}$ komponente B. Klasa r_3 intenziteta λ_3 obrađuje se operacijom $o_{A,2}$ na komponenti A koja dodatno koristi operaciju $o_{B,2}$ komponente B. Drugi primjer konfiguracije gdje su obje komponente u istoj elastičnoj grupi prikazan je na slici 5.6. U ovom primjeru postoji jedan virtualni poslužitelj koji je pridodan jednoj elastičnoj grupi, na kojemu se pak obje komponente A i B.

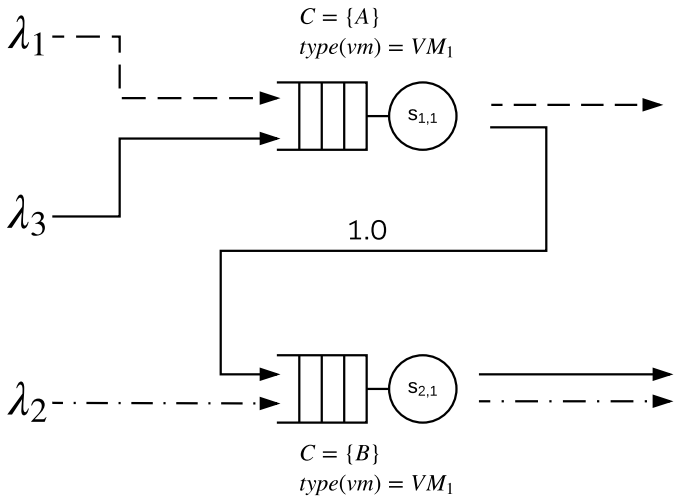
Po modelu iz poglavlja 6.2 svaka elastična grupa sama sebi određuje broj poslužitelja zavisno o opterećenju. Tijekom simulacije, konstantno se prati opterećenje

$$\rho_{vm} = \sum_{i=1}^{R} \frac{\lambda_i}{\mu_i}$$

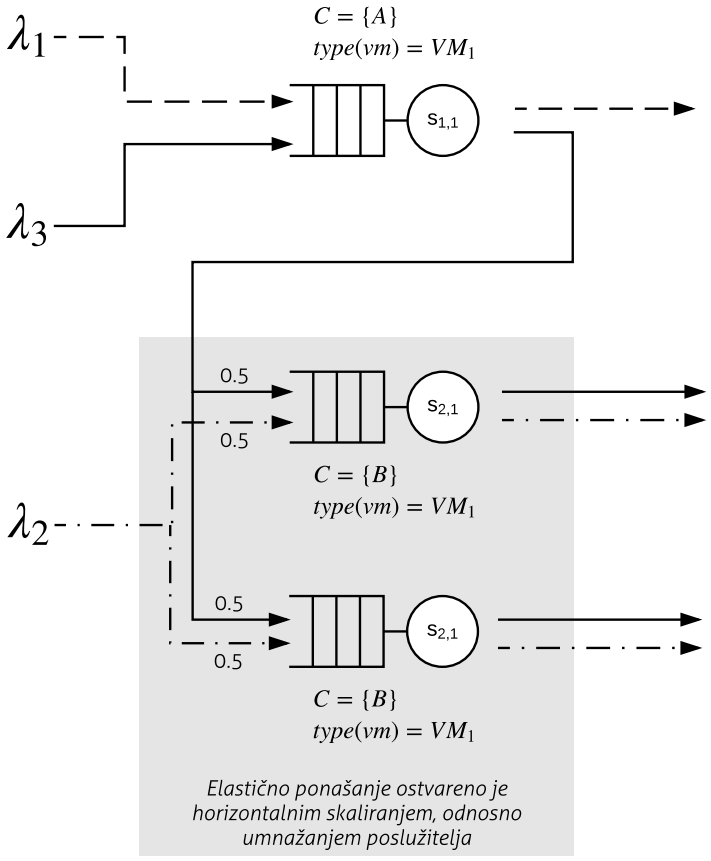
svakog servisnog poslužitelja gdje je $\mu_i \sim D_{vm,r_i}$ iz formule 5.2. Dakle opterećenje ρ_{vm} određeno je ukupnim opterećenjem izvršavanja svih zahtjeva pripadajućih klasa.

Trenutna implementacija koristi strategiju skaliranja određenu gornjim i donjim limitom opterećenja ρ_{vm}. Ako prosječno opterećenje svih poslužitelja u elastičnoj grupi prijeđe gornju granicu dolazi do horizontalnog skaliranja. Slika 5.5b prikazuje primjer gdje je u drugoj elastičnoj grupi dodan novi servisni čvor. Samim time, simulira se i postojanje raspoređivača opterećenja time što se sav ulazni promet u servisni čvor dijeli između dva poslužitelja.

Različite vrste poslužitelje (u ovom primjeru $VM = \{VM_1, VM_2\}$) karakterizirane su brojem kanala posluživanja kod svakog servisnog čvora. Na slici 5.5b poslužitelj $s_{1,1}$ koji modelira poslužitelj tipa VM_2 s dva procesora, predstavljen je servisnim čvorom s dva kanala.

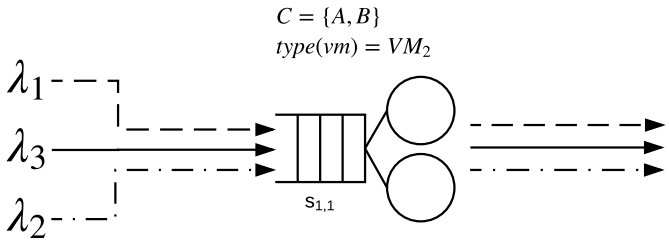


(a) Primjer mreže koja modelira konfiguraciju sačinjenu od dvije elastične grupe



(b) Primjer gdje druga elastična grupa horizontalno skalira na dva poslužitelja

Slika 5.5: Primjeri otvorenih mreža redova čekanja s višestrukim klasama zahtjeva



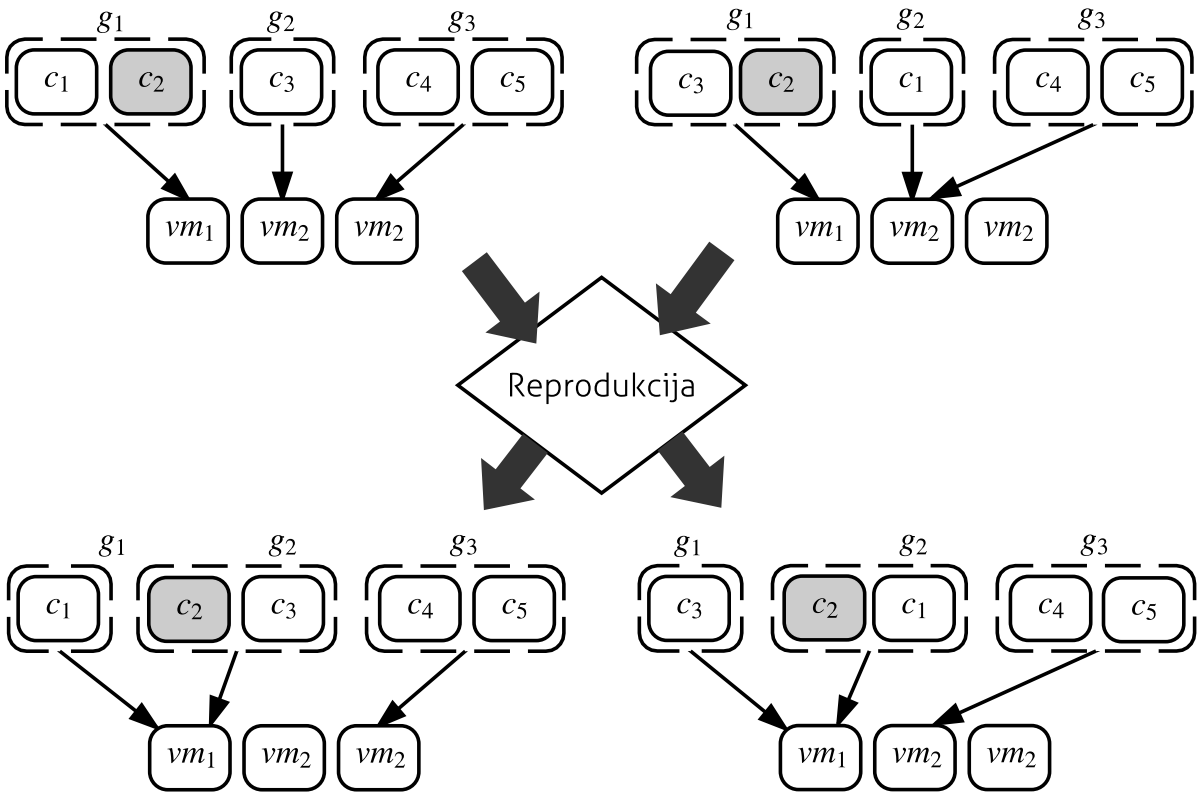
Slika 5.6: Primjer otvorene mreže redova čekanja s jednom elastičnom grupom za koju je određena vrsta poslužitelja sa dvije procesorske jezgre

5.6 Reprodukcija kandidata

Evolucijski algoritam uključuje provedbu operatora mutacije i reprodukcije [185]. Obje operacije se nad kandidatima provode s određenom vjerojatnošću. Za potrebe optimizacije postavljena je vjerojatnost mutacije na $p_{mut} = 0.2$ te vjerojatnost reprodukcije na $p_{rep} = 0.7$. Operacija reprodukcije stvara od dva kandidata *roditelja* dva kandidata *potomka*. Postupak kojim se stvara potomak sastoji se od nasumičnog odabira komponente koja se premjesti iz elastične grupe prvog roditelja i smjesti u elastičnu grupu s komponentama iz elastične grupe drugog roditelja. Vrsta poslužitelja pripadne elastične grupe također se određuje pomoću drugog roditelja. Operacija mutacije premješta nasumično odabranu komponentu u nasumično odabranu grupu te vrši nasumičnu promjenu vrste poslužitelja.

Slika 5.7 prikazuje primjer reprodukcije između dva roditelja gdje je slučajno odabrana komponenta c_2 za premještanje. Lijevi potomak nastao je od lijevog roditelja premještanjem komponente c_2 u grupu s komponentom c_3 prema desnom roditelju. Analogno, desni potomak nastao je od desnog roditelja premještanjem komponente c_2 u grupu s komponentom c_1 prema prvom roditelju. Vrste poslužitelja također su određene prema lokaciji komponente c_2 kod roditelja.

Opisane tehnike reprodukcije i mutacije omogućuju pronalazak i evaluaciju novih konfiguracije. U ovom radu primijenjena je $(\lambda + \mu)$ tehnika evolucije što znači da nakon reprodukcije i mutacije roditelji s potomcima ulaze u postupak selekcije za sljedeću generaciju rješenja. Time je osigurano da potomci koji pokazuju lošije rezultate od roditelja ne budu uključeni u sljedeću generaciju.



Slika 5.7: Primjer reprodukcije dva kandidata roditelja

5.7 Kriterij zaustavljanja

Kod evolucijskih algoritma potrebno je odrediti adekvatni kriterij zaustavljanja. Zadaća takvog kriterija jest odrediti pravi trenutak za zaustavljanje daljnje optimizacije. Ako dođe do preranog zaustavljanja, moguće je da neće biti otkriveni kvalitetni kandidati, a ako se zaustavljanje dogodi kasnije, dolazi do dugog izvršavanja optimizacije bez poboljšanja u rezultatima. Zadaća kriterija je uspostavljanje ravnoteže između vremena izvođenja i kvalitete rezultata [188].

Neki od kriteriji zaustavljanja mogu biti:

- maksimalni broj generacija - gdje se optimizacija zaustavlja nakon n iteracija
- vrijeme izvođenja - gdje se optimizacija zaustavlja nakon određenog vremena
- ostvarenje zadane kvalitete - gdje se može po svakom kriteriju optimizacije odrediti minimalno zadovoljavajuća granica; optimizacija se zaustavlja nakon što željeni broj jedinki ostvari minimalnu kvalitetu

Predloženi algoritam koristi metodu zaustavljanja prema maksimalnom dozvoljenom broju generacija koja osigurava gornju granicu duljine izvođenja optimizacije te dodatno po ostvarenju minimalno zadane kvalitete gdje se za granice kvalitete uzimaju SLA ograničenja u vremenu odaziva.

5.8 Zaključak

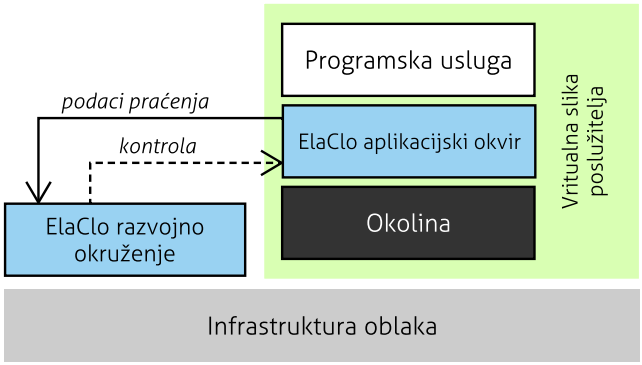
U ovom poglavlju opisan je evolucijski algoritam koji koristi postupak diskretne simulacije sustava modeliranoga pomoću mreže redova čekanja u potrazi za skupom Pareto optimalnih rješenja. Takav se skup rješenja može koristiti kao ulazni skup kandidata za provođenje postupka ocjene i odabira optimalne konfiguracije metodom iznesenom u poglavlju 4. Postupak simulacije i učinkovita pretrage prostora rješenja predstavljenoga evolucijskoga algoritma omogućuje pronalazak optimalnih rješenja u značajno kraćem vremenu nego što je to moguće ostvariti mjerenjem nad stvarnim sustavom te samim time čini postupak optimizacije izvedivim.

Poglavlje 6

Izvedbena okolina za optimiranje konfiguracije

Ovo poglavlje iznosi model izvedbenog sustava za optimiranje konfiguracije aplikacije u oblaku koja slijedi arhitekturu predloženu u poglavlju 3. Sustav prati metodu za ocjenu i odabir konfiguracije iznesenu u poglavlju 4 te omogućuje predlaganje optimalnih kandidata pomoću genetskog algoritma iz poglavlja 5. Programska usluga ili aplikacija čiju konfiguraciju je potrebno optimirati u daljnjem se tekstu navodi kao *ciljana aplikacija* ili *ciljana programska usluga*. Sustav *ElaClo* (izvedenica od ***Ela**stic **Clo**ud*) ostvaren je kao distribuirani sustav čije se komponente mogu podijeliti na:

- *aplikacijske komponente* u obliku aplikacijskog okvira (engl. *application programming framework*) za razvoj komponenti ciljane aplikacije u predstavljenoj arhitekturi. Aplikacijske komponente integriraju se u ciljanu aplikaciju te dodatno omogućuju sustavu ElaClo uvid u različite radne karakteristike.
- komponente *razvojne okoline* u obliku mrežne aplikacije za upravljanje postupkom optimizacije od strane korisnika te pozadinskog sustava koji omogućuje izvođenje i optimiranje ciljane aplikacije.



Slika 6.1: Pregled izvedbene okoline ElaClo

Slika 6.1 prikazuje izvedbenu okolinu ElaClo podijeljenu na aplikacijski okvir i razvojnu okolinu. Cjelokupno rješenje koristi infrastrukturu u oblaku tako da ElaClo autonomno postavlja, instalira i aktivira (termini definirani u poglavlju 2.1.3) sve komponente ciljanog aplikacije tijekom procesa optimiranja. Dodatno, ElaClo autonomno iznajmljuje infrastrukturu oblaka u ostvarivanju elastičnosti za komponente ciljane aplikacije. Time je dodatno olakšan razvoj novih aplikacija gdje se ostvarivanje elastičnosti može odgoditi i obaviti tek nakon otkrivanja optimalne konfiguracije pomoću ElaClo-a. *Okolina* ciljane aplikacije uključuje sve potrebne zavisnosti koje promatrani sustav može imati poput zavisnih komponenti, komponentne platforme, knjižnica ili operacijskog sustava. Za korištenje sustava ElaClo razvojni tim ciljane aplikacije mora stvoriti slike virtualnih strojeva (engl. *virtual machine image*) koje sadrže potrebnu okolinu s aplikacijom koja uključuje komponente aplikacijskog okvira.

Pojedinosti aplikacijskog okvira iznosi poglavlje 6.1, dok poglavlje 6.2 opisuje centralnu razvojnu okolinu ElaClo. Poglavlje 6.3 iznosi detalje o izvođenju ciljane aplikacije pomoću navedenih komponenti sustava ElaClo te na koji način sustav ostvaruje elastičnost u odnosu na promjenjivo radno opterećenje. Poglavlje 6.4 opisuje korisničko sučelje za upravljanje sustavom ElaClo.

6.1 Aplikacijski okvir za ostvarivanje elastične konfiguracije

Aplikacijski okvir sustava ElaClo sudjeluje u procesu ostvarivanje željene elastične konfiguracije tako da usmjeruje komunikaciju između komponenti ciljane aplikacije. Trenutna inačica sustava ElaClo pruža aplikacijski okvir za optimiranje programske usluge implementirane pomoću komponentnog modela JavaEE*. Postupak implementacije može se proširiti i na aplikacije koje koriste ostale komponentne modele.

Aplikacijski okvir koristi se tako da naznačuje koja programska sučelja predstavljaju komponente za optimizaciju pomoću ključne riječi *@ElasticComponent* (prikazano na slici 6.2). Za označavanje zavisnih komponenti koristi se ključna riječ *@ElasticComponentProvided* (prika-

```
// IInvoiceService.java

@ElasticComponent // oznaka komponente
@WebService(targetNamespace=...)
public interface IInvoiceService {
  ...
}
```

Slika 6.2: Oznake koje su potrebne pri definiciji komponente

*JavaEE komponentni model dostupan je na *http://www.oracle.com/technetwork/java/javaee/overview/index.html*

```
// IInvoiceService.java
@Stateless
@WebService(...)
public class InvoiceService implements IInvoiceService {

   // potrebne zavisne komponente
   @Inject @ElasticComponentProvided IFiscalizationService fService;
   @Inject @ElasticComponentProvided IDataService dataService;

   public CreateInvoiceResponse createInvoice(CreateInvoiceRequest req) {
      // pozivanje operacije iz zavisne komponente
      fiscalizationService.fiscalizeInvoice(request);
   }
}
```

Slika 6.3: Potrebne oznake kod implementacije komponente sa zavisnim komponentama

zano na slici 6.3) pomoću čega ElaClo dobiva potrebne informacije o komponentama koje ciljana aplikacija ostvaruje te njihovim zavisnostima.

Komunikacija između komponenti može biti lokalna ili udaljena zavisno o trenutnoj konfiguraciji k^M. Za komponente unutar iste elastične grupe sustav ElaClo podešava lokalnu komunikaciju dok za komponente različite elastične grupe sustav podešava udaljenu komunikaciju pomoću protokola SOAP[†]. Pojedinosti o komunikaciji između komponenti iznesene su u poglavlju 6.3.2.

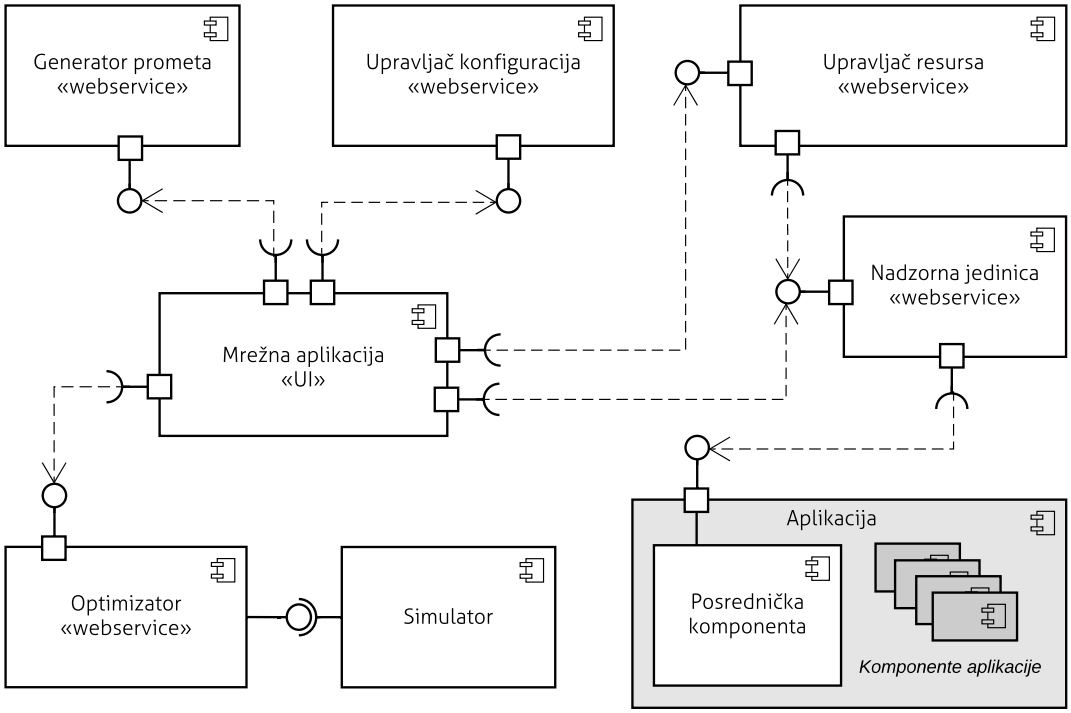
6.2 Razvojno okruženje za optimiranje konfiguracije u skladu sa sporazumom o razini usluge

Razvojno okruženje ElaClo sustava ostvareno je u uslužno orijentiranoj arhitekturi gdje je svaka komponenta ostvarena kao zasebna mrežna usluga, dok je prezentacijski sloj ostvaren kao mrežna aplikacija koja koristi te usluge. Slika 6.4 prikazuje strukturu komponenti ElaCloa. U nastavku je opisana svaka komponenta zasebno.

Generator prometa

Komponenta **generator prometa** je implementacija algoritma za generiranje mrežnih zahtjeva po protokolu HTTP i modelu radnog opterećenja iz poglavlja 4.1. Generator prometa je osnova za evaluaciju kvalitete aplikacije unutar promjenjivog radnog opterećenja nad mrežnim uslugama aplikacije. Razlika ove komponente od postojećih generatora prometa je u tome što omogućuje definiciju modela opterećenja kao nehomogenog Poissonovog procesa čime se dobiva radno opterećenje stohastičke prirode, definiranim intervalima promjenjivog intenziteta. U tu svrhu koristi se ulazno-izlazna asinkrona petlje [189] (engl. *Asynchronous*

[†]SOAP protokol definiran je na *https://www.w3.org/TR/soap/*



Slika 6.4: Komponente ElaClo sustava

I/O event loop) koja je efikasnija od postojećih rješenja zasnovanih na dretvama, poput alata JMeter[‡] ili Apache Benchmark[§]. Ulazno-izlazne asinkrone petlje efikasne su u kontekstu veće količine ulazno-izlaznih zadataka jer model dretvi uključuje niz operacija promjene konteksta prilikom promjene trenutne dretve izvršavanja.

Kako za Poissonov proces vrijedi teorem spajanja [190] u svrhu generiranja velikih opterećenja možemo koristiti nekoliko instanci generatora prometa na različitim računalima tako da funkciju opterećenja $\lambda(t)$ iz poglavlja 4.1 razložimo na nekoliko manjih funkcija $\lambda_1(t), \lambda_2(t), \ldots, \lambda_n(t)$ za koje u svakom trenutku t vrijedi

$$\lambda(t) = \sum_{i}^{n} \lambda_i(t)$$

što čini komponentu generiranja prometa horizontalno skalabilnom te se u slučaju većih intenziteta može rasporediti na više virtualnih poslužitelja.

Nadzorna jedinica

ElaClo komponenta za nadzor mrežnih usluga ostvaruje kontinuirano praćenje svake komponente aplikacije u svrhu omogućavanja elastičnosti nad aplikacijom te evaluacije kriterija kvalitete aplikacije u udovoljavanju ugovora. Komponenta za nadzor ostvaruje algoritme iz-

[‡]JMeter - *http://jmeter.apache.org/*
[§]Apache Benchmark - dostupan na *https://httpd.apache.org/docs/2.4/programs/ab.html*

nesene u poglavlju 4.3.2. Metrike koje komponenta prati jesu sljedeće:

- Vrijeme odziva svih zahtjeva prema komponentama aplikacije te izgradnja aproksimativne razdiobe $f_T(t)$ iz poglavlja 4.3.2;
- Trenutna propusnost svih operacija komponenti aplikacije;
- Srednje opterećenje (engl. *average system load*) virtualnih strojeva na kojima su komponente postavljene;
- Trenutno opterećenje procesora virtualnih strojeva;

Trenutna propusnost i opterećenje računaju se pomoću metode pomičnih prosjeka (engl. *moving average*) [191]. Takva se metoda često primjenjuje u reduciranju velikih oscilacija u podacima. Podaci se s komponenata aplikacije prate pomoću *posredničke komponente* aplikacijskoga okvira ElaClo. Podaci sa svih posredničkih komponenti se objedinjuju te omogućuju praćenje podataka nad trenutnom konfiguracijom aplikacije. Ako se aplikacijska komponenta koju pratimo nalazi na više virtualnih strojeva, računa se prosječno vrijeme odaziva po svakoj operaciji ili prosječno opterećenje procesora. Nadzorna jedinica ostvaruje praćenje kvalitete konfiguracije metodologijom iznijetom prethodnim istraživanjima [192, 193]. Sumirane podatke o opterećenju zadane konfiguracije koristi komponenta *upravljač resursa*.

Upravljač resursa

Kako bi ElaClo mogao ispitati kriterije kvalitete mora omogućiti elastično horizontalno skaliranje svake elastične grupe ciljane aplikacije. Tu funkcionalnost vrši komponenta **upravljač resursa**: koristeći podatke nadzorne jedinice koji predstavljaju trenutno opterećenje ciljane aplikacije, donosi odluke o potrebnom broju resursa oblaka koje je potrebno upotrijebiti. Akcije za skaliranje određene su gornjim i donjim granicama željenih praćenih metrika. Primjerice, može se odrediti da prosječno opterećenje procesora u elastičnoj grupi koje premašuje 70% pokreće zakup dodatnoga poslužitelja za tu grupu.

Sustav ElaClo omogućuje parametrizaciju upravljača resursa pomoću sljedećih postavki:

- Duljina intervala u kojem se računaju prosječne vrijednosti opterećenja procesora.
- Granične vrijednosti za pokretanje akcija zakupa (engl. *scale-in*) ili otpuštanja (engl. *scale-out*) računalnih resursa.
- Vrijeme čekanja nakon provođenja akcije zakupa dodatnih resursa u kojem se sustav stabilizira. Ova tehnika osigurava novo zakupljenim resursima potrebno vrijeme za dosizanje radne točke čime se se smanjuju oscilacije u broju korištenih resursa.

Ako postoji potreba za neke aplikacije da ostvare naprednije tehnike postizanja elastičnosti ElaClo omogućuje postavljanje korisničkih komponenti za upravljanje resursima u obliku proširenja (engl. *add-on*).

Upravljač konfiguracija

Ova komponenta zadužena je za ostvarivanje trenutne konfiguracije prema modelu iz poglavlja koristeći dostupne komponente aplikacije i infrastrukturu oblaka. Upravljač konfiguracija provodi grupiranje komponenti u elastične grupe prema funkciji $f_{C \to P}$ te vrši instalaciju komponenti elastičnih grupa na poslužitelje prema $f_{P \to W}$ i $f_{P \to R}$.

Optimizator i simulator

Komponente optimizatora implementiraju genetski algoritam iz poglavlja 5.3 pri čemu se koristi paket za oblikovanje genetskih algoritama DAEP[¶]. Komponenta vrši transformaciju modela konfiguracije ciljane aplikacije u simulacijski model za potrebe evaluacije svake simulirane konfiguracije.

Simulator je izveden nadogradnjom postojećeg simulatora Ciw-a[‖] u programskom jeziku Python. Nadogradnja je ostvarena u simulaciji postupka elastičnosti. Proces simulacije opisan je u poglavlju 5.5.1.

6.3 Izvođenje programske usluge u sklopu postupka optimizacije

Kako bi se programska usluga mogla ocijeniti potrebno ju je postaviti na infrastrukturu oblaka. To podrazumijeva integraciju programskih komponenti usluge s virtualiziranim sklopovljem infrastrukture. Zbog postizanja elastičnosti horizontalnim skaliranjem svaka se programska komponenta integrira s više sklopovskih komponenti — virtualnih strojeva infrastrukture u oblaku. Procesom integracije komponenti upravlja sustav ElaClo autonomno. Kako se ciljanu aplikaciju može postaviti u više različitih konfiguracija, sustav ElaClo dodatno mora omogućiti ispravno postavljanje usluge u željenu konfiguraciju, ostvarenje komunikaciju između komponenti te njihovu elastičnost. U sljedećim poglavljima iznose se metode pomoću kojih se navedeno ostvaruje.

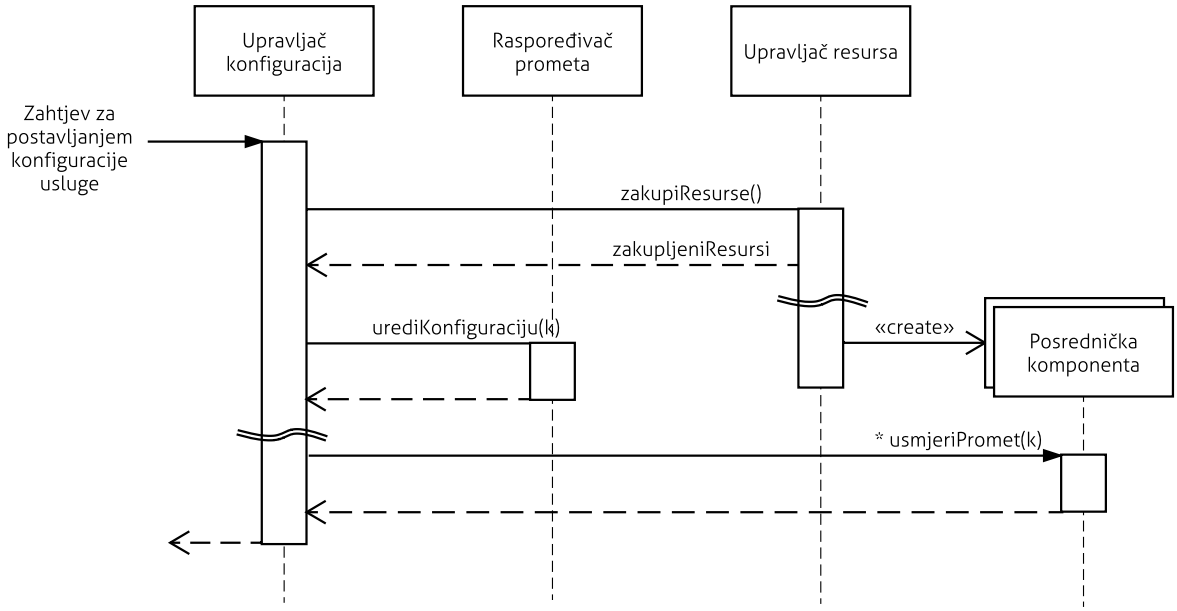
6.3.1 Postavljanje ciljane programske usluge na infrastrukturu oblaka

Postavljanje programske usluge u željenoj konfiguraciji k^M odvija se u tri faze:

- Zakup virtualnih strojeva te postavljanje u skladu sa zadanom slikom virtualnoga stroja
- Konfiguracija posredničke komponente na svim instancama komponenti ciljane aplikacije
- Konfiguracija raspoređivača prometa prema instancama komponenti ciljane aplikacije

[¶] DAEP je dostupan na *https://github.com/DEAP/deap*

[‖] *https://pypi.python.org/pypi/Ciw* - Simulator mreže redova čekanja *Ciw*



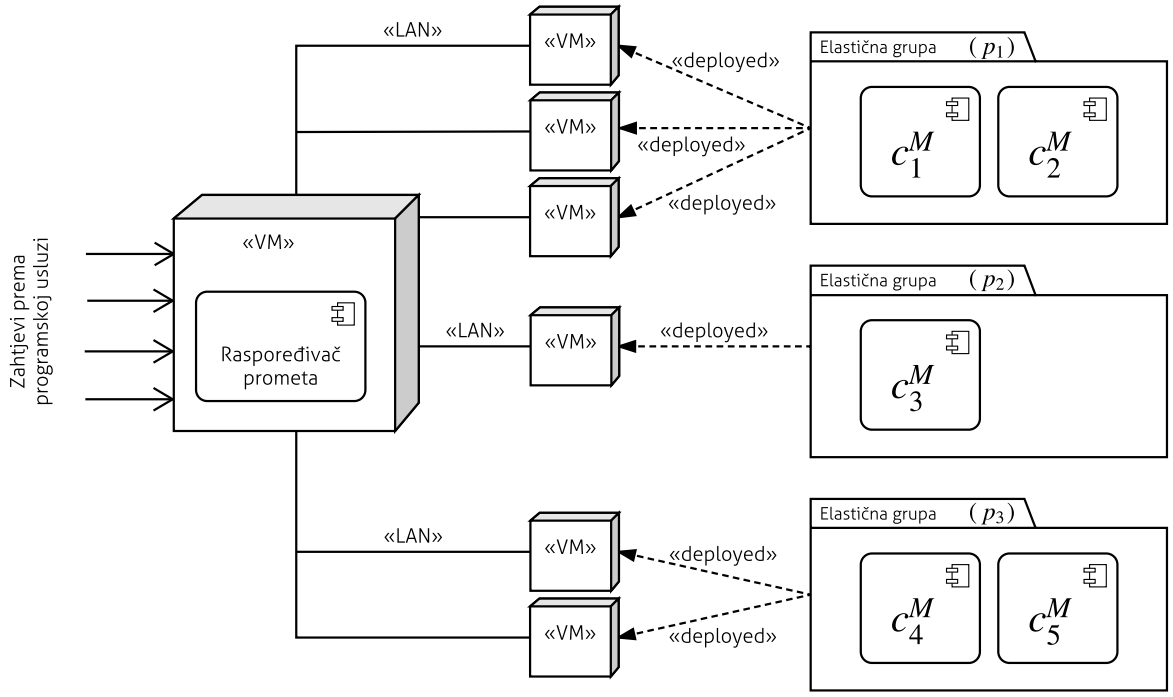
Slika 6.5: Postupak postavljanja konfiguracija programske usluge

Slika 6.5 prikazuje navedene korake pomoću slijednog dijagrama UML-a. U prvom koraku vrši se zakup i pokretanje virtualnih strojeva. U tu svrhu koristi se mrežna usluga davatelja infrastrukture. Različiti davatelji usluga poput Amazona[**], Rackspacea[††] ili Scalewaya[‡‡] pružaju aplikacijsko programsko sučelje (engl. *application programming interface*) za sve oblike upravljanja infrastrukturom što omogućuje ostvarivanje elastičnosti. ElaClo također koristi sučelja oblaka pri upravljanju infrastrukturom.

ElaClo omogućuje ocjenu svake generirane konfiguracije $k^M \in K^M$ programske usluge određenog modela M tako da osigurava elastičnost u skladu s trenutnom konfiguracijom k^M. Slika 6.6 prikazuje primjer raspodjele komponenti na virtualne strojeve unutar oblaka. Prikazana je programska usluga koja se sastoji od komponenti $C^M = \{c_1^M, c_2^M, c_3^M, c_4^M, c_5^M\}$ u konfiguraciji $k^M = \{p_0, p_1, p_2\},\ p_0 = \{c_1^M, c_2^M\},\ p_1 = \{c_3^M\},\ p_2 = \{c_4^M, c_5^M\}$. Elastičnost je ostvarena postojanjem tri regulacijska kruga MAPE-K za svaku pojedinu particiju p_1, p_2, p_3. Regulacijski krugovi implementirani su unutar ElaCla. Tako u svakom trenutku izvršavanja programske usluge dodijeljena je minimalno potrebna količina virtualnih strojeva $r \in R$. Skup R sadrži sve korištene virtualne strojeve u nekom trenutku te samim time $\sum_{r \in R} cost(r)$ određuje trenutni trošak izvođenja usluge prema modelu naplate. Razvojno okruženje ElaClo upravlja veličinom skupa R zakupom resursa koristeći programsko sučelje infrastrukture u oblaku u skladu sa zahtjevima svake skupine p.

Za raspodjelu zahtjeva prema komponentama na različitim resursima r koristi se raspoređivač prometa (engl. *load balancer*). Raspoređivač prometa usmjerava dolazne zahtjeve na

[**]Amazon Web Services - *https://aws.amazon.com/*
[††]Rackspace - *http://www.rackspace.com*
[‡‡]Scaleway - *https://www.scaleway.com/*



Slika 6.6: Primjer izvršavanja programske usluge u razvojnoj okolini ElaClo

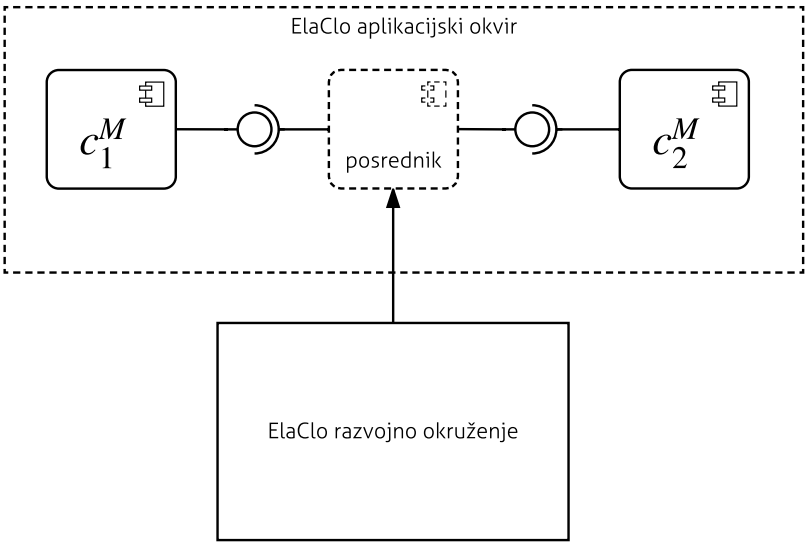
instance određene grupe p, a samim time i na svaki virtualni stroj $r \in R$ na kojem je trenutno postavljena grupa p. Tako se ukupno opterećenje raspodjeljuje na više resursa r što predstavlja implementaciju horizontalne skalabilnosti.

6.3.2 Komunikacija između komponenti

Kako bi se osigurala komunikacija između komponenti u bilo kojoj konfiguraciji koristi se aplikacijski okvir koji integriran u ciljnu aplikaciju služi za postavljanje komunikacijskog kanala između dviju komponenti. U tu svrhu koristi se posrednička komponenta koja se nalazi u sklopu aplikacijskog okvira. Posrednička komponenta određuje hoće li se komunikacija ostvariti direktnim pozivanjem metode druge komponente ili preko mrežne usluge, zavisno o trenutnoj konfiguraciji. Slika 6.7 prikazuje primjer komunikacije između dviju komponenti c_1^M i c_2^M koja se odvija preko posredničke komponente. Izvedbena okolina sustava ElaClo upravlja svim posredničkim komponentama te tako određuje komunikacijske tokove u skladu sa željenom konfiguracijom programske usluge.

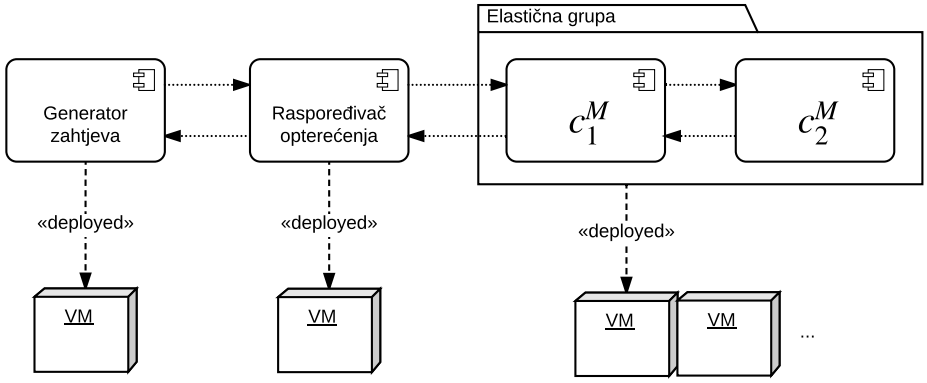
Komponente unutar iste grupe p međusobno komuniciraju preko istog adresnoga prostora na način koji je propisan komponentnim modelom. Slika 6.8a prikazuje lokalnu komunikaciju komponenti unutar grupe.

Ako se komponente nalaze u različitim grupama, tada one međusobno komuniciraju preko raspoređivača prometa koji ujedno vrši ulogu usmjeravanja prometa. Kako se u ovom slučaju komunikacija odvija putem lokalne mreže infrastrukture oblaka, potrebno je koristiti proto-

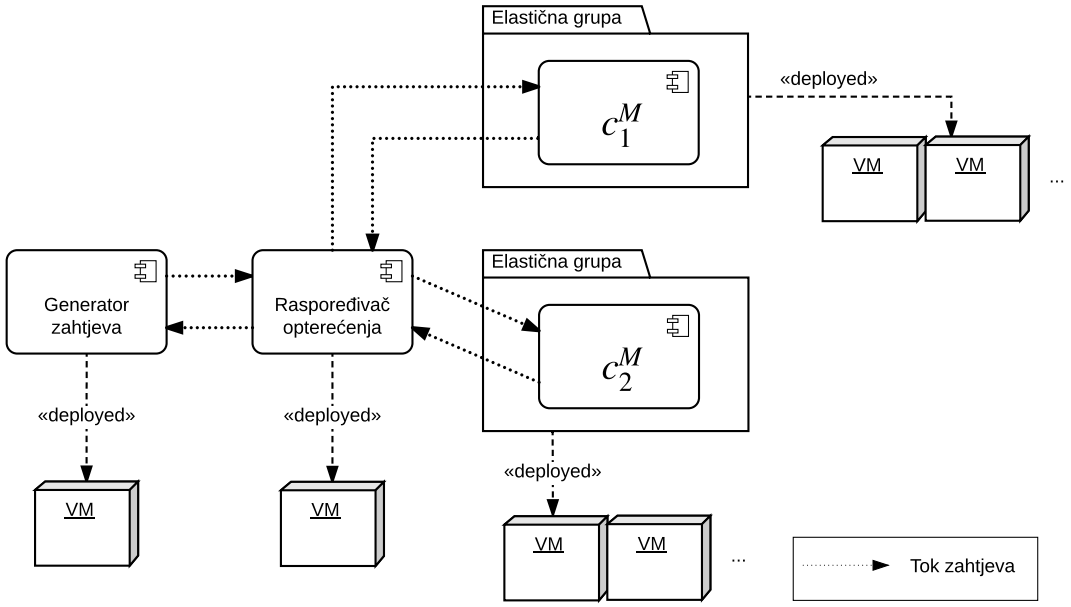


Slika 6.7: Posrednička komponenta programskog okvira ElaClo za povezivanje komponenti u skladu s željenom konfiguracijom

kole za udaljenu komunikaciju poput SOAP-a, RPC-a, ili već zavisno o korištenoj tehnologiji implementacije. Slika 6.8b prikazuje kako udaljene komponente iz različitih grupa komuniciraju putem raspoređivača opterećenja.



(a) Lokalna komunikacija unutar istog adresnog prostora



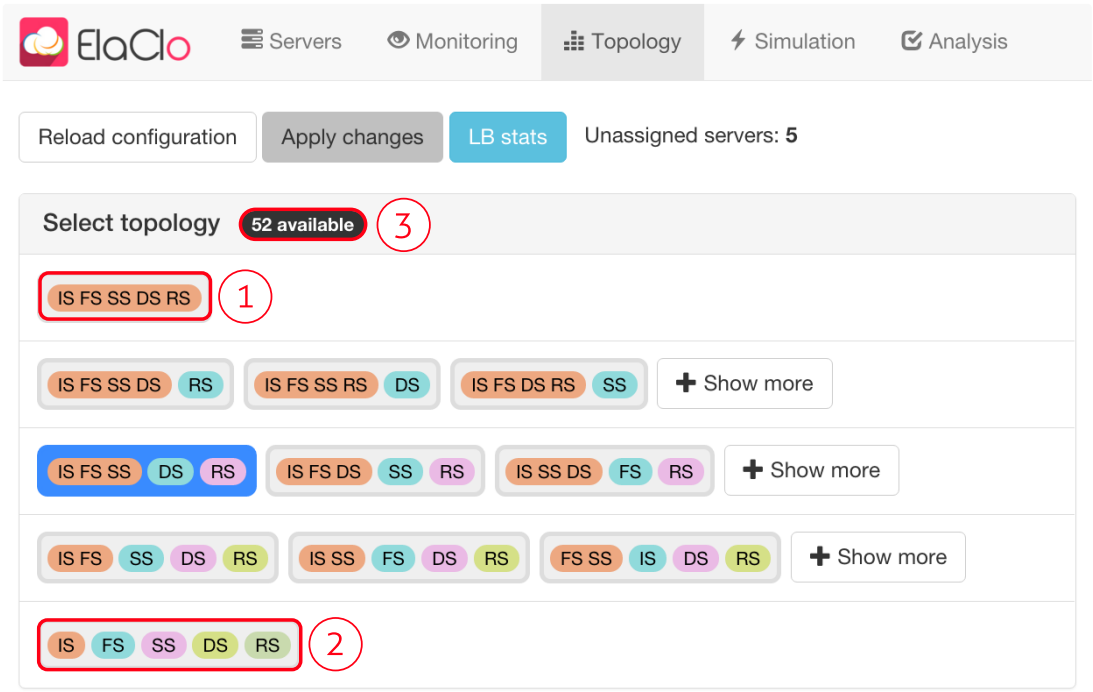
(b) Udaljena komunikacija između komponenti

Slika 6.8: Dvije moguće vrste komunikacije između komponenti k_1 i k_2 PUOP-a

6.4 Korisničko sučelje

Alat ElaClo omogućuje automatski ili ručni način rada. U automatskom načinu rada, nakon definiranja radnog opterećenja provodi se postupak optimizacije iznesen kroz poglavlja 4 i 5. Moguće je također ručno odabrati i evaluirati željenu konfiguraciju koristeći za to predviđeno korisničko sučelje opisano u nastavku.

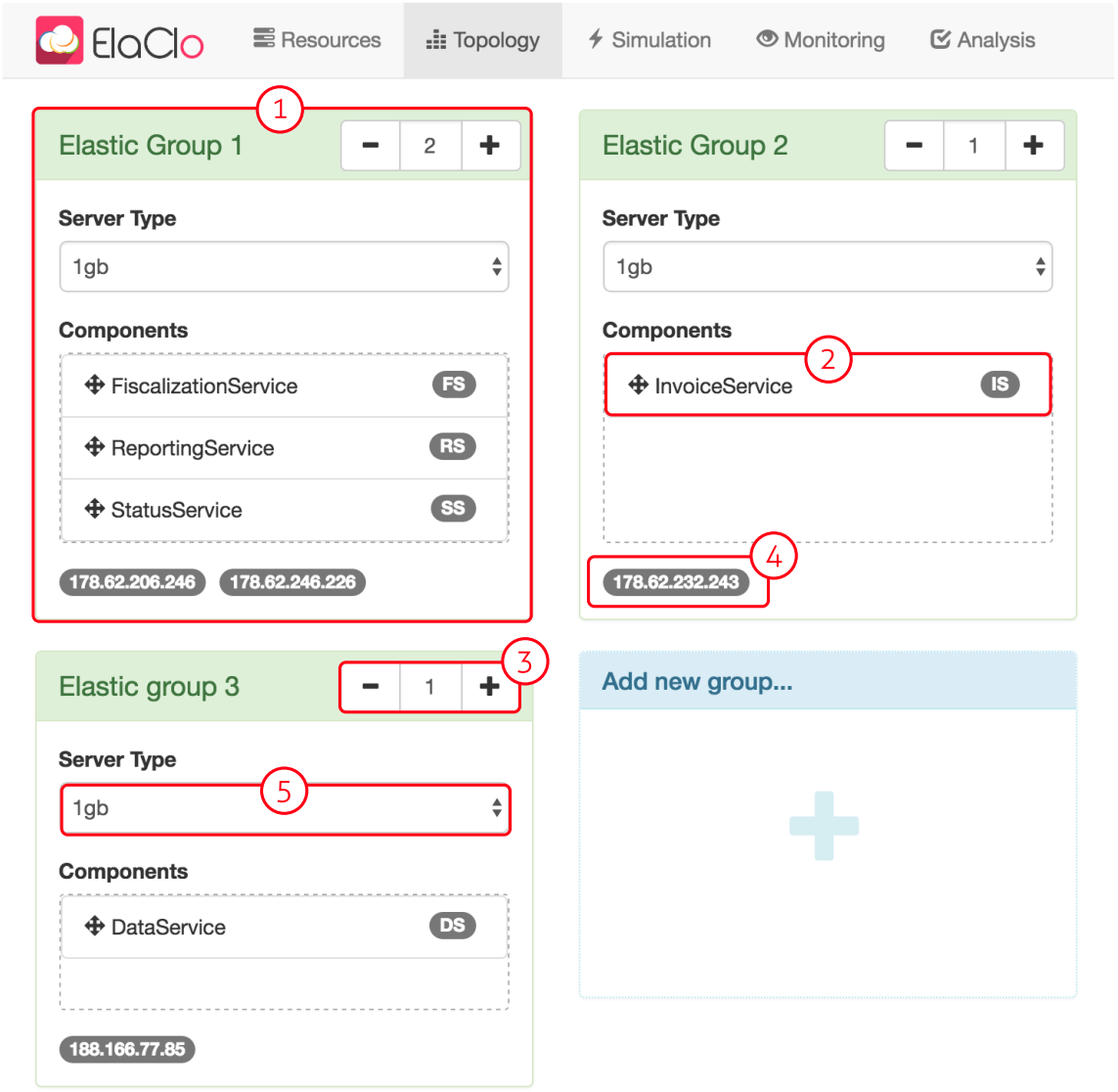
Slika 6.9 prikazuje optimalne konfiguracije iz simulacijskoga okruženja pripremljene za evaluaciju nad stvarnim sustavom. Konfiguracije su vizualno odvojene sukladno broju particija. Primjerice, (1) prikazuje konfiguraciju s jednom particijom, a (2) konfiguraciju s pet particija. Ukupan broj mogućih konfiguracija označen je s (3).



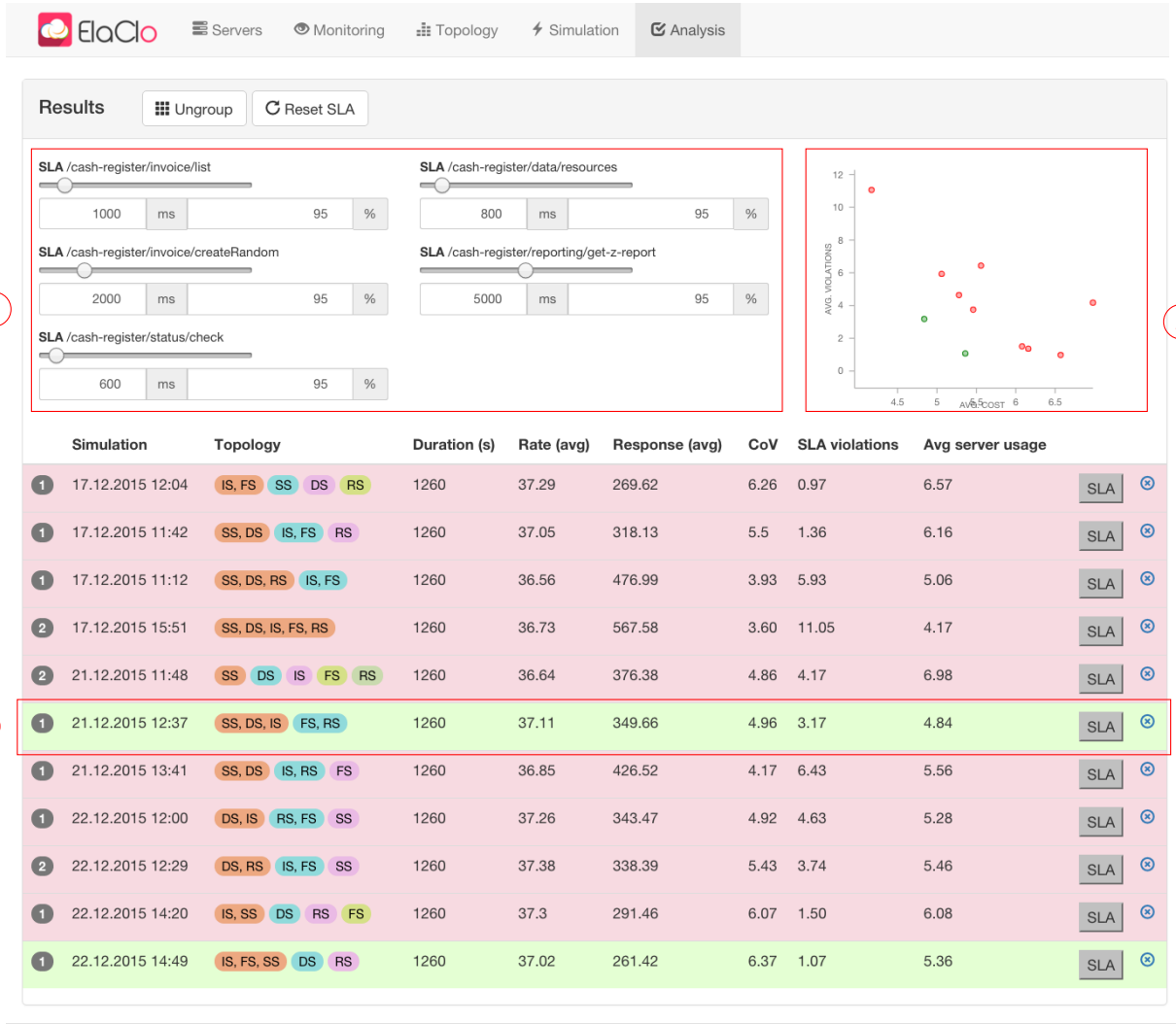
Slika 6.9: Korisničko sučelje razvojnog okruženja ElaClo za pregled i odabir generiranih konfiguracija

Slika 6.10 prikazuje korisničko sučelje za ostvarivanje ostalih željenih konfiguracija neovisno o simulacijskom postupku. Konfiguracija se određuje popunjavanjem komponenti u particije (1), tako da se komponente mogu premjestiti (2). Svaku particiju određuje i inicijalna količina poslužitelja (3) koja je podložna autonomnoj promjeni zbog elastičnosti. Za svaku particiju prikazane su mrežne adrese dodijeljenih virtualnih strojeva (4). Vrsta virtualnog stroja može se definirati za svaku grupu zasebno (5).

U konačnici, slika 6.11 prikazuje sučelje za analizu krajnje dobivenih konfiguracija. Sučelje omogućuje: (1) Dodatnu parametrizaciju modela ugovora SLA nakon optimiranja, (2) grafičku usporedbu ocjena i troškova konfiguracija (trošak nasuprot kvalitete u obliku broja prekršaja stavki SLA) te (3) listu svih konfiguracija koje udovoljavaju ugovor s mogućnošću sortiranja po raznim kriterijima.



Slika 6.10: Korisničko sučelje ElaClo razvojnog okruženja za podešavanje konfiguracije programske usluge



Slika 6.11: Korisničko sučelje razvojnog okruženja ElaClo za analizu rezultata.

Poglavlje 7

Validacija

Ovo poglavlje iznosi postupak validacije predstavljene metode optimiranja pomoću studije slučaja stvarnoga informacijskoga sustava. Postupak validacije strukturiran je prema dva ključna cilja: a) ispitati validnost predloženog postupka optimiranja konfiguracija te b) evaluirati radne karakteristike postupka optimizacije kvantitativnim metodama. U studiji slučaja koristi se informacijski sustav u oblaku *CashRegister* kojeg koristi više tisuća malih i srednjih organizacija na tri tržišta (Hrvatska, Slovenija i Češka). Odabran informacijski sustav predstavlja *tipični slučaj* [194] aplikacije uže namjene čime ispitujemo primjenjivost sustava ElaClo i sve srodne aplikacije. Za potrebe ovog istraživanja izvedena je replika spomenutog sustava s ograničenim brojem funkcionalnosti, ostvarena po modelu predstavljenom u poglavlju 3.4. Model radnog opterećenja izveden je analizom stvarnog radnog opterećenja *CashRegister* aplikacije [180].

U poglavlju 7.1 pobliže se opisuju ciljevi validacije te definiraju istraživačka pitanja. Poglavlje 7.2 predstavlja metodologiju validacije koja se primjenjuje u odgovaranju postavljenih pitanja. Poglavlje 7.3 iznosi detalje implementacije sustava ElaClo, dok se u poglavlju 7.4 opisuje *CashRegister* sustav korišten u studiji slučaja. Poglavlje 7.5 iznosi detalje provođenja optimizacije nad *CashRegister* aplikacijom te daje odgovore na postavljena istraživačka pitanja. Naposljetku, poglavlje 7.5.4 iznosi ograničenja provedenog istraživanja.

7.1 Ciljevi validacije

Cilj optimizacijskog postupka predloženog u ovom radu jest pružiti arhitektu informacijskoga sustava automatizirani postupak utvrđivanja optimalne konfiguracije zavisne o željenom SLA-u. Dosadašnja istraživanja razlikuju nekoliko razina validacije sustava za predikciju kriterija kvalitete sustava modeliranjem koje možemo primijeniti pri validaciji optimiranja simulacijskim modelima [195]:

1. **Validacija preciznosti** gdje se uspoređuju relevantne metrike dobivene pomoću mo-

dela s metrikama stvarnog sustava. U kontekstu ovoga rada — provedbe automatiziranog postupka pronalaska optimalnih kandidata potrebno je za mjeru preciznosti izraziti preciznost rezultata postupka optimizacije.

2. **Validacija primjenjivosti** ispituje je li korisnik sustava u mogućnosti provesti postupak automatizacije. U ovom slučaju, kako se radi o automatiziranom postupku, dovoljno je ispitati je li korisnik u mogućnosti zadovoljiti sve preduvjete za provedbu optimizacije. Kako sustav ElaClo automatski utvrđuje model aplikacije na korisniku je da primijeni aplikacijski okvir opisan u poglavlju 6.1.
3. **Validacija isplativosti** ispituje isplativost postupka uključujući sve troškove koji sudjeluju u provedbi postupka. Ovo je ujedno i najzahtjevniji dio validacije zbog velike količine faktora koje je potrebno sagledati te se u sklopu ovoga rada nije provelo takvo istraživanje.

U skladu s validacijom preciznosti potrebno je ispitati jesu li kandidati dobiveni postupkom optimizacije doista optimalni kandidati. Kao što je prethodno izneseno, postupak optimizacije u ovom radu provodi se u dvije faze: a) evolucijskim algoritmom uz pomoć metode simulacije za predlaganje skupa kandidata za daljnju evaluaciju nad stvarnim sustavom te b) evaluacija nad stvarnim sustavom pomoću alata ElaClo. Kako izvedbena okolina ElaClo ocjenjuje konfiguracije nad stvarnim sustavom u oblaku ne postoji postupak predikcije u tom koraku te je potrebno provesti samo ispitivanje preciznosti rezultata dobivenih pomoću evolucijskog algoritma i simulacijskog okruženja. U skladu s time postavljaju se sljedeća istraživačka pitanja:

RQ1: Koliko je prosječno odstupanje rezultata evolucijskog algoritma od optimalnog Pareto skupa rješenja?

RQ2: U kojoj su mjeri predloženi evolucijski algoritam i postupak simulacije odgovarajući surogatni model stvarnoj aplikaciji u oblaku?

Sustav ElaClo predložene konfiguracije genetskog algoritma ispituje iterativno koristeći infrastrukturu u oblaku. Vrijeme izvođenja evaluacije pojedine konfiguracije određeno je trajanjem modela radnog opterećenja (poglavlje 4.1). Kako bi se kvalitetno ispitalo elastično ponašanje sustava, potrebno je odrediti opterećenje u trajanju od barem nekoliko sati za svaku konfiguraciju. Samim time moguće je provesti evaluaciju manjeg broja konfiguracija što znači da evolucijski algoritam mora odrediti relativno uzak set predloženih kandidata. Broj konfiguracija koje polučuje postupak evolucijskog algoritma zavisi o odabranom parametru veličine populacije μ.

Samim time efikasnost provođenja postupka optimizacije određena je brzinom konvergencije evolucijskog algoritma čime izvodimo sljedeće istraživačko pitanje:

RQ3: Kakve su vremenske karakteristike evolucijskog algoritma u određivanju optimalnih konfiguracija metodom simulacije?

7.2 Metodologija validacije

Kao mjeru koja kvantificira preciznost pojedinih rezultata u svrhu odgovaranja na **RQ1** koristi se mjera *pokrivenosti*:

$$\mathscr{C}(\mathbb{K}^*_{result}, \mathbb{K}^*_{true})$$

predstavljene u istraživanju [196]. Mjera pokrivenosti koristi se za usporedbu pojedinog Pareto skupa dobivenog kao rezultat izvođenja evolucijskog algoritma $\mathbb{K}^*_{result}$ koja čine Pareto front PF_{result} u odnosu na pravi skup Pareto rješenja $\mathbb{K}^*_{true}$ i fronta PF_{true}. Mjera pokrivenosti izražava omjer dominantnih rezultata između dva skupa, $\mathbb{K}^*_{result}$ u odnosu na $\mathbb{K}^*_{true}$. Za konfiguraciju K_A kažemo da dominira konfiguraciju K_B ako po svim promatranim kriterijima kvalitete za K_A vrijedi da su bolji nego kriteriji koje postiže K_B što naznačujemo kao

$$K_A \succ K_B.$$

Zbog stohastičke prirode simulacije kriterija kvalitete u postupku optimizacije, mjera kvalitete varira kroz različita provođenja eksperimenata te zbog toga uvodimo modificiranu mjeru pokrivenosti koja konstatira zastupljenost rezultata iz $\mathbb{K}^*_{result}$ u $\mathbb{K}^*_{true}$:

$$\mathscr{C}^*(\mathbb{K}^*_{result}, \mathbb{K}^*_{true}) = \frac{|\mathbb{K}^*_{result} \cap \mathbb{K}^*_{true}|}{|\mathbb{K}^*_{true}|} \in [0,1].$$

Mjera $\mathscr{C}^*$ označava koliki je udio pojedinog rezultata izvođenja zastupljen u rješenju $\mathbb{K}^*_{true}$ gdje slučaj kada je $\mathscr{C}(\mathbb{K}^*_{result}, \mathbb{K}^*_{true}) = 1$ označava $\mathbb{K}^*_{result} = \mathbb{K}^*_{true}$.

Minimalna mjera pokrivenosti $\mathscr{C}^*$ koju je potrebno ostvariti pojedinim izvođenjem evolucijskog algoritma postavlja se na 0.7 što znači da prosječno izvođenje evolucijskog algoritma mora dati barem 70% kandidata iz pravog skupa rješenje $\mathbb{K}^*_{true}$. U tu svrhu testira se hipoteza jednostranim Studentovim t-testom nulte hipoteze da prosječna vrijednost indikatora $\mathscr{C}^*$ iznosi 0.7, uz alternativnu hipotezu da je prosječna vrijednost veća od 0.7 te p-vrijednost manju od 1%. Preduvjet Studentovog t-testa je normalna distribucija testirane varijable što se dodatno provjerava Kolmogorov–Smirnovim testom.

Pravi skup Pareto rješenja PF_{true} i pripadnih konfiguracija $\mathbb{K}^*_{true}$ nije unaprijed poznat te ga je zbog velikog prostora pretrage, računske zahtjevnosti i stohastičke prirode provođenja simulacije nije moguće egzaktno odrediti. Aproksimacija istog određuje se višestrukim ponavljanjem postupka optimizacije te određivanjem skupa Pareto rješenja od svih dobivenih rezultata što je uobičajeni postupak u stohastičkoj višekriterijskoj optimizaciji [197].

Dobivanje egzaktnog odgovora na **RQ2** zahtjeva provođenje višestrukih ponavljanja evaluaciju svake pojedine konfiguracije te usporedbu rezultata iz simulacijske okoline s rezultatima nad stvarnom okolinom. Kako evaluacija svakog kandidata u stvarnom okruženju traje više sati te zbog velikog prostora pretrage to nije izvedivo. Zbog toga se primjenjuje alter-

nativna metoda odgovaranja na **RQ2**: provode se dodatni postupci optimizacije, ali s obrnutim ciljevima za svaki kriterij kvalitete što rezultira setom rješenja $\mathbb{K}^*_{inverse}$. Takva provedba optimizacije također pruža dodatne odgovore: mogu li veća ulaganja u trošak polučiti bolje rezultate te obrnuto, mogu li lošija rješenja polučiti niže troškove te ukazati na eventualno postojanje rješenja s povećanim troškovima i niskom razinom kvalitete. Rezultati optimizacije trebali bi dominirati tako dobivena rješenja.

Oba skupa rješenja: $\mathbb{K}^*_{inverse}$ te prethodni skup $\mathbb{K}^*_{result}$ evaluiraju se pomoću alata ElaClo nad stvarnim okruženjem što pridružuje stvarno ostvarene kriterije kvalitete svakoj konfiguraciji: $f_{cloud}(\mathbb{K}^*_{result})$ i $f_{cloud}(\mathbb{K}^*_{inverse})$. Tada se utvrđuje omjer dominantnih rješenja iz $\mathbb{K}^*_{inverse}$ nad rješenjima iz $\mathbb{K}^*_{result}$ te ispituje je li taj omjer statistički značajniji od omjera dominantnih konfiguracija iz $\mathbb{K}^*_{result}$ nad $\mathbb{K}^*_{inverse}$. Ako konfiguracija K_A dominira konfiguraciju K_B prema kriterijima kvalitete utvrđenim nad stvarnim sustavom označujemo to sa:

$$K_A \overset{*}{\succ} K_B.$$

Kako bi kvantificirali omjer dominantnosti između tih dvaju skupova, uvodimo mjeru Ocjena-Dominantnosti (OD) za pojedinu konfiguraciju T u odnosu na set konfiguracija $\mathbb{K}$:

$$OD(T,\mathbb{K}) = \sum_{T_i \in \mathbb{K}} f_{DOM}(T,T_i)$$

gdje je

$$f_{DOM}(K_A,K_B) = \begin{cases} 1 & \text{if } K_A \overset{*}{\succ} K_B \\ 0 & \text{otherwise} \end{cases}.$$

OD izražava količinu konfiguracija iz nekog skupa koje su dominirane promatranom konfiguracijom. Samim time postavljamo hipotezu da je prosječna vrijednost OD za konfiguracije iz $\mathbb{K}^*_{result}$ naspram $\mathbb{K}^*_{inverse}$ veća od prosječne vrijednosti OD u obrnutom slučaju.

Ta se hipoteza ispituje Mann–Whiteneyevim U testom [198], poznatim kao Wilcoxonovim *rank-sum* testom gdje se za nultu hipotezu uzima da je vrijednost OD za nasumičnog kandidata iz prve grupe veća od nasumičnog kandidata iz druge grupe. Statistička značajnost postavlja se na $p < 0.01$. Prikazuje se i parametar veličine učinka (engl. effect-size) [199] za bolje intuitivno tumačenje rezultata.

U svrhu odgovaranja na **RQ3** određuje se minimalni broj iteracija genetskog algoritma koji ispunjava pokrivenost od najmanje 0.7 uz p-vrijednost manju od 1%. Također, prikazuje se prosječna vrijednost i odstupanje indikatora pokrivenosti tijekom iteracija algoritma te prosječna vrijednost kriterija kvalitete pripadnih populacija iz svake iteracije.

7.3 Implementacija prototipnog alata ElaClo

ElaClo razvojno okruženje ostvareno je kao mrežna aplikacija za optimiranje strukture aplikacija ostvarenih pomoću komponentnog modela *JavaEE*.

Pri izradi mrežne usluge za generiranje prometa korišten je programski jezik Python te knjižnica *Tornado*[*] za slanje mrežnih zahtjeva prema aplikaciji protokolom HTTP.

Upravljač konfiguracijom ostvaren je programskim jezikom PHP, a kao raspoređivač prometa upotrebljava se postojeća komponenta Haproxy[†] u inačici 1.6. Upravljač je ostvaren tako da omogućuje automatsku promjenu konfiguracije Haproxy raspoređivača prometa u tijeku izvođenja.

Upravljač resursima također je mrežna usluga ostvarena programskim jezikom Python koji koriste mrežne usluge davatelja infrastrukture u oblaku za automatsku prilagodbu konfiguracije u skladu s potrebama izvođenja različitih mjerenja ili prilagodbi trenutnom opterećenju aplikacije. Trenutno je implementirano upravljanje nad davateljem usluge *DigitalOcean*[‡].

Sučelje mrežne aplikacije ElaClo ostvareno je pomoću tehnologije *HTML5* te knjižnice *Knockout*[§] za programski jezik JavaScript koji je integralni dio tehnologije HTML5. Odlika knjižnice Knockout jest mogućnost ostvarivanje elemenata korisničkog sučelja u *Model-View-ViewModel* paradigmi [200]. Za iscrtavanje grafikona korištena je knjižnica *jQuery flot*[¶]. Elementi korisničkog sučelja ostvareni su knjižnicom *Bootstrap*[‖].

7.4 Studija slučaja

Informacijski sustav *CashRegister* je rješenje ostvareno po principu programske podrške kao usluge (engl. *Software-as-a-Service*, skraćeno SaaS). CashRegister je dizajniran pomoću klijentsko-poslužiteljske arhitekture u višekorisničkom načinu rada (engl. *multi-tenant application*) gdje istu instalaciju aplikacije koristi nekoliko tisuća manjih i srednjih organizacija na hrvatskom, slovenskom i češkom tržištu.

Cjelokupni informacijski sustav CashRegister ostvaren je u tri sloja komponenti. Komponente prezentacijskog sloja implementirane su u sklopu *Android*, *iPhone* i *Windows PC* aplikacija koje komuniciraju s komponentama srednjeg aplikacijskog sloja realiziranih u obliku mrežnih usluga. Komponente podatkovnog sloja ne ulaze u postupak optimizacije jer su ostvarene pomoću relacijskih baza podataka koje u potpunosti ne posjeduju svojstvo elastičnosti

[*] Knjižnica *Tornado* dostupna je *https://pypi.python.org/pypi/tornado*

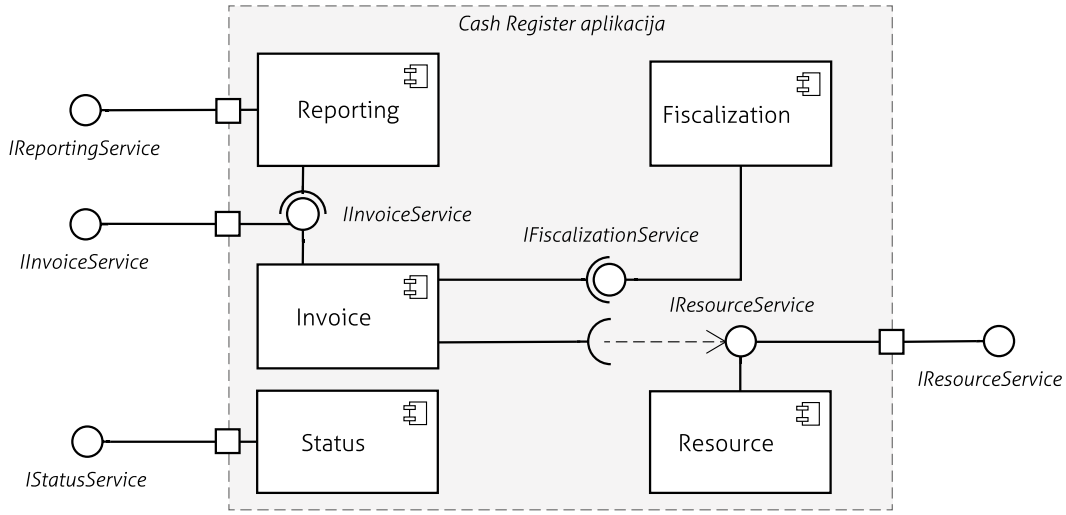
[†] Haproxy komponenta dostupna je na *http://www.haproxy.org*

[‡] DigitalOcean - davatelj infrastrukture u oblaku s internetskim sjedištem na adresi *http://www.digitalocean.com*

[§] Knockout knjižnica dostupna je na *http://knockoutjs.com/*

[¶] jQuery flot knjižnica dostupna je na *http://www.flotcharts.org/*

[‖] Bootstrap knjižnica dostupna je na *http://www.flotcharts.org/*



Slika 7.1: Komponente *CashRegister* sustava

već su kapacitirane prema najvećem opterećenju sustava. U postupak optimizacije ulaze komponente aplikacijskog sloja koje pružaju mrežne servise klijentskim komponentama prezentacijskog sloja. Slika 7.1 prikazuje komponente koje su ostvarene u replici stvarnoga sustava za potrebe evaluacije sustava *ElaClo*:

- komponenta *Reporting* pruža mrežne usluge vezane uz pregled izvještaja o poslovanju poput dnevnih i mjesečnih rekapitulacija
- komponenta *Invoice* služi za zaprimanje dokumenata poput računa ili ponuda izrađenih na uređajima klijentima. Komponenta također pruža operacije pregleda izrađenih dokumenata
- komponenta *Status* pruža mrežne usluge za sinkronizaciju postavki između komponenti prezentacijskog sloja i aplikacijskog sloja
- komponenta *Fiscalization* vrši fiskalizaciju izdanih računa pomoću mrežnih usluga ministarstva financija
- komponenta *Resource* pruža mrežne usluge za upravljanje zapisima o artiklima za prodaju.

Cilj studije slučaja jest prikazati primjenu predstavljenih metoda, algoritama i alata na postupku optimizacije konfiguracija sustava CashRegister. Studija također definira SLA koji je definiran višegodišnjim praćenjem rada korisnika na sustavu i njihovih zahtjeva u brzini odaziva sustava. Iako stvarna instalacija CashRegister sustava nema definirane sporazume o razini korištenja s korisnicima, želja je davatelja usluge unaprijediti uslugu SLA-om. U tu svrhu provodi se analiza postupka pronalaska optimalnih konfiguracija tako da se minimiziraju troškovi najma infrastrukture i broj prekršaja pojedinih stavki sporazuma.

7.4.1 Definiranje sporazuma SLA

Sustav CashRegister klijenti upotrebljavaju kao kritičan dio poslovanja gdje svaka nemogućnost izdavanja računa znači gubitak prihoda. U skladu s time definiran je prijedlog SLA prema modelu iz poglavlja 4.2 prikazan tablicom 7.1. Za sve stavke ugovora definiran je 95-percentil ($\tau_{\%} = 0.95$) od t_{SLA} što znači da u 95% slučajeva vrijeme odaziva mora biti manje od t_{SLA}. Iznimku predstavlja scenarij UC_3 čija je pouzdanost kritična te je za njega definiran 98-percentil.

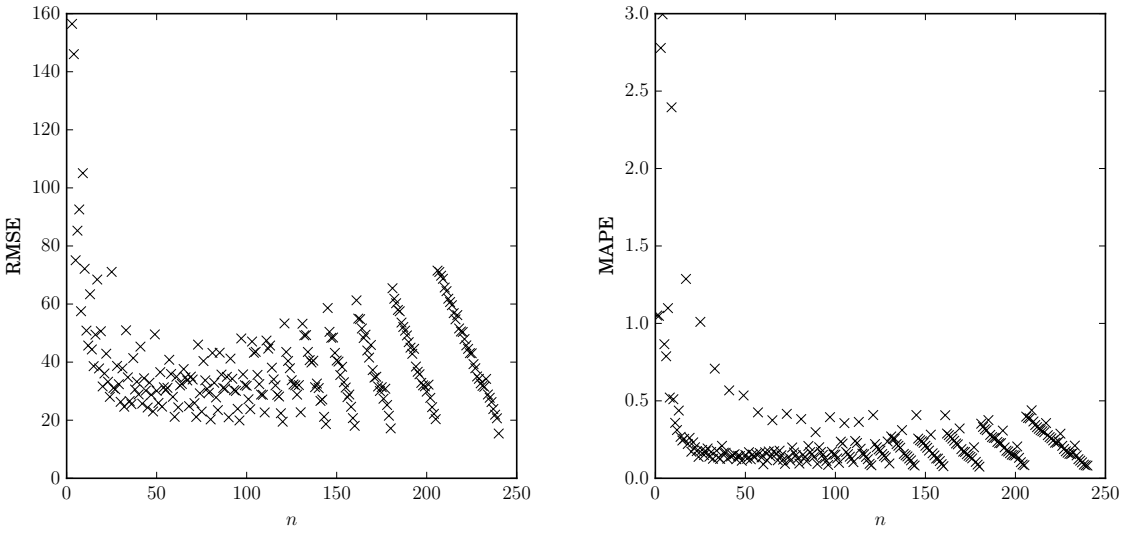
Scenarij	**Komponenta**	**Opis**	t_{SLA}(ms)	$\tau_{\%}$
UC_1	Invoice	Dohvat računa	500	0.95
UC_2	Resource	Dohvat podataka o artiklima i partnerima	500	0.95
UC_3	Invoice	Izrada računa	2500	0.98
UC_4	Report	Dnevni izvještaj o radu	4000	0.95
UC_5	Status	Dohvat stanja sustava	300	0.95

Tablica 7.1: Prijedlog ugovora SLA po scenarijima korištenja

7.4.2 Radno opterećenje

Promjenjivo radno opterećenje dobiveno je mjerenjem intenziteta korištenja stvarnog sustava te modeliranjem intenziteta $\lambda(t)$ kao funkcije linearne po odsječcima (engl. *piecewise-linear function*). Iz zapisa o radu stvarnog sustava (engl. *log*) prikupljeno je preko 13 milijuna korisničkih zahtjeva upućenih prema sustavu u razdoblju od 40 dana. Pomoću prikupljenih podataka određeno je prosječno dnevno opterećenje po vremenskim intervalima od jedne minute, dakle ukupno 1440 intervala za 24 sata. Prema prosječnom dnevnom opterećenju napravljena je linearna interpolacija po odsječcima $s_1, s_2, ..., s_n$ za ukupno n odsječaka. Broj odsječaka n odabran je tako da bude dovoljno velik da greška između modela i stvarnog opterećenja bude adekvatno mala. Slika 7.2 prikazuje odnos između broja odsječaka n i korijena srednje kvadratne pogreške (*Root Mean Square Error*, skraćeno RMSE) i srednje apsolutne postotne pogreške (engl. *Mean Absolute Percentage Error*, skraćeno MAPE) između modela i stvarnog intenziteta opterećenja.

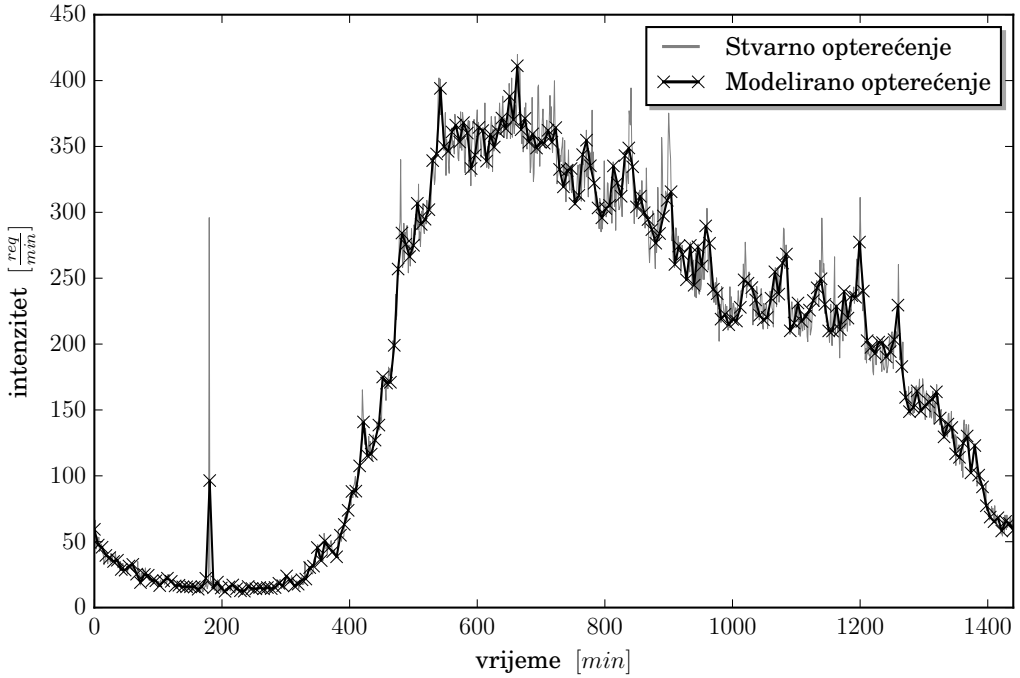
Pri modeliranju radnog opterećenja odabrana je interpolacija na $n = 240$ odsječaka, a rezultirajući se model koristi kao funkcija intenziteta $\lambda(t)$ u modelu opterećenja prilikom izvođenja simulacija i stvarnih mjerenja u izvedbenoj okolini. Slika 7.3 prikazuje stvarno prosječno dnevno opterećenje uz model linearne interpolacije uz $n = 240$.



(a) RMSE u odnosu na n

(b) MAPE u odnosu na n

Slika 7.2: Pogreška između modela i stvarnog intenziteta u odnosu na odabrani broj odsječaka n



Slika 7.3: Usporedba izmjerenog stvarnog opterećenja i modela opterećenjae

7.4.3 Cilj optimizacije

Cilj optimizacije u studiji slučaja jest odrediti konfiguracije koje zadovoljavaju traženi sporazum SLA ili ostvaruju još bolji SLA s minimalnim troškovima zakupa infrastrukture u oblaku. Ako ne postoji konfiguracija koja zadovoljava zahtjeve potrebno je pronaći najbolje kandidate te u skladu s njima predložiti nove parametre sporazuma.

7.5 Postupak optimizacije

Aplikacija *CashRegister* sastoji se od pet komponenti. U optimizaciji se razmatraju tri vrste poslužitelja prema tablici 7.3 što znači da ukupan broj konfiguracija iznosi 1561 (prema formuli 3.2). Ručno postavljanje sustava i mjerenje radnih karakteristika u svakoj mogućoj konfiguraciji zahtijevalo bi oko 400 inženjer-dana pod pretpostavkom da jedan ciklus postavljanja konfiguracije i evaluacije traje dva sata. Također, iako metoda iz 4. poglavlja pruža automatizaciju procesa konfiguracije i ocjene radnih karakteristika, za svih 1561 konfiguracija trajala bi 65 dana (u neprekidnom izvođenju) zbog čega je potrebno primijeniti postupak višekriterijske optimizacije evolucijskim algoritmom iz 5. poglavlja čime se određuje manji broj kandidata za analizu u stvarnoj okolini.

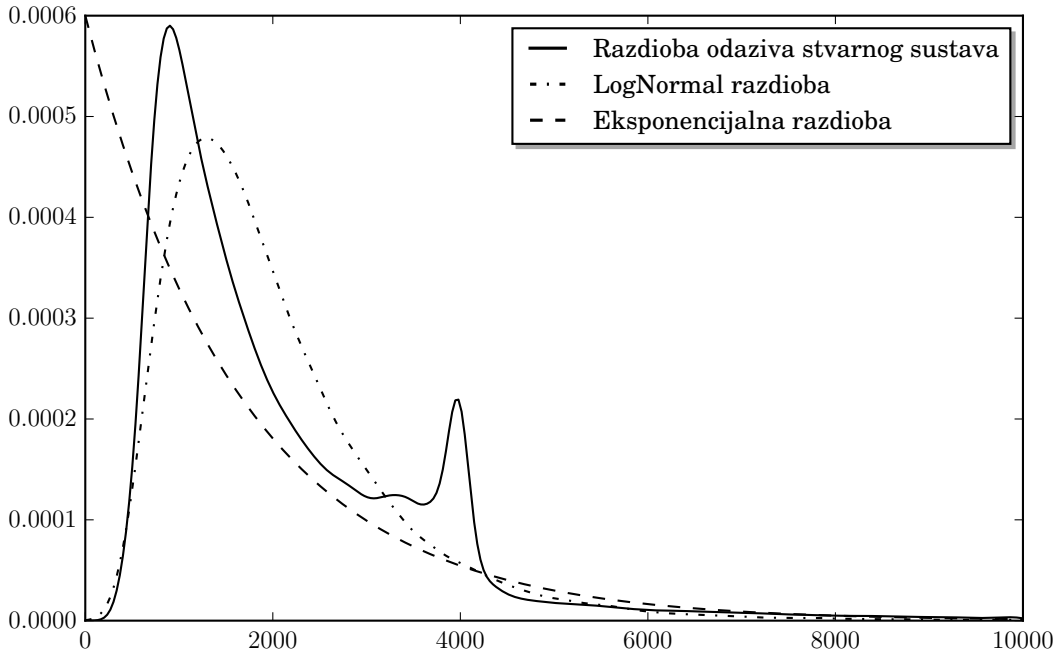
Za modeliranje vremena odaziva pojedinih operacija koje pružaju komponente odabrana je log-normalna distribucija koja puno vjernije reprezentira distribucije vremena odaziva kod informacijskih sustava od često korištene eksponencijalne distribucije [201]. Slika 7.4 prikazuje usporedbu distribucije vremena odazive za metodu *SaveInvoice* komponente *Invoice* određenu prema setu podataka dobivenim mjerenjem stvarnog sustava kroz 40 dana rada. Na istoj slici prikazana je i log-normalna distribucija korištena u simulaciji te najbliža eksponencijalna distribucija. Može se primijetiti kako eksponencijalna distribucija ne uspijeva dobro modelirati vjerojatnosti za vrijeme odaziva ispod 1000 *ms*. Log-normalna distribucija u simulaciji za sve je metoda parametrizirana tako da se izjednači prvi i drugi moment distribucije: srednja vrijednost i varijanca.

Vremena odaziva pojedinih operacija komponenata aplikacije *CashRegister* korištena prilikom simulacije prikazani su tablicom 7.2. Log-normalna distribucija vremena odaziva određena je s dva parametra:

$$\ln \mathcal{N}(\mu, \sigma^2),$$

parametar lokacije $\mu \in \mathbb{R}$ i skale $\sigma > 0$ koji su određeni prema prvom i drugom momentu izmjerene distribucije vremena odaziva metoda stvarnog sustava

$$\mu = \ln\left(\frac{m^2}{\sqrt{v+m^2}}\right), \sigma = \sqrt{\ln\left(1+\frac{v}{m^2}\right)},$$



Slika 7.4: Usporedba distribucije vremena odaziva za metodu *SaveInvoice* između stvarnog sustava i modela sa lognormalnom i eksponencijalnom distribucijom

gdje je m predstavlja mjerenu srednju vrijednost, a v varijanca vremena odaziva stvarnog sustava.

Tablica 7.2: Parametri lognormalne distribucije vremena odaziva pojedinih operacija

Operacija	**Komponenta**	**Opis**	μ	σ
CreateInvoice	*Invoice*	Izrada računa	-0.39	0.76
Fiscalize	*Fiscalization*	Fiskalizacija računa	-1.12	0.64
GetStatus	*Status*	Sinkronizacija podataka	-2.58	1.39
GetReport	*Report*	Dohvat izvještaja	0.49	0.71
InvoiceList	*Invoice*	Dohvat računa	-1.13	0.94
GetResources	*Resource*	Dohvat artikala	-1.99	1.46

U nastavku, poglavlje 7.5.1 iznosi postupak optimizacije predloženim evolucijskim algoritmom u reduciranju početnog broja kandidata na set najpovoljnijih kandidata, a poglavlje 7.5.2 iznosi validaciju navedenog postupka. Evaluacija predloženih kandidata u stvarnoj okoline opisana je u poglavlju 7.5.3.

Tablica 7.3: Vrste virtualnih poslužitelja korištenih u optimizaciji

Vrsta	Naziv	# CPU	RAM [MB]	Cijena/h
VM_1	Small	1	1	0.015
VM_2	Medium	2	2	0.03
VM_3	Large	4	4	0.06

7.5.1 Primjena evolucijskog algoritma u reduciranju broja kandidata

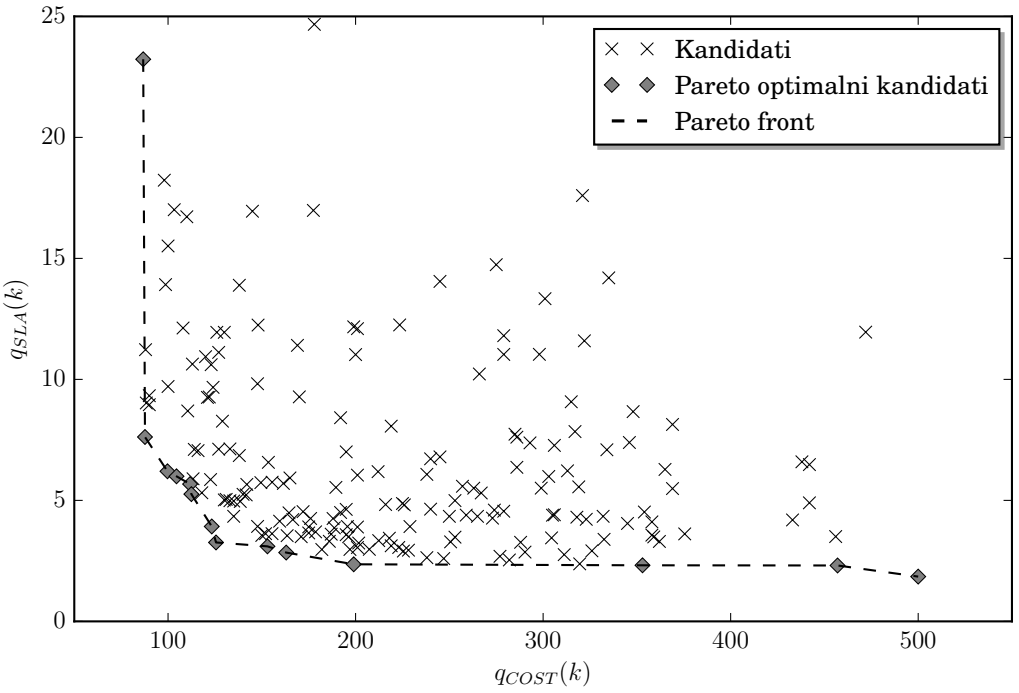
Pomoću implementiranog evolucijskog algoritma prema postupku iz poglavlja 5.3 određen je set od 11 kandidata. Tablica 7.4 prikazuje dobivene konfiguracije unutar seta Pareto optimalnih rješenja koje se dalje evaluiraju pomoću sustava ElaClo. Parametri evolucijskog algoritma postavljeni su na sljedeće vrijednosti:

$$\mu = 30,\ \lambda = 12,\ n = 100\ .$$

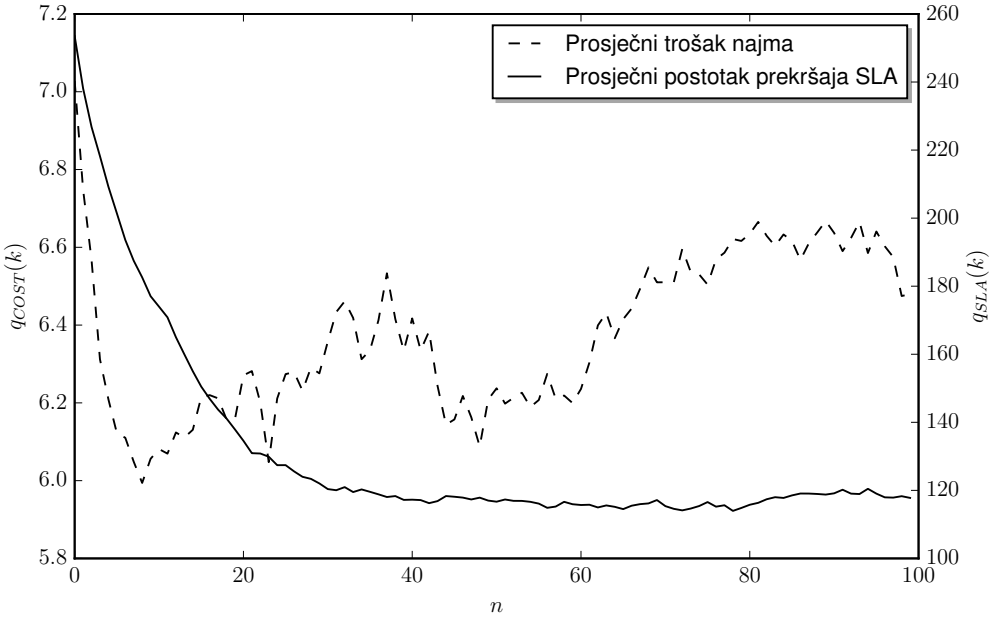
Prosječno vrijeme izvođenja postupka na računalu s jednom procesorskom jezgrom iznosilo je 12.5 sati. Izvođenje postupka na računalu s više jezgri može ubrzati postupak posljedično tome da se procesorsko vrijeme uglavnom troši na evaluaciju kandidata. Samim time postupak se može ubrzati korištenjem broja jezgri do veličine parametra λ koji označava maksimalan broj novih evaluacija po generaciji. Slika 7.6 prikazuje prosječne vrijednosti kriterija kvalitete populacije unutar svake generacije gdje se može zamijetiti napredak kod budućih generacija.

Slika 7.5 prikazuje iznose simuliranih kriterija kvalitete između različitih kandidata: q_{SLA} - udio zahtjeva čije je vrijeme izvođenja veće od vremena t_{SLA} izraženih tablicom 7.1 te q_{COST} - ukupni trošak najma infrastrukture u obliku virtualnih strojeva u sklopu simulacije. Između svih evaluiranih kandidata posebno su naznačeni kandidati unutar skupa Pareto optimalnih rješenja.

Među dobivenim rezultatima možemo primijetiti da je za opterećenje i komponente u ovoj studiji slučaja poželjnije koristiti manje kapacitirane virtualne strojeve s manje procesorskih jezgara čime se može uštedjeti pri niskim opterećenjima, a dodatno omogućuju i preciznije ostvarenje elastičnosti čime raspoloživi kapacitet bliže odgovara trenutnom radnom opterećenju. Samim time može se ubrzati postupak optimizacije ako ograničimo korištenje samo virtualnog stroja VM_1. Korištenje jačih virtualnih strojeva te samim time povećanje troškova najma ne ostvaruje značajno bolje rezultate u vidu vremena odaziva.



Slika 7.5: Usporedba kriterija kvalitete pretraženih kandidata tijekom optimizacije evolucijskim algoritmom



Slika 7.6: Prosječne vrijednosti kriterija kvalitete unutar populacija svake generacije

Tablica 7.4: Rezultat izvođenja evolucijskog algoritma za optimizaciju

Konfiguracija k		Trošak $q_{COST}(k)$	Udio prekršaja $q_{SLA}(k)$
$G_1 = \{Report, Resource\}$ $G_2 = \{Fiscalization, Invoice, Status\}$	$type(G_1) = VM_1$ $type(G_2) = VM_1$	78.00	33.18
$G_1 = \{Fiscalization, Invoice\}$ $G_2 = \{Report, Resource, Status\}$	$type(G_1) = VM_1$ $type(G_2) = VM_1$	79.00	13.37
$G_1 = \{Fiscalization, Report, Resource, Status\}$ $G_2 = \{Invoice\}$	$type(G_1) = VM_1$ $type(G_2) = VM_1$	82.00	6.83
$G_1 = \{Invoice, Status\}$ $G_2 = \{Fiscalization, Report, Resource\}$	$type(G_1) = VM_1$ $type(G_2) = VM_1$	83.00	6.77
$G_1 = \{Fiscalization, Report, Status\}$ $G_2 = \{Invoice, Resource\}$	$type(G_1) = VM_1$ $type(G_2) = VM_1$	85.00	5.25
$G_1 = \{Fiscalization, Status\}$ $G_2 = \{Report, Resource\}$ $G_3 = \{Invoice\}$	$type(G_1) = VM_1$ $type(G_2) = VM_1$ $type(G_3) = VM_1$	95.00	3.78
$G_1 = \{Fiscalization, Report, Status\}$ $G_2 = \{Invoice\}$ $G_3 = \{Resource\}$	$type(G_1) = VM_1$ $type(G_2) = VM_1$ $type(G_3) = VM_1$	106.00	2.53
$G_1 = \{Report, Status\}$ $G_2 = \{Resource\}$ $G_3 = \{Fiscalization\}$ $G_4 = \{Invoice\}$	$type(G_1) = VM_1$ $type(G_2) = VM_1$ $type(G_3) = VM_1$ $type(G_4) = VM_1$	127.00	1.62
$G_1 = \{Status\}$ $G_2 = \{Report, Resource\}$ $G_3 = \{Fiscalization\}$ $G_4 = \{Invoice\}$	$type(G_1) = VM_1$ $type(G_2) = VM_1$ $type(G_3) = VM_2$ $type(G_4) = VM_1$	161.00	1.61
$G_1 = \{Status\}$ $G_2 = \{Resource\}$ $G_3 = \{Invoice\}$ $G_4 = \{Fiscalization, Report\}$	$type(G_1) = VM_1$ $type(G_2) = VM_1$ $type(G_3) = VM_1$ $type(G_4) = VM_2$	170.00	1.52
$G_1 = \{Fiscalization\}$ $G_2 = \{Report\}$ $G_3 = \{Resource\}$ $G_4 = \{Invoice\}$ $G_5 = \{Status\}$	$type(G_1) = VM_1$ $type(G_2) = VM_4$ $type(G_3) = VM_1$ $type(G_4) = VM_1$ $type(G_5) = VM_2$	259.00	1.09

7.5.2 Validacija rezultata evolucijskog algoritma

Prije nego što validiramo rezultate evolucijskog algoritma potrebno je validirati ispravnost simulacijskog postupka. Kako bi ispitali konzistentnost rezultata simulacije proveli smo Kruskal-Wallisov H neparametrijski test [202] pri nultoj hipotezi koja podrazumijeva da svi rezultati pripadaju istoj distribuciji neovisno o simuliranoj konfiguraciji. Prihvaćanje ovakve hipoteze značilo bi da postupak simulacije ne uspijeva razlučiti kriterije kvalitete pri različitim konfiguracijama sustava.

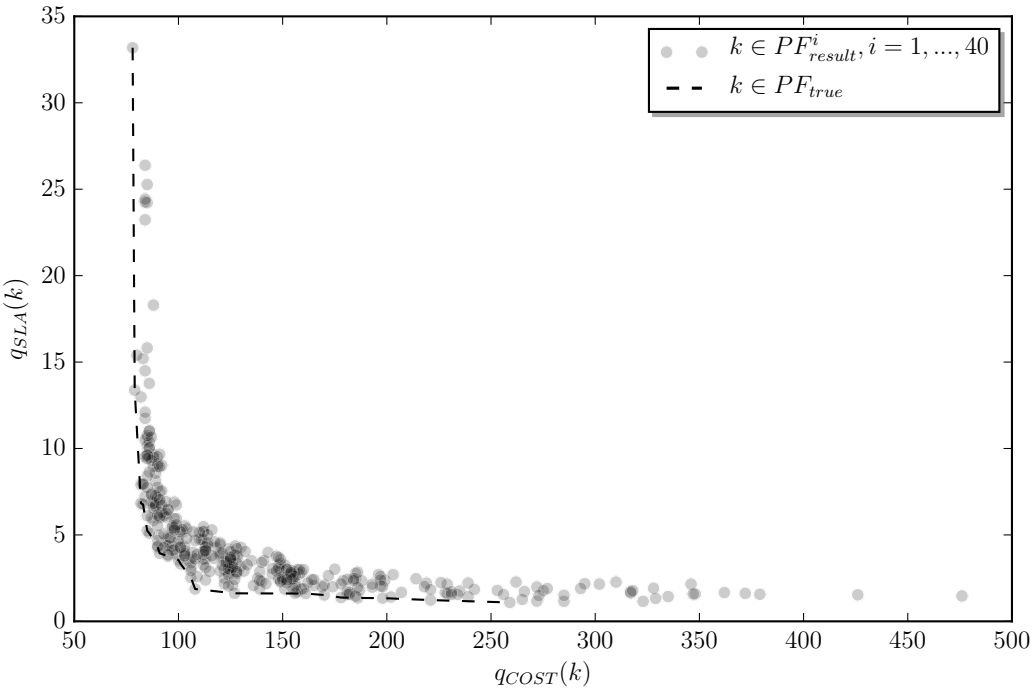
Tablica 7.5 prikazuje rezultate Kruskal-Wallisov H neparametrijski test provedenog nad 58 konfiguracija za kojih postoji barem 10 provedenih simulacijskih testova što je dovoljan uvjet za provođenje ovog neparametrijskog testa. Ukupno je kroz test uspoređeno 1653 parova konfiguracija ($\frac{58*57}{2}$) te je utvrđeno sa statističkom značajnošću od $p < 0.01$ da više od 91% parova različitih konfiguracija ostvaruje različite troškove te više od 69% različit broj SLA

Tablica 7.5: Rezultati Kruskal-Wallisovog H testa

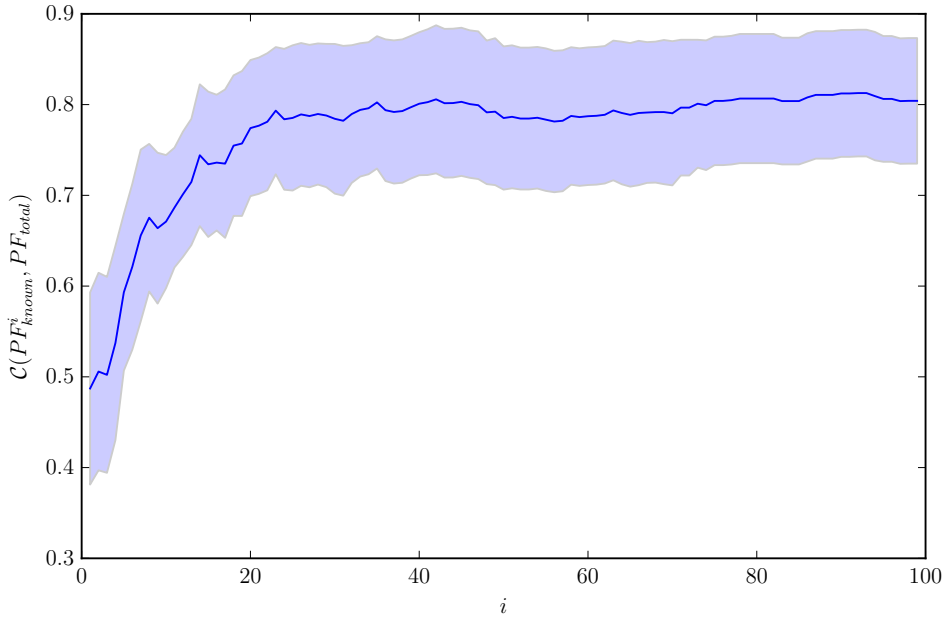
Kriterij kvalitete	Rezultat testa	
	p-vrijednost	postotak konfiguracija
q_{COST}	<0.05	93.5%
	<0.01	91.3%
q_{SLA}	<0.05	76.2%
	<0.01	69.6%

prekršaja čime je pokazano da su rezultati simulacije kriterija kvalitete konzistentni te da postupak simulacije može prikazati utjecaj konfiguracije na kriterije kvalitete.

Za validaciju rezultata genetskog algoritma proveden je niz od 40 uzastopnih optimizacija. Slika 7.7 prikazuje kriterije kvalitete svih rezultirajućih konfiguracija $\mathbb{K}^{i}_{result}, i \in [1,40]$ u usporedbi s kriterijima kvalitete optimalnog Pareto fronta PF_{true}. Pri odgovoru na **RQ1** i **RQ3** ispitan je kriterij ostvarivanja minimalne pokrivenosti $\mathscr{C}^* \geq 0.7$. Izvršavanjem genetskog algoritma kroz 100 generacija ($gen \in [1,n], n = 100$) dobiveno je 40 vrijednosti $\mathscr{C}^*_i$ metrike za svaku iteraciju $i \in [1,40]$. Zatim je za svako izvođenje i proveden Studentov t-test u svrhu ispitivanja nulte hipoteze koja tvrdi je $\mathscr{C}^* \leq 0.7$ uz $p < 0.01$. Za generacije $gen \leq 78$ nije odbačena nulta hipoteza, dok je za sve $gen \geq 78$ ista odbačena uz $p = 0.0086$ pri $gen = 78$ sve do $p = 0.0078$ za $gen = 100$. Zaključujemo kako prosječno izvođenje genetskog algoritma treba barem 78 generacija u postizanju zadanog cilja mjere pokrivenosti $\mathscr{C}^* = 0.7$. Na slici 7.8 prikazano je kretanje prosječne pokrivenosti i standardne devijacije tijekom generacija.



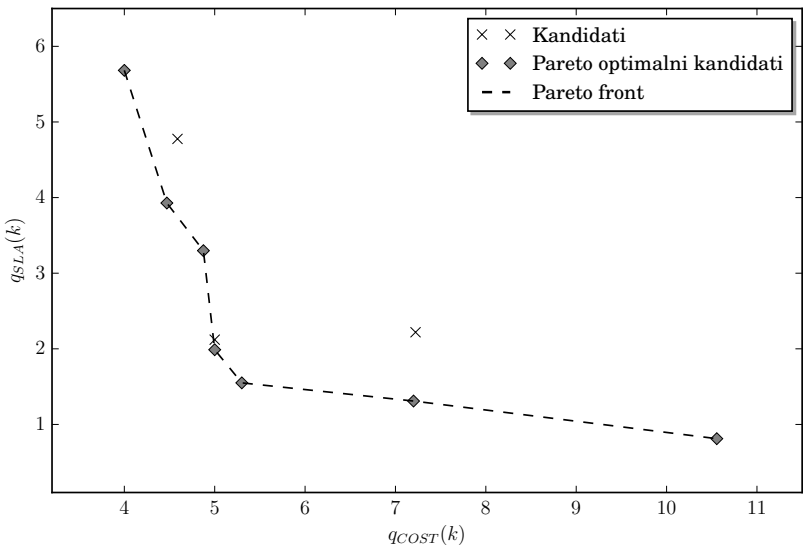
Slika 7.7: Prikaz svih Pareto skupova tokom 40 ponavljanja izvršavanja te usporedba sa globalnim Pareto skupom izvedenim pomoću kandidata svih izvršavanja



Slika 7.8: Prosječna vrijednost indikatora pokrivenosti $\mathscr{C}^*(\mathbb{K}^*_{result}, \mathbb{K}^*_{true})$ tijekom iteracija genetskog algoritma; osjenčane vrijednosti predstavljaju interval standardne devijacije

7.5.3 Evaluacija odabranih kandidata pomoću alata ElaClo

U svrhu odgovaranja na **RQ2** te utvrđivanja u kojoj mjeri odgovaraju optimalni kandidati iz simulacijskog okruženja (Tablica 7.4) optimalnim kandidatima nad stvarnim okruženjem uspoređeni su kriteriji kvalitete u stvarnom okruženju koristeći ElaClo. Slika 7.9 prikazuje ostvarene kriterije kvalitete u stvarnom okruženju kandidata dobivenih evolucijskim algoritmom.

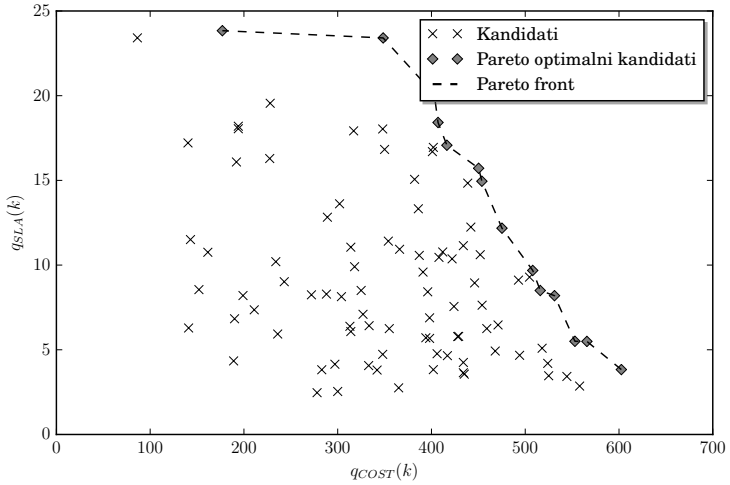


Slika 7.9: Kvaliteta kandidata dobivenih putem GA ocijenjena u stvarnom okruženju pomoću alata ElaClo

Za usporedbu proveden je postupak evolucijskog algoritma za optimalne kandidate suprotnih ciljeva optimizacije što je rezultiralo skupom kandidata prikazanih na slici 7.10. Nadalje, kriteriji kvalitete i za takve kandidate također su određeni nad stvarnim sustavom koristeći ElaClo. Kriteriji kvalitete za oba skupa kandidata nad stvarnim okruženjem prikazani su slikom 7.11.

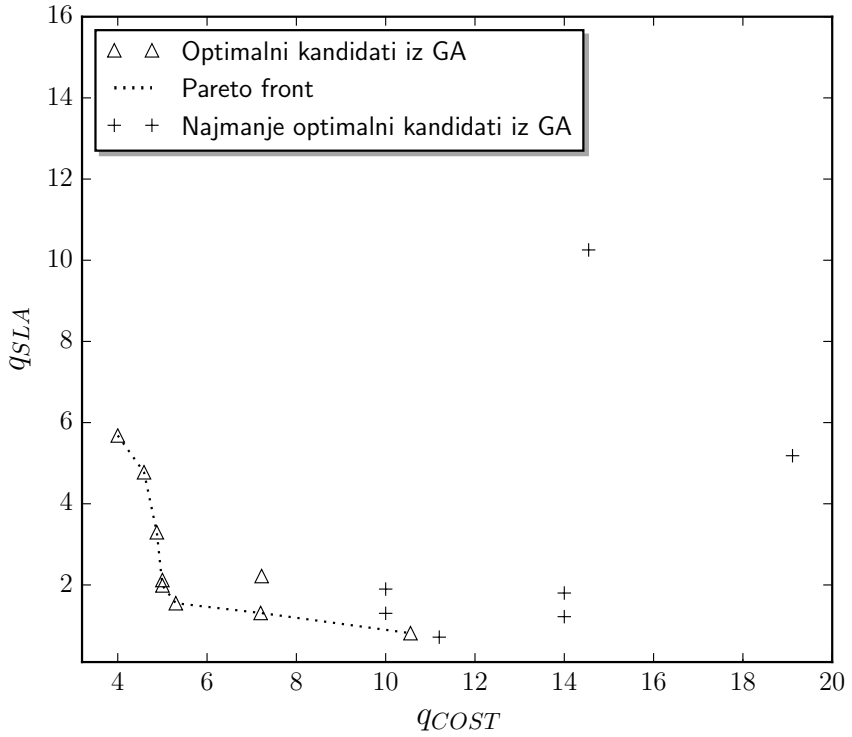
Izračunom mjere OD definirane u poglavlju 7.2 te statističkom usporedbom testom Mann-Whitney U odbačena je nulta hipoteza u korist alternativne hipoteze sa statističkom značajnošću od $p < 10^{-4}$. Time možemo zaključiti da su odnosi kriterija kvalitete između kandidata u simulacijskom okruženju zadržani i u stvarnom okruženju. Mjere učinka također su određene te pokazuju da u 38% slučajeva optimalne konfiguracije određene evolucijskim algoritmom pokazuju niže troškove uz bolje radne karakteristike (manje prekršaja dopuštenih gornjih granica vremena odaziva). Nadalje, ako promatramo po samo jednoj dimenziji (samo trošak ili samo radne karakteristike), kandidati u 97% slučajeva ostvaruju niže troškove infrastrukture, dok u 41% slučajeva bolje radne karakteristike.

U predstavljenoj studiji slučaja, optimizacijski proces evolucijskim algoritmom se pokazao učinkovitim pri preporuci konfiguracija koje ostvaruju niže troškove izvođenja uz jednake ili



Slika 7.10: Pareto-front kriterija kvalitete suprotnih ciljevima optimizacije određenih u simulacijskom okruženju evolucijskim algoritmom

bolje radne karakteristike većine skupljih konfiguracija. Ovim eksperimentom pokazano je kako veća ulaganja u infrastrukturu ne moraju nužno rezultirati boljim radnim karakteristikama te kako je predstavljeni postupak optimizacije opravdan.



Slika 7.11: Usporedni prikaz najboljih i najlošijih kandidata iz simulacijskog okruženja s kriterijima kvalitete određenim u sklopu stvarnog okruženja

7.5.4 Ograničenja istraživanja

Predmet predložene metode optimizacije jesu aplikacije u oblaku koje odlikuje elastična prilagodba radnoga kapaciteta sukladno promjenjivom opterećenju i predloženoj arhitekturi. Kako postoje brojne varijante u načinu na koji takva aplikacija može biti realizirana navode se ograničenja postupka optimizacije i primijenjene validacije.

Izvedbena okolina ElaClo namijenjena je da se koristi za aplikacije čija je arhitektura zasnovana na komponentama ostvarenim u obliku mrežnih usluga uz ostala ograničenja predstavljene arhitekture. Na velika industrijska rješenja gdje su komponente zasnovane nad više razina kompozicija manjih komponenata ili organizirane primjenom ESB-a (engl. *Enterprise Service Bus*) trenutno nije moguće primijeniti ElaClo. Nadalje, ElaClo ostvaruje elastičnost za komponente koje ne pohranjuju stanje (engl. *stateless*) što je glavni preduvjet za ostvarenje horizontalne skalabilnosti. Primjena elastičnosti na komponente poput baza podataka nije omogućeno. Takve komponente valja izuzeti iz postupka optimizacije. Primjena pohrane podataka pomoću komponenata koje omogućuju elastično ponašanje omogućena je dodatnim uslugama davatelja infrastrukture u oblaku [203].

Daljnji napredak simulacijskoga modela može se ostvariti modeliranjem zastoja prema ulazno-izlaznim jedinicama poput diskovlja ili mrežnih sklopovlja te zastoja programske prirode poput sinkronizacije dretvi.

Također treba sagledati i možebitne nedostatke u valjanosti istraživanja (engl. *threats to validity*) [204]:

- **Sastavna valjanost** (engl. *structural validity*). Sastavnu valjanost može narušiti pretpostavka modela da se elastičnost može ostvariti jednostavnim horizontalnim skaliranjem komponente. Istraživanje bi trebalo proširiti na kompleksnije aplikacije gdje bi se moglo ukazati na nepredviđene poteškoće kod pojedinih komponenata koje je teško ostvariti tako da ne sadrže *stanje*
- **Vanjska valjanost** (engl. *external validity*). ElaClo je validiran pomoću studije slučaja tipične aplikacije. Dodatni studiji slučajeva mogu ukazati na neke druge poteškoće u primjeni ElaCla [205].

Poglavlje 8

Zaključak

Ova disertacija predstavlja okvir ElaClo za optimiranje konfiguracija informacijskih sustava prema vremenima odziva definiranim sporazumom o razini usluga te popratnim troškovima najma infrastrukture u oblaku. Okvir ElaClo ostvaren je kao izvedbena okolina u oblaku koja sadrži sve potrebne komponente za ostvarivanje automatiziranog procesa optimizacije.

ElaClo podržava razvoj informacijskih sustava prema predloženom modelu arhitekture koji omogućuje dinamičku rekonfiguraciju u sklopu prilagodbe promjenjivom radnom opterećenju. Model predlaže idempotentnost operacija pojedinih komponenti i asinkronu komunikaciju između komponenti kao podlogu za ostvarenje dinamičke rekonfiguracije. Različite moguće konfiguracije ostvarene su pridruživanjem komponenti u elastične grupe. Pojedine grupe odlikuje mogućnost autonomnog horizontalnog skaliranja u prilagodbi trenutnom intenzitetu radnog opterećenja. Nastavno na danu elastičnu arhitekturu, predložen je proces razvoja koji uključuje određivanje optimalnih konfiguracija prema ciljanom sporazumu o razini usluge.

Za određivanje radnih značajki pojedinih konfiguracija predstavljena je metoda koja omogućuje ostvarenje konfiguracije, simuliranje radnog opterećenja prema zadanom modelu opterećenja, prikupljanje potrebnih metrika tijekom ocjene atributa kvalitete, usporedbu rezultata koje pojedina konfiguracija ostvaruje te prilagodbu sporazuma o razini usluge prema željenoj konfiguraciji.

ElaClo koristi genetski algoritam i metodu simuliranja radnih značajki zasnovanu na mreži redova čekanja radi bržeg pretraživanja prostora mogućih konfiguracija i određivanja optimalnih kandidata. U tu svrhu ostvaren je simulator koji raspolaže mogućnošću prilagodbe simulacijskog modela tijekom izvođenja zbog simuliranja elastičnog horizontalnog skaliranja.

Naposljetku, konstruiran je model izvedbene okoline koja ostvaruje navedene doprinose svim potrebnim komponentama koristeći infrastrukturu računarstva u oblaku. Navedena okolina implementirana je u obliku alata te je provedena validacija cijelog postupka koristeći

reprezentativni primjer informacijskoga sustava u domeni maloprodaje. Izvršena je validacija preciznosti simulacijskog modela u određivanju optimalnih kandidata, kao i radne karakteristike samog postupka. Pomoću statističkih testova potvrđene su željene pretpostavke o preciznosti i efikasnosti predloženoga postupka na uzorku od 40 provedenih optimizacija.

8.1 Implikacije istraživanja

ElaClo nadopunjuje postojeća istraživanja analizirajući primjenu postupka optimiranja nad stvarnim sustavom u kasnijim fazama razvoja evolucijskim algoritmom i simulacijskim modelom. Za razliku od prethodnih istraživanja gdje je potrebno ekspertno znanje u oblikovanju i parametrizaciji simulacijskih modela, ElaClo izlučuje simulacijski model prema predloženom modelu arhitekture. Time je postignuta veća pouzdanost rezultata optimizacije jer se završna evaluacija reduciranog broja kandidata vrši nad stvarnim sustavom u stvarnoj okolini.

Dosadašnji postupci optimizacije izvođenjem stvarnog sustava nisu uzeli u obzir sustavnu pretragu svih mogućih konfiguracija već su se doticali samo horizontalne i vertikalne skalabilnosti infrastrukture pri određivanju optimalne količine korištenih resursa. Razlog tomu je veličina prostora svih mogućih konfiguracija te duljina trajanja evaluacije pojedine konfiguracije. ElaClo rješava taj problem tako da se postupak optimizacije dijeli na dvije faze: (1) optimiranje koristeći simulacijski model aplikacije te (2) optimiranje pomoću rezultirajućih kandidata iz prvog koraka gdje se ocjena provodi u stvarnoj okolini oblaka. Primjenjivost cijelog postupka demonstrirala se realizacijom sustava u obliku prototipnog alata te provođenjem postupka optimiranja u sklopu studije slučaja.

Cijeli sustav omogućuje razvojnim timovima automatizirani postupak određivanja konfiguracija koje udovoljavaju ograničenjima postavljenim sporazumima o razini usluge uz optimalne troškove. Koristeći predloženi model arhitekture, razvojni timovi prepuštaju sustavu ElaClo automatski ekstrakciju simulacijskih modela potrebnih prvoj fazi optimizacije. Ukoliko se ne utvrdi konfiguracija koja udovoljava željenom sporazumu može se odrediti novi sporazum prema najefikasnijoj konfiguraciji, zavisno o planiranim troškovima izvođenja.

Samim time, predloženi postupak optimizacije može ostvariti značajne uštede u završnim fazama testiranja i puštanja informacijskih sustava u rad. Postupak također nadopunjuje klasične metode testiranja kapaciteta sustava u izvedbenoj okolini računarstva u oblaku gdje kapacitet više nije unaprijed određene već je podložan promjenama u tijeku rada. Koristeći ElaClo određuju se upravo one konfiguracije koje najefikasnije provode dinamičku promjenu kapaciteta čime povećavaju ukupnu iskoristivost uloženih infrastrukturnih resursa tijekom izvođenja. Tako se ostvaruje ne samo ušteda u vremenu koje je potrebno za plasirati pojedini proizvod na tržište već i ušteda u potrebnim ulaganjima u infrastrukturu koristeći optimalne konfiguracije.

8.2 Budući pravci razvoja

Budući razvoj na sustavu ElaClo ima za cilj ostvariti integraciju s tehnikom virtualizacije u obliku spremnika (engl. *container*) poput sustava Docker*. Time se dodatno olakšava inkrementalni razvoj informacijskog sustava čije su komponente heterogene. Trenutni prototip alata ElaClo podržava samo komponenti model *JavaEE* čime mu je ograničena primjena.

Valja se također razmotriti mogućnosti generiranja modela TOSCA [20] prema određenim optimalnim konfiguracijama zbog daljnje automatizacije u postupku instalacije sustave u produkcijsku okolinu. Integracija sa sustavom za upravljanje infrastrukture u oblaku OpenStack [206] s ciljem evaluacije konfiguracija koje uključuju korištenje više dislociranih podatkovnih centara [206] dodatno bi doprinijela trenutnom znanstvenom dosegu.

Najvažnije područje budućeg razvoja je dodatno unaprjeđenje u brzini ocjene mogućih konfiguracija koristeći aproksimacijske analitičke modele uz programske knjižnice poput LQNS-a [207]. Budući da sporazumi o razini usluga dotiču kvantile vremena odaziva, potrebno je koristiti modele koji omogućuju analitički izračun razdiobe vremena odaziva. U tu svrhu valja istražiti primjenu diferencijalnih jednadžbi koje opisuju kretanje fluida [208] pri modeliraju radnih karakteristika.

Općenito, potrebno je uspostaviti okvir za korištenje većeg broja evaluacijskih modela sustava s različitim karakteristikama u vidu brzine određivanja kriterija kvalitete i preciznosti rezultata. Primjena heterogenih modela omogućila bi dinamičko izvođenje cjelokupnog procesa optimizacije u skladu s postavljenim ciljevima, veličinom prostora pretrage te vremenskom ograničenja u trajanju cijelog postupka optimizacije.

Dinamička optimizacija tijekom izvođenja također predstavlja zanimljivo područje budućeg istraživanja. Trenutni postupak optimizacije može se nadopuniti metodama strojnog učenja čime se mogu odrediti karakteristične značajke koje određuju kriterije kvalitete. Takvo prikupljeno znanje moglo bi se koristiti za kontinuiranu optimizaciju tijekom izvođenja, pri čemu predloženi model već ostvaruje nužan preduvjet time što podržava dinamičku promjenu elastičnih grupa.

*Docker — dostupno na *https://www.docker.com/*

Literatura

[1] International Organization for Standardization., International Electrotechnical Commission., Institute of Electrical and Electronics Engineers., IEEE-SA Standards Board., ISO/IEC/IEEE 24765:2010 - Systems and Software Engineering - Vocabulary. ISO/IEC, 2011, dostupno na: http://ieeexplore.ieee.org/document/5733835/

[2] Sommerville, I., Software Engineering, 2010.

[3] Perry, D. E., Wolf, A. L., "Foundations for the study of software architecture", ACM SIGSOFT Software Engineering Notes, Vol. 17, No. 4, 1992, str. 40–52.

[4] Ieee, Guide to the Software Engineering Body of Knowledge Version 3.0 (SWEBOK Guide V3.0), 2014.

[5] Ameller, D., Galster, M., Avgeriou, P., Franch, X., "A survey on quality attributes in service-based systems", Software Quality Journal, 2015, str. 1–29.

[6] Khazaei, H., Misic, J., Misic, V. B., "A Fine-Grained Performance Model of Cloud Computing Centers", IEEE Transactions Parallel and Distributed Systems, Vol. 24, No. 11, 2013, str. 2138–2147.

[7] Gartner Group, "Hype Cycle for Cloud Computing", Tech. Rep., 2014, dostupno na: https://www.gartner.com/doc/2807621/hype-cycle-cloud-computing-

[8] Armbrust, M., Fox, O., Griffith, R., Joseph, A. D., Katz, Y., Andy Konwinski, Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M., "Above the Clouds: A Berkeley View of Cloud Computing", University of California, Berkeley, Tech. Rep. UCB, 2009, str. 07–013.

[9] Armbrust, M., Stoica, I., Zaharia, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., "A view of cloud computing", Communications of the ACM, Vol. 53, No. 4, apr 2010, str. 50, dostupno na: http://dl.acm.org/ft{_}gateway.cfm?id=1721672{&}type=html

[10] Valacich, J., Schneider, C., Information Systems Today. Prentice Hall, 2011.

[11] D'Atri, A., De Marco, M., Casalino, N., Interdisciplinary Aspects of Information Systems Studies, 2008, dostupno na: http://www.scopus.com/inward/record.url?eid=2-s2.0-84892063577{&}partnerID=tZOtx3y1

[12] Rimal, B. P., Choi, E., Lumb, I., "A Taxonomy and Survey of Cloud Computing Systems", 2009 Fifth International Joint Conference on INC, IMS and IDC, 2009, str. 44–51, dostupno na: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5331755

[13] Herbst, N. R., Huber, N., Kounev, S., Amrehn, E., "Self-Adaptive Workload Classification and Forecasting for Proactive Resource Provisioning", Concurrency and Computation: Practice and Experience, Vol. 26, No. 12, 2014, str. 2053–2078.

[14] Mell, P., Grance, T., "The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology", National Institute of Standards and Technology, Information Technology Laboratory, Vol. 145, 2011, str. 7, dostupno na: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[15] Herbst, N. R., Kounev, S., Reussner, R., "Elasticity in Cloud Computing : What It Is, and What It Is Not", in 10th International Conference on Autonomic Computing, 2013, str. 23–27, dostupno na: http://sdqweb.ipd.kit.edu/publications/pdfs/HeKoRe2013-ICAC-Elasticity.pdf

[16] Dustdar, S., Guo, Y., Satzger, B., Truong, H.-L., "Principles of Elastic Processes", IEEE Internet Computing, Vol. 15, No. 5, sep 2011, str. 66–71.

[17] Wu, L., Buyya, R., "Service Level Agreement (SLA) in Utility Computing Systems", IGI Global, 2010, str. 27.

[18] Buco, M. J., Chang, R. N., Luan, L. Z., Ward, C., Wolf, J. L., Yu, P. S., "Utility computing SLA management based upon business objectives", IBM Systems Journal, Vol. 43, No. 1, 2004, str. 159–178, dostupno na: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5386770

[19] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., Stafford, J., Documenting Software Architectures Views and Beyond, 2nd ed. Addison-Wesley, 2010.

[20] Binz, T., Breitenbücher, U., Kopp, O., Leymann, F., Binz, T., Breitenb, U., Kopp, O., Binz, T., Breitenb, U., "TOSCA: Portable Automated Deployment and Management of Cloud Applications", in Advanced Web Services, Bouguettaya, A., Sheng, Q. Z., Daniel, F., (ur.). New York: Springer, jan 2014, str. 527–549.

[21] Martens, A., Koziolek, H., Becker, S., Reussner, R. H., "Automatically Improve Software Models for Performance, Reliability and Cost Using Genetic Algorithms", Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering, 2010, str. 105–116, dostupno na: http://www.inf.pucrs.br/wosp

[22] Tanković, N., Galinac Grbac, T., Truong, H.-l., Dustdar, S., "Transforming Vertical Web Applications Into Elastic Cloud Applications", in International Conference on Cloud Engineering (IC2E 2015). IEEE, mar 2015, str. 135–144.

[23] Basili, V. R., "The Experimental Paradigm in Software Engineering", Experimental Software Engineering Issues: Critical Assessment and Future Directives, No. August, 1993, str. 1–12, dostupno na: http://www.springerlink.com/index/p353387611764860.pdf

[24] Basili, V. R., Shull, F., Lanubile, F., "Building Knowledge through Families of Software Studies :", Software Engineering, IEEE Transactions on, Vol. 25, No. 4, 1999, str. 456—-473.

[25] Basili, V. R., "The Role of Controlled Experiments in Software Engineering Research", Empirical Software Engineering, 2007, str. 33 – 37.

[26] Kitchenham, B., Charters, S., "Guidelines for performing Systematic Literature Reviews in Software Engineering", Engineering, Vol. 2, 2007, str. 1051.

[27] Runeson, P., Host, M., Rainer, A., Regnell, B., Case Study Research in Software Engineering: Guidelines and Examples, 2012, dostupno na: http://books.google.pt/books?id=T7rXoaxqPIAC

[28] Wohlin, C., Experimentation in Sofware Engineering, 2013, Vol. 53, No. 9.

[29] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J., Software Architectures, 2002.

[30] Parnas, D. L., "On the criteria to be used in decomposing systems into modules", Communications of the ACM, Vol. 15, No. 12, 1972, str. 1053–1058, dostupno na: http://portal.acm.org/citation.cfm?doid=361598.361623

[31] Feiler, P. H., Gluch, D. P., "The Architecture Analysis & Design Language (AADL): An Introduction", No. February, 2006, str. CMU/SEI—-2006—-TN—-011, dostupno na: http://www.sei.cmu.edu/library/abstracts/reports/06tn011.cfm

[32] Taylor, R. N., Medvidović, N., Dashofy, E. M., Software Architecture: Foundations, Theory, and Practice. John Wiley & Sons, Inc, 2009.

[33] Szyperski, C., Grunz, D., Murer, S., Component Software: Beyond Object-oriented Programming. Addison-Wesley Longman Publishing Co., Inc., oct 2002, dostupno na: http://dl.acm.org/citation.cfm?id=515228

[34] Gamma, E., Helm, R., Johnson, R., Vlissides, J., Design Patterns: Elements of Reusable Object-Oriented Software. Pearson Education, 1994.

[35] ISO/IEC/IEEE, "ISO/IEC 24765:2010 Systems and software engineering - Vocabulary", Tech. Rep., 2010.

[36] Heineman, George T. Councill, W. T., Component-Based Software Engineering: Putting the Pieces Together, 1st ed. Addison-Wesley Professional, 2001.

[37] Crnkovic, I., Sentilles, S., Vulgarakis, a., Chaudron, M. R. V., "A Classification Framework for Software Component Models", IEEE Transactions on Software Engineering, Vol. 37, No. 5, 2011, str. 593–615.

[38] Carzaniga, A., Fuggetta, A., Hall, R. S., Heimbigner, D., Van Der Hoek, A. A., Wolf, A. L., "A characterization framework for software deployment technologies", DTIC Document, Tech. Rep., 1998.

[39] Cheesman, J., Daniels, J., UML Components: A simple process for specifying component-based software, 2000, No. 3, dostupno na: http://dsrg.mff.cuni.cz/{~}mencl/projects/uml20components-thesis.html

[40] Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B., Wüst, J., Zettel, J., "Component-based product line engineering with UML", jan 2002, dostupno na: http://dl.acm.org/citation.cfm?id=502972

[41] Hasselbring, W., "Integration for Applications", 2002, str. 16–25.

[42] Rational, "Rational Unified Process: Best Practices for Software Development Teams", Tech. Rep., 2001, dostupno na: http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:Rational+Unified+Process+Best+Practices+for+Software{#}4

[43] Bass, L., "Software architecture in practice", 2012, str. 640.

[44] ISO/IEC, "ISO/IEC 25010:2011 - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models", Tech. Rep., 2011.

[45] Jain, R., The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling., 1991, Vol. 491.

[46] Voas, J. M., Miller, K. W., "Testability: The New Verification", IEEE Software, Vol. 12, No. 3, 1995, str. 17–28.

[47] Kurose, J. F., Ross, K. W., "Computer Networking A Top-Down Approach Featuring the Internet", Pearson Education India, Vol. 1, 2005, str. 712, dostupno na: http://www.amazon.com/dp/B0026579FQ

[48] Harchol-Balter, M., Performance Modeling and Design of Computer Systems: Queueing Theory in Action. Cambridge University Press, 2013.

[49] Dean, J., "Latency Comparison Numbers", 2012.

[50] Keller, a., Ludwig, H., "Defining and monitoring service level agreements for dynamic e-business", Proceedings of LISA '02: Sixteenth Systems Administration Conference, No. November, 2002, str. 189–204.

[51] Kosiński, J., Radziszowski, D., Zieliński, K., Zieliński, S., Przybylski, G., Niedziela, P., "Definition and evaluation of penalty functions in SLA management framework", Proceedings - 4th International Conference on Networking and Services, ICNS 2008, 2008, str. 176–181.

[52] Brunnström, K., Beker, S. A., Moor, K. D., Dooms, A., Egger, S., Garcia, M.-N., Hossfeld, T., Jumisko-Pyykkö, S., Keimel, C., Larabi, M.-C., Lawlor, B., Callet, P. L., Möller, S., Pereira, F., Pereira, M., Perkis, A., Pibernik, J., Pinheiro, A., Raake, A., Reichl, P., Reiter, U., Schatz, R., Schelkens, P., Skorin-Kapov, L., Strohmeier, D., Timmerer, C., Varela, M., Wechsung, I., You, J., Zgank, A., "Qualinet White Paper on Definitions of Quality of Experience", Tech. Rep., 2013.

[53] Varela, M., Skorin-kapov, L., Ebrahimi, T., "Quality of Service Versus Quality of Experience", in Quality of Experience, 2014, str. 1–2, dostupno na: http://link.springer.com/10.1007/978-3-319-02681-7

[54] Nielsen, J., Usability engineering. Elsevier, 1994.

[55] Chellappa, R., "Intermediaries in cloud-computing: A new computing paradigm", in INFORMS Annual Meeting, Dallas, 1997.

[56] Buyya, R., Buyya, R., Yeo, C. S., Yeo, C. S., Venugopal, S., Venugopal, S., Broberg, J., Broberg, J., Brandic, I., Brandic, I., "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility", Future Generation Computer Systems, Vol. 25, No. June 2009, 2009, str. 17, dostupno na: http://portal.acm.org/citation.cfm?id=1528937.1529211

[57] Bellur, U., Rao, C. S., Sd, M. K., "Optimal Placement Algorithms for Virtual Machines", ArXiv e-prints, No. Vm, 2010, str. 1–16, dostupno na: http://arxiv.org/abs/1011.5064

[58] Rathod, C., "A Survey on Different Virtual Machine Placement Algorithms", IInternational Journal of Advance Research in Computer Science and Management Studies, Vol. 2, No. 2, 2014, str. 266–271, dostupno na: http://ijarcsms.com/docs/paper/volume2/issue2/V2I2-0075.pdf

[59] Piao, J. T., Yan, J., "A network-aware virtual machine placement and migration approach in cloud computing", Proceedings - 9th International Conference on Grid and Cloud Computing, GCC 2010, 2010, str. 87–92.

[60] Yeo, C. S., Venugopal, S., Chu, X., Buyya, R., "Autonomic metered pricing for a utility computing service", Future Generation Computer Systems, Vol. 26, No. 8, 2010, str. 1368–1380.

[61] Weinhardt, C., Anandasivam, A., Blau, B., Stöer, J., "Business models in the service world", IT Professional, Vol. 11, No. 2, 2009, str. 28–33.

[62] Chun, S.-H., Choi, B.-S., "Service models and pricing schemes for cloud computing", Cluster Computing, Vol. 17, No. 2, sep 2013, str. 529–535, dostupno na: http://link.springer.com/10.1007/s10586-013-0296-1

[63] Fujiwara, I., "Study on Combinatorial Auction Mechanism for Resource Allocation in Cloud Computing Environment", Nii.Jp, 2011, dostupno na: http://www.nii.jp/graduate/thesis/pdf/201203/fujiwara{_}Dr{_}thesis.pdf

[64] Zaman, S., Grosu, D., "Combinatorial auction-based allocation of virtual machine instances in clouds", Journal of Parallel and Distributed Computing, Vol. 73, No. 4, 2013, str. 495–508, dostupno na: http://dx.doi.org/10.1016/j.jpdc.2012.12.006

[65] Legillon, F., Melab, N., Renard, D., Talbi, E.-G., "Cost Minimization of Service Deployment in a Public Cloud Environment", in 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum. IEEE, may 2013, str. 491–498, dostupno na: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6650923

[66] Arlitt, M. F., Williamson, C. L., "Web server workload characterization", ACM SIGMETRICS Performance Evaluation Review, Vol. 24, No. 1, 1996, str. 126–137, dostupno na: http://portal.acm.org/citation.cfm?doid=233008.233034

[67] Kistowski, J., Herbst, N. R., Kounev, S., "Modeling variations in load intensity over time", Proceedings of the third international workshop on Large scale testing (LT'14), 2014, str. 1–4, dostupno na: http://dl.acm.org/citation.cfm?id=2577036.2577037

[68] Barry, D. K., Web Services and Service-Oriented Architecture: The Savvy Manager's Guide, 2003, Vol. 2003.

[69] Herbst, N. R., "Workload Classification and Forecasting", 2012.

[70] Lehrig, S., Eikerling, H., Becker, S., "Scalability, Elasticity, and Efficiency in Cloud Computing: A Systematic Literature Review of Definitions and Metrics", Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures, No. MAY, 2015, str. 83–92.

[71] Hill, M. D., "What is scalability?", ACM SIGARCH Computer Architecture News, Vol. 18, No. 4, 1990, str. 18–21, dostupno na: ftp://ftp.cs.wisc.edu/markhill/Papers/can90{_}scalability.pdf

[72] Jogalekar, P., Woodside, M., Member, S., "Evaluating the Scalability of Managed Distributed Systems", IEEE Transactions on Parallel and Distributed Systems, Vol. 11, No. 6, 2000, str. 589–603.

[73] Duboc, L., Rosenblum, D. S., Wicks, T., "A framework for modelling and analysis of software systems scalability", in Proceeding of the 28th international conference on Software engineering - ICSE '06. New York, New York, USA: ACM Press, may 2006, str. 949, dostupno na: http://dl.acm.org/citation.cfm?id=1134285.1134460

[74] Zaharia, M., "An Architecture for Fast and General Data Processing on Large Clusters", Berkeley Technical Report, 2014, str. 128.

[75] Lehrig, S., Becker, S., "The CloudScale Method for Software Scalability, Elasticity, and Efficiency Engineering: A Tutorial", Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, 2015, str. 329–331, dostupno na: http://doi.acm.org/10.1145/2668930.2688818

[76] Bientinesi, P., Iakymchuk, R., Napper, J., "Handbook of Cloud Computing (2010)", Handbook of Cloud Computing, 2010, str. 493–516.

[77] Vaquero, L. M., Rodero-Merino, L., Buyya, R., "Dynamically scaling applications in the cloud", ACM SIGCOMM Computer Communication Review, Vol. 41, No. 1, jan 2011, str. 45, dostupno na: http://dl.acm.org/citation.cfm?id=1925861.1925869

[78] Verma, A., Kumar, G., Koller, R., "The Cost of Reconfiguration in a Cloud", Proceedings of the 11th International Middleware Conference Industrial track on Middleware Industrial Track 10, No. November, 2010, str. 11–16, dostupno na: http://portal.acm.org/citation.cfm?doid=1891719.1891721

[79] Weber, A., Herbst, N., Groenda, H., Kounev, S., "Towards a Resource Elasticity Benchmark for Cloud Environments", Proceedings of the 2nd International Workshop on Hot Topics in Cloud service Scalability - HotTopiCS '14, No. March, 2014, str. 1–8, dostupno na: https://sdqweb.ipd.kit.edu/publications/pdfs/WeHeGrKo2014-HotTopicsWS-ElaBench.pdf{%}5Cnhttp://dl.acm.org/citation.cfm?doid=2649563.2649571

[80] Huebscher, M. C., McCann, J. a., "A survey of autonomic computing—degrees, models, and applications", ACM Computing Surveys, Vol. 40, No. 3, 2008, str. 1–28, dostupno na: http://portal.acm.org/citation.cfm?doid=1380584.1380585

[81] IBM, "Autonomic Computing White Paper: An Architectural Blueprint for Autonomic Computing", Tech. Rep. June, 2005, dostupno na: http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:An+architectural+blueprint+for+autonomic+computing+.{#}0

[82] Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., Meedeniya, I., "Software Architecture Optimization Methods: A Systematic Literature Review", IEEE Transactions on Software Engineering, Vol. 39, No. 5, may 2013, str. 658–683, dostupno na: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6311410

[83] Malekimajd, M., Rizzi, A. M., Ardagna, D., Ciavotta, M., Passacantando, M., Movaghar, A., "Optimal capacity allocation for executing mapreduce jobs in cloud systems", Proceedings - 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2014, 2015, str. 385–392.

[84] Palanisamy, B., Singh, A., Liu, L., "Cost-Effective Resource Provisioning for MapReduce in a Cloud", IEEE Transactions on Parallel and Distributed Systems, Vol. 26, No. 5, 2015, str. 1265–1279.

[85] Krippendorff, K. H., Content Analysis: An Introduction to Its Methodology, 3rd ed. SAGE Publications, Inc, 2013.

[86] Chadegani, A. A., Salehi, H., Yunus, M., Farhadi, H., Fooladi, M., Farhadi, M., "A Comparison between Two Main Academic Literature Collections: Web of Science and Scopus Databases", Asian Social Science, Vol. 9, No. 5, 2013, str. 18–26.

[87] Wang, D., Yang, Y., Mi, Z., "A genetic-based approach to web service composition in geo-distributed cloud environment", Computers & Electrical Engineering, Vol. 43, No. August, 2015, str. 129–141, dostupno na: http://www.sciencedirect.com/science/article/pii/S0045790614002419

[88] Ciavotta, M., Ardagna, D., Gibilisco, G. P. G., "A Mixed Integer Linear Programming Optimization Approach for Multi-Cloud Capacity Allocation", The Journal of Systems & Software, Vol. 123, 2016, dostupno na: http://dx.doi.org/10.1016/j.jss.2016.10.001

[89] Wang, H., Wang, X., Hu, X., Zhang, X., Gu, M., "A multi-agent reinforcement learning approach to dynamic service composition", Information Sciences, Vol. 363, 2016, str. 96–119, dostupno na: http://dx.doi.org/10.1016/j.ins.2016.05.002

[90] Ashraf, A., Byholm, B., Porres, I., "A Multi-objective ACS Algorithm to Optimize Cost, Performance, and Reliability in the Cloud", Proceedings - 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing, UCC 2015, 2016, str. 341–347.

[91] Abbes, W., Kechaou, Z., Alimi, A. M., "A New Placement Optimization Approach in Hybrid Cloud Based on Genetic Algorithm", 2016.

[92] Bhardwaj, S., "A Particle Swarm Optimization Approach for Cost Effective SaaS Placement on Cloud", IEEE International conference on Computing and Automation (ICCCA 2015), 2015, str. 686–690.

[93] Li, J., Woodside, M., Chinneck, J., Litoiu, M., "Adaptive cloud deployment using persistence strategies and application awareness", IEEE Transactions on Cloud Computing, Vol. PP, No. 99, 2015, str. 1–14.

[94] Diaz-Sanchez, F., Al Zahr, S., Gagnaire, M., "An Exact Placement Approach for Optimizing Cost and Recovery Time under Faulty Multi-cloud Environments", 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, Vol. 2, 2013, str. 138–143, dostupno na: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6735409

[95] Sottet, J. S., Vagner, A., Frey, A. G., Architecture Optimization with SysML Modeling: A Case Study Using Variability, 2015, Vol. 580, dostupno na: http://www.scopus.com/inward/record.url?eid=2-s2.0-84955261216{&}partnerID=tZOtx3y1

[96] Sun, Y., White, J., Li, B., Walker, M., Turner, H., Automated QoS-oriented cloud resource optimization using containers, 2016.

[97] Walker, M., Reiser, M. O., Tucci-Piergiovanni, S., Papadopoulos, Y., Lönn, H., Mraidha, C., Parker, D., Chen, D., Servat, D., "Automatic optimisation of system architectures using EAST-ADL", Journal of Systems and Software, Vol. 86, No. 10, 2013, str. 2467–2487, dostupno na: http://dx.doi.org/10.1016/j.jss.2013.04.001

[98] Grabarnik, G. Y., Shwartz, L., Tortonesi, M., "Business-driven optimization of component placement for complex services in federated Clouds", IEEE/IFIP NOMS 2014 - IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World, 2014.

[99] Dastjerdi, A. V., Garg, S. K., Rana, O. F., Buyya, R., "CloudPick: a framework for QoS-aware and ontology-based service deployment across clouds", Software - Practice and Experience, Vol. 45, No. 2, 2015, str. 197–231.

[100] Kokkinos, P., Varvarigou, T. A., Kretsis, A., Soumplis, P., Varvarigos, E. A., "Cost and utilization optimization of Amazon EC2 instances", IEEE International Conference on Cloud Computing, CLOUD, 2013, str. 518–525.

[101] Bellur, U., Malani, A., Narendra, N. C., "Cost optimization in multi-site multi-cloud environments with multiple pricing schemes", IEEE International Conference on Cloud Computing, CLOUD, 2014, str. 689–696.

[102] Lucas-Simarro, J. L., Moreno-Vozmediano, R., S.Montero, R., Llorente, I., "Cost optimization of virtual infrastructures in dynamic multi-cloud scenarios", Concurrency Computation: Practice and Experience, Vol. 27, No. 9, 2015, str. 2260–2277.

[103] Li, W., Svärd, P., Tordsson, J., Elmroth, E., "Cost-optimal cloud service placement under dynamic pricing schemes", Proceedings - 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC 2013, 2013, str. 187–194.

[104] Eyers, D., Cross-Tier Application and Data Partitioning of Web Applications for Hybrid Cloud Deployment, 2013, No. December.

[105] Kashef, M. M., Yoon, H., Keshavarz, M., Hwang, J., "Decision support tool for IoT service providers for utilization of multi clouds", International Conference on Advanced Communication Technology, ICACT, Vol. 2016-March, 2016, str. 91–96.

[106] Srirama, S. N., Iurii, T., Viil, J., "Dynamic Deployment and Auto-scaling Enterprise Applications on the Heterogeneous Cloud", 2016.

[107] Silva, P., Perez, C., Desprez, F., "Efficient Heuristics for Placing Large-Scale Distributed Applications on Multiple Clouds", 2016 16th IEEE/ACM International Symposium

on Cluster, Cloud and Grid Computing (CCGrid), 2016, str. 483–492, dostupno na: http://ieeexplore.ieee.org/document/7515725/

[108] Jamshidi, P., Sharifloo, A., Pahl, C., Arabnejad, H., Metzger, A., Estrada, G., "Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures", Proceedings - 2016 12th International ACM SIGSOFT Conference on Quality of Software Architectures, QoSA 2016, 2016, str. 70–79.

[109] Sahni, J., Vidyarthi, D. P., "Heterogeneity-aware adaptive auto-scaling heuristic for improved QoS and resource usage in cloud environments", Computing, 2016, dostupno na: http://link.springer.com/10.1007/s00607-016-0530-9

[110] Unuvar, M., Steinder, M., Tantawi, A. N., "Hybrid Cloud Placement Algorithm", Proceedings of the 22nd IEEE International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems - MASCOTS'14, 2014, str. 197–206, dostupno na: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7033655

[111] Koziolek, A., Ardagna, D., Mirandola, R., "Hybrid multi-attribute QoS optimization in component based software systems", Journal of Systems and Software, Vol. 86, No. 10, 2013, str. 2542–2558.

[112] Li, J. Z., Lu, Q., Zhu, L., Bass, L., Xu, X., Sakr, S., Bannerman, P. L., Liu, A., "Improving availability of cloud-based applications through deployment choices", IEEE International Conference on Cloud Computing, CLOUD, 2013, str. 43–50.

[113] Sidhanta, S., Mukhopadhyay, S., "Infra: SLO aware elastic auto-scaling in the cloud for cost reduction", Proceedings - 2016 IEEE International Congress on Big Data, BigData Congress 2016, 2016, str. 141–148.

[114] Casalicchio, E., Silvestri, L., "Mechanisms for SLA provisioning in cloud-based service providers", Computer Networks, Vol. 57, No. 3, 2013, str. 795–810, dostupno na: http://dx.doi.org/10.1016/j.comnet.2012.10.020

[115] Mireslami, S., Rakai, L., Wang, M., Far, B. H., "Minimizing deployment cost of cloud-based web application with guaranteed QoS", 2015 IEEE Global Communications Conference, GLOBECOM 2015, 2015.

[116] Gandhi, A., Dube, P., Karve, A., Kochut, A., Zhang, L., "Model-driven optimal resource scaling in cloud", Software & Systems Modeling, 2017, dostupno na: http://link.springer.com/10.1007/s10270-017-0584-y

[117] Huber, N., van Hoorn, A., Koziolek, A., Brosig, F., Kounev, S., "Modeling run-time adaptation at the system architecture level in dynamic service-oriented environments", Service Oriented Computing and Applications, Vol. 8, No. 1, 2014, str. 73–89.

[118] Kritikos, K., Plexousakis, D., "Multi-cloud Application Design through Cloud Service Composition", 2015 IEEE 8th International Conference on Cloud Computing, 2015, str. 686–693, dostupno na: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7214106

[119] Andrikopoulos, V., Saez, S. G., Leymann, F., Wettinger, J., "Optimal Distribution of Applications in the Cloud", in Lecture Notes in Computer Science 8484. Springer International Publishing, 2014, str. 75–90.

[120] Srirama, S. N., Ostovar, A., "Optimal resource provisioning for scaling enterprise applications on the cloud", 2014 IEEE 6th International Conference on Cloud Computing Technology and Science, 2014, str. 262–271, dostupno na: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7037676

[121] Liu, X., Xia, C., Wei, Z., Sun, X., Zhan, Z., "Optimal Service Selection Based on Business for Cloud Computing", 2013 International Conference on Cloud and Service Computing, 2013, str. 92–97, dostupno na: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6693184

[122] Potena, P., "Optimization of adaptation plans for a service-oriented architecture with cost, reliability, availability and performance tradeoff", Journal of Systems and Software, Vol. 86, No. 3, 2013, str. 624–648, dostupno na: http://dx.doi.org/10.1016/j.jss.2012.10.929

[123] Dubois, D. J., Casale, G., "OptiSpot: minimizing application deployment cost using spot cloud resources", Cluster Computing, Vol. 19, No. 2, 2016, str. 1–17.

[124] Kaviani, N., Wohlstadter, E., Lea, R., "Partitioning of web applications for hybrid cloud deployment", Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 8275 LNCS, 2013, str. 226–246.

[125] Nanda, S., Hacker, T. J., Lu, Y.-h., "Predictive Model for Dynamically Provisioning Resources in Multi-Tier Web Applications", in 2016 IEEE 8th International Conference on Cloud Computing Technology and Science, 2016, str. 326–335.

[126] Ghavamipoor, H., Golpayegani, S. A. H., "QoS-aware provider selection in e-services supply chain", in 2016 Eighth International Conference on Information and Knowledge Technology (IKT), 2016, str. 258–262.

[127] Bjorkqvist, M., Spicuglia, S., Chen, L., Binder, W., "QoS-Aware Service VM Provisioning in Clouds: Experiences, Models, and Cost Analysis", in Lecture Notes in Computer Science 8274, 2013, str. 69–83.

[128] Liu, Z. Z., Jia, Z. P., Xue, X., An, J. Y., "Reliable Web service composition based on QoS dynamic prediction", Soft Computing, Vol. 19, No. 5, 2015, str. 1409–1425.

[129] Sun, Y., White, J., Eade, S., Schmidt, D. C., "ROAR: A QoS-oriented modeling framework for automated cloud resource allocation and optimization", Journal of Systems and Software, Vol. 116, 2016, str. 146–161, dostupno na: http://dx.doi.org/10.1016/j.jss.2015.08.006

[130] Frey, S., Fittkau, F., Hasselbring, W., "Search-based genetic optimization for deployment and reconfiguration of software in the cloud.", in International Conference on Software Engineering (ICSE-13), 2013, str. 512–521.

[131] Incerto, E., Tribastone, M., Trubiani, C., "Symbolic performance adaptation", Proceedings of the 11th, 2016, dostupno na: http://dl.acm.org/citation.cfm?id=2897060

[132] Rachkidi, E. E., Agoulmine, N., Belaid, D., Chendeb, N., "Towards an Efficient Service Provisioning in Cloud of Things (CoT)", 2016.

[133] Souza, V. B., Masip-bruin, X., Marín-tordera, E., Ramírez, W., Sánchez, S., "Towards Distributed Service Allocation in Fog-to-Cloud (F2C) Scenarios", 2016, str. 0–5.

[134] Hwang, C. L., Yoon, K., Multiple attribute decision making : methods and applications : a state-of-the-art survey. Springer-Verlag, 1981, dostupno na: https://books.google.hr/books?id=Hz-yAAAAIAAJ{&}redir{_}esc=y

[135] Censor, Y., "Applied Mathematics and Optimization Pareto Optimality in Multiobjective Problems", Applied Mathematics and Optimization, 1977, str. 41– 59.

[136] Reussner, R., Becker, S., Burger, E., Happe, J., Hauck, M., Koziolek, A., Koziolek, H., Krogmann, K., Kuperberg, M., "The Palladio Component Model", Karlsruhe, Tech. Rep. March, 2011, dostupno na: http://portal.acm.org/citation.cfm?doid=1712605.1712651

[137] Koziolek, A., Koziolek, H., Reussner, R., "PerOpteryx: Automated Application of Tactics in Multi-Objective Software Architecture Optimization", in Proceedings of the joint ACM SIGSOFT conference – QoSA and ACM SIGSOFT symposium –

ISARCS on Quality of software architectures – QoSA and architecting critical systems – ISARCS - QoSA-ISARCS '11, 2011, str. 33, dostupno na: http://portal.acm.org/citation.cfm?doid=2000259.2000267

[138] Korte, B., Vygen, J., Combinatorial Optimization, 2012, Vol. 21, dostupno na: https://www.zib.de/groetschel/pubnew/paper/groetschellovasz1995.pdf{%}5Cnhttp://link.springer.com/10.1007/978-3-642-24488-9

[139] Huang, K.-C., Shen, B.-J., "Service deployment strategies for efficient execution of composite SaaS applications on cloud platform", Journal of Systems and Software, Vol. 107, sep 2015, str. 127–141.

[140] Hadded, L., Charrada, F. B., Tata, S., "An efficient optimization algorithm of autonomic managers in service-based applications", in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), ser. Lecture Notes in Computer Science, Debruyne, C., Panetto, H., Meersman, R., Dillon, T., Weichhart, G., An, Y., Ardagna, C. A., (ur.), Vol. 9415, No. October. Cham: Springer International Publishing, 2015, str. 19–37.

[141] Sun, Y., White, J., Eade, S., Schmidt, D. C., "ROAR: A QoS-oriented modeling framework for automated cloud resource allocation and optimization", Journal of Systems and Software, aug 2015.

[142] Andrikopoulos, V., Gómez Sáez, S., Leymann, F., Wettinger, J., "Optimal Distribution of Applications in the Cloud", in Advanced Information Systems Engineering SE - 6, ser. Lecture Notes in Computer Science, Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J., (ur.). Springer International Publishing, 2014, Vol. 8484, str. 75–90.

[143] Utting, M., Pretschner, A., Legeard, B., "A taxonomy of model-based testing approaches", Software Testing, Verification and Reliability, Vol. 22, No. 5, aug 2012, str. 297–312, dostupno na: http://doi.wiley.com/10.1002/stvr.456

[144] Franceschelli, D., "Space4Cloud. An approach to system performance and cost evaluation for CLOUD", 2012, dostupno na: https://www.politesi.polimi.it/handle/10589/72688

[145] Etemaadi, R., Chaudron, M. R. V., "New degrees of freedom in metaheuristic optimization of component-based systems architecture: Architecture topology and load balancing", Science of Computer Programming, Vol. 97, No. P3, 2015, str. 366–380, dostupno na: http://dx.doi.org/10.1016/j.scico.2014.06.012

[146] Aleti, A., Björnander, S., Grunske, L., Meedeniya, I., "ArcheOpterix: An extendable tool for architecture optimization of AADL models", Proceedings of the 2009 ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software, MOMPES 2009, 2009, str. 61–71.

[147] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., "A fast and elitist multiobjective genetic algorithm: NSGA-II", IEEE Transactions on Evolutionary Computation, Vol. 6, No. 2, 2002, str. 182–197.

[148] Franceschelli, D., Ardagna, D., Ciavotta, M., Di Nitto, E., "SPACE4CLOUD: A tool for system performance and cost evaluation of CLOUD systems", MultiCloud 2013 - Proceedings of the International Workshop on Multi-Cloud Applications and Federated Clouds, 2013, str. 27–34.

[149] Ciavotta, M., Ardagna, D., Koziolek, A., "Palladio optimization suite: QoS optimization for component-based cloud applications", 9th EAI International Conference on Performance Evaluation Methodologies and Tools, ValueTools 2015, No. 1, 2015, str. 3–4.

[150] Ciavotta, M., Gianniti, E., Ardagna, D., "D-SPACE4Cloud: A Design Tool for Big Data Applications", Lecture Notes in Computer Science 10048, 2016, str. 614–629, dostupno na: http://arxiv.org/abs/1605.07083

[151] Kruntchen, P., "Architectural blueprints - the "4+1" view model of software architecture", IEEE Software, Vol. 12, No. November, 1995, str. 42–50, dostupno na: http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:Architectural+Blueprints+?+The+?+4+++1+?+View+Model+of+Software+Architecture{#}0

[152] Fuentes-Fernández, L., Vallecillo-Moreno, A., "An Introduction to UML Profiles", European Journal for the Informatics Professional, Vol. V, No. 2, 2004, str. 6–13.

[153] Metamodeling, A., "Model-Driven Development: A Metamodeling Foundation", 2003.

[154] Fielding, R. T., Taylor, R. N., "Principled design of the modern Web architecture", ACM Transactions on Internet Technology, Vol. 2, No. 2, may 2002, str. 115–150, dostupno na: http://dl.acm.org/citation.cfm?id=514183.514185

[155] Marakas, G. M., O'Brien, J. A., Introduction to Information Systems.

[156] Dennis, A., Wixom, B. H., Tegarden, D., System Analysis and Design with UML Version 2.0. John Wiley & Sons, Inc., 2005.

[157] Waldo, J., Wyant, G., Wollrath, A., Kendall, S., "A Note on Distributed Computing A Note on Distributed Computing", in Mobile Object Systems Towards the

Programmable Internet, Vitek, J., Tschudin, C., (ur.). Springer Berlin Heidelberg, 1997, Vol. 21, No. SMLI TR-94-29, str. 49–64, dostupno na: http://www.springerlink.com/index/9217063478752518.pdf

[158] Fielding, R., Reschke, J., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", 2013.

[159] Tran, N. L., Skhiri, S., Zimányi, E., "EQS: An elastic and scalable message queue for the cloud", Proceedings - 2011 3rd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2011, 2011, str. 391–398.

[160] Brewer, E., "Pushing the cap: Strategies for consistency and availability.", Computer, Vol. 45, No. 2, 2012, str. 23–29.

[161] OMG, "UML Superstructure Specification, v2.4.1", Tech. Rep. August, 2011.

[162] Sharp, H., "Cardinality of finite topologies", Journal of Combinatorial Theory, Vol. 5, No. 1, jul 1968, str. 82–86, dostupno na: http://www.sciencedirect.com/science/article/pii/S0021980068800316

[163] Bianchi, L., Dorigo, M., Gambardella, L. M., Gutjahr, W. J., "A survey on metaheuristics for stochastic combinatorial optimization", Natural Computing, Vol. 8, No. 2, 2009, str. 239–287.

[164] Miettinen, K., Multiobjective Optimization: Interactive and Evolutionary Approaches, Branke, J., Deb, K., Miettinen, K., Słowiński, R., (ur.), 2008, Vol. 1070, dostupno na: http://link.springer.com/chapter/10.1007/3-540-68339-9{_}34{%}5Cnhttp://link.springer.com/content/pdf/10.1007/3-540-68339-9{_}34.pdf

[165] Woodside, M., Franks, G., Petriu, D., "The Future of Software Performance Engineering", in Future of Software Engineering, 2007. FOSE'07. IEEE, 2007, str. 171–187, dostupno na: http://ieeexplore.ieee.org/xpl/login.jsp?tp={&}arnumber=4221619{&}url=http{%}3A{%}2F{%}2Fieeexplore.ieee.org{%}2Fxpls{%}2Fabs{_}all.jsp{%}3Farnumber{%}3D4221619

[166] Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M., "Model-based performance prediction in software development: a survey", IEEE Transactions on Software Engineering, Vol. 30, No. 5, 2004, str. 295–310, dostupno na: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1291833

[167] Avritzer, A., Kondek, J., Liu, D., Weyuker, E. J., "Software performance testing based on workload characterization", WOSP '02: Proceedings of the 3rd international workshop on Software and performance, 2002, str. 17–24.

[168] Kingman, J., Poisson processes. Oxford university press, 1992.

[169] Daley, D. J., Vere-Jones, D., An Introduction to the Theory of Point Processes: Volume I: Elementary Theory and Methods, 2008, Vol. I, dostupno na: http://www.springerlink.com/content/978-0-387-21337-8

[170] Ecuyer, P. L., "Software for Uniform Random Number Generation", in Proceedings of the Winter Simulation Conference, Vol. 1, 2001, str. 95–105.

[171] Dean, J., Barroso, L. A., "The tail at scale", Communications of the ACM, Vol. 56, No. 2, 2013, str. 74, dostupno na: http://dl.acm.org/citation.cfm?id=2408794{%}5Cnhttp://dl.acm.org/citation.cfm?doid=2408776.2408794

[172] Fiedler, M., Hossfeld, T., Tran-Gia, P., "A generic quantitative relationship between Quality of Experience and Quality of Service", Blekinge Tekniska hogskola, Vol. 24, No. March, 2010, str. 36–41.

[173] Bouch, a., Bhatti, N., Kuchinsky, a., "Quality is in the eye of the beholder: meeting users' requirements for Internet Quality of Service", Proceedings of CHI 2000, Vol. 2, No. 1, 2000, str. 297–304, dostupno na: http://discovery.ucl.ac.uk/173028/

[174] The Metro Ethernet Forum, "MEF 10.3 Ethernet Services Attributes Phase 3", Tech. Rep. November, 2013, dostupno na: http://www.metroethernetforum.org/PDF{_}Documents/technical-specifications/MEF10.2.pdf

[175] Almes, G., Kalidindi, S., Zekauskas, M., Morton, A., "RFC 7679: A One-Way Delay Metric for IP Performance Metrics (IPPM)", Tech. Rep., 2016.

[176] Van Zandt, T., "How to fit a response time distribution", Psychonomic Bulletin & Review, Vol. 7, No. 3, 2000, str. 424–465, dostupno na: http://link.springer.com/10.3758/BF03214357

[177] Menascé, D. a., "Composing Web services: A QoS view", IEEE Internet Computing, Vol. 8, No. 6, 2004, str. 88–90.

[178] Andrews, G. E., Eriksson, K., Integer Partitions, 2nd ed. Cambridge University Press, 2004, Vol. 1.

[179] Greenwald, M., Khanna, S., "Space-efficient online computation of quantile summaries", ACM SIGMOD Record, Vol. 30, No. 2, 2001, str. 58–66, dostupno na: http://dl.acm.org/citation.cfm?id=376284.375670

[180] Tanković, N., Bogunović, N., Galinac Grbac, T., Žagar, M., "Analyzing incoming workload in Cloud business services", in 2015 23rd International Conference on Software, Telecommunications and Computer Networks (SoftCOM). IEEE, sep 2015, str. 300–304.

[181] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., Introduction to algorithms, 2009.

[182] Donoso, Y., Fabregat, R., Multi-Objective Optimization in Computer Networks Using Metaheuristics, 2013, Vol. 53, No. 9.

[183] Serafini, P., Mathematics of Multi Objective Optimization, 2012, Vol. 25, No. 2.

[184] Deb, K., "Multi-Objective Optimization Using Evolutionary Algorithms", str. 497, dostupno na: https://books.google.co.in/books/about/Multi{_}Objective{_}Optimization{_}Using{_}Evolu.html?id=OSTn4GSy2uQC{&}pgis=1{%}5Cnhttps://books.google.com/books?id=OSTn4GSy2uQC{&}pgis=1 2001.

[185] Back, T., Fogel, D., Michalewicz, Z., "Evolutionary Computation 1: Basic Algorithms and Operators", Evolutionary Computation, 2000, str. 378.

[186] Gross, D., Shortle, J. F. J., Thompson, J. J. M., Harris, C. C. M., "Fundamentals of Queueing Theory", aug 2008, dostupno na: http://dl.acm.org/citation.cfm?id=1972549http://books.google.com/books?hl=en{&}lr={&}id=h7Az7KAGI2cC{&}oi=fnd{&}pg=PP1{&}dq=Fundamentals+of+queueing+theory{&}ots=PowkyJ0bmZ{&}sig=PZdUN{_}Jqr9rOjmtATctfPmfFAMk

[187] Lazowska, E. D., Quantitative System Performance: Computer System Analysis Using Queueing Network Models, 1984.

[188] Eiben, A. E., Smith, J. E., Introduction to Evolutionary Computing, 2003, dostupno na: http://books.google.com/books?id=RRKo9xVFW{_}QC{&}pgis=1

[189] Tilkov, S., Vinoski, S., "Node.js: Using JavaScript to build high-performance network programs", IEEE Internet Computing, Vol. 14, No. 6, 2010, str. 80–83.

[190] Booker, J., "Poisson Processes", Technometrics, Vol. 37, No. 1, 1995, str. 124.

[191] Novakowski, K. S., Gillham, R. W., Wei, W., "Time series analysis", Ulb.Tu-Darmstadt.De, Vol. 97, 1990, str. 23–32, dostupno na: http://www.ulb.tu-darmstadt.de/tocs/130292508.pdf{%}5Cnpapers2://publication/uuid/F09FAB0C-2CAA-46EA-B649-F986B866BF15

[192] Moldovan, D., Copil, G., Truong, H.-L., Dustdar, S., "MELA: Monitoring and Analyzing Elasticity of Cloud Services", in 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, 2013, str. 80–87.

[193] Trihinas, D., Pallis, G., Dikaiakos, M. D., "JCatascopia: Monitoring Elastically Adaptive Applications in the Cloud", 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, may 2014, str. 226–235.

[194] Easterbrook, S., Singer, J., Storey, M.-A., Damian, D., "Selecting Empirical Methods for Software Engineering Research", Guide to Advanced Empirical Software Engineering, 2008, str. 285–311, dostupno na: http://www.springerlink.com/index/n815725515063p2m.pdf

[195] Böhme, R., Reussner, R., "Validation of predictions with measurements", Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 4909 LNCS, 2008, str. 14–18.

[196] Zitzler, E., Thiele, L., "Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach", IEEE Transactions on Evolutionary Computation, Vol. 3, No. 4, 1999, str. 257–271.

[197] Knowles, J. D., Thiele, L., Zitzler, E., "A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers", Tech. Rep., 2006.

[198] Mann, H. B., Whitney, D. R., "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other", The Annals of Mathematical Statistics, Vol. 18, No. 1, 1947, str. 50–60.

[199] McGraw, K. O., Wong, S. P., "A common language effect size statistic.", Psychological Bulletin, Vol. 111, No. 2, 1992, str. 361–365, dostupno na: http://doi.apa.org/getdoi.cfm?doi=10.1037/0033-2909.111.2.361

[200] Smith, J., "Advanced MVVM", The Journal of infectious diseases, Vol. 207, No. 2, 2013, str. NP, dostupno na: http://www.ncbi.nlm.nih.gov/pubmed/23275637

[201] Paxson, V., "Empirically-Derived Analytic Models of Wide-Area TCP Connections", Statistical Methodology, Vol. 2, No. August, 1994, str. 316–336.

[202] Kruskal, W. H., Wallis, W. A., "Use of Ranks in One-Criterion Variance Analysis", Journal of the American Statistical Association, Vol. 47, No. 260, 1952, str. 583–621.

[203] Beach, B., "Elastic Block Storage", in Pro Powershell for Amazon Web Services. Berkeley, CA: Apress, 2014, str. 49–66, dostupno na: http://link.springer.com/10.1007/978-1-4302-6452-1{_}4

[204] Jedlitschka, A., Ciolkowski, M., Pfahl, D., "Reporting experiments in software engineering", Guide to Advanced Empirical Software Engineering, 2008, str. 201–228.

[205] Yin, R. K., Case Study Research: Design and Methods, 2009, Vol. 5, No. 5, dostupno na: http://books.google.com/books?id=FzawIAdilHkC{&}pgis=1

[206] Katsaros, G., Menzel, M., Lenk, A., Revelant, J. R., Skipp, R., Eberhardt, J., "Cloud Application Portability with TOSCA, Chef and Openstack", in 2014 IEEE International Conference on Cloud Engineering. IEEE, mar 2014, str. 295–302.

[207] Franks, G., Al-Omari, T., Woodside, M., Das, O., Derisavi, S., "Enhanced modeling and solution of layered queueing networks", IEEE Transactions on Software Engineering, Vol. 35, No. 2 SPEC. ISS., 2009, str. 148–161.

[208] Perez, J. F., Casale, G., "Assessing SLA compliance from Palladio component models", Proceedings - 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2013, 2014, str. 409–416.

Popis slika

2.1. Odnosi između termina *arhitektura*, *arhitekturalni stil* i *konfiguracija* 12
2.2. Dijelovi sustava zasnovanoga na komponentama 13
2.3. Aktivnosti vezane uz postavljanje komponente u rad 15
2.4. Model procesa razvoja sustava zasnovanih na komponentama 17
2.5. Odnosi između termina koji definiraju kvalitetu programske podrške 21
2.6. Odnosi između korištenih termina u definiranju radnog opterećenja 27
2.7. Dvije dimenzije skalabilnosti: vertikalna i horizontalna 28
2.8. Usporedba klasičnih metoda elastičnosti s metodama koje omogućuje virtualizacija resursa u oblaku 30
2.9. Referentna arhitektura elastičnog sustava 31
2.10. Usporedba vremena odaziva običnog i elastičnog sustava 32
2.11. Referentni model upravljačkog kruga MAPE-K 33
2.12. Model procesa koji se koristio pri provođenju sustavnog pregleda literature . 34
2.13. Korištene ključne riječi i uvjeti prilikom pretrage 36
2.14. Taksonomija pristupa optimiranju konfiguracije informacijskih sustava 38

3.1. Usporedba vremena odaziva kod sinkrone i asinkrone komunikacije između komponenti. 51
3.2. Grafička reprezentacija funkcija $f_{C\to P}$, $f_{P\to R}$ i $f_{P\to W}$ 53
3.3. Profil jezika UML za modeliranje predložene arhitekture 54
3.4. Primjer modela prema predloženom profilu jezika UML koji prikazuje komponente iz studije slučaja 55
3.5. Određivanje optimalnih funkcija $f^*_{C\to P}$, $f^*_{P\to W}$ i $f^*_{P\to R}$ u dvije međusobno zavisne faze 56
3.6. Određivanje optimalne funkcije f^*_{KP} u dvije etape 59
3.7. Model procesa razvoja sustava zasnovanog na komponentama uz optimizaciju konfiguracije 61

4.1. Metoda određivanja prikladne konfiguracije 64
4.2. Trenuci dolazaka zahtjeva prema usluzi 65

4.3. Model radnog opterećenja koji se sastoji od više nasumičnih procesa, svaki proces označuje pozive prema jednoj operaciji komponente 66
4.4. Nelinearna veza između objektivnog i subjektivnog kriterija kvalitete 69
4.5. Primjer odnosa između prosječnog vremena odaziva i različitih P-percentila vremena odaziva. Vrijednosti su prikupljene na stvarnoj programskoj usluzi iz studije slučaja iz poglavlja 7.4 70
4.6. Postupak optimizacije konfiguracije 71

5.1. Postupak optimizacije elitističkim evolucijskim algoritmom NSGA-II 82
5.2. Primjer jednog kandidata (konfiguracije) 83
5.3. Meta-model sustava za provođenje simulacije 84
5.4. Postupak provođenja simulacije 85
5.5. Primjeri otvorenih mreža redova čekanja s višestrukim klasama zahtjeva . . . 88
5.6. Primjer otvorene mreže redova čekanja s jednom elastičnom grupom za koju je određena vrsta poslužitelja sa dvije procesorske jezgre 89
5.7. Primjer reprodukcije dva kandidata roditelja 90

6.1. Pregled izvedbene okoline ElaClo 93
6.2. Oznake koje su potrebne pri definiciji komponente 94
6.3. Potrebne oznake kod implementacije komponente sa zavisnim komponentama 95
6.4. Komponente ElaClo sustava 96
6.5. Postupak postavljanja konfiguracija programske usluge 99
6.6. Primjer izvršavanja programske usluge u razvojnoj okolini ElaClo 100
6.7. Posrednička komponenta programskog okvira ElaClo za povezivanje komponenti u skladu s željenom konfiguracijom 101
6.8. Dvije moguće vrste komunikacije između komponenti k_1 i k_2 PUOP-a 102
6.9. Korisničko sučelje razvojnog okruženja ElaClo za pregled i odabir generiranih konfiguracija 103
6.10. Korisničko sučelje ElaClo razvojnog okruženja za podešavanje konfiguracije programske usluge 104
6.11. Korisničko sučelje razvojnog okruženja ElaClo za analizu rezultata. 105

7.1. Komponente *CashRegister* sustava 112
7.2. Pogreška između modela i stvarnog intenziteta u odnosu na odabrani broj odsječaka n 114
7.3. Usporedba izmjerenog stvarnog opterećenja i modela opterećenjae 114
7.4. Usporedba distribucije vremena odaziva za metodu *SaveInvoice* između stvarnog sustava i modela sa lognormalnom i eksponencijalnom distribucijom . . . 116

7.5. Usporedba kriterija kvalitete pretraženih kandidata tijekom optimizacije evolucijskim algoritmom 118
7.6. Prosječne vrijednosti kriterija kvalitete unutar populacija svake generacije . . 118
7.7. Prikaz svih Pareto skupova tokom 40 ponavljanja izvršavanja te usporedba sa globalnim Pareto skupom izvedenim pomoću kandidata svih izvršavanja . . . 121
7.8. Prosječna vrijednost indikatora pokrivenosti $\mathscr{C}^*(\mathbb{K}^*_{result}, \mathbb{K}^*_{true})$ tijekom iteracija genetskog algoritma; osjenčane vrijednosti predstavljaju interval standardne devijacije 121
7.9. Kvaliteta kandidata dobivenih putem GA ocijenjena u stvarnom okruženju pomoću alata ElaClo 122
7.10. Pareto-front kriterija kvalitete suprotnih ciljevima optimizacije određenih u simulacijskom okruženju evolucijskim algoritmom 123
7.11. Usporedni prikaz najboljih i najlošijih kandidata iz simulacijskog okruženja s kriterijima kvalitete određenim u sklopu stvarnog okruženja 124

Popis tablica

2.1. Primjer različitih latencija kod različitih aspekata informacijskih sustava . . . 22
2.2. Broj prikupljenih radova po fazama selekcije 36
2.3. Konačni popis uključene literature 37
2.4. Podjela literature po dimenzijama s obzirom na formulaciju problema 40
2.5. Podjela literature po dimenzijama s obzirom na formulaciju rješenja 41
2.6. Podjela literature po dimenzijama s obzirom na način validacije 41

3.1. Korištena notacija 60

4.1. Operacije nad OST stablom 75

5.1. Radno opterećenje za primjer simulacije 86

7.1. Prijedlog ugovora SLA po scenarijima korištenja 113
7.2. Parametri lognormalne distribucije vremena odaziva pojedinih operacija . . . 116
7.3. Vrste virtualnih poslužitelja korištenih u optimizaciji 117
7.4. Rezultat izvođenja evolucijskog algoritma za optimizaciju 119
7.5. Rezultati Kruskal-Wallisovog H testa 120

Popis algoritama

1. Algoritam za generiranje zahtjeva prema programskoj usluzi 67
2. Prva faza određivanja konfiguracija 72
3. Druga faza određivanja konfiguracija: smještanje komponenti u grupe određene prvom fazom 73
4. Procedure za učinkovitu aproksimaciju percentila iz skupa vremena odaziva . 76

Životopis

Nikola Tanković rođen je 14.3.1986. u Puli, Hrvatska. Fakultet elektrotehnike i računarstva Sveučilišta u Zagrebu upisuje 2004. godine te 2009. godine stječe titulu diplomiranog inženjera računarstva. U kolovozu 2009. godine upisuje doktorski studij elektrotehnike i računarstva Fakulteta elektrotehnike i računarstva. U listopadu 2009. godine prima priznanje rektora kao voditelj pobjedničkog tima na međunarodnom natjecanju SCORE 2009, održanog u Vancouveru (Kanada).

Od 2009. do 2014. zaposlen je u organizaciji Superius d.o.o. kao programski inžinjer gdje 2015. postaje tehnički direktor. Godine 2016. zapošljava se kao asistent na Odjelu za informacijske i komunikacijske tehnologije Sveučilišta u Puli gdje sudjeluje u održavanju nastave na kolegijima *Programsko inženjerstvo*, *Razvoj informacijskih sustava*, *Izrada informatičkih projekata* te *Dizajn i razvoj računalnih igara*. Tijekom 2016. i 2017. godine sudjeluje kao istraživač u znanstvenom projektu *Programski sustavi u evoluciji: analiza i inovativni pristupi pametnom upravljanju* kojeg vodi izv. prof. dr. sc. Tihana Galinac Grbac. Objavio je niz znanstvenih radova iz područja modeliranja programske podrške i optimizacije kvalitete informacijskih sustava. Istraživački interesi uključuju modeliranje programske podrške te predviđanje i optimiranje radnih karakteristika programskih sustava. Član je međunarodnih strukovnih udruženja ACM i IEEE.

Popis objavljenih radova

Radovi u časopisima

1. Tanković, N.; Galinac Grbac, T.; Žagar, M., "ElaClo : A Framework for Optimizing Software Application Topology in the Cloud Environment", Expert systems with applications, Vol. 90, Prosinac 2017, str. 62-86.

Radovi na međunarodnim znanstvenim skupovima

1. Tanković, Nikola; Galinac Grbac, Tihana. AGM: A DSL for Mobile Cloud Computing Based On Directed Graph, Proceedings of the 4th symposium on computer languages,

implementations and tools / Budimac, Zoran (ur.). Rhodes, Greece : AIP Conf. Proc., 2015.

2. Tanković, Nikola; Bogunović, Nikola; Galinac Grbac, Tihana; Žagar, Mario. Analyzing incoming workload in Cloud business services // Software, Telecommunications and Computer Networks (SoftCOM), 2015 23rd International Conference on / Rožić, Nikola ; Begušić, Dinko; Šarić, Matko; Šolić, Petar (ur.). Split : IEEE, 2015., str. 300-304
3. Tanković, Nikola; Grbac, Tihana Galinac; Truong, Hong-Linh; Dustdar, Schahram. Transforming Vertical Web Applications into Elastic Cloud Applications, 2015 IEEE International Conference on Cloud Engineering. IEEE Computer Society Conference Publishing Services (CPS), 2015., str. 135-144
4. Tanković, Nikola; Galinac Grbac, Tihana; Žagar, Mario. Experiences from Building an EUD Business Portal // MIPRO, 2014 Proceedings of the 37th International Convention, Petar Biljanović (ur.). Rijeka : Croatian Society for Information and Communication Technology, Electronics and Microelectronics - MIPRO, 2014., str. 635-640
5. Vukotić, Dražen; Tanković, Nikola; Žagar, Mario. Novel ICT Trends that can Empower Business Again // An Enterprise Odyssey: Corporate governance and public policy - path to sustainable future / Lovorka Galetić, Jurica Šimurina (ur.).Zagreb : Faculty of Economics and Business, University of Zagreb, 2012., str. 103-104
6. Tanković, Nikola; Vukotić, Dražen; Žagar, Mario. Rethinking Model Driven Development: Analysis and Opportunities // Information Technology Interfaces (ITI), Proceedings of the ITI 2012 34th International Conference on / Luzar-Stiffler, Vesna ; Jarec, Iva ; Bekic, Zoran (ur.). Zagreb : SRCE, 2012., str. 505-510 Tanković, Nikola; Vukotić, Dražen; Žagar, Mario.
7. Executable Graph Model for Building Data-Centric Applications // Information Technology Interfaces (ITI), Proceedings of the ITI 2011 33th International Conference on, Luzar-Stiffler, Vesna; Jarec, Iva; Bekic, Zoran (ur.). Zagreb : SRCE, 2011., str. 577-582

Biography

Nikola Tanković was born on 14th of March 1986 in Pula, Croatia. He enrolls to Faculty of Electrical Engineering and Computing in 2004 where he graduates in 2009 and starts his doctoral study at same faculty. In October 2009, he received an award from University of Zagreb for leading the winning team at the internation software engineering competition SCORE with finals held during ICSE 2009 conference in Vancouver, Canada.

In the period of 2009 to 2014 he was working at Superius d.o.o. in Pula as a software engineer where he became a chief technology officer in 2015. In 2016 he starts a teaching assistant position at the Department for Information and Communication Technologies at University of Pula, Croatia, lecturing *Software Engineering*, *Computer Games Design and Implementation*, and *Information Systems Development* courses. During 2016 and 2017 he participates in research project *Evolving Software Systems: Analysis and Innovative Approaches for Smart Management (EVOSOFT)* financed by the Croatian Science Foundation and lead by Associate Professor Tihana Galinac Grbac, PhD. He published a series of scientific papers in the field of model-driven software engineering and information system performance optimization. His research interests are in the field of modeling software application systems and software performance prediction and optimization. He is a member of ACM and IEEE international organizations.