

Disertacija Borivoj Bogdanovic (2).docx

 Singidunum University

Document Details

Submission ID

trn:oid::3117:627965933

Submission Date

Sep 22, 2026, 11:31 AM GMT+2

Download Date

Oct 20, 2026, 2:14 PM GMT+2

File Name

Disertacija Borivoj Bogdanovic (2).docx

File Size

744.0 KB

176 Pages

57,609 Words

388,414 Characters

2% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Quoted Text
- Cited Text

Top Sources

- 1%  Internet sources
- 1%  Publications
- 0%  Submitted works (Student Papers)

Integrity Flags

1 Integrity Flag for Review

-  **Replaced Characters**
27 suspect characters on 7 pages
Letters are swapped with similar characters from another alphabet.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Top Sources

- 1%  Internet sources
- 1%  Publications
- 0%  Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Internet	lara.epfl.ch	<1%
2	Internet	portal.sinteza.singidunum.ac.rs	<1%
3	Publication	Vujovic, Vladimir. "Modelom upravljani razvoj arhitekture Senzor Veb mreza.", Un...	<1%
4	Publication	Anghel Leonard. "Spring Boot Persistence Best Practices", Springer Science and B...	<1%
5	Internet	repozitorij.unizd.hr	<1%
6	Publication	Vidakovic, Milan. "Prosirivo agentsko okruzenje bazirano na java tehnologiji.", Un...	<1%
7	Internet	d-nb.info	<1%
8	Internet	www.qucosa.de	<1%
9	Internet	repozitorij.unin.hr	<1%
10	Internet	ebuah.uah.es	<1%
11	Internet	fedora.unibl.org	<1%

12	Internet	imt-atlantique.hal.science	<1%
13	Publication	Аврамовић, Дарко. "Razvoj Modela Informacionog Sistema za Podršku Upravlja...	<1%
14	Internet	bibbase.org	<1%
15	Internet	web.math.pmf.unizg.hr	<1%
16	Internet	www.cris.uns.ac.rs	<1%
17	Internet	www.ftn.uns.ac.rs	<1%
18	Internet	eclipse-esmf.github.io	<1%
19	Internet	link.springer.com	<1%
20	Internet	newsletter.pragmaticengineer.com	<1%
21	Internet	projects.eclipse.org	<1%
22	Publication	Kamps, . "BACK MATTER", Abstract Homotopy and Simple Homotopy Theory, 1997.	<1%
23	Internet	www.inf.unioeste.br	<1%
24	Publication	François Goasdoué. "Answering queries using views", ACM Transactions on Inter...	<1%
25	Publication	Kovačević, Miljan M.. "Model za prognozu i procenu troškova izgradnje armirano-...	<1%

26	Internet	docslib.org	<1%
27	Internet	mcgill.scholaris.ca	<1%
28	Internet	1library.net	<1%
29	Internet	dev.to	<1%
30	Internet	nardus.mpn.gov.rs	<1%
31	Internet	open.spotify.com	<1%
32	Internet	ris.utwente.nl	<1%
33	Internet	vdocuments.com.br	<1%
34	Internet	www.its.edu.rs	<1%
35	Publication	Bodor, Slađana M.. "Analiza Perzistentnosti i Divergentnosti Tekućeg Računa u Ev...	<1%
36	Publication	Hadnadev, Miroslav. "Funkcionalne karakteristike smesa maltodekstrina i namen...	<1%
37	Publication	Lecture Notes in Computer Science, 2002.	<1%
38	Publication	Martin Brandenburg. "Einführung in die Kategorientheorie", Springer Science an...	<1%
39	Internet	eur-lex.europa.eu	<1%

40	Internet	hr.wikipedia.org	<1%
41	Internet	katalog.bibliothek.kit.edu	<1%
42	Internet	l3.elfak.ni.ac.rs	<1%
43	Internet	repozitorij.unizg.hr	<1%
44	Internet	suvadisaakovic.ba	<1%
45	Internet	tel.archives-ouvertes.fr	<1%
46	Internet	www.fefa.edu.rs	<1%
47	Internet	www.norveskazavas.me	<1%
48	Publication	Đorđije Ivanišević. "NAMJENSKI JEZIK I OKRUŽENJE ZA MODELOVANJE GREŠAKA U ...	<1%
49	Publication	"The Semantic Web", Springer Science and Business Media LLC, 2023	<1%
50	Publication	Besic, Carisa H.. "Istrazivanje atributa benchmarkinga koji determinisu poslovnu iz...	<1%
51	Publication	Michell Guzman, Stefan Haar, Salim Perchy, Camilo Rueda, Frank D. Valencia. "Bel...	<1%
52	Publication	Xinyu Liu. "Mathematics in Programming", Springer Science and Business Media ...	<1%
53	Internet	bib.irb.hr	<1%

54	Internet	fedorabg.bg.ac.rs	<1%
55	Internet	multimedia.elsevier.es	<1%
56	Internet	repozitorij.pmf.unizg.hr	<1%
57	Internet	www.asus.com	<1%
58	Internet	www.gaf.ni.ac.rs	<1%
59	Internet	www.igi-global.com	<1%
60	Publication	"Graph Transformation, Specifications, and Nets", Springer Science and Business ...	<1%
61	Publication	Mandic, Milinko. "Razvoj platforme za standardizaciju obrazovanja nastavnika inf...	<1%
62	Publication	"The Semantic Web: ESWC 2018 Satellite Events", Springer Science and Business ...	<1%
63	Publication	Semantic Web Information Management, 2010.	<1%



Univerzitet Singidunum
Departman za poslediplomske studije

Studijski program
Inteligentno softversko inženjerstvo

Doktorska disertacija

**METODOLOGIJA ZA UBRZANI RAZVOJ
PROCESNO-SVESNIH INFORMACIONIH
SISTEMA BAZIRANA NA ONTOLOŠKOM
MODELOVANJU I TEORIJI KATEGORIJA**

Mentor:

Prof. dr Vidan Marković

Kandidat:

Borivoj Bogdanović

Broj indeksa: 2022/480143

Beograd, 2026. godine

Predgovor

Ova disertacija objedinjuje višegodišnje istraživanje razvoja procesno svesnih informacionih sistema, ontološkog modelovanja, teorije kategorija i automatizovanog generisanja programskog koda. Polazište rada bila je potreba da se znanje o domenu, arhitekturi i tehnološkim izborima zabeleži u obliku koji je istovremeno razumljiv istraživaču, proverljiv formalnim sredstvima i upotrebljiv u izgradnji softvera.

Predložena metodologija razvijana je uporedo sa softverskim okvirom BAB i njegovim primenama u različitim domenima. Zbog toga se u radu teorijski rezultati, ontološki modeli, transformacioni postupak i primeri primene razmatraju kao delovi jedne celine. Implementacioni primeri služe za proveru izvodljivosti i razjašnjenje arhitektonskih odluka, ali ne zamenjuju formalne pretpostavke i granice zaključaka koje su izričito navedene u tekstu.

Ovom prilikom zahvaljujem mentoru i ostalim članovima komisije na korisnim sugestijama pri izradi i pisanju ove disertacije.

Takođe zahvaljujem prof. dr Zoranu Petriću, naučnom savetniku na Matematičkom institutu SANU, na pomoći koja je značajno doprinela kvalitetu dela koji se odnosi na primenu teorije kategorija u predloženoj metodologiji.

Posebnu zahvalnost dugujem prof. dr Zori Konjović na stručnom usmeravanju, pažljivom čitanju rukopisa i podršci tokom celog istraživačkog procesa.

Na kraju, zahvaljujem porodici i prijateljima koji su bili uz mene tokom četiri godine rada, imali razumevanja za moj manjak slobodnog vremena i pružali mi podršku da istrajem i završim ovaj poduhvat.

Beograd, 2026. godine Borivoj Bogdanović

50

61

36

SADRŽAJ

Predgovor.....	i
1. Uvod.....	1
1.1 Motivacija i predmet istraživanja.....	1
1.2 Domet i ograničenja istraživanja	2
1.3 Istraživačka hipoteza.....	3
1.4 Istraživačka pitanja	3
1.5 Metodologija istraživanja.....	4
1.6 Naučni doprinos	5
1.7 Struktura rada.....	6
2. Pregled oblasti i srodnih istraživanja	7
2.1 Procesno svesni informacioni sistemi.....	7
2.1.1 Koncepti i klasifikacija	7
Izvorna definicija i kontekst nastanka.....	7
Četiri ortogonalne dimenzije klasifikacije	8
Karakterizacija PAIS-a relevantnih za ovu disertaciju	9
Odnos prema srodnim konceptima	10
Ograničenja tradicionalnog PAIS koncepta.....	11
2.1.2 Od PAIS ka inteligentnim PAIS	12
Prvi talas: rudarenje procesa kao most ka inteligenciji (2009).....	12
Drugi talas: fleksibilnost i dinamičko prilagođavanje (2010-2012)	13
Treći talas: paradigma usmerena na čoveka (2012).....	13
Četvrti talas: konsolidacija u IPAIS paradigmu (2017).....	14
Semantičke tehnologije kao druga osa IPAIS razvoja.....	14
Neurosimbolička veštačka inteligencija kao savremena sinteza	15
Kontinuitet istraživačkog programa i prelaz ka ABPMS-u	15
2.2 Unapređenje poslovnih procesa pomoću veštačke inteligencije i manifest ABPMS-a	16
2.2.1 Manifest ABPMS-a: vizija i pet ključnih osobina	16
Vizija i definicija ABPMS-a.....	16
Životni ciklus ABPMS-a.....	17
Pet ključnih osobina ABPMS-a	18

Istraživački izazovi u manifestu.....	20
Značaj manifesta za ovu disertaciju	21
2.2.2 Postojeći pristupi unapređenju poslovnih sistema	22
Rani pristupi zasnovani na znanju (2000-2010)	22
Ontološki pristupi (2005-2020).....	22
Pristupi zasnovani na mašinskom učenju (2015-2023)	23
Pristupi zasnovani na velikim jezičkim modelima (2022-2026)	24
Sintetički pristupi: neurosimboličko upravljanje poslovnim procesima.....	24
2.2.3 Jaz između ABPMS vizije i postojećih implementacija	25
Specifična praznina: nedostatak formalnog mosta između modela i izvršavanja..	26
2.3 Modelom vođeni razvoj softvera	26
2.3.1 OMG MDA: PIM, PSM, transformacije.....	27
Tri nivoa apstrakcije	27
UML kao osnovni notacioni jezik.....	28
Sistematska dokumentacija semantičkih problema UML-a	28
Pokušaji formalizacije: fUML, PSSM i ALF	29
Object Constraint Language kao dodatni sloj.....	29
QVT kao standardni mehanizam transformacija	30
Osnovna ograničenja MDA paradigme	31
2.3.2 Eclipse EMF/Ecore ekosistem	31
Ecore kao praktični metamodel	31
EMF kao široko korišćeni industrijski okvir	32
Platformska nezavisnost modela i prenosivost alata.....	32
Problem evolucije metamodela i modela	33
Proširenja: Xtext, Xtend i ekosistem alata.....	33
2.3.3 QVT, Aceleo i imperativne transformacije: ograničenja	33
Aceleo i pristupi zasnovani na šablonima za M2T transformacije	34
Osnovni problem imperativne prirode transformacija	34
Put ka deklarativnim i formalno zasnovanim pristupima	35
Rezime: argumenti za alternativni pristup	35
2.4 Ontološko modelovanje u softverskom inženjerstvu	36
2.4.1 OWL 2 DL i deskriptivna logika	36

Pojam ontologije u informatici	36
Razvoj OWL-a kao standardnog ontološkog jezika	37
Tri profila OWL 2: EL, QL, RL	37
OWL 2 DL i deskriptivna logika SROIQ	38
Pretpostavka otvorenog sveta i odsustvo pretpostavke o jedinstvenosti imena.....	38
2.4.2 Ontologije u razvoju softvera: pregled pristupa.....	40
Ontološki vođeni informacijski sistemi.....	40
Sistematski pregled uloga formalnih ontologija	40
OntoUML i drugi ontološki profili UML-a	40
Neurosimboličko povezivanje ontologija i savremenog mašinskog učenja	41
2.4.3 CodeOntology: ontološko modelovanje programskog koda	42
Motivacija za ontološko modelovanje koda	42
Pregled pristupa ontološkom modelovanju koda	42
CodeOntology	42
Primene i razvoj CodeOntology pristupa.....	43
Značaj CodeOntology za ovu disertaciju	43
2.5 Teorija kategorija kao formalni okvir za transformacije modela.....	44
2.5.1 Osnovni pojmovi teorije kategorija	44
Istorijski nastanak i motivacija	44
Pojam kategorije	45
Pojam funktora.....	46
Pojam prirodne transformacije.....	46
Napredniji pojmovi	47
2.5.2 Primena teorije kategorija u softverskom inženjerstvu.....	47
Rani pristupi: apstraktna specifikacija i semantika.....	47
Kategorijalna formalizacija obrazaca projektovanja	48
Sistemski okvir Kovaljova za softversko inženjerstvo	48
2.5.3 Teorija kategorija u modelom vođenom inženjerstvu	50
Programski rad Diskina i Majbauma	50
Trostruke grafovske gramatike i kategorijalna osnova.....	50
Savremeni razvoj: kategorijalna šema za softversko inženjerstvo	50
Kategorijalna arhitektura za preduzeće zasnovano na modelima	51

Značaj za metodologiju ove disertacije.....	51
2.6 Automatizacija generisanja koda	52
2.6.1 Paradigme generisanja koda u modelom vođenom inženjerstvu.....	52
Istorijski kontekst i tipologija transformacija	52
Transformacija modela u tekst kao osnova generisanja koda.....	52
Sistematski pregled automatskih tehnika generisanja koda.....	53
2.6.2 Šablonsko generisanje koda.....	53
Principi i arhitektonski okvir	53
Klasifikacija TBCG tehnika.....	54
Poređenje TBCG alata	54
Zrelost i industrijska prihvaćenost	55
2.6.3 FreeMarker kao izabrani šablonski mehanizam	55
Karakteristike i mesto FreeMarkera među alatima.....	55
Arhitektura i princip rada.....	55
Ključne funkcionalnosti relevantne za BAB metodologiju	56
Obrazloženje izbora FreeMarkera u okviru BAB-a.....	56
Zaključno o izboru FreeMarkera	57
2.7 Pozicioniranje disertacije u odnosu na stanje oblasti.....	58
2.7.1 Objedinjavanje stanja oblasti i utvrđene istraživačke praznine	58
Zrelost posebnih oblasti	58
Četiri posebne istraživačke praznine na spojevima temeljnih oblasti	59
Sažetak utvrđenih istraživačkih praznina.....	60
2.7.2 Specifični doprinosi disertacije.....	60
Prvi doprinos: ontološki PIM za procesno svesne informacione sisteme.....	60
Drugi doprinos: kategorijalna formalizacija MDA lanca iz OWL ontologije	60
Treći doprinos: reifikovana preslikavanja kao mehanizam platformске nezavisnosti.....	61
Četvrti doprinos: BAB okvir kao konkretna instanca metodologije.....	62
Peti doprinos: delimična realizacija osobina ABPMS-a u sloju softverskih agenata	62
2.7.3 Razgraničenje u odnosu na srodne pristupe.....	63
Razgraničenje od ontoloških profila UML-a	63

Razgraničenje od platformi za razvoj uz malo programiranja ili bez programiranja	64
Razgraničenje od MDA transformacija iz UML-a	64
Razgraničenje od LLM-zasnovanog generisanja koda	65
Razgraničenje od kategorijalnih formalizacija sistemskog inženjerstva	65
Razgraničenje od ontološkog modelovanja koda	65
Sažetak poglavlja 2	66
3. Ontologije kao osnova za specifikaciju modela.....	67
3.1 Arhitektura BAB ontološkog okvira.....	67
3.1.1 Modularni BAB PIM 2.0	67
3.1.2 Objedinjena kategorijalna interpretacija jezgra	68
3.1.3 BAB Mapping 2.0.....	68
3.1.4 BABj 2.0 kao platformski specifična realizacija	69
3.2 Konceptualni model PIM-a.....	70
3.2.1 Aplikacija, entitet, akter i atribut	70
3.2.2 Neutralni model binarnih relacija	70
3.2.3 Pogledi, forme i projekcije.....	71
3.2.4 Neobavezni aspekti i ugovori sposobnosti.....	71
3.3 BABj 2.0 kao izvršni PSM profil.....	72
3.3.1 Deklaracija ciljnog skupa tehnologija.....	72
3.3.2 Tipovi podataka i BAB ugovori.....	72
3.3.3 Profili preslikavanja i uloge artefakata	73
3.3.4 Izbor i objedinjavanje preslikavanja	73
3.4 Veza PIM-a, preslikavanja i PSM-a.....	73
3.4.1 Tri odgovornosti u fizički odvojenim rečnicima	73
3.4.2 Granice platformske nezavisnosti	74
3.5 Validacija ontološkog i realizacionog ugovora.....	74
3.5.1 Granica OWL zaključivanja	74
3.5.2 Provere zatvorenog sveta	75
3.5.3 TBox i ABox način modelovanja	75
3.5.4 Značenje uspešne validacije.....	75
3.6 Sažetak i prelaz na kategorijalnu formalizaciju	76
4. Teorija kategorija kao osnova za transformacije modela	77

4.1 Konstrukcija kategorija iz ontoloških skeleta	77
4.1.1 Ontološki skelet i slobodna kategorija	77
4.1.2 Objedinjena kategorija jezgra i njene pune potkategorije	78
4.1.3 Kategorija dorada preslikavanja	79
4.1.4 Prezentovana kategorija realizacionog potpisa	79
4.1.5 Kategorija generisanog koda	81
4.2 Konformansni funktor i bifunktor transformacije modela u model	81
4.2.1 Realizaciono preslikavanje	81
4.2.2 Konformansni funktor i prirodnost dorada	81
4.2.3 Evaluacioni bifunktor	83
4.3 Funktor transformacije modela u kod i kompozicija	85
4.3.1 Funktor transformacije platformski specifičnog modela u kod	85
4.3.2 Kompozitna transformacija	85
4.4 Uslovno očuvanje šematske strukture	86
4.4.1 Centralni rezultat	86
4.4.2 Posledice	87
4.4.3 Granice formalne tvrdnje	88
4.5 Validacija modela i verifikacija transformacije	88
4.5.1 Validacija domenske ontologije	88
4.5.2 Pet obaveza verifikacije	88
4.5.3 Status sadašnje implementacije	89
4.6 Sažetak	90
5. BAB softverski okvir: implementacija metodologije	91
5.1 Arhitektura BAB softverskog okvira	91
5.1.1 Slojevita arhitektura poslovnog sistema	91
5.1.2 AbstractEntity, AbstractHome, AbstractDTO: apstrakcije jezgra	93
5.1.3 GenericView, GenericWindow, GenericTable: prezentacione apstrakcije	96
5.1.4 Sposobnosti kao ugovori između PIM-a i biblioteke	98
5.1.5 Modularna realizacija biblioteke BAB 2	99
5.2 BABj: Java/Jakarta EE/Vaadin realizacija	100
5.2.1 BABj 2.0 kao profil realizacije modularnog PIM-a	100
5.2.2 Perzistentni sloj, neutralne relacije i sposobnosti entiteta	101

5.2.3 Aplikacioni, servisni i procesni sloj.....	102
5.2.4 Prezentacioni i bezbednosni sloj i omotač aplikacije	102
5.3 BAB generator: transformator modela.....	103
5.3.1 Operativni transformacioni lanac.....	103
5.3.2 Ekstrakcija pomoću Apache Jena i validacija ugovora.....	104
5.3.3 PSM razrešavanje i FreeMarker šabloni	105
5.3.4 Generisani projekat i bezbedna ponovna generacija.....	106
5.3.5 Ponovljivost, testiranje i granice prenosivosti	107
5.3.6 Sertifikat transformacione specifikacije	107
5.4 Operativni PSM ugovor i sledljivost realizacije	108
5.4.1 Preslikavanje ključnih BAB apstrakcija	108
5.4.2 Uloga WoC rečnika i granice ontološkog opisa koda.....	109
5.5 Mehanizam unapređenja pomoću veštačke inteligencije.....	110
5.5.1 AbstractAI: ontološki okvir	110
5.5.2 Preslikavanje osobina ABPMS na arhitekturu BAB	112
5.5.3 Povezivanje sa spoljnim uslugama veštačke inteligencije.....	116
6. Validacija i evaluacija.....	120
6.1 Potvrda koncepta 1: PostOffice informacioni sistem.....	121
6.1.1 Opis domena i ontološki model	121
6.1.2 Generisanje koda i rezultati	122
6.2 Potvrda koncepta 2: WasteX: kooperativni IS za sekundarne sirovine	123
6.2.1 Procesni model: sakupljanje, reciklaža, cirkulacija	123
6.2.2 Evolucija od centralizovanog ka međuorganizacionom sistemu	125
6.2.3 Složenost integracije i stabilnost arhitektonskih ugovora.....	125
6.2.4 Objavljena uporedna evaluacija.....	126
6.3 Dodatne aplikacije zasnovane na BAB bibliotekama.....	126
6.3.1 Primena BAB ugovora u istraživačkoj aplikaciji.....	126
6.3.2 Granica istraživačke evaluacije i predviđanja u stvarnoj primeni	127
6.3.3 DentaScape 2: primena modularne biblioteke BAB 2.....	127
6.4 Objavljena validaciona studija: statička klasifikacija u dva domena.....	128
6.4.1 Domet studije	128
6.4.2 Domen vrednovanja istraživačkih rezultata.....	128

6.4.3	Provera prenosivosti u prenatalnom domenu.....	129
6.4.4	Delimična merenja vremena izvršavanja.....	129
6.5	Evaluacija delimične operativne realizacije ABPMS osobina.....	130
6.5.1	Objašnjivost: upiti nad modelom i transparentnost transformacija	130
6.5.2	Adaptivnost: promena skupa tehnologija i ponašanja	131
6.5.3	Sposobnost delovanja kroz razgovor i formalne osnove za autonomiju.....	132
6.6	Inženjerska verifikacija i kvantitativni pokazatelji	133
6.6.1	Aktuelna inženjerska verifikacija	133
6.6.2	Metrike: tehnički dug, pokrivenost koda testovima i vreme razvoja.....	135
6.6.3	Poređenje sa konvencionalnim razvojem.....	136
6.6.4	Ispunjenje istraživačkih pitanja.....	137
7.	Diskusija rezultata.....	139
7.1	Doprinos u odnosu na stanje oblasti	139
7.2	Ograničenja pristupa	140
7.3	Pretnje validnosti	141
8.	Zaključak i pravci daljih istraživanja	143
8.1	Zaključak.....	143
8.2	Pravci daljih istraživanja.....	144
8.2.1	Dvosmerni inženjering.....	144
8.2.2	Puna realizacija ABPMS osobina	144
8.2.3	Realizacija alternativnih skupova platformskih tehnologija.....	144
8.2.4	Domenski specifični jezici i grafički alati za ontološko modelovanje	145
8.2.5	Proširenje regresionog korpusa.....	145
	Literatura.....	146
	Prilozi.....	157
	Prilog A: Modularni BAB PIM 2.0 (RDF/OWL).....	157
	Prilog B: BAB Mapping 2.0 i BABj 2.0 (RDF/OWL)	157
	Prilog C: Validacija, PSM razrešavanje i sertifikat transformacije	158
	Prilog D: FreeMarker šabloni	159
	Prilog E: Generisani kod: PostOffice primer	159
	Prilog F: Matrica preslikavanja osobina ABPMS-a.....	159
	Prilog G: Tehnička evidencija i reproduktivnost provera.....	160

Bibliotečki podaci o doktorskoj disertaciji	162
Sažetak	163
Library information about the doctoral dissertation	164
Abstract	165

1. Uvod

1.1 Motivacija i predmet istraživanja

Savremeni poslovni sistemi se nalaze u okruženju koje karakterišu neprekidne promene zahteva, globalizovana konkurencija, kao i sve izraženija potreba za automatizacijom operativnih procesa. Informacioni sistemi koji podržavaju takve sisteme moraju biti sposobni da upravljaju složenim poslovnim procesima koji uključuju ljude, aplikacije i izvore podataka. Koncept procesno svesnih informacionih sistema, PAIS (*Process-Aware Information Systems*), nudi odgovarajući okvir za razvoj takvih sistema, definisan kao softverski sistem koji upravlja i izvršava operativne procese na osnovu procesnih modela [1].

Najnovija istraživanja, podstaknuta razvojem savremenih informacionih tehnologija i veštačke inteligencije, ukazuju na mogućnost novog pogleda na poslovne procese. Dumas i saradnici [2] uvode koncept sistema ABPMS (*AI-Augmented Business Process Management Systems*), kod kojih tokovi izvršavanja nisu unapred određeni, prilagođavanja ne zahtevaju eksplicitne promene softverskih aplikacija, a sistem ima potencijal da autonomno otkriva i proverava mogućnosti za unapređenje, a zatim sprovodi odgovarajuća unapređenja u realnom vremenu. ABPMS manifest definiše pet ključnih osobina ovakvih sistema: autonomiju, sposobnost delovanja kroz razgovor (engl. *conversational actionability*), adaptivnost, samousavršavanje i objašnjivost.

Razvoj softvera vođen modelima MDE (*Model-Driven Engineering*) predstavlja ustaljeni pristup od kog se očekuje povećanje produktivnosti i konzistentnosti kroz automatizovanu transformaciju apstraktnih modela u izvršivi kod [3, 4]. Međutim, dominantni industrijski pristupi, OMG Model-Driven Architecture (MDA), definisan vodičem [5], i Eclipse Modeling Framework (EMF) sa metamodelom Ecore [6], pokazuju značajna ograničenja u kontekstu automatizovanog generisanja složenih poslovnih sistema.

UML kao celina ostavlja određene semantičke odluke nedovoljno preciziranim, dok fUML (*Foundational UML*) definiše preciznu izvršnu semantiku izabranog podskupa, a ALF (*Action Language for Foundational UML*) njegovu tekstualnu sintaksu [7, 8]. QVT (*Query/View/Transformation*) obuhvata deklarativne i imperativne jezike; složenost konkretnih transformacija zahteva zasebnu proveru njihovih svojstava [9, 10]. EMF ima razvijenu implementaciju u jeziku Java, ali vezanost alata za platformu treba razlikovati od platformске nezavisnosti modela koji se njime opisuje [6].

Motivacija ovog istraživanja leži u prevazilaženju ovih arhitektonskih prepreka uvođenjem pristupa zasnovanog na deskriptivnoj logici i teoriji kategorija. Predložena metodologija zamenjuje poluformalnu UML notaciju formalnom ontologijom zasnovanom na standardu OWL 2 DL [11], a transformacioni lanac opisuje kategorijalnom specifikacijom i realizuje determinističkim generatorom. BAB okvir, na kome se zasniva realizacija metodologije, predstavljen je u radu Bogdanović i saradnici [12]. Pritom, polazeći od PAIS-a kao konceptualne osnove i usvajajući ABPMS kao

dugoročni cilj, metodologija povezuje sledljivost transformacije sa zasebnim slojem agenata BAB okvira, čije komponente tokom rada sistema delimično realizuju elemente četiri ABPMS osobine; obim i granice te realizacije ispituju se u evaluaciji. Rezultat je deterministički pristup razvoju softvera u kome ontologija predstavlja jedinstven merodavni izvor specifikacije (engl. *Single Source of Truth*). Funktorska konstrukcija čuva identitete i kompoziciju, dok se očuvanje odabranog šematskog (TBox) dela ontologije obuhvaćenog transformacionom specifikacijom, odnosno njegovih tipova, relacija i strukturnih zavisnosti, tvrdi uslovno, pod pretpostavkama Propozicije 4.2. OWL aksiomi i ograničenja koji nisu uključeni u izabrani kategorijalni potpis proveravaju se zasebno u ontološkoj validaciji.

1.2 Domet i ograničenja istraživanja

S obzirom na interdisciplinarnu prirodu istraživanja, koja povezuje formalnu matematiku, ontološko inženjerstvo i razvoj poslovnih informacionih sistema, neophodno je precizno definisati predmet i opseg disertacije. Rad se usredsređuje na:

1. definisanje teorijskih osnova metodologije (ontološko modelovanje i teorija kategorija), nezavisnih od konkretne implementacije;
2. kategorijalnu formalizaciju procesa transformacije modela u kod, uključujući originalni bifunktor i mehanizam konformansnog funktora;
3. konstruisanje BAB Core OWL modela kao platformski nezavisne osnove za opis procesno svesnih informacionih sistema, zajedno sa apstraktnom hijerarhijom realizacionih strategija;
4. implementaciju metodologije kroz BAB softverski okvir, koji predstavlja jednu konkretnu instancu predložene metodologije, sa potpunom realizacijom na skupu tehnologija Java/Jakarta EE/Vaadin (BABj)¹;
5. analizu delimične realizacije elemenata ABPMS osobina tokom rada sistema kroz sledljivost generatora i komponente zasebnog sloja agenata BAB okvira; stepen podrške za objašnjivost, adaptivnost, sposobnost delovanja kroz razgovor, autonomiju u zadatim okvirima i samousavršavanje ocenjuje se prema raspoloživoj evidenciji u Poglavlju 6.

U radu se insistira na suštinskoj razlici između predložene metodologije i pratećeg softverskog okvira. Dok metodologija, utemeljena na ontološkom modelovanju, kategorijalnoj formalizaciji i reifikovanim preslikavanjima, predstavlja trajni teorijski doprinos nezavistan od skupa tehnologija, BAB softverski okvir (BABj) služi kao potvrda izvodljivosti metodologije i instrument za njenu praktičnu primenu. Ova razlika između apstraktnog modela i konkretne instance dosledno se održava kroz čitavu disertaciju.

¹ Zvanična dokumentacija: Java: <https://docs.oracle.com/javase/>; Jakarta EE: <https://jakarta.ee/>; Vaadin: <https://vaadin.com/docs/latest>

1.3 Istraživačka hipoteza

Polazeći od PAIS pristupa kao konceptualne osnove i ABPMS koncepta kao dugoročnog cilja, uz oslonac na ontološko modelovanje zasnovano na OWL 2 DL standardu za formalizaciju domenskog znanja i na teoriju kategorija kao formalni aparat za dokazivanje očuvanja eksplicitno modelovanih struktura i relacija kroz transformacioni lanac od modela do koda, moguće je izgraditi proširiv softverski okvir za ubrzani razvoj procesno svesnih informacionih sistema koji:

1. omogućava formalno modelovanje kako strukturnih tako i procesnih aspekata procesno svesnih informacionih sistema kroz jedinstven koherentan ontološki model zasnovan na OWL 2 DL standardu, sa eksplicitnom logičkom semantikom i podrškom za automatsko ontološko zaključivanje;
2. automatizovano generiše kompletne višeslojne poslovne aplikacije iz tog formalnog modela;
3. pod eksplicitno navedenim uslovima verifikacije obezbeđuje matematički dokazivo očuvanje odabranog šematskog dela ontologije obuhvaćenog transformacionom specifikacijom, odnosno njegovih tipova, relacija i strukturnih zavisnosti, prilikom transformacije ka platformski specifičnom modelu i generisanim softverskim artefaktima; OWL ograničenja koja nisu uključena u izabrani kategorijalni potpis nisu obuhvaćena ovom garancijom;
4. obezbeđuje arhitektonske preduslove, kroz mehanizam reifikovanih preslikavanja (*ImplementationMapping*), čime pokazuje da se isti model može preslikati na različite skupove tehnologija;
5. tokom rada sistema delimično realizuje elemente četiri od pet ABPMS osobina kroz konkretne komponente sloja agenata: objašnjivost kroz transparentnost ontološkog modela i nadzor aktivnosti agenata tokom izvršavanja, adaptivnost kroz reifikovana preslikavanja i dinamičke kontrole agenata tokom izvršavanja, sposobnost delovanja kroz razgovor putem SPARQL interfejsa i agenta UserHelpDeskAgent, i autonomiju u zadatim okvirima kroz prototip predlaganja ispravki i ograničene kriterijume njihovog prihvatanja; petu osobinu, samousavršavanje, podržava infrastrukturno kroz perzistentnu memoriju agenata i operativne metrike izvršavanja, dok puna realizacija svih osobina na nivou celog sistema ostaje pravac daljeg istraživanja;
6. dovodi do merljivog poboljšanja brzine razvoja, smanjenja tehničkog duga i konzistentnosti implementacije u poređenju sa konvencionalnim razvojem.

1.4 Istraživačka pitanja

Radi sistematskog razmatranja postavljene hipoteze, ova disertacija se usredsređuje na sledećih pet istraživačkih pitanja:

RQ1 (Ontologija kao PIM za PAIS): Kako se može konstruisati OWL 2 DL ontologija koja istovremeno modeluje strukturne i procesne aspekte procesno svesnih informacionih sistema, i koje konceptualne odluke takvo modelovanje zahteva?

RQ2 (Kategorijalna formalizacija transformacija): Koja se svojstva transformacionog lanca od ontologije do izvršnog koda mogu formalno garantovati kategorijalnom aparaturom (slobodne kategorije, funktori, bifunktori), a koja svojstva su izvan dometa te aparature i zavise od inženjerske discipline?

RQ3 (Reifikacija i platformska nezavisnost): Kako se mehanizam reifikovanih preslikavanja može realizovati tako da omogući zamenu ciljnog skupa tehnologija bez izmene platformski nezavisnog modela, i koja ograničenja takav pristup nameće?

RQ4 (Delimična realizacija ABPMS osobina tokom rada sistema): U kojoj meri predložena metodologija, kroz sloj agenata BAB okvira, tokom rada sistema realizuje elemente četiri ABPMS osobine, objašnjivosti, adaptivnosti, sposobnosti delovanja kroz razgovor i autonomije u zadatim okvirima, na nivou konkretnih komponenti, koje su granice te realizacije u odnosu na potpunu ABPMS viziju i u čemu se metodologija razlikuje od pristupa zasnovanih isključivo na statističkim modelima?

RQ5 (Validacija i evaluacija): U kojim aspektima i u kojoj meri se pristup razvoja zasnovan na predloženoj metodologiji razlikuje od konvencionalnog razvoja procesno svesnih informacionih sistema, kada se primeni na realnim industrijskim slučajevima?

Veza istraživačkih pitanja sa poglavljima u kojima se ona prvenstveno i dopunski razmatraju prikazana je u Tabeli 1.1.

RQ	Glavno poglavlje	Sporedno poglavlje
RQ1	3.1, 3.2	2.1, 2.4, 6.1, 6.2.1
RQ2	4	2.5, 5.3.6, 6.6.1, 7.2
RQ3	3.3, 4.2, 5.3.5	2.3, 6.2.3, 7.2
RQ4	5.5, 6.5	2.2, 2.7.3
RQ5	6.1, 6.2, 6.3, 6.4, 6.6	7.1, 7.3

Tabela 1.1. Mapiranje istraživačkih pitanja na poglavlja.

1.5 Metodologija istraživanja

Predloženo rešenje zasnovano je na sintezi principa ontološkog inženjerstva i teorije kategorija, čime uvodi suštinski drugačiji pristup od standardnih MDE praksi kroz četiri ključna mehanizma:

Formalna semantika umesto poluformalnih dijagrama. Zamena UML-a OWL 2 DL standardom omogućava redefinisavanje PIM-a. Umesto skupa statičkih dijagrama, model postaje strogo formalan sistem aksioma, podložan automatskom zaključivanju i validaciji pre samog generisanja koda. Ova formalna osnova istovremeno doprinosi objašnjivosti sistema, jer svaka komponenta generisanog softvera ima eksplicitnu semantiku koja se može dobiti upitima nad ontologijom, što je osnovni preduslov ABPMS vizije.

Kategorijalni lanac transformacija. Objedinjena kategorija C_{Core} , izvedena iz unije svih učitanih PIM modula i BAB Mapping šeme, njene pune potkategorije, kategorija dorada preslikavanja, PSM i specifikacije koda modeluju se kategorijama. Bifunktor F_s preslikava par koji čine PIM element i realizaciona strategija u platformski specifičnu specifikaciju, dok H_s tu specifikaciju projektuje u artefakt koda. Kompozicija $T_s = H_s \circ F_s$ daje formalno određen lanac. Tvrdnja o očuvanju je uslovna i odnosi se na odabranu šematsku strukturu kada su ispunjeni uslovi kompletnosti, injektivnosti, determinističnosti i kompatibilnosti transformacionih artefakata.

Reifikacija preslikavanja. Apstraktna pravila realizacije predstavljena su zasebnim, eksplicitno definisanim klasama hijerarhije `mapping:ImplementationMapping` u zasebnom BAB Mapping 2.0 rečniku, dok konkretni BABj PSM profil njihove individue specifične za dati skup tehnologija vezuje za ciljne tipove, anotacije, uloge artefakata i šablone. Time se modularni PIM 2.0 ne opterećuje detaljima jezika Java. Promena ciljne platforme zahteva novi PSM profil i odgovarajuću podršku generatora, ali ne zahteva promenu domenskog modela.

Ontologija kao interfejs za sposobnost delovanja kroz razgovor. Formalni ontološki model, zasnovan na standardima RDF i OWL [13, 11], podržava SPARQL upite nad celokupnom strukturom sistema [14]. Ovo omogućava programsku interakciju sa modelom i, potencijalno, interakciju kroz razgovor posredstvom buduće komponente koja bi upite korisnika na prirodnom jeziku prevodila u SPARQL upite. Korisnik ili agent veštačke inteligencije tada bi mogao da postavlja pitanja o sistemu, njegovoj strukturi i ponašanju, što predstavlja osnovu za *sposobnost delovanja kroz razgovor* (engl. *conversational actionability*) definisanu ABPMS manifestom.

1.6 Naučni doprinos

Ova disertacija donosi sledeće originalne naučne doprinose:

1. **Modularni BAB PIM 2.0 za PAIS.** OWL model čije strukturno jezgro BAB Core obuhvata aplikaciju, entitete, aktere, atribut i neutralne binarne relacije, dok odvojeni moduli opisuju prikaze, ponašanje, validaciju, bezbednost, servise, klasifikaciju i platformski neutralne ugovore sposobnosti.
2. **Kategorijalna formalizacija generisanja koda.** Formalizacija procesa kao kompozicije bifunktora za PIM, preslikavanje i PSM i funktora transformacije PSM-a u kod, sa uslovnim rezultatom o očuvanju odabranog šematskog dela.
3. **Reifikovana preslikavanja kao mehanizam platformske nezavisnosti.** Apstraktna hijerarhija `mapping:ImplementationMapping` u BAB Mapping 2.0 rečniku i realizacione individue specifične za dati skup tehnologija u BABj PSM ontologiji čine pravila transformacije eksplicitnim i proverljivim. Izvodljivost je potvrđena za BABj, dok drugi skupovi tehnologija ostaju arhitektonska mogućnost.
4. **BAB softverski okvir.** Implementacija predložene metodologije kao proširivog softverskog okvira za skup tehnologija Java/Jakarta EE/Vaadin. Generator je

proveren testovima, validacijom referentnog PIM-a, sertifikatom transformacione specifikacije i uspešnom izgradnjom generisanog WAR projekta pomoću alata Maven². Generisana aplikacija PostOffice raspoređena je i pokrenuta na serveru WildFly 41.0.1³, gde su ručno provereni prijavljivanje, CRUD operacije, navigacija, filtriranje i modelovano ponašanje. BAB softverski okvir je konkretna instanca univerzalne metodologije opisane u doprinosima od 1 do 3.

5. **Sloj agenata BAB okvira i delimična realizacija ABPMS osobina tokom rada sistema.** Zasebni sloj obuhvata upravljanje agentima, nadzor događaja, podršku za razgovor i prototip predlaganja ispravki. Zajedno sa sledljivošću ontološke transformacije, ove komponente tokom rada sistema realizuju ograničene elemente četiri ABPMS osobine. Evaluacija u potpoglavlju 6.5 razlikuje postojanje i ponašanje komponente tokom izvršavanja od potpune realizacije osobine na nivou sistema: kriterijumi prihvatanja predloga nisu formalni dokaz bezbednosti radnje, SPARQL konzola i UserHelpDeskAgent ne realizuju u potpunosti sposobnost delovanja kroz razgovor, a memorija razgovora nije dokaz samousavršavanja.

Formalnu osnovu disertacije neposredno razvija rad Bogdanović i saradnici [15]. Opšti kategorijalni pojmovi na kojima se taj okvir zasniva provereni su prema nezavisnim izvorima, dok su potpun razvoj aparata i granice tvrdnji izloženi u Poglavlju 4. Rad [16] pruža objavljenu empirijsku proveru statičke klasifikacije vođene ontologijom. Poglavlja od 3 do 5 opisuju implementaciju generatora koja je predmet ove disertacije.

1.7 Struktura rada

Disertacija je organizovana u osam osnovnih poglavlja, iza kojih slede literatura i prilozi. Nakon uvoda, u Poglavlju 2 dat je pregled oblasti i srodnih istraživanja. Poglavlja 3, 4 i 5 čine centralni deo disertacije. Poglavlje 3 predstavlja modularni BAB PIM, zaseban rečnik BAB Mapping i BABj PSM, a Poglavlje 4 njihovu kategorijalnu formalizaciju. Poglavlje 5 opisuje BAB softverski okvir i generator kao realizaciju metodologije. Poglavlje 6 sadrži validaciju i evaluaciju, Poglavlje 7 diskusiju rezultata, a Poglavlje 8 zaključak i pravce daljih istraživanja.

² Zvanična stranica alata Apache Maven: <https://maven.apache.org/>

³ Zvanična stranica aplikacionog servera WildFly: <https://www.wildfly.org/>

2. Pregled oblasti i srodnih istraživanja

2.1 Procesno svesni informacioni sistemi

2.1.1 Koncepti i klasifikacija

Procesno svesni informacioni sistemi (engl. *Process-Aware Information Systems*, PAIS) predstavljaju klasu softverskih sistema koja objedinjuje upravljanje operativnim poslovnim procesima sa integracijom ljudskih i softverskih učesnika na osnovu eksplicitnih procesnih modela. Pojam je formalno uveden u monografiji koju su uredili Dumas, Van der Aalst i Ter Hofstede [1], koja je ostala jedna od temeljnih referenci u ovoj oblasti. Njena definicija koristi se i dalje razvija u kasnijoj literaturi, od pregledne studije Van der Aalsta [17] do savremenog manifesta koji su objavili Dumas i drugi [2]. Odeljak koji sledi izlaže izvornu definiciju i klasifikaciju, razmatra odnos ovog pojma prema srodnim sistemima (radnim tokovima, sistemima upravljanja poslovnim procesima i sistemima za upravljanje resursima preduzeća) i identifikuje ključna ograničenja tradicionalnog pristupa PAIS-u koja motivišu istraživanje predstavljeno u ovoj disertaciji.

Izvorna definicija i kontekst nastanka

U uvodnom poglavlju monografije [1], urednici Dumas, Van der Aalst i Ter Hofstede usvajaju sledeću definiciju: „*Softverski sistem koji na osnovu procesnih modela upravlja operativnim procesima i izvršava ih, uz učešće ljudi, aplikacija i/ili izvora informacija*“ [1, str. 7]. Autori odmah u nastavku preciziraju da procesni modeli obično bivaju predstavljeni vizuelnim jezicima (na primer Petrijevim mrežama), da se tipično instanciraju više puta (za svaku novu pojavu, poput porudžbine) i da se svaka instanca obrađuje na unapred definisan način, uz mogućnost varijacija [1, str. 7].

Sam pojam procesne svesnosti (engl. *process-awareness*) autori suprotstavljaju aplikacijama koje podržavaju pojedinačne zadatke bez razumevanja šireg procesnog konteksta. Kao ilustraciju navode da uređivač teksta ili klijent elektronske pošte nisu procesno svesni, jer „zadatak u procesu može dovesti do slanja poruke elektronske pošte, ali klijent elektronske pošte nije svestan procesa u kojem se koristi“ [1, str. 7]. Upravo ta razlika, između sistema koji podržava zadatke i sistema koji razume procese, čini konceptualno jezgro paradigme PAIS-a.

Motivacija za uvođenje koncepta proistekla je iz uočenog pomeranja težišta u informacionim sistemima tokom devedesetih godina dvadesetog veka. U predgovoru monografije autori govore o „tihoj revoluciji“ (engl. *silent revolution*) u kojoj se težište sa podataka premešta na procese. Kao ključne prepoznaju tri pojave: prelazak sa programiranja pojedinačnih modula na sastavljanje (engl. *from programming to assembling*), premeštanje težišta sa podataka na procese i kretanje od pažljivo projektovanih rešenja ka redizajnu i organskom rastu [1, odeljak 1.1]. Drugi važan razlog za uvođenje jedinstvenog pojma bilo je mnoštvo neusaglašenih pristupa, koje autori nazivaju *Babel of approaches*: slične, ali u pojedinostima različite tehnologije za

procesnu svesnost informacionih sistema trebalo je objediniti pod zajedničkim teorijskim okriljem. Monografija je istovremeno zamišljena kao udžbenik i kao manifest koji ujedinjuje srodne tehnologije: sisteme upravljanja radnim tokovima, sisteme za podršku saradnji, platforme za integraciju preduzeća, jezike poslovne orkestracije servisa i slične pristupe [1, str. 13].

Četiri ortogonalne dimenzije klasifikacije

U odeljku 1.4 monografije (str. 11-16) autori uvode četiri ortogonalne dimenzije za klasifikaciju PAIS rešenja. Te dimenzije nisu međusobno isključive, već zajedno obrazuju višedimenzioni klasifikacioni prostor u kojem se može odrediti mesto svakog konkretnog sistema. Van der Aalst [17] preuzima isti četvorodimenzioni okvir, što potvrđuje njegovu kasniju upotrebu u literaturi.

Prva dimenzija, orijentacija prema životnom ciklusu procesa, razlikuje sisteme orijentisane ka dizajnu (engl. *design-oriented*) od sistema orijentisanih ka implementaciji (engl. *implementation-oriented*). Životni ciklus PAIS-a obuhvata četiri faze: dizajn procesa, implementaciju procesa, izvršavanje (engl. *enactment*) i dijagnozu. Alati orijentisani ka dizajnu (tipično alati za modelovanje poslovnih procesa poput sistema ARIS, engl. *Architecture of Integrated Information Systems*) fokusiraju se na gornju polovinu ciklusa i najčešće nude biblioteke referentnih modela pogodnih za prilagođavanje [18]. Alati orijentisani ka implementaciji, među koje spada i sistem Staffware [19], pokrivaju donju polovinu ciklusa i podržavaju implementaciju i izvršavanje, ali slabije podržavaju dizajn i dijagnozu [1, str. 11-12]. Autori pominju i hibridne sisteme koji pokrivaju širi deo ciklusa pod cenu sužavanja domena (na primer alati za upravljanje projektima).

Druga dimenzija, priroda učesnika, klasifikuje sisteme prema tome da li procesi uključuju primarno ljude, primarno softverske aplikacije ili njihovu kombinaciju. Autori razlikuju tri podtipa: *person-to-person* (P2P), *person-to-application* (P2A) i *application-to-application* (A2A). U P2P procesima učesnici su pretežno ljudi, a aplikacije (kao što su alati za praćenje projekata, klijenti elektronske pošte ili sistemi za video-konferencije) služe posredovanju interpersonalne komunikacije. P2A procesi „obuhvataju i ljudske zadatke i interakcije među ljudima, kao i zadatke i interakcije u kojima učestvuju aplikacije koje deluju bez ljudske intervencije. Sistemi za upravljanje radnim tokovima pripadaju kategoriji P2A jer im je prvenstveni cilj da omoguće ljudima i aplikacijama da rade na integrisan način“ [1, str. 12]. U A2A procesima učesnici su isključivo softverski sistemi, što je tipično za distribuirano računarstvo i integraciju aplikacija na nivou preduzeća. Autori eksplicitno upozoravaju da granice između P2P, P2A i A2A nisu oštre, već postoji kontinuum tehnika i alata od P2P (ručno, vođeno ljudima) do A2A (automatizovano, vođeno aplikacijama) [1, str. 12].

Treća dimenzija, struktura i predvidljivost procesa, razlikuje četiri vrednosti: neuokvirene (engl. *unframed*), ad hoc uokvirene (engl. *ad hoc framed*), labavo uokvirene

(engl. *loosely framed*) i čvrsto uokvirene (engl. *tightly framed*) procese. Definicije izložene u radu Van der Aalsta [17] glase:

- *Neuokvireni* proces nema eksplicitni procesni model; tipično se javlja u kolaborativnim procesima podržanim generičkim alatima za grupni rad bez ugrađene podrške za procesno modelovanje.
- *Ad hoc uokvireni* proces ima unapred definisan model koji se izvršava samo jednom ili svega nekoliko puta pre nego što bude izmenjen ili odbačen. Primeri su upravljanje projektima (gde je „procesni model“ zapravo plan projekta) i naučni radni tokovi (engl. *scientific workflows*) koji se pokreću po potrebi za konkretnu analizu.
- *Labavo uokvireni* proces jeste proces „za koji unapred postoji definisan procesni model i skup ograničenja, tako da unapred definisan model opisuje uobičajen način rada, dok stvarnim izvršavanjima procesa dopušta da u određenim granicama odstupaju od tog modela“ [1, str. 14]. Tipičan primer su sistemi za obradu slučajeva (engl. *case handling*).
- *Čvrsto uokvireni* proces „dosledno prati unapred definisan procesni model. To je slučaj kod tradicionalnih sistema za upravljanje radnim tokovima“ [1, str. 14].

Granice između ove četiri vrednosti takođe nisu oštre: proces koji je prvobitno čvrsto uokviren može vremenom postati labavo uokviren ako se instance počnu učestalo odvajati od modela; obrnuto, proučavanje istorije izvršavanja labavo uokvirenog procesa može otkriti zajedničku strukturu koja opravdava preciznije definisanje okvira [1, 17, str. 14].

Četvrta dimenzija, organizacioni domet, razlikuje unutar-organizacione (engl. *intraorganizational*) i među-organizacione (engl. *interorganizational*) sisteme. Autori ističu istorijski zaokret: „U početku su PAIS sistemi bili pretežno usmereni na okruženja unutar jedne organizacije. [...] Poslednjih godina težište se pomerilo ka procesima koji prelaze organizacione granice. Takvi međuorganizacioni procesi mogu biti jedan-prema-jedan, odnosno zasnovani na bilateralnim odnosima, jedan-prema-više [...] ili više-prema-više“ [1, str. 15]. Taj trend su u osnovi pokretali usvajanje servisno-orijentisane arhitekture i pojava jezika za orkestraciju Veb servisa BPEL (engl. *Business Process Execution Language*) [20].

Karakterizacija PAIS-a relevantnih za ovu disertaciju

Na osnovu četiri izvorne dimenzije iz 2005. godine može se precizno odrediti klasa PAIS rešenja koja je predmet ove disertacije: poslovni informacioni sistemi za mala i srednja preduzeća, sa posebnim naglaskom na podršku razvoju i isporuci softverskih rešenja. Takvi sistemi su specifični za određeni domen, u smislu koji navode Dumas i drugi [21], jer procesna logika odražava posebnosti konkretnog poslovnog domena i ne može se svesti na generičke tokove. Istovremeno su labavo uokvireni, u smislu koji navode Dumas i drugi [1, str. 14]: postoje referentni modeli procesa, ali stvarna izvršavanja značajno variraju u zavisnosti od konteksta, uz dopuštena odstupanja od normativnog modela. Procesni su međuorganizacioni jer obuhvataju saradnju različitih poslovnih subjekata

(klijenata, dobavljača, partnera i regulatornih tela). Sistemi su usmereni na implementaciju, pa su njihovi primarni ciljevi izvršavanje, automatizacija i nadgledanje stvarnih poslovnih procesa, a ne samo njihovo modelovanje. Konačno, učesnici su mešoviti: sistem kombinuje ljudske aktere i softverske aplikacije, tako da u Dumasovoj taksonomiji P2P/P2A/A2A pripada kategoriji P2A sa elementima toka A2A.

Odnos prema srodnim konceptima

Sistemi za upravljanje radnim tokovima (engl. *Workflow Management Systems*, WfMS) istorijski su prethodili PAIS-u i predstavljaju njegov najuži podskup. Van der Aalst i Van Hee [22] definišu WfMS kao sisteme koji obezbeđuju da prava informacija dospeva do pravog lica ili računarske aplikacije u pravo vreme. Georgakopoulos, Hornick i Sheth [23] utvrdili su metodološki okvir od modelovanja i reinženjeringa do automatizacije. Ključna razlika je u tome što WfMS podržava pretežno strukturirane, ponovljive procese sa strogom automatizacijom kontrolnog toka, dok PAIS obuhvata i nestrukturirane scenarije zasnovane na znanju i podacima, kao i scenarije u kojima čovek učestvuje u odlučivanju. Van der Aalst [24] izričito tvrdi da PAIS obuhvata tradicionalne WFM sisteme i moderne BPM sisteme, ali i sisteme koji pružaju veću fleksibilnost ili podržavaju specifične procese.

Sistemi za upravljanje poslovnim procesima (engl. *Business Process Management Systems*, BPMS) predstavljaju tehnološku realizaciju discipline BPM i takođe čine podklasu PAIS-a. Referentno delo je monografija Weskea [25], čije treće izdanje određuje BPM kao disciplinu koja se bavi podrškom procesima iz šire perspektive, uključujući različite vrste analize (simulaciju, verifikaciju i rudarenje procesa) i povezivanje procesa sa poslovnim i društvenim aspektima. Razlika u odnosu na WfMS leži u obuhvatu životnog ciklusa: BPMS pokriva identifikaciju, otkrivanje, analizu, redizajn, implementaciju i nadgledanje, dok je WfMS usmeren na izvršavanje. U odnosu na PAIS, BPMS je tehnološki realizovana podklasa koja izričito podržava celokupan ciklus BPM-a.

Sistemi za upravljanje resursima preduzeća (engl. *Enterprise Resource Planning*, ERP), poput platformi SAP i Oracle, takođe se mogu smatrati vidom PAIS-a, ali njegovim najrigidnijim vidom. Van der Aalst [24] ih klasifikuje zajedno sa CRM sistemima i velikom međuslojnom infrastrukturom kao procesno svesne sisteme koji ne moraju nužno kontrolisati procese pomoću generičkog mehanizma radnih tokova; umesto toga, karakteriše ih postojanje eksplicitnog pojma procesa i činjenica da je informacioni sistem svestan procesa koje podržava. U taksonomiji koju navode Dumas i drugi [1], ERP sistemi pripadaju sistemima usmerenim na implementaciju sa čvrsto uokvirenim procesima, u kojima je izmena procesne logike moguća samo parametrizacijom ili programskim izmenama. Van der Aalst [17, str. 2] to precizira: iako ERP sistemi poput SAP-a imaju mehanizam za radne tokove, većina procesa je programski kodirana u sistem i može se menjati samo programiranjem ili izmenom konfiguracionih parametara, zbog čega su izmene ili vremenski zahtevne ili ograničene na unapred definisan skup parametara.

Ograničenja tradicionalnog PAIS koncepta

Kritička analiza savremene literature ukazuje na četiri međusobno povezana ograničenja tradicionalnog pristupa PAIS-u koja podstiču njegov dalji razvoj i neposredno su značajna za istraživački problem ove disertacije.

Prvo ograničenje, kruta struktura i teško prilagođavanje promenama. Reichert i Weber [26, poglavlje 3] ističu da u mnogim oblastima primene korisnici neće prihvatiti PAIS ako ih on ograničava. Unapred specificirani imperativni modeli (mreže radnih tokova, notacija BPMN) ne mogu bez znatne tehničke intervencije da podrže nepredvidive procese zasnovane na znanju, što je posebno izraženo u razvoju softvera. Adams i drugi [27] kritikuju tradicionalne WfMS jer tipično zahtevaju da model bude potpuno definisan pre nego što se može instancirati, dok se promene moraju unositi statičkim izmenama modela.

Drugo ograničenje, nedostatak semantičke strogosti. Hepp, Leymann, Domingue, Wahler i Fensel [28] ukazuju na to da je stepen mehanizacije u BPM-u i dalje vrlo ograničen, kao i da BPM ne pruža zaista jedinstven pogled na sve procese organizacije. Autori taj problem posmatraju pre svega kao ontološki, odnosno kao posledicu nedostatka mašinski dostupne semantike. Oznake na prirodnom jeziku u modelima koji nemaju formalnu ontološku utemeljenost onemogućavaju automatsko rezonovanje, otkrivanje i verifikaciju procesa.

Treće ograničenje, ograničena sposobnost učenja iz podataka. Tradicionalni PAIS je projektovan odozgo-nadole, na osnovu pretpostavljenih procesa, bez mehanizma za otkrivanje modela iz tragova izvršavanja, proveru usaglašenosti i obogaćivanje modela pokazateljima učinka. Van der Aalst [29] u drugom izdanju monografije formalizuje rudarenje procesa (engl. *process mining*) kao odgovor na ovu prazninu, ali konstatuje da klasični PAIS ne uključuje analitiku podataka u jezgro svoje arhitekture.

Četvrto ograničenje, slaba integracija sa tehnikama veštačke inteligencije. Dumas i drugi [2] najjasnije formulišu ovu prazninu, naglašavajući da u tradicionalnom BPMS-u svaki zadatak ili odluku u procesu pokreće ili ljudski akter ili softverska aplikacija koja izvršava zadatak prema unapred određenoj poslovnoj logici, dok u sistemima koji su unapređeni veštačkom inteligencijom sistem sam odlučuje da li će se zadatak uopšte izvršiti i ako hoće, kako će biti izvršen, bez unapred definisanih skripti izvršavanja.

U odnosu na savremene arhitektonske trendove, tradicionalni PAIS pokazuje slabu usklađenost sa mikroservisnim arhitekturama i arhitekturama zasnovanim na događajima (engl. *event-driven*), kao i sa načelima sistema izvorno projektovanih za oblak (engl. *cloud-native*). Primer nove generacije platformi je Camunda Platform 8, objavljena u aprilu 2022. godine, sa izvršnim jezgrom Zeebe, razvijenim kao distribuirani mehanizam zasnovan na tokovima događaja umesto na relacionoj bazi podataka. Zeebe je projektovan za horizontalno skaliranje i otpornost na otkaze, dok je za samostalno produkciono postavljanje platforme Camunda 8 zvanično podržan Kubernetes [30, 31]. Takva

arhitektura pomera paradigmu PAIS-a ka distribuiranoj, elastično skalabilnoj obradi događaja.

Četiri utvrđena ograničenja (kruta struktura, semantička siromašnost, slaba sposobnost učenja iz podataka i nedovoljna integracija sa veštačkom inteligencijom) zajedno određuju pravac razvoja koncepta PAIS-a od 2005. godine do danas. Razvoj ka inteligentnim procesno svesnim sistemima (IPAIS), koji je predmet narednog potpoglavlja, predstavlja sistematski odgovor na ta ograničenja kroz uvođenje prilagodljivosti, usmerenosti na čoveka i integracije sa tehnikama veštačke inteligencije.

2.1.2 Od PAIS ka inteligentnim PAIS

Koncept PAIS-a, čija je definicija ustaljena 2005. godine, tokom dve decenije prošao je kroz nekoliko talasa istraživanja. U svakom od njih izvorne ortogonalne dimenzije klasifikacije dopunjavane su novim stanovištima, pri čemu osnovna definicija iz monografije Dumas, Van der Aalst i Ter Hofstede [1] nije dovedena u pitanje. Savremena istraživanja nisu bila usmerena na zamenu paradigme PAIS-a, već na njeno proširenje uključivanjem fleksibilnosti, usmerenosti na čoveka i kognitivnih slojeva veštačke inteligencije u postojeći klasifikacioni okvir. Tako je nastao termin inteligentni procesno svesni informacioni sistemi (engl. *Intelligent Process-Aware Information Systems*, IPAIS), koji označava klasu PAIS rešenja obogaćenu kognitivnim i adaptivnim sposobnostima.

Prvi talas: rudarenje procesa kao most ka inteligenciji (2009)

Pregledna studija Van der Aalsta [17] predstavlja važno polazište istraživanja koja povezuju PAIS i rudarenje procesa. Studija kritički razmatra unapred specificirane modele: „*Uvidi dobijeni rudarenjem procesa veoma su dragoceni za razvoj sledeće generacije PAIS sistema jer jasno pokazuju nesaglasnost između modela predloženih za upravljanje tim sistemima i stvarnosti. S jedne strane, modeli često previše pojednostavljaju stvarnost, zbog čega sistemi postaju suviše ograničavajući. S druge strane, modeli ne obuhvataju važne aspekte poslovnih procesa*“ [17, str. 2]. Ova dijagnoza opisuje neslaganje između projektovanih modela i empirijskih tragova izvršavanja i motiviše dalje povezivanje procesnog modelovanja sa podacima o izvršavanju.

Rudarenje procesa (engl. *process mining*), čije osnove Van der Aalst kasnije kodifikuje u monografiji [29] i Manifestu IEEE radne grupe o rudarenju procesa, usvojenom 2011. i objavljenom 2012. godine [32], uvodi tri osnovne klase aktivnosti koje bogate klasični PAIS: otkrivanje modela (engl. *discovery*), induktivno izvođenje procesnog modela iz dnevnika izvršavanja; proveru usaglašenosti (engl. *conformance checking*), poređenje zabeleženog ponašanja sa normativnim modelom radi detekcije odstupanja; i obogaćivanje modela (engl. *enhancement*), proširenje postojećeg modela podacima o performansama, uzrocima kašnjenja ili rizicima. Za razliku od tradicionalnog PAIS pristupa, u kojem se model projektuje odozgo-nadole i zatim primenjuje na instance,

rudarenje procesa uvodi odozdo-nagore perspektivu u kojoj model evoluirao na osnovu stvarnog ponašanja sistema. Time se otvara put ka PAIS-u koji izvršava procese i uči na osnovu sopstvenog ponašanja.

Drugi talas: fleksibilnost i dinamičko prilagođavanje (2010-2012)

U drugoj fazi razvoja naglasak je bio na fleksibilnosti. Reichert i Weber [26] uvode taksonomiju koja obuhvata četiri oblika fleksibilnosti: varijabilnost (engl. *variability*), labavost (engl. *looseness*), adaptaciju (engl. *adaptation*) i evoluciju (engl. *evolution*). Ovi prevodi se u nastavku koriste kao nazivi kategorija iz izvorne taksonomije. Varijabilnost označava potrebu za različitim varijantama istog procesa u različitim kontekstima; labavost dozvoljava odstupanja od normativnog modela tokom izvršavanja; adaptacija se odnosi na izmene instance procesa u toku rada; evolucija obuhvata promene samog modela tokom vremena. Autori izričito ukazuju na ograničenje tradicionalnog pristupa: tradicionalni PAIS sistemi bili su usmereni na podršku predvidljivim i ponovljivim poslovnim procesima, dok se sve više koriste za podršku manje strukturiranim procesima zasnovanim na znanju [26, poglavlje 3].

Paralelno sa Reichertovim i Weberovim radom, Adams, Ter Hofstede, Russell i Van der Aalst [27] predstavljaju paradigmu *Worklet Service*, zasnovanu na teoriji aktivnosti. Rad menja način posmatranja procesnog modela: umesto unapred definisanog plana koji se mora strogo pratiti, model se posmatra kao „vodič ka cilju aktivnosti“ (prevod autora), koji učesnici mogu ponovo tumačiti u skladu sa kontekstom. *Worklet Service* podržava dinamičku i proširivu fleksibilnost, evoluciju i obradu izuzetaka u poslovnim procesima na osnovu saznanja o tome kako ljudi zaista obavljaju svoje radne zadatke. Ovaj zaokret od procesa kao unapred zadatog izvršnog scenarija ka procesu kao strukturiranom kontekstu odlučivanja postavio je osnovu za kasnije radove o inteligentnim PAIS sistemima.

Treći talas: paradigma usmerena na čoveka (2012)

Fazu razvoja usmerenu na čoveka predstavlja rad Kabicher-Fuchs, Rinderle-Ma, Recker, Indulska, Charoy, Christiaanse, Dunkl, Grambow, Kolb, Leopold i Mendling [33]. Reč je o tehničkom izveštaju zasnovanom na rezultatima istraživanja u fokusnim grupama, koji definiše humano-centrične procesno svesne informacione sisteme i izdvaja buduće istraživačke izazove. Motivacija polazi od činjenice da potpuna automatizacija procesa u pojedinim oblastima nije praktična, pa su ljudske sposobnosti, procene i odluke nužan deo procesa. Okvir HC-PAIS razmatra radno iskustvo, motivaciju, zadovoljstvo i procesnu pismenost učesnika kao relevantne činioce projektovanja takvih sistema.

Paralelno, Kabicher-Fuchs i Rinderle-Ma [34] predlažu pokazatelje radnog iskustva i prikazuju kako se oni mogu primeniti pri dodeli zadataka na osnovu prethodnih aktivnosti. Radno iskustvo se ne posmatra kao nepromenljiva karakteristika korisnika, određena godinama staža ili položajem, već kao skup sposobnosti koji se razvija kroz obavljene aktivnosti i njihove ishode. Ova perspektiva omogućava sistemu da uči o

sposobnostima ljudskih učesnika, što se može tumačiti kao rana forma „inteligencije“ u kontekstu PAIS-a.

Četvrti talas: konsolidacija u IPAIS paradigmu (2017)

Kulminacija prethodnih talasa dolazi 2017. godine objavljivanjem zbornika Grambow, Oberhauser i Reichert (ur.) [35]. Uvodno poglavlje urednika, „On the Fundamentals of Intelligent Process-Aware Information Systems“, formalizuje pojam IPAIS i predstavlja ga kao sistematski odgovor na ograničenja tradicionalnog PAIS-a. Urednici konstatuju: dok je poslovno okruženje izloženo visokom stepenu promena, sa razvojem kao što su globalizacija koja zahteva nove pristupe i tehnologije poput računarstva u oblaku, velikih podataka, mobilnih aplikacija ili obrade događaja, PAIS sistemi nisu pratili te trendove kako bi adekvatno upravljali procesima [35, poglavlje 1].

Urednici definišu IPAIS kao PAIS obogaćen sposobnostima koje se mogu grupisati u četiri dimenzije inteligencije: kontekstualnu svest (sposobnost sistema da prepozna promene u radnom okruženju i odgovori na njih, uključujući obeležja poput lokacije, vremena, dostupnih resursa i sklonosti korisnika); semantičko rezonovanje (sposobnost formalnog predstavljanja domenskog znanja i automatskog zaključivanja nad njim, tipično zasnovanog na ontologijama i semantičkim veb servisima); prediktivnu analitiku (sposobnost predviđanja ishoda instance procesa, vremena izvršavanja ili verovatnoće izuzetaka na osnovu istorijskih podataka, tipično primenom mašinskog učenja); i autonomne sposobnosti (sposobnost sistema da samostalno donosi odluke o toku procesa, na primer, da automatski ponovo dodeljuje zadatke, dinamički restrukturira proces ili pokreće kompenzacije aktivnosti kao odgovor na otkrivene izuzetke).

Pored definicije uloge, ovaj zbornik objedinjuje niz konkretnih tehničkih doprinosa koji realizuju zamisao IPAIS-a: od okvira za prediktivno nadgledanje, preko semantički obogaćenih mehanizama za izvršavanje poslovnih procesa, do sistema za automatsko otkrivanje odstupanja procesa. Glavni doprinos zbornika jeste sistematizacija IPAIS-a kao jedinstvenog istraživačkog programa sa ustaljenom terminologijom i zajedničkim metodološkim osnovama.

Semantičke tehnologije kao druga osa IPAIS razvoja

Paralelno sa razvojem rudarenja procesa i fleksibilnosti, od sredine dvehiljaditih godina, oblikuje se i drugi talas istraživanja koji tretira nedostatak semantičke rigoroznosti kao ključno ograničenje tradicionalnog PAIS-a. Hepp, Leymann, Domingue, Wahler i Fensel [28] u već pomenutom radu „Semantic Business Process Management“ formulišu tezu da se problem mehanizacije BPM-a suštinski svodi na ontološki problem. Ovaj uvid otvara prostor za sistematsko istraživanje ontoloških osnova procesnog modelovanja koje će kasnije doneti radovi poput Singer [36] i Annane, Aussenac-Gilles i Kamel [37]. Ontologije ovde prevazilaze ulogu pomoćnog alata za dokumentovanje procesa i postaju osnovni mehanizam predstavljanja znanja, koji omogućava automatsko zaključivanje, proveru konzistentnosti i semantičku interoperabilnost između heterogenih PAIS sistema.

Ovaj pravac predstavlja neposrednu prethodnicu ontološkog pristupa usvojenog u ovoj disertaciji, kojem se vraćamo u poglavlju 2.4.

Neurosimbolička veštačka inteligencija kao savremena sinteza

Noviji teorijski pomak predstavlja objedinjavanje ontoloških (simboličkih) i mašinskih (subsimboličkih) pristupa veštačkoj inteligenciji u paradigmu neurosimboličke veštačke inteligencije. Sarker, Zhou, Eberhart i Hitzler [38] sistematizuju ovu oblast, ističući njen potencijal za rešavanje problema objašnjivosti u složenim sistemima mašinskog učenja. Breit, Waltersdorfer, Ekaputra, Sabou, Ekelhart, Iana, Paulheim, Portisch, Revenko, Ten Teije i Van Harmelen [39] sistematski prikazuju hibridne sisteme koji kombinuju simboličko znanje i statističke modele. Za oblast IPAIS-a ovaj pravac je dvostruko značajan: s jedne strane, omogućava da se prediktivna analitika zasnovana na mašinskom učenju tumači kroz domensko znanje kodirano u ontologiji; s druge strane, otvara put ka proverljivim i objašnjivim odlukama, koje predstavljaju jedan od osnovnih zahteva savremenih poslovnih okruženja.

Kontinuitet istraživačkog programa i prelaz ka ABPMS-u

Četiri prikazana talasa (rudarenje procesa, fleksibilnost i dinamičko prilagođavanje, usmerenost na čoveka i objedinjavanje u paradigmi IPAIS-a) ne predstavljaju uzastopne zamene, već slojeve istraživačkog programa koji se nadograđuju na izvorni koncept PAIS-a iz 2005. godine. Paradigma IPAIS-a iz 2017. godine obuhvata sve prethodne doprinose kao svoje sastavne elemente: fleksibilnost i adaptacija iz okvira Reicherta i Webera ugrađuju se kao preduslov kontekstualne svesnosti; rudarenje procesa postaje empirijska osnova prediktivne analitike; stanovište usmereno na čoveka ulazi u semantičko modelovanje učesnika; a semantičke tehnologije obezbeđuju jezgro za predstavljanje znanja nad kojim se odvijaju sve ostale vrste inteligencije.

Istovremeno, paradigma IPAIS-a iz 2017. godine otvara prostor za sledeću generaciju sistema koju oblikuje nagli razvoj generativne veštačke inteligencije u periodu 2020-2026. godine. Kada Dumas, Fournier, Limonad, Marrella, Montali, Rehse i saradnici [2] uvode koncept ABPMS-a (engl. *AI-Augmented Business Process Management Systems*) kao novu podklasu PAIS-a u nastajanju, obogaćenu pouzdanom tehnologijom veštačke inteligencije, oni ne zamenjuju IPAIS, već ga proširuju novim sposobnostima: sposobnošću delovanja kroz razgovor (engl. *conversational actionability*) zasnovanom na velikim jezičkim modelima, samousavršavanjem (engl. *self-improvement*) kroz povratne sprege učenja i autonomijom u zadatim okvirima koja obuhvata čitave procesne tokove. Detaljno razmatranje koncepta ABPMS-a, njegovih pet osobina i odnosa prema prethodnom konceptu IPAIS-a predmet je sledećeg potpoglavlja.

Metodologija BAB zauzima mesto između IPAIS-a i ABPMS-a: ona usvaja arhitektonski okvir IPAIS-a kao konceptualnu osnovu, a kroz sloj agenata iznad generisanog koda delimično realizuje elemente četiri osobine ABPMS-a na nivou konkretnih komponenti. Za petu osobinu, samousavršavanje, obezbeđuje infrastrukturnu podlogu kroz trajnu

memoriju agenata i operativne pokazatelje. Ovo mesto u odnosu na IPAIS i ABPMS teorijski je obrazloženo u Poglavlju 3, dok Poglavlje 6 proverava raspoložive komponente i izričito navodi granice njihove realizacije.

2.2 Unapređenje poslovnih procesa pomoću veštačke inteligencije i manifest ABPMS-a

2.2.1 Manifest ABPMS-a: vizija i pet ključnih osobina

Paralelno sa sazrevanjem paradigme inteligentnih procesno svesnih informacionih sistema (IPAIS), u periodu 2020-2022. godine ostvaren je niz tehnoloških napredaka u oblasti velikih jezičkih modela i generativne veštačke inteligencije, koji su suštinski promenili pretpostavke o sposobnostima koje savremeni informacioni sistemi zaista mogu da imaju. Kao odgovor na ove promene, grupa od četrnaest istraživača iz vodećih svetskih institucija (Univerziteta u Tartuu, IBM Research, Univerziteta Sapijenca u Rimu, Slobodnog univerziteta u Bocen-Bolzanu, Univerziteta u Manhajmu, Eindhovenskog tehnološkog univerziteta, Technion-a, Univerziteta u Melburnu, Apromore-a i TU Berlin-a) objavila je 2023. godine manifest koji formalizuje sledeću generaciju procesno svesnih sistema. Radi se o radu Dumas, Fournier, Limonad, Marrella, Montali, Rehse, Accorsi, Calvanese, De Giacomo, Fahland, Gal, La Rosa, Völzer i Weber [2]. Manifest je proizašao iz dva istraživačka projekta koja je finansirao Evropski istraživački savet (projekti PIX 834141 i WhiteMech 834228) i predstavlja saglasnost ključnih istraživača u oblasti BPM-a o tome kako bi procesno svesni informacioni sistemi trebalo da se razvijaju u vreme pouzdane veštačke inteligencije.

Vizija i definicija ABPMS-a

Vizija manifesta polazi od zapažanja da su istovremeno sazrela dva činioca: veća dostupnost podataka o izvršavanju poslovnih procesa i napredak tehnika veštačke inteligencije. To je omogućilo pojavu informacionih sistema u kojima tokovi izvršavanja nisu unapred određeni, prilagođavanja ne zahtevaju neposredne izmene softverskih aplikacija, a mogućnosti za unapređenje autonomno se otkrivaju, proveravaju i primenjuju tokom rada sistema. Takve sisteme autori nazivaju sistemima za upravljanje poslovnim procesima unapređenim veštačkom inteligencijom (engl. *AI-Augmented Business Process Management Systems*, ABPMS).

Manifest uvodi definiciju ABPMS-a koja je u kasnijim radovima često služila kao polazište: „*ABPMS je procesno svesni informacioni sistem koji se oslanja na pouzdanu tehnologiju veštačke inteligencije radi zaključivanja i delovanja na osnovu podataka, u okviru skupa ograničenja, sa ciljem stalnog prilagođavanja i unapređenja skupa poslovnih procesa prema jednom ili više pokazatelja uspešnosti*“ [2, str. 2].

Manifest jasno određuje ABPMS kao novu klasu procesno svesnih informacionih sistema, a ne kao potpuno novu paradigmu: „*Sistemi za upravljanje poslovnim procesima prošireni veštačkom inteligencijom (ABPMS) predstavljaju novu klasu procesno svesnih*

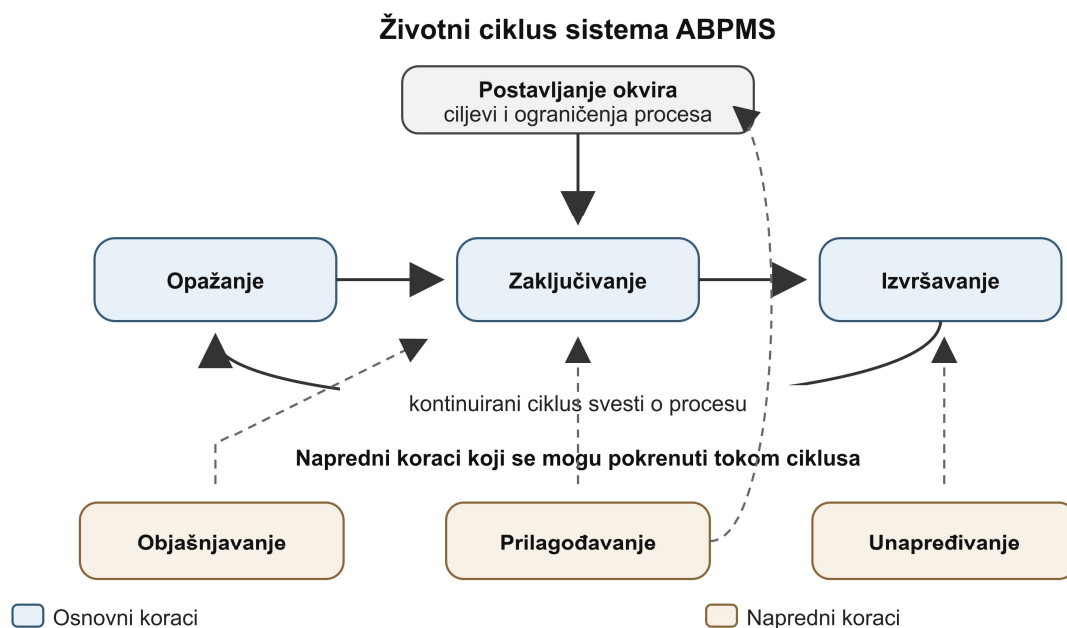
informacionih sistema, osnaženu pouzdanom tehnologijom veštačke inteligencije“ [2, str. 1]. Ovim se uspostavlja jasan konceptualni kontinuitet sa prvobitnom definicijom PAIS-a iz 2005. godine i sa paradigmom IPAIS-a iz 2017. godine, pri čemu ABPMS predstavlja najnoviju razvojnu fazu procesno svesnih sistema, a ne zamenu za prethodne faze. Ovaj kontinuitet određuje teorijsko mesto disertacije: PAIS se usvaja kao konceptualna polazna tačka, a ABPMS kao dugoročni cilj.

Autori preciziraju da se ABPMS razlikuje od klasičnih sistema za upravljanje poslovnim procesima (BPMS) prvenstveno po prirodi donošenja odluka: *„U tradicionalnom BPMS-u svaki zadatak ili odluku u procesu usmerava ljudski akter ili softverska aplikacija koja izvršava zadatak prema unapred određenoj poslovnoj logici. Nasuprot tome, u ABPMS-u sistem automatski odlučuje da li će zadatak uopšte biti izvršen i, ako hoće, kako će biti izvršen, na primer posredstvom ljudskog agenta ili samog sistema, bez unapred definisanih skripti izvršavanja“* [2, str. 3].

Životni ciklus ABPMS-a

Manifest uvodi prošireni životni ciklus ABPMS-a, prikazan u prilagođenom obliku na Slici 2.1. Ciklus razlikuje dve kategorije koraka: osnovne korake (*basic steps*), koji se mogu naći i u životnom ciklusu tradicionalnog BPMS-a, ali su ovde obogaćeni tehnikama veštačke inteligencije, i napredne korake (*advanced steps*), koji su specifični za ABPMS i postoje samo kada je veštačka inteligencija sastavni deo sistema.

Životni ciklus obuhvata četiri osnovna koraka: *uokvirivanje* (engl. *frame*), *izvršavanje* (engl. *enact*), *opažanje* (engl. *perceive*) i *rezonovanje* (engl. *reason*). Njegov napredni deo čine tri dodatna koraka: *objašnjavanje* (engl. *explain*), *prilagođavanje* (engl. *adapt*) i *unapređivanje* (engl. *improve*). Centralni mehanizam izvršavanja u ABPMS-u organizovan je prema ciklusu opažanja, razmišljanja i delovanja (engl. *sense-think-act*), paradigmi široko prihvaćenoj za definisanje arhitektura dinamičkih sistema veštačke inteligencije [40].



Slika 2.1. Pojednostavljen životni ciklus sistema ABPMS, prilagođeno prema Dumas i saradnici [2].

Pojam okvira (engl. *frame*) zauzima centralno mesto u manifestu i zahteva posebno objašnjenje. Oslanjajući se na Minsky [41], autori usvajaju gledište da je okvir „*struktura podataka za predstavljanje stereotipne situacije [...] koja obuhvata više vrsta informacija [...] o načinu upotrebe okvira [...] o onome što se može očekivati kao sledeći događaj [i] o tome šta treba učiniti ako se ta očekivanja ne potvrde*“ [2, str. 2]. U kontekstu ABPMS-a, okvir definiše granice unutar kojih sistem treba da deluje, a te granice mogu biti strogo ili labavo određene. Petrijeva mreža koja propisuje ponašanje procesa primer je izvršavanja u strogo zadatim okvirima, dok skup vremensko-logičkih ograničenja koja moraju biti zadovoljena tokom izvršavanja procesa predstavlja izvršavanje u labavije zadatim okvirima. Ovakvo tumačenje okvira neposredno se nastavlja na klasifikaciju iz 2005. godine (Dumas, Van der Aalst i Ter Hofstede), čime se dodatno učvršćuje konceptualna veza između PAIS-a i ABPMS-a.

Pet ključnih osobina ABPMS-a

U odeljku 3 manifesta autori navode da sistem, kako bi ostvario prethodno opisani životni ciklus, mora posedovati pet ključnih osobina: autonomiju u zadatim okvirima (engl. *(framed) autonomy*), sposobnost delovanja kroz razgovor (engl. *conversational actionability*), prilagodljivost (engl. *adaptive*), samousavršavanje (engl. *self-improving*) i objašnjivost (engl. *explainable*). Ove osobine nisu nezavisne, jer između njih postoje međuzavisnosti, već su kumulativne: svaka naredna osobina počiva na mehanizmima koje obezbeđuju prethodne.

Autonomija u zadatim okvirima. Sistem izvršava korake životnog ciklusa nezavisno i proaktivno unutar datog okvira. ABPMS koristi sposobnosti veštačke inteligencije da proceni okolnosti konkretne situacije i zatim nezavisno izabere i sprovede najprimereniji postupak, što je posebno izraženo u koraku rezonovanja, gde sistem može da se osloni na

simboličke ili subsimboličke tehnike da pronađe sopstvene obrasce i izvuče zaključke. Autori uvode i pojam ponovnog određivanja okvira (engl. *meta-framing*): projektant može ograničiti sposobnost ABPMS-a da samostalno menja okvir, odnosno odrediti koji su delovi okvira izmenjivi, a koji se mogu menjati samo ljudskom intervencijom [2, str. 7]. Ponovno određivanje okvira omogućava postepenu izgradnju poverenja između ljudskih agenata i sistema, od potpuno autonomnog režima rada do režima u kojem sistemu nije dozvoljeno da samostalno menja okvir.

Sposobnost delovanja kroz razgovor. Očekuje se da ABPMS deluje u velikoj meri autonomno, ali ljudski akteri ostaju suštinski deo životnog ciklusa. Sposobnost delotvorne interakcije sa ljudima zato predstavlja jednu od centralnih osobina. Autori navode: „*ABPMS treba da bude sposoban za delovanje kroz razgovor, odnosno za proaktivnu komunikaciju sa ljudskim akterima o radnjama, ciljevima i namerama povezanim sa procesom, korišćenjem prirodnog jezika koji se po potrebi može dopuniti bogatijim interfejsima*“ [2, str. 7]. Izraz „kroz razgovor“ (engl. *conversational*) označava da ABPMS pruža interfejs za pisani i govorni prirodni jezik, prilagođene potrebama korisnika i situaciji u kojoj se komunikacija odvija. Izraz „sposobnost delovanja“ (engl. *actionable*) znači da sistem daje konkretne preporuke i razmatra njihove prednosti i nedostatke. Dakle, sistem ima dve uloge: odgovara na upite i, kao sagovornik, sam pokreće interakciju.

Prilagodljivost. Poslovni procesi često se odvijaju u promenljivim okruženjima koja karakterišu neizvesnost i promene. ABPMS mora da bude sposoban da funkcioniše u takvim okolnostima, što znači da mora da uočava promene u svom okruženju i da reaguje na njih tokom izvršavanja. Autori jasno suprotstavljaju ABPMS tradicionalnom BPMS-u u ovom pogledu: „*Za razliku od konvencionalnih BPMS sistema, u kojima se promene pojedinačnih instanci procesa po potrebi mogu izvršiti samo ljudskom intervencijom, ABPMS može samostalno pokrenuti prilagođavanje*“ [2, str. 8]. Ova osobina neposredno se nadovezuje na mehanizme fleksibilnosti i dinamičkog prilagođavanja koje su razvili Reichert i Weber [26] i Adams i drugi [27], ali ih proširuje dimenzijom autonomije. Odluka o prilagođavanju više nije rezervisana za ljude.

Samousavršavanje. Dok se prilagodljivost odnosi na sposobnost sistema da reaguje na spoljašnje promene, samousavršavanje označava sposobnost blagovremenog predviđanja promena i stalnog unapređivanja. ABPMS treba da predviđa promene i da bude optimalno pripremljen za njih: na primer, sistem može uočiti postepeno povećanje temperature u rashladnom kamionu i predvideti da će biti potrebno održavanje u narednim nedeljama, a zatim ga planirati za dan kada je predviđeno transportno opterećenje nisko, uz rezervisanje kapaciteta u drugim kamionima za preuzimanje porudžbina [2, str. 8]. Cilj ABPMS-a je postizanje optimalnih kompromisa između suprotstavljenih pokazatelja uspešnosti procesa, kao što su troškovi rada i vreme čekanja. Samim tim, sistem samostalno odlučuje šta treba učiniti da bi se postigao takav kompromis. Takve odluke mogu uticati na ceo proces ili na pojedinačne slučajeve i mogu biti privremene ili trajne.

Objašnjivost. Prihvatanje ABPMS-a među korisnicima zahteva poverenje. Korisnik mora verovati da će sistem izvršiti proces na način koji smatra razumnim i opravdanim. Autori se oslanjaju na definiciju poverenja koju su dali Mayer, Davis i Schoorman [42]: spremnost jedne strane da bude ranjiva na postupke druge strane na osnovu očekivanja da će druga strana obaviti određenu važnu radnju. Da bi bio dostojan poverenja, ABPMS mora da omogući korisnicima da razumeju njegove odluke, što zahteva verodostojna objašnjenja prilagođena situaciji, kao i mogućnost da operatori obezbede da sistem izvršava poslovne procese kako je predviđeno i da zadrži sposobnost oporavka iz nepoželjnih ili nedoslednih stanja. Autori usvajaju okvir Toreini i saradnika [43], koji utvrđuje četiri aspekta pouzdanosti: pravednost, objašnjivost, mogućnost provere i bezbednost (engl. *Fairness, Explainability, Auditability, Safety*, FEAS). U kontekstu ABPMS-a, objašnjivost se posmatra u širem poslovnom smislu koji obuhvata i pravednost i zahteve provere, dok se bezbednost rešava u koraku uokvirivanja. Autori uvode pojam procesno svesne objašnjivosti (engl. *process-aware explainability*), koji podrazumeva sposobnost sistema da nezavisno i stalno zaključuje o ishodima izvršavanja procesa, što uključuje neprekidno beleženje ključnih uslova i sposobnost izvođenja zaključaka o zavisnostima između tih uslova i rezultata izvršavanja procesa.

Istraživački izazovi u manifestu

Manifest postavlja pet osobina, a u odeljku 4 utvrđuje i pet istraživačkih izazova čije prevazilaženje predstavlja preduslov za realizaciju vizije ABPMS-a. Ovi izazovi neposredno utiču na metodološki pristup u ovoj disertaciji i zato ih treba sažeto prikazati.

Prvi izazov je situaciono svesna objašnjivost (engl. *situation-aware explainability*), koja zahteva da objašnjenja ne budu zasnovana samo na lokalnim zaključcima, kao što su tekući zadatak ili odluka, već da uključe izričito vođenje zapisa koji povezuju trenutni izvršni okvir sa odgovarajućim lancem uzročnih zaključaka, omogućavajući praćenje do osnovnih uzroka. Manifest posebno ističe da grafovi znanja zasnovani na događajima [44] omogućavaju simboličko predstavljanje situacija za situaciono svesno rezonovanje, što je značajno za ontološki pristup usvojen u ovoj disertaciji.

Drugi izazov je unapređena automatizacija procesa (engl. *augmented process automation*), koju autori suprotstavljaju robotskoj automatizaciji procesa (RPA). Dok RPA automatizuje ponovljive zadatke pomoću softverskih robota koji oponašaju interakcije čoveka sa korisničkim interfejsom, ABPMS ne teži potpunoj zameni čoveka, već dopunjavanju njegovih sposobnosti pomoću takozvane *hibridne procesne inteligencije* [45], paradigme u kojoj veštačka inteligencija deluje kao „učenik u praksi“ koji se prilagođava radnim praksama korisnika, a ne obrnuto [2].

Treći izazov je automatizovano prilagođavanje procesa. Autori pregledaju istorijski razvoj ove oblasti, od ranih radova Casati, Ceri, Paraboschi i Pozzi [46] o rukovanju izuzecima, preko sistema ADEPT [47, 48], zasnovanog na strukturnim promenama po potrebi, do pristupa SmartPM [49, 50], koji primenjuje planere zasnovane na VI za izgradnju postupaka oporavka tokom izvršavanja. Manifest jasno navodi da ABPMS

zahteva tehnike koje prevazilaze unapred kodirana rešenja i omogućavaju uvođenje novih tipova podataka i zadataka tokom izvršavanja.

Četvrti izazov je prilagodljivost različitim perspektivama (engl. *perspective agility*), koja zahteva da ABPMS podržava procese čija je struktura u vreme projektovanja nepoznata ili nepotpuna i koji mogu da se pojave tokom izvršavanja. Ovaj zahtev se rešava kombinacijom formalizama, imperativnih, deklarativnih i ciljno orijentisanih, kao i međusobno povezanih tipova artefakata. Autori pominju notaciju za upravljanje slučajevima i modelovanje (CMMN) i paradigmu agenata sa uverenjima, željama i namerama (BDI) kao dva primera realizacije ovog zahteva [2].

Peti izazov su razgovori usmereni na delovanje (engl. *actionable conversations*), koji zahtevaju razvoj sistema za interakciju korisnika i ABPMS-a koji prevazilaze reaktivne razgovorne agente, kao što su tipični četbotovi. Autori predlažu objedinjeno korišćenje obrade prirodnog jezika i mašinskog učenja za oblikovanje dinamičkih razgovora koji odgovaraju na upite korisnika i samostalno pokreću komunikaciju radi obaveštavanja o napretku procesa, upozoravanja na značajne promene i davanja preporuka za intervencije.

Značaj manifesta za ovu disertaciju

Manifest koji su objavili Dumas i saradnici [2] predstavlja centralnu savremenu teorijsku osnovu na koju se ova disertacija oslanja iz tri razloga. Prvo, on jasno određuje ABPMS kao podklasu PAIS-a, čime se uspostavlja teorijski kontinuitet između dve paradigme i opravdava istraživački pristup koji polazi od PAIS-a kao osnove i teži ABPMS-u kao cilju. Drugo, pet utvrđenih osobina, autonomija u zadatim okvirima, sposobnost delovanja kroz razgovor, prilagodljivost, samousavršavanje i objašnjivost, obrazuju okvir za vrednovanje doprinosa metodologija i softverskih okvira koji teže statusu rešenja ABPMS-a. Treće, pet istraživačkih izazova, posebno situaciono svesna objašnjivost i prilagodljivost različitim perspektivama, u okviru ove disertacije motivišu razmatranje semantičkih tehnologija i ontoloških osnova kao dela infrastrukture za buduću realizaciju vizije ABPMS-a.

Manifest predstavlja istraživački program, a ne gotovo rešenje: autori ga jasno formulišu kao „*podsticaj za buduća istraživanja*“ (prevod autora) i navode da predložene osobine i izazovi nisu iscrpni, već ih treba posmatrati kao prvi korak ka oblikovanju pojma ABPMS-a. Ova disertacija ne tvrdi da pruža potpunu implementaciju vizije ABPMS-a. Umesto toga, ona pokazuje delimičnu realizaciju elemenata četiri osobine tokom rada sistema pomoću konkretnih komponenti agentskog sloja: objašnjivosti pomoću transparentnosti formalnog ontološkog modela i nadzora aktivnosti agenata, prilagodljivosti pomoću reifikovanih preslikavanja i nadzora agenata tokom izvršavanja, sposobnosti delovanja kroz razgovor pomoću interfejsa SPARQL i agenta UserHelpDeskAgent, kao i autonomije u zadatim okvirima pomoću prototipa za predlaganje ispravki i ograničenih kriterijuma prihvatanja. Za petu osobinu, samousavršavanje, obezbeđeni su infrastrukturni preduslovi, ali ona nije potvrđena tokom rada sistema.

Pre nego što se prikaže kako predložena metodologija obezbeđuje ove preduslove, neophodno je sažeto prikazati postojeće pristupe unapređenju poslovnih sistema pomoću veštačke inteligencije, što je predmet sledećeg potpoglavlja, i utvrditi prazninu između vizije ABPMS-a i trenutnog stanja implementacija.

2.2.2 Postojeći pristupi unapređenju poslovnih sistema

Istraživanje uključivanja elemenata veštačke inteligencije (VI) u poslovne informacione sisteme nije novo i prati istoriju discipline upravljanja poslovnim procesima duže od dve decenije. Ono što je novo u periodu nakon 2020. godine jeste kombinacija tri faktora: raspoloživost velikih skupova podataka o izvršavanju procesa, zrelost tehnika mašinskog učenja i pojava velikih jezičkih modela kao generičkog reprezentacionog mehanizma. Ovo potpoglavlje sistematizuje ključne talase integracije veštačke inteligencije u poslovne sisteme, grupisane u četiri kategorije: pristupe zasnovane na znanju, ontološke pristupe, pristupe zasnovane na mašinskom učenju, i pristupe zasnovane na velikim jezičkim modelima.

Rani pristupi zasnovani na znanju (2000-2010)

Prvi talas integracije veštačke inteligencije u poslovne sisteme odvijao se pod okriljem paradigme sistema zasnovanih na znanju i bio je usmeren na rukovanje izuzecima u sistemima za upravljanje radnim tokovima. Klein i Dellarocas [51] uvode pristup zasnovan na skladištu opštih obrazaca izuzetaka i pridruženih strategija za njihovo rešavanje, čime se znanje o rukovanju izuzecima izdvaja iz sistema i čini ponovo upotrebljivim. Kearney [52] integriše tehnike VI planiranja u mehanizme za izvršavanje radnih tokova, omogućavajući dinamičko generisanje planova izvršavanja na osnovu ciljeva i ograničenja. Cao, Yang, Chan i Xu [53] koriste mobilne agente kao mehanizam za distribuirano rukovanje izuzecima.

Ograničenje ove rane generacije pristupa jeste to što su oni u osnovi reaktivni: rukovanje izuzecima zahteva unapred definisane obrasce izuzetaka i pridružene strategije, što znači da projektant sistema mora da predvidi sve moguće izuzetne situacije. Ovakav pristup suštinski je nespojiv sa vizijom ABPMS-a, u kojoj sistem samostalno otkriva nepredviđene situacije i oblikuje strategije odgovora.

Ontološki pristupi (2005-2020)

Paralelno sa razvojem pristupa zasnovanih na znanju, od sredine dvehiljaditih godina oblikuje se istraživački pravac koji ontologije tretira kao osnovni mehanizam za predstavljanje pri procesnom modelovanju. U ključnom programskom tekstu ovog pravca Hepp, Leymann, Domingue, Wahler i Fensel [28] ukazuju da se problem automatizacije BPM-a suštinski svodi na ontološki problem, odnosno nedostatak mašinski dostupne semantike u modelima procesa.

Konkretni ontološki pristupi razvijali su se u dve glavne smernice. Prva smernica obuhvata ontološku formalizaciju postojećih notacija za modelovanje procesa. De Nicola,

Lezoche i Missikoff [54] razvijaju ontološki okvir koji preslikava konstrukte poslovnog procesnog modelovanja na formalne ontološke aksiome. Singer [36] razvija OWL ontologiju standarda BPMN, prikazuje serijalizaciju BPMN modela u OWL-u i razmatra njihovo preslikavanje u akterski prikaz procesa. Annane, Aussenac-Gilles i Kamel [37] razvijaju OWL ontologiju čije jezgro predstavlja fragment metamodela BPMN 2.0, a zatim ga proširuju pojmovima potrebnim za detaljno predstavljanje i praćenje poslovnih procesa.

Druga smernica je semantička interoperabilnost poslovnih informacionih sistema u kontekstu pojma informacionih sistema vođenih ontologijama (engl. *Ontology-Driven Information Systems*, ODIS). Sharman, Kishore i Ramesh (ur., 2007) definišu ODIS kao sisteme u kojima ontologija pokriva kako strukturnu dimenziju (mehanizmi za generisanje i pristup komponentama informacionog sistema) tako i vremensku dimenziju (smernice za razvoj novog informacionog sistema, pomoć u izboru adekvatnih procesa, algoritama, softverskih komponenti). Husáková i Bureš [55] sistematski pregledaju 27 studija i identifikuju pet glavnih uloga formalnih ontologija u razvoju informacionih sistema: interoperabilnost, upravljanje ljudskim resursima, predstavljanje domenskog znanja, semantika u modelovanju zasnovanom na UML-u, i upravljanje programskim kodom i dokumentacijom. Abonyi, Nagy i Ruppert [56] primenjuju ontološke osnove na domen pametne proizvodnje, dok Yang, Zheng i saradnici [57] razvijaju ontološku reprezentaciju industrijskih proizvodnih radnih tokova.

Ograničenje ontoloških pristupa, uočeno u nizu radova, jeste takozvani ontološki jaz (engl. *ontology gap*): formalne ontologije obezbeđuju strogo određeno jezgro za predstavljanje znanja, ali samo predstavljanje ne generiše automatski ponašanje informacionog sistema. Potreban je mehanizam koji povezuje ontološki model i izvršnu infrastrukturu, što je upravo problem koji rešava metodologija koja je predmet ove disertacije.

Pristupi zasnovani na mašinskom učenju (2015-2023)

Treći talas integracije veštačke inteligencije u poslovne sisteme, oblikovan razvojem tehnika mašinskog učenja i dubokog učenja, u velikoj meri je odvojen od ontoloških pristupa i usmeren na statističko izdvajanje obrazaca iz podataka o izvršavanju procesa. Kratsch, Manderscheid, Röglinger i Seyfried [58] upoređuju klasične pristupe, slučajne šume i mašine sa potpornim vektorima, sa dubokim neuronskim mrežama sa unaprednim prostiranjem i mrežama sa dugom kratkoročnom memorijom za predviđanje ishoda procesnih instanci. Na pet javno dostupnih dnevnika događaja duboko učenje je u celini dalo bolje rezultate, pri čemu prednost zavisi od svojstava dnevnika, kao što su odnos broja varijanti i instanci, neuravnoteženost ciljnih klasa i raspodela podataka između nivoa aktivnosti i instance. Bharadiya [59] daje pregled primena mašinskog učenja i veštačke inteligencije u poslovnoj inteligenciji, uključujući prediktivnu analitiku, virtuelne pomoćnike, otkrivanje odstupanja, prognoziranje i objašnjivost modela.

Ključno ograničenje pristupa zasnovanih isključivo na mašinskom učenju, u kontekstu vizije ABPMS-a, jeste odsustvo eksplicitne semantičke podloge. Statistički modeli uspešno prepoznaju obrasce u podacima, ali ne raspolažu domenskim znanjem koje bi im omogućilo da protumače značenje tih obrazaca. Ovo ograničenje je suština problema objašnjivosti: model može tačno da predvidi ishod, ali ne može da objasni zašto je taj ishod verovatan pojmovima razumljivim ljudskim korisnicima. Upravo ova praznina podstiče objedinjavanje ontoloških pristupa i mašinskog učenja u okviru paradigme neurosimboličke veštačke inteligencije, koja je razmatrana u potpoglavlju 2.1.2.

Pristupi zasnovani na velikim jezičkim modelima (2022-2026)

Četvrti i najnoviji talas integracije veštačke inteligencije u poslovne sisteme vezan je za pojavu velikih jezičkih modela (engl. *Large Language Models*, LLM), generativnih modela obučanih na obimnim korpusima tekstualnih podataka koji ispoljavaju nove sposobnosti u različitim zadacima obrade prirodnog jezika. Teubner, Flath, Weinhardt, Van der Aalst i Hinz [60] navode da nastupa novo razdoblje u upravljanju poslovnim procesima, u kojem se obrada prirodnog jezika pomera sa sporednog na centralno mesto u skupu tehnologija. Busch, Rochlitz, Sola i Leopold [61] uvode pojam inženjerstva upita (engl. *prompt engineering*) kao metodologije interakcije sa LLM-ovima u kontekstu zadataka BPM-a i pokazuju kako pažljivo formulisani upiti mogu zameniti tradicionalne, specijalizovane modele mašinskog učenja u zadacima poput izdvajanja procesnih entiteta iz tekstualnih opisa.

U kontekstu interakcije LLM-ova i ontologija, Neuhaus [62] analizira odnos između ove dve tehnologije i obrazlaže da je reč o komplementarnim, a ne konkurentskim pristupima: LLM-ovi obezbeđuju prilagodljivo razumevanje prirodnog jezika, ali imaju problem sa halucinacijama i neobjašnjivošću, dok ontologije obezbeđuju stroge simboličke predstave, ali zahtevaju ručnu izgradnju i održavanje. Babaei Giglou, D'Souza i Auer [63] procenjuju devet porodica LLM-ova na zadacima tipizacije termina, otkrivanja taksonomije i izdvajanja netaksonomskih relacija. U suprotnom smeru, Palagin, Kaverinskiy, Litvin i Malakhov [64] razvijaju ontologijom vođene strukturisane upite i prikazuju njihovu primenu na izdvajanju i klasifikaciji entiteta i generisanju odgovora na ukrajinskom jeziku u domenu rehabilitacije.

U oblasti poslovnih informacionih sistema, Nasiri, Adadi i Lahmer [65] predlažu pristup u kojem se iz korisničkih priča automatski generišu procesni modeli pomoću tehnika obrade prirodnog jezika. Schüller i Alpers [66] sistematski pregledaju stanje oblasti automatskog generisanja procesnih modela i izdvajaju LLM-ove kao tehnologiju koja obećava značajan napredak.

Sintetički pristupi: neurosimboličko upravljanje poslovnim procesima

Jedan od novijih pravaca jeste povezivanje simboličkih prikaza znanja i statističkog učenja. Sarker, Zhou, Eberhart i Hitzler [38] sistematizuju neurosimboličke pristupe, a Breit, Waltersdorfer, Ekaputra i saradnici [39] analiziraju sisteme koji kombinuju

mašinsko učenje i resurse semantičkog veba. Posebna mogućnost, prikazana u radu Palagina i saradnika [64], jeste korišćenje ontologije za izgradnju strukturisanih upita za veliki jezički model. Za metodologiju koja je predmet ove disertacije značajna je mogućnost budućeg povezivanja razgovornog sloja sa formalnim ontološkim jezgrom i SPARQL interfejsom; takvo povezivanje ovde se razmatra kao pravac daljeg razvoja, a ne kao već realizovana sposobnost BAB generatora.

2.2.3 Jaz između ABPMS vizije i postojećih implementacija

Četiri talasa pristupa integracije veštačke inteligencije u poslovne sisteme, koji su prikazani u prethodnom potpoglavlju, doneli su značajne doprinose: od pristupa zasnovanih na znanju za rukovanje izuzecima, preko ontoloških reprezentacija procesa, do sofisticiranih modela mašinskog učenja i velikih jezičkih modela. Kada se rezultati pregledanih radova sistematski uporede sa pet osobina ABPMS-a utvrđenih u manifestu koji su objavili Dumas i saradnici [2], u analiziranom korpusu ne nalazi se integrisani okvir koji istovremeno obezbeđuje preduslove za svih pet osobina. Ovo potpoglavlje taj nalaz strukturira prema pet osobina.

Praznina u objašnjivosti. Pristupi zasnovani isključivo na mašinskom učenju i velikim jezičkim modelima otežavaju davanje objašnjenja izraženih pojmovima razumljivim čoveku, čak i kada uspešno predviđaju ishode. Ontološki pristupi nude strožu semantičku osnovu za objašnjenja, ali u pregledanim radovima najčešće ne pokrivaju celokupan put od modela do izvršive poslovne aplikacije. Praznina se zato odnosi na povezivanje objašnjivosti na nivou domenskih pojmova sa izvršivom realizacijom celovitog sistema.

Praznina u prilagodljivosti. Okvir koji su razvili Reichert i Weber [26] i pristup SmartPM [49] obezbeđuju mehanizme za prilagođavanje modela i instanci procesa tokom izvršavanja. Promenu implementacione platforme, programskog jezika ili tehnologije za čuvanje podataka, međutim, ne razmatraju kao predmet prilagođavanja. Takve promene zahtevaju transformaciju aplikativne arhitekture i programskog koda, pa izlaze iz okvira prilagođavanja procesa tokom izvršavanja. Zbog toga se arhitektonska prilagodljivost, odnosno sposobnost preslikavanja istog platformski nezavisnog modela na različite skupove tehnologija, u ovoj disertaciji posmatra kao poseban istraživački problem.

Praznina u sposobnosti delovanja kroz razgovor. Razgovorni asistenti mogu poslužiti kao interfejs iznad postojećih sistema, ali pregledani pristupi ne uspostavljaju dosledno vezu sa formalnim modelom celokupne aplikacije. Zato upiti o stanju pojedinačnog poslovnog objekta ne podrazumevaju i mogućnost postavljanja pitanja o strukturi procesa, preduslovima izvršavanja zadatka ili uticaju promene jednog entiteta na druge komponente sistema. Praznina se odnosi na razgovorni pristup semantičkom modelu sistema, a ne samo podacima ili tekstualnoj dokumentaciji.

Praznina u autonomiji. Preliminarno istraživanje BAB okvira [12] polazi od zapažanja da se mogućnosti veštačke inteligencije u tada razmatranim platformama BPM-a uvode kao izdvojene funkcionalnosti, bez dubljeg povezivanja unutar izvršavanja procesa. Za pouzdano delovanje autonomnih agenata unutar poslovnih procesa potrebne su jasne

granice delovanja, determinističke transformacije i eksplicitno definisani, proverljivi uslovi usaglašenosti apstraktnih modela i konkretnih realizacija. Ova disertacija zato razmatra formalnu i generatorsku infrastrukturu kao osnovu za ograničeno delovanje agenata, a ne kao dokaz potpune autonomije.

Praznina u samousavršavanju. U analiziranom korpusu ova osobina je najslabije pokrivena. Samousavršavanje zahteva neprekidno praćenje učinka procesa, predviđanje promena i proaktivno poboljšavanje. Dok rudarenje procesa [29] obezbeđuje naknadnu analizu podataka, a mašinsko učenje prediktivno nadgledanje [58], njihovo povezivanje u neprekidan ciklus samousavršavanja tokom rada sistema ostaje otvoren istraživački problem, izvan dometa ove disertacije i eksplicitno priznat kao pravac za dalji rad.

Specifična praznina: nedostatak formalnog mosta između modela i izvršavanja

Pored pet praznina povezanih sa osobinama ABPMS-a, za ovu disertaciju značajno je povezivanje formalnog modela aplikacije sa proverljivim postupkom generisanja. Postojeći pristupi nude različite formalne osnove: fUML precizira izvršnu semantiku podskupa UML-a, QVT uključuje deklarativne i imperativne jezike, a EMF pruža infrastrukturu za modelovanje i generisanje. Njihovo postojanje ne obezbeđuje samo po sebi dokaz očuvanja izabranih svojstava konkretne transformacije. Istraživački problem zato je definisanje ontološkog PIM-a, realizacionog ugovora i eksplicitnih uslova očuvanja za posmatranu klasu procesno svesnih informacionih sistema.

Za realizaciju vizije ABPMS-a, u kojoj objašnjivost počiva na transparentnosti formalnog modela, prilagodljivost na reifikovanim preslikavanjima, sposobnost delovanja kroz razgovor na modelu nad kojim se mogu izvršavati upiti, a autonomija u zadatim okvirima na pouzdanim formalnim osnovama, neophodan je suštinski drugačiji pristup. Umesto imperativnih, poluformalnih modela i transformacija definisanih prema trenutnoj potrebi, potrebno je objediniti ontološko modelovanje i kategorijalno definisane transformacije. Upravo je ovakav pristup predmet ove disertacije: on ne popunjava sve praznine ABPMS-a, ali pomoću konkretnih komponenata zasnovanih na agentima delimično realizuje elemente četiri osobine tokom rada sistema i postavlja infrastrukturnu osnovu za buduću realizaciju samousavršavanja.

U narednim potpoglavljima 2.3 do 2.6 razmatraju se konkretne oblasti koje zajedno čine metodološku osnovu ovog pristupa: modelom vođeni razvoj softvera, ontološko modelovanje, teorija kategorija i automatizacija generisanja koda. Zatim se u potpoglavlju 2.7 određuje mesto sopstvenog istraživanja u odnosu na prikazano stanje oblasti.

2.3 Modelom vođeni razvoj softvera

Modelom vođeni razvoj softvera (engl. *Model-Driven Engineering*, MDE) predstavlja metodološku paradigmu u kojoj modeli nisu pomoćni artefakti namenjeni dokumentaciji, već primarni, izvršivi i merodavni izvori iz kojih se automatizovano generišu izvorni kod, konfiguracije i prateća dokumentacija softverskog sistema. Selic [4] jasno formuliše osnovnu tezu ovog pristupa: potencijalne prednosti korišćenja modela u softverskom

inženjerstvu značajno su veće nego u drugim inženjerskim disciplinama zahvaljujući mogućnosti uspostavljanja neprekidne veze između modela i sistema koje oni predstavljaju. Međutim, isti autor naglašava i da ta obećanja često nisu ostvarena u praksi zbog praktičnih problema povezanih sa artefaktima i kulturom prethodne generacije softverskih tehnologija.

U industrijskoj i akademskoj praksi MDE je realizovan kroz dva glavna tehnološka pravca: **Model-Driven Architecture (MDA)** organizacije **Object Management Group (OMG)** i **Eclipse Modeling Framework (EMF)** sa Ecore metamodelom. Ovo potpoglavlje sistematski prikazuje oba pravca uz kritičku analizu njihovih ograničenja, pri čemu su u svakom koraku analize navedene recenzirane publikacije nezavisnih autora koje su dokumentovale konkretne nedostatke. Analiza pokazuje da, uprkos zrelosti i širokoj industrijskoj prihvaćenosti ovih pristupa, postoje tri međusobno povezana problema: semantička neodređenost notacionih jezika, ograničenja tehnologije transformacije modela i zavisnost od konkretnih tehnoloških platformi. Ti problemi podstiču razvoj alternativnog pristupa predloženog u ovoj disertaciji. Srodan industrijski pravac predstavlja pristup *Software Factories* Greenfielda i saradnika [67], koji objedinjuje razvoj zasnovan na komponentama, modelima i softverskim proizvodnim linijama radi ponovljive izrade porodica aplikacija. Za razliku od pristupa ove disertacije, njegova osnova nisu ontološki PIM i kategorijalna formalizacija transformacionog lanca.

2.3.1 OMG MDA: PIM, PSM, transformacije

Model-Driven Architecture je standardizovana metodološka paradigma koju je razvila i održava organizacija OMG. Formalni dokument koji definiše MDA je „MDA Guide“, čija je prva verzija (1.0.1) objavljena u junu 2003. godine [68], dok su članovi OMG-a revidiranu verziju 2.0 usvojili u junu 2014. godine [5]. Ovaj dokument oslanja se na zaseban MDA Foundation Model i predstavlja važeću referentnu specifikaciju arhitekture. Od 2014. godine nije bilo većih revizija dokumenta, pa je njegova normativna osnova ostala stabilna.

Tri nivoa apstrakcije

MDA postulira tri osnovna nivoa apstrakcije sistema, koje OMG dokumentacija naziva arhitektonskim perspektivama (engl. *architectural viewpoints*): model nezavisan od izračunavanja (engl. *Computation-Independent Model*, CIM), **platformski nezavisan model** (engl. *Platform-Independent Model*, PIM) i platformski specifičan model (engl. *Platform-Specific Model*, PSM). CIM je najapstraktniji nivo i opisuje sistem sa stanovišta poslovnog domena, bez ulaženja u tehnološke detalje. PIM opisuje računarske aspekte sistema (strukturu, ponašanje i interakcije), ali bez pozivanja na konkretne tehnologije realizacije. PSM obezbeđuje tehničke detalje neophodne za realizaciju sistema na konkretnoj platformi (na primer Java EE, .NET, CORBA ili veb-servisi).

OMG-ov MDA Guide jasno naglašava da ova tri nivoa nisu iscrpan skup mogućih perspektiva, već korisna trijada. Da Silva [3] koristi CIM, PIM i PSM kao nivoe

32 apstrakcije za klasifikaciju jezika i pristupa zasnovanih na modelima, a MDA transformacije razvrstava na CIM-CIM, CIM-PIM, PIM-PIM, PIM-PSM, PSM-PSM i PSM-tekst. U ovoj disertaciji težište je na prelasku sa PIM-a na PSM i izvršne artefakte, dok CIM nije formalizovan kao zaseban deo transformacionog lanca.

UML kao osnovni notacioni jezik

Jezgro MDA paradigme čini Unified Modeling Language (UML), notacioni standard čija je aktuelna verzija UML 2.5.1 [69] objavljena 2017. godine (OMG, 2017b). U MDA kontekstu, UML ne služi isključivo za vizualizaciju; on je formalizovan preko *Meta-Object Facility* (MOF) metamodela, koji obezbeđuje ujedinjenu osnovu za sve OMG notacione standarde [70]. Prema dokumentu organizacije OMG [71], UML je i dalje ključna tehnologija koja omogućava MDA i, prema proceni te organizacije, predstavlja osnovu za oko 99% razvojnih projekata zasnovanih na MDA pristupu. Iako MDA formalno ne zahteva UML, u praksi je veoma tesno povezan sa njim.

Snažno oslanjanje MDA paradigme na UML istovremeno donosi prednosti i ograničenja. S jedne strane, UML je omogućio veliku industrijsku prihvaćenost i raspoloživost brojnih komercijalnih i otvorenih alata. S druge strane, UML nasleđuje ozbiljne slabosti koje je akademska zajednica sistematski dokumentovala u nizu recenziranih publikacija. Glavna kritika odnosi se na semantičku neodređenost njegovih dijagramskih konstrukata: dijagrami klasa, dijagrami aktivnosti i dijagrami stanja u standardnom UML-u nemaju formalno definisanu operativnu semantiku, već samo neformalne opise na prirodnom jeziku.

Sistematska dokumentacija semantičkih problema UML-a

Selic [4] razmatra pragmatične preduslove i industrijske kriterijume za uspešnu primenu modelom vođenog razvoja, uz naglasak na značaju automatizacije i kvaliteta modela. Detaljna dokumentacija pojedinačnih problema UML semantike razvijena je kroz niz recenziranih publikacija. Broy i Cengarle [8] sistematizuju iskustva iz višegodišnjih pokušaja da se UML-u pridruži formalna semantika i konstatuju da problem nije u nedostatku teorijskih pristupa, već u osnovnim odlukama pri projektovanju jezika koje su dovele do nagomilavanja dvosmislenosti.

Konkretni problemi dokumentovani su u nizu posebno usmerenih istraživanja. Fuentes, Quintana, Llorens, Génova i Prieto-Díaz [72] identifikovali su konkretne greške u UML metamodelu koje se prenose u svaku implementaciju zasnovanu na njemu. Fecher, Schönborn, Kys i De Roeper [73] dokumentovali su dvadeset devet novootkrivenih nejasnih mesta u semantici UML 2.0 dijagrama stanja, uključujući dvosmislenosti, nedoslednosti i nepotrebno stroga ograničenja ilustrovana na šest konkretnih primera. Rączkowska i Walkowiak-Gall [74], uprkos revizijama standarda tokom decenije između ove dve studije, dokumentuju petnaest novih problema u tumačenju UML 2.5.1 [69] dijagrama aktivnosti. Autori jasno zaključuju da nedostatak formalne semantike dovodi

do dvosmislenosti koje su posebno kritične pri automatizaciji razvoja, potvrđivanju specifikacija sistema, proveru modela, transformacijama i generisanju koda.

Nezavisno istraživanje sa praktične strane obavio je Nacionalni institut za standarde i tehnologiju (NIST) u publikaciji NISTIR 8358, Padi i saradnici [75]. Autori eksplicitno konstatuju da orijentacija UML-a na dijagrame, iako privlačna, ima svoju cenu jer UML kao celina nema formalnu semantiku, što unosi dvosmislenosti u jezik i otežava interoperabilnost sa drugim jezicima za modelovanje, konkretnim implementacijama, pa čak i sa različitim tipovima dijagrama unutar samog UML-a. Kao konkretnu ilustraciju, autori navode da strelice u UML dijagramima predstavljaju proizvoljne (više-prema-više) odnose, za razliku od strelica u teoriji kategorija koje predstavljaju funkcionalne (više-prema-jedan) odnose, što je upravo tipičan primer semantičke neodređenosti kakvu kritikuju Broy i Cengarle.

Pokušaji formalizacije: fUML, PSSM i ALF

Prepoznajući problem semantičke neodređenosti, OMG je u periodu 2008-2021. godine standardizovao tri proširenja namenjena uvođenju izvršive semantike UML-a. Prvo je *Semantics of a Foundational Subset for Executable UML Models* (fUML) [76], čija je aktuelna verzija 1.5 objavljena 2021. godine [76]. Drugo je *Precise Semantics of UML State Machines* (PSSM) [77], čija je verzija 1.0 objavljena 2019. godine i koja proširuje fUML izvršnom semantikom za dijagrame stanja [77]. Treće je *Action Language for Foundational UML* (ALF) [78], čija je verzija 1.1 objavljena 2017. godine i koja obezbeđuje konkretnu tekstualnu sintaksu za specifikaciju ponašanja unutar fUML modela [78].

Uprkos ovim naporima, kritike semantičke nepreciznosti UML-a u okviru MDA ostaju aktuelne. Sistematsku analizu specifikacije PSSM standarda sproveli su Elekes, Molnár i Micskei [7]. Autori konstatuju da razumevanje zasnovano na poluformalnoj semantici jeste preciznije od neformalnog teksta, ali može biti ograničeno ako ne pokriva ceo jezik. PSSM, na primer, ne uključuje vremenski zavisna ponašanja iz UML-a. Broy, Crane, Dingel, Hartman, Rumpe i Selic [79] prikazuju tadašnje stanje istraživanja u projektu UML 2 Semantics i različite pristupe formalnoj semantici UML-a, čime dokumentuju da je formalizacija jezika bila aktivna i višestruka istraživačka oblast.

U preglednoj studiji UML-a i ontoloških reprezentacija Mkhinini, Labbani-Narsis i Nicolle [80] sistematski analiziraju 66 radova objavljenih od 1999. godine. Odnose između UML-a i ontologija razvrstavaju prema motivaciji pristupa, a posebno razmatraju transformacije UML modela u ontologije i ontologija u UML modele, njihove praktične primene i složenost pravila transformacije.

Object Constraint Language kao dodatni sloj

Object Constraint Language (OCL) [81], čiju je aktuelnu verziju 2.4 OMG usvojio 2014. godine [81], predstavlja formalni jezik za izražavanje ograničenja nad UML modelima. OCL omogućava da se specifikuju invarijante klase, preduslovi i postuslovi metoda, kao i

53

8

60

uslovi čuvara u dijagramima stanja. U MDA praksi OCL je nužan dodatak osnovnim UML dijagramima kada se želi postići stroga semantika. Iskustva primene dokumentovana su u nizu radova, među kojima je naročito ilustrativan Wilke i Demuth [82]. Rezultati rada ukazuju na problem: uprkos zreloj verziji UML 2.3, model i dalje nije konzistentan, pa autori predlažu kako se OCL ograničenja mogu poboljšati da bi se konzistentnost zaista ostvarila.

Dokumentacija projekta Eclipse OCL, verzija 6.6.0 [83], navodi da je klasični evaluator tesno vezan za Ecore i da odsustvo razdvajanja semantike specifikacije OCL-a od semantike realizacije u Javi otežava tačnu realizaciju jednakosti u ugnježdenim kolekcijama (prevod autora). Ovaj primer pokazuje konkretno ograničenje klasične realizacije, a ne opštu neispravnost standarda OCL.

QVT kao standardni mehanizam transformacija

Transformacije između CIM, PIM i PSM modela u MDA paradigmi specifikuju se pomoću standarda *Query/View/Transformation* (QVT). QVT definiše tri različita jezika za pisanje transformacija: *QVT-Relations* (deklarativni jezik visokog nivoa za specifikaciju dvosmernih transformacija), *QVT-Core* (deklarativni jezik nižeg nivoa koji služi kao semantička osnova za QVT-Relations) i *QVT-Operational Mappings* (imperativni jezik za pisanje jednosmernih transformacija).

Kritičku analizu QVT standarda daje Kurtev [9]. Autor sistematski upoređuje zahteve različitih transformacionih scenarija sa onim što QVT pruža i konstatuje da je tadašnja podrška alata za QVT bila ograničena, što je odražavalo ranu fazu i razvoj standarda. Kasniji pregled Kahani i saradnika [84] dokumentuje da je podrška alata ostala neujednačena, pri čemu su četiri od devet razmatranih alata usmerenih na QVT u vreme njihove studije bila obustavljena.

Ozbiljan kritički osvrt na QVT pruža i rad Jouault, Allilaire, Bézivin, Kurtev i Valduriez [85]. Autori predstavljaju svoj jezik *ATLAS Transformation Language* (ATL) kao pragmatičnu alternativu QVT-u, čiji je razvoj podstaknut ograničenjima QVT-a u industrijskoj primeni. Van Amstel, Bosems, Kurtev i Ferreira Pires [10], eksperimentalno upoređuju performanse ATL-a, QVT-Operational i QVT-Relations i identifikuju konkretne faktore koji utiču na opadanje performansi transformacija sa porastom veličine i složenosti sistema.

Detaljnu savremenu klasifikaciju MDE transformacionih alata pružaju Kahani, Bagherzadeh, Cordy, Dingel i Varró [84]. Pregled obuhvata 60 alata i poredi ih prema 45 obeležja u šest grupa. Analiza održavanja pokazala je i da su četiri od devet alata usmerenih na standard QVT bila obustavljena u vreme sprovođenja studije. Noviji rad Abdelmalek, Khriss i Jakimi [86] predlaže pristup sastavljanju transformacija modela i pokazuje da podrška za izgradnju složenijih transformacionih tokova ostaje relevantan istraživački problem.

Osnovna ograničenja MDA paradigme

Zbirna analiza prethodno dokumentovanih nezavisnih studija ukazuje na tri osnovna ograničenja MDA paradigme koja su neposredno značajna za istraživački problem ove disertacije.

Prvo, UML kao osnova PIM-a pati od semantičke neodređenosti koja je sistematski dokumentovana u recenziranim radovima, od Fuentes i saradnika [72], preko Fecher i saradnika [73], Broy i Cengarle [8], do Rączkowska i Walkowiak-Gall [74] i NIST IR 8358 [75]. Ova neodređenost može se delimično ublažiti kroz fUML, PSSM, ALF i OCL anotacije, ali kako Elekes i saradnici [7] pokazuju na primeru PSSM-a, preciziranje semantike pojedinačnog podskupa ne pokriva celokupan UML. Pregled Mkhinini i saradnika [80] potvrđuje da se UML i ontologije u literaturi povezuju upravo zbog njihovih različitih svojstava i namena; ova disertacija zato ispituje mogućnost upotrebe OWL 2 DL ontologije kao primarnog PIM formalizma.

Drugo, ekosistem QVT transformacija ima neujednačenu i promenljivu podršku alata. Kahani i saradnici [84] u pregledu 60 alata beleže da su četiri od devet razmatranih alata usmerenih na QVT bila obustavljena. Kurtev [9], Jouault i saradnici [85], Van Amstel i saradnici [10] i Abdelmalek i saradnici [86] ukazuju na zasebna pitanja podrške alata, ovladavanja jezicima, učinka i sastavljanja transformacija. Imperativne varijante QVT transformacija same po sebi ne daju matematičku garanciju da će proces transformacije očuvati strukturni integritet i relacione zavisnosti izvornog modela u ciljnom domenu. U takvim okolnostima proveru se u praksi oslanja na testiranje, što nije dovoljno kada se zahteva formalno dokaziva svojstva.

Treće, MDA paradigma tretira transformacije kao spoljne artefakte u odnosu na modele. Transformaciona pravila čuvaju se u zasebnim QVT ili ATL datotekama, zbog čega se promene u ciljnoj platformi ili arhitektonskom obrascu moraju ručno uskladiti sa transformacionim skriptama. Nema standardnog mehanizma koji bi pravila transformacije tretirao kao ravnopravne entitete unutar samog modela, što je upravo problem koji rešava koncept reifikovanih preslikavanja iz metodologije ove disertacije.

2.3.2 Eclipse EMF/Ecore ekosistem

Paralelno sa razvojem MDA paradigme od strane OMG-a, Eclipse fondacija je razvila alternativni pristup modelom vođenom razvoju softvera zasnovan na *Eclipse Modeling Framework* (EMF). Steinberg, Budinsky, Paternostro i Merks [6] sistematski opisuju EMF, njegov metamodel Ecore, generisanje koda i povezivanje sa drugim alatima platforme Eclipse. Ecore i OMG-ov *Essential MOF* (EMOF) dele osnovne koncepte metamodelovanja, ali pripadaju različitim tehnološkim ekosistemima.

Ecore kao praktični metamodel

Jezgro EMF-a čini *Ecore*; metamodel za definisanje modela u stilu objektno-orijentisanog modelovanja. Osnovne metaklase Ecore-a su *EClass* (koja predstavlja klasu modela),

EAttribute (atribute tih klasa), *EReference* (relacije između klasa), *EPackage* (logički paket koji grupiše klase) i *EDataType* (primitivne tipove podataka). Ecore modeli mogu biti serijalizovani u XMI formatu (XML Metadata Interchange), što omogućava njihovo čuvanje, razmenu i verzionisanje u standardnim repozitorijumima izvornog koda [87]. EMF alati za generisanje koda mogu iz Ecore modela automatski generisati Java klase sa metodama za pristup atributima i njihovu izmenu, mehanizmima za praćenje izmena i infrastrukturom za serijalizaciju.

EMF kao široko korišćeni industrijski okvir

EMF je osnova razvijenog skupa alata za modelom vođeno inženjerstvo u okruženju Eclipse. Projekti kao što su *Graphical Modeling Framework* (GMF), *Eclipse Sirius*, *Eclipse Papyrus*, *Xtext* i *Epsilon Modeling Framework*⁴ koriste Ecore modele ili obezbeđuju integraciju sa njima. Takva povezanost čini EMF praktično značajnim ekosistemom, ali sama po sebi ne predstavlja dokaz da je jedan modelom vođeni pristup zamenio sve ostale.

Platformska nezavisnost modela i prenosivost alata

Praktično ograničenje prenosivosti izvornog EMF ekosistema opisuju Schwichtenberg, Jovanovikj, Gerth i Engels [88] u radu [88]. Autori polaze od činjenice da se aplikacije namenjene većem broju platformi, poput Androida, iOS-a, Windowsa i veb-pregledača, obično realizuju u različitim programskim jezicima, dok EMF generiše Java kod iz platformski nezavisnih modela sa ugrađenim OCL izrazima. Potpuni Ecore i OCL aplikativno-programski interfejsi (API) za izvršavanje nisu dostupni za sve ove platforme, pa se njihova funkcionalnost mora ponovo realizovati. CrossEcore rešava ovaj problem generisanjem C#, Swift, TypeScript i JavaScript koda iz Ecore modela. Postojanje takvog okvira pokazuje da primena istog modela van platforme Java zahteva dodatnu generatorsku i izvršnu infrastrukturu, a ne da Ecore model prestaje da bude platformski nezavisan.

Razliku između platformske nezavisnosti modela i prenosivosti alata pokazuju i drugi okviri. *.NET Modeling Framework* (NMF) generiše C# kod iz Ecore modela [89]; *PyEcore* obezbeđuje Python implementaciju EMF-a koja omogućava rad sa Ecore metamodelima i modelima u Pythonu; projekat *Model-Driven Engineering for C++* (MDE4CPP) razvija okruženje nalik EMF-u za C++. Ova rešenja pokazuju trošak prenosa generatorske i izvršne infrastrukture na druge platforme, ali ne opovrgavaju platformski nezavisan status samog Ecore modela.

Dodatni primer tehnološke vezanosti konkretne implementacije nalazi se u zvaničnoj dokumentaciji projekta *Eclipse OCL* [83]. U njoj se navodi da je klasični evaluator tesno vezan za Ecore i da odsustvo razdvajanja semantike specifikacije OCL-a od semantike

⁴ Zvanične stranice projekata: Eclipse GMF Tooling: <https://projects.eclipse.org/projects/modeling.gmf.gmf-tooling>; Eclipse Sirius: <https://eclipse.dev/sirius/>; Eclipse Papyrus: <https://eclipse.dev/papyrus/>

realizacije u Javi otežava tačnu realizaciju jednakosti u ugnježdenim kolekcijama. Reč je, dakle, o precizno određenom ograničenju klasičnog evaluatora.

Dokumentovana vezanost izvornog EMF alata i izvršnih API-ja za Javu pokazuje da platformska nezavisnost modela sama po sebi ne obezbeđuje prenosivu realizaciju. U metodologiji ove disertacije ontološki PIM zasnovan na OWL 2 DL ne bira se zato što Ecore nije PIM, već zato što pruža formalnu semantiku i mogućnost zaključivanja nad opštim pojmovima, kao što su individue, klase, odnosi i aksiomi. Prenosivost realizacije rešava se zasebno, eksplicitnim preslikavanjima, transformacionim ugovorom i generatorskim strategijama za izabrane ciljne tehnologije.

Problem evolucije metamodela i modela

Pored ograničenja prenosivosti alata, EMF/Ecore ekosistem suočava se i sa problemom usklađenog razvoja metamodela i modela (engl. *co-evolution*). Herrmannsdoerfer, Benz i Juergens [90] dokumentuju da se nakon promene metamodela postojeći modeli moraju migrirati. Taj zadatak je u praksi zahtevan, podložan greškama i samo delimično automatizovan. Problem je posebno izražen u dugoročnom razvoju poslovnih informacionih sistema, u kojima se metamodel menja zajedno sa poslovnim zahtevima. Namenski pristup migraciji predstavlja Epsilon Flock, jezik koji su Rose, Kolovos, Paige, Polack i Poulding [91] razvili za ažuriranje instanci modela nakon promene metamodela.

Proširenja: Xtext, Xtend i ekosistem alata

EMF ekosistem nije statičan; tokom dve decenije razvio je bogato okruženje pratećih alata. *Xtext* [92] je alat za izgradnju domenski-specifičnih jezika koji automatski generiše sintaksički analizator (parser), uređivač sa automatskim dopunjavanjem i postupak provere i mehanizme za izgradnju apstraktnih sintaksnih stabala iz gramatičke specifikacije. *Xtend* je programski jezik koji se prevodi u izvorni kod na jeziku Java i specijalizovan je za pisanje generatora koda iz EMF modela, nudeći šablonski pristup sa imperativnom semantikom. *Epsilon Modeling Framework*, razvijen na Univerzitetu u Jorku i održavan kao Eclipse projekat, obezbeđuje porodicu jezika za rad sa EMF modelima (Eclipse Foundation [93]).

Navedeni alati Xtext, Xtend i Epsilon neposredno su povezani sa okruženjem Eclipse, pri čemu se uobičajeno oslanjaju na EMF i JVM. Pregled Kahani i saradnika [84] pokazuje, međutim, da širi skup transformacionih alata obuhvata različite jezike, okruženja za izvršavanje i razvojne platforme. Tehnološku vezanost zato treba procenjivati za konkretan skup alata, a ne pripisivati je MDE-u kao celini.

2.3.3 QVT, Acceleo i imperativne transformacije: ograničenja

Treći pravac u oblasti modelom vođenog razvoja softvera odnosi se na konkretne alate za izvršavanje transformacija modela u model (engl. *model-to-model*, M2M) i modela u tekst (engl. *model-to-text*, M2T). Dok je QVT u prethodnim potpoglavljima razmatran kao OMG standard, ovo potpoglavlje razmatra kako se jezici slični QVT-u i alati za

šablonsko generisanje koda ponašaju u praksi i koja ograničenja ta praksa otkriva, sa osloncem na recenziranu akademsku literaturu koja je ta ograničenja sistematski dokumentovala.

Acceleo i pristupi zasnovani na šablonima za M2T transformacije

Acceleo, trenutno deo Eclipse ekosistema i pod održavanjem Obeo organizacije, predstavlja implementaciju OMG-ovog *Model-to-Text Transformation Language* (MOFM2T) [94] standarda, čija je verzija 1.0 usvojena 2008. godine [94]. *Acceleo* koristi šablonski pristup u kojem se tekstualni šabloni sa ugnježđenim instrukcijama za pristup modelima kombinuju sa EMF modelima radi generisanja izvornog koda.

Sistematski pregled pristupa transformacijama dali su Czarnecki i Helsen [95], pri čemu su šablonski zasnovane transformacije izdvojene kao posebna kategorija. Rose, Paige, Kolovos i Polack [96] predstavljaju Epsilon Generation Language (EGL) kao jezik za transformaciju modela u tekst, opisuju njegova posebna svojstva i porede ga sa drugim jezicima iste namene. Ti izvori potvrđuju postojanje različitih šablonskih pristupa, ali ne znače da svi konkretni jezici imaju ista ograničenja.

Za ovu disertaciju važna su tri inženjerska rizika šablonskog pristupa: (1) mešanje poslovne logike sa sintaksnom projekcijom, kada šabloni pored same projekcije sintakse sadrže i odluke o tome *šta* treba generisati; (2) vezivanje šablonskih instrukcija za konkretan metamodel i njegov izvršni ekosistem; i (3) odsustvo formalne garancije korektnosti samo na osnovu činjenice da se koristi šablon. Greška u šablonu može proizvesti kod koji se ne prevodi ili kod koji se prevodi, ali se ponaša drugačije od očekivanog.

Osnovni problem imperativne prirode transformacija

QVT-Operational i Xtend dopuštaju proceduralno definisanje transformacije, dok *Acceleo* koristi šablone za transformaciju modela u tekst. Imperativni, deklarativni i šablonski pristupi razlikuju se po načinu zadavanja pravila, ali nijedna od tih oznaka sama ne dokazuje očuvanje izabranih svojstava. Czarnecki i Helsen [95] sistematizuju ove razlike; za konkretnu transformaciju potrebno je zasebno povezati pravila sa njenim ugovorom i postupkom provere.

ATL (Atlas Transformation Language), razvijen u INRIA grupi [85], kombinuje deklarativna pravila sa imperativnim konstrukcijama. Zato se ne svodi na imperativne skripte. Za razmatrani problem važno je da izbor ATL-a ili pojedinog QVT jezika sam po sebi ne predstavlja dokaz očuvanja izabranih struktura konkretne transformacije [84].

Detaljnu sistematizaciju pristupa transformacijama dali su Mens i Van Gorp [97]. Autori razlikuju *endogene* transformacije (između modela istog metamodela) i *egzogene* transformacije (između modela različitih metamodela), kao i *horizontalne* i *vertikalne* transformacije na osnovu nivoa apstrakcije izvornog i ciljnog modela. Ova taksonomija razdvaja vrste transformacija i njihova svojstva. Očuvanje izabranih semantičkih osobina

predstavlja zasebnu dokaznu obavezu i ne sledi samo iz izbora imperativne ili deklarativne sintakse transformacije.

U imperativnoj paradigmi transformacija se definiše kao niz koraka. Ovakav pristup je intuitivan, ali otvara nekoliko inženjerskih pitanja. Prvo, iz samog izbora imperativnog jezika ne sledi formalna garancija očuvanja strukture; za takvu tvrdnju potrebni su odgovarajuća semantika, specifikacija i dokazne obaveze. Drugo, promene metamodela zahtevaju usklađivanje postojećih modela, što sistematski dokumentuju Herrmannsdoerfer, Benz i Juergens [90] u radu o usklađenom razvoju metamodela i modela. Treće, podrška za više ciljnih platformi zahteva odgovarajuća pravila i realizacije za svaku od njih. Pregled Kahani i saradnika [84] pokazuje da se alati znatno razlikuju prema podržanim jezicima, okruženjima za izvršavanje i mogućnostima ponovne upotrebe, pa se obim tog posla mora ocenjivati za konkretan transformacioni sistem.

Put ka deklarativnim i formalno zasnovanim pristupima

U akademskoj literaturi već dve decenije postoji svest o ograničenjima imperativnih transformacija i interesovanje za deklarativne, formalno zasnovane alternative. Czarnecki i Helsen [95] sistematizuju ove pristupe. Na teorijskoj strani, Diskin i Majbaum (Maibaum) [98] programski argumentuju za primenu teorije kategorija kao teorijske osnove za MDE. Ovu argumentaciju produbljuje i sistematizuje Kovaljov (Kovalyov) [99].

Glavna ideja kategorijalnog pristupa jeste da se modeli posmatraju kao kategorije, matematičke strukture u kojima objekti predstavljaju elemente modela, a morfizmi relacije između njih, dok se transformacije između modela posmatraju kao funktori. Funktori po definiciji čuvaju identitetne morfizme i kompoziciju. Iz toga ne slede automatski vernost, injektivnost na objektima niti očuvanje svih razlika između elemenata modela; te jače osobine u ovoj disertaciji važe uslovno, za odabrani šematski deo i pod pretpostavkama utvrđenim u Poglavlju 4. Nezavisan argument za kategorijalni pristup modelovanju klasa dali su Padi i saradnici [75] u već pomenutoj NIST publikaciji.

Rezime: argumenti za alternativni pristup

Sistematska analiza tri dominantna pravca u MDE-u (MDA, EMF i imperativne transformacije), zasnovana na pregledanim istraživačkim radovima, standardima i zvaničnoj dokumentaciji, otkriva zajednički skup ograničenja koja podstiču razvoj metodološkog pristupa ove disertacije. UML pati od semantičke nepreciznosti, što dokumentuju Fuentes i saradnici [72], Fecher i saradnici [73], Broy i Cengarle [8], Rączkowska i Walkowiak-Gall [74], Elekes i saradnici [7] i Padi i saradnici [75]. Platformska nezavisnost Ecore modela sama po sebi ne obezbeđuje prenosivost generatorske i izvršne infrastrukture, što pokazuju višeplatformski okviri NMF i CrossEcore [89, 88], kao i zvanična dokumentacija projekta Eclipse OCL. QVT, ATL i Acceleo nude različite deklarativne, imperativne i šablonske mehanizme, pa njihove mogućnosti treba razmatrati odvojeno [95, 84, 9, 97]. Za ovu disertaciju ključno je

eksplicitno povezivanje konkretnog transformacionog ugovora sa svojstvima koja se proveravaju, uz razlikovanje specifikacije od njene programske realizacije.

Metodologija koja je predmet ove disertacije predlaže sistematski odgovor na svaki od ovih problema. Prvo, UML i Ecore se zamenjuju OWL 2 DL ontologijom, čime se obezbeđuje osnova za semantičku strogost, automatsko zaključivanje i proveru logičke konzistentnosti. Drugo, transformacioni lanac formalizuje se kategorijalnim funktorima u skladu sa teorijskim okvirom Diskina i Majbauma [98] i Kovaljova [99]: identiteti i kompozicija čuvaju se po definiciji funktora, dok se vernost i odsustvo neželjenog sažimanja odabranih šematskih razlika dobijaju samo pod izričitim uslovima Propozicije 4.2. Treće, konfiguracije se zamenjuju reifikovanim preslikavanjima, odnosno pravilima transformacije predstavljenim kao ontološki entiteti nad kojima su mogući upiti i upravljanje verzijama. Četvrto, isti mehanizam razdvaja platformski nezavisan model od realizacionih odluka specifičnih za određeni skup tehnologija. Podrška novom skupu tehnologija zahteva odgovarajući PSM profil, razrešavač, fazu transformacije modela u kod i proveru, bez promene domenskog PIM-a.

Detaljno izlaganje svakog od ovih odgovora je predmet poglavlja 3 (modularni BAB ontološki okvir), 4 (kategorijalna formalizacija) i 5 (BAB softverski okvir). Pre nego što se prikaže predložena metodologija, neophodno je razmotriti i ostale teorijske osnove koje čine njene konstitutivne elemente: ontološko modelovanje (potpoglavljje 2.4), teoriju kategorija (potpoglavljje 2.5) i automatizaciju generisanja koda (potpoglavljje 2.6).

2.4 Ontološko modelovanje u softverskom inženjerstvu

Ontološko modelovanje predstavlja jednu od tri osnove metodološkog pristupa ove disertacije, uz modelom vođeni razvoj softvera (potpoglavljje 2.3) i teoriju kategorija (potpoglavljje 2.5). Dok MDE paradigma pruža strukturni okvir za razdvajanje apstraktnih modela od konkretnih implementacija, ontološki formalizmi obezbeđuju formalno definisanu semantiku, čiji je izostanak prepoznat kao ključni nedostatak pristupa zasnovanih na UML-u i EMF-u. Ovo potpoglavljje prikazuje teorijske osnove ontološkog modelovanja, OWL 2 DL i deskriptivne logike, ulogu ontologija u razvoju informacionih sistema i, posebno, ontološko predstavljanje programskog koda.

2.4.1 OWL 2 DL i deskriptivna logika

Pojam ontologije u informatici

Moderno razumevanje pojma *ontologije* u informatici počinje radom Grubera [100], u kojem je formulisana jedna od najčešće korišćenih definicija: „*Ontologija je eksplicitna specifikacija konceptualizacije*“ (prevod autora). Pojam je preuzet iz filozofije, gde ontologija označava sistematičan prikaz postojanja, ali je u kontekstu deljenja znanja među sistemima veštačke inteligencije prilagođen opisivanju pojmova i odnosa koji formalno postoje za jednog ili više agenata.

Gruber [101] razradio je i proširio ovu definiciju u radu [101], u kojem je formulisao pet ključnih kriterijuma za projektovanje ontologija: jasnoću, koherentnost, proširivost, minimalnu pristrasnost pri kodiranju i minimalnu ontološku obavezu. Borst [102] je u doktorskoj disertaciji odbranjenoj na Univerzitetu u Twenteu dodao važan atribut „deljenosti“ (engl. *shared*) i formulisao dopunjenu definiciju: ontologija je „*formalna specifikacija deljene konceptualizacije*“ (prevod autora). Studer, Benjamins i Fensel [103] sintetizovali su Gruberovu i Borstovu formulaciju u ono što je postalo standardna definicija u oblasti: „*Ontologija je formalna, eksplicitna specifikacija deljene konceptualizacije*“ (prevod autora). Ovu definiciju sa četiri odrednice (formalna, eksplicitna, deljena, konceptualizacija) Guarino, Oberle i Staab [104] produbljuju formalnom analizom odnosa između konceptualizacije i njenog ontološkog predstavljanja.

Razvoj OWL-a kao standardnog ontološkog jezika

Praktična primena ontoloških ideja u veb-okruženju počinje početkom 2000-ih godina sa inicijativama DAML (DARPA Agent Markup Language) i OIL (Ontology Inference Layer), koje su se 2001. godine objedinile u jezik DAML+OIL [105]. World Wide Web konzorcijum (W3C) je 2001. godine osnovao *Web Ontology Working Group* pod suvoditeljstvom Džejmsa Hendlera (James Hendler) i Gusa Šrajbera (Guus Schreiber),⁵ koja je radila na standardizaciji ontološkog jezika za semantički veb. Rezultat tog napora je *Web Ontology Language* (OWL), koji je zvanično postao W3C preporuka 10. februara 2004. godine [106]. Prvobitni standard OWL 1 definisao je tri varijante jezika sa različitim nivoom izražajnosti: OWL Lite, OWL DL i OWL Full.

U oktobru 2007. godine pokrenuta je nova W3C radna grupa sa zadatkom da razvije sledeću generaciju standarda, a nova verzija, OWL 2, zvanično je usvojena 27. oktobra 2009. godine. Aktuelna referentna specifikacija je drugo izdanje iz 11. decembra 2012. godine. Dokument [11] daje uvod u modelovanje pomoću OWL-a 2 i upućuje na zasebne dokumente o strukturi, sintaksi, semantici, profilima i usaglašenosti. OWL 2 dodaje OWL 1-u niz novih mogućnosti: proširenu izražajnost svojstava (uključujući lokalnu refleksivnost i lance svojstava), proširenu podršku za tipove podataka, jednostavne mogućnosti metamodelovanja, proširene mogućnosti označavanja i ključeve.

Tri profila OWL 2: EL, QL, RL

OWL 2 definiše *tri profila* (sintaktičke podjezike) koji uspostavljaju kompromis između izražajnosti i efikasnosti zaključivanja, dokumentovane u specifikaciji [107]. OWL 2 EL naročito je pogodan za aplikacije koje zahtevaju ontologije sa velikim brojem klasa i svojstava, i posebno je usmeren ka medicinskim terminologijama poput SNOMED CT. Osnova mu je porodica deskriptivnih logika EL (*Existential quantification*), koja omogućava algoritme polinomijalne složenosti za sve standardne zadatke zaključivanja.

⁵ Zvanična stranica radne grupe W3C *Web Ontology Working Group*: <https://www.w3.org/2001/sw/WebOnt/>

OWL 2 QL namenjen je aplikacijama sa velikim količinama podataka o individuuama, gde je odgovaranje na upite najvažniji zadatak zaključivanja. U OWL 2 QL, odgovori na konjunktivne upite mogu se dobiti pomoću konvencionalnih relacionih sistema za upravljanje bazama podataka, čija je osnova porodica deskriptivnih logika DL-Lite [108]. OWL 2 RL (Rule Language) je profil namenjen implementaciji pomoću tehnologija zasnovanih na pravilima, poput sistema za upravljanje bazama podataka proširenih pravilima, sa polinomijalnom složenošću zaključivanja.

Među alternativnim formalizmima i platformama za specifikaciju sistema, koji nisu izabrani kao osnova ove disertacije, nalaze se Alloy [109], zasnovan na relacionoj logici prvog reda, Clafer [110] za modelovanje obeležja sa atributima, te JetBrains MPS⁶ kao platforma za jezičko inženjerstvo zasnovana na projekcionim uređivačima. OWL 2 DL je izabran zbog usklađenosti sa standardima i alatima semantičkog veba, među kojima su HermiT⁷, SPARQL, SHACL i SWRL, kao i zbog podrške za automatsko zaključivanje koja je značajna za argumentaciju ove disertacije.

OWL 2 DL i deskriptivna logika SROIQ

Za svrhe ove disertacije centralna varijanta je OWL 2 DL (DL = *Description Logic*), koja obezbeđuje visoku izražajnu moć uz očuvanje odlučivosti zaključivanja. Formalni temelj OWL 2 DL čini deskriptivna logika SROIQ, koja je uvedena u radu Horrocks, Kutz i Sattler [111]. Ime SROIQ je akronim koji označava ključne izražajne konstrukcije logike: *S* (ALC sa tranzitivnim ulogama), *R* (složena uključenja uloga, refleksivne i asimetrične uloge), *O* (nominali, individualna imena u klasnim izrazima), *I* (inverzna svojstva) i *Q* (kvalifikovana ograničenja kardinalnosti).

U zvaničnom dokumentu [112] izričito se navodi da je direktna semantika OWL 2 kompatibilna sa semantikom deskriptivne logike SROIQ zasnovanom na teoriji modela, pri čemu je SROIQ fragment logike prvog reda sa korisnim računskim svojstvima. Prednost ovako tesne matematičke veze jeste to što OWL 2 alati mogu neposredno koristiti obimnu literaturu o deskriptivnim logikama i iskustvo stečeno tokom više decenija razvoja zaključivača.

Referentna monografija za deskriptivne logike je Baader, Calvanese, McGuinness, Nardi i Patel-Schneider (ur., 2003, drugo izdanje 2007) [113]. Hitzler, Krötzsch i Rudolph [114] daju sistematski uvod u deskriptivne logike u kontekstu semantičkog veba.

Pretpostavka otvorenog sveta i odsustvo pretpostavke o jedinstvenosti imena

Dve karakteristike OWL 2 DL semantike su od posebne važnosti za metodološki pristup ove disertacije. Prva je pretpostavka otvorenog sveta (engl. *Open World Assumption*, OWA), princip po kom odsustvo eksplicitnog iskaza samo po sebi ne implicira njegovu neistinitost; iskaz može biti izveden iz drugih aksioma ili ostati neodređen. Ovo je suštinska razlika u odnosu na relacione baze podataka i logičko programiranje tipa Prolog

⁶ Zvanična stranica platforme JetBrains MPS: <https://www.jetbrains.com/mps/>

⁷ Zvanična stranica zaključivača HermiT: <https://www.hermit-reasoner.com/>

ili Datalog, koji rade pod pretpostavkom zatvorenog sveta (engl. *Closed World Assumption*). Druga karakteristika je odsustvo pretpostavke o jedinstvenosti imena (engl. *Unique Name Assumption*): dva različita IRI identifikatora mogu označavati istu individuu; njihova različitost mora biti eksplicitno navedena, na primer pomoću *owl:differentFrom*, ili izvedena iz drugih aksioma, kako navode dokumenti [11] i [112].

Ove dve karakteristike neposredna su posledica izvorne namene OWL-a kao jezika za semantički veb, u kojem je znanje raspodeljeno među različitim izvorima i ne može se očekivati potpuna, zatvorena slika sveta. Za razvoj poslovnih informacionih sistema, međutim, pretpostavka otvorenog sveta predstavlja i ograničenje: kada validacija zahteva tvrdnje oblika „*entitet nije kategorija X*“, takva nepripadnost mora biti eksplicitno izražena, na primer pripadnošću komplementu klase, ili izvedena iz drugih aksioma; odsustvo tvrdnje o pripadnosti nije dovoljno. Sirin, Parsia, Cuenca Grau, Kalyanpur i Katz [115] predstavljaju Pellet kao potpun OWL DL zaključivač koji podržava proveru konzistentnosti, zadovoljivosti i zaključivanje nad individuuama; takvo zaključivanje i dalje sledi OWL semantiku otvorenog sveta.

Navedena ograničenja izražajnosti OWL 2 DL (otvoreni svet, nemogućnost izražavanja određenih klasa pravila, odsustvo neposredne podrške za validaciju instanci prema potpunom skupu podataka) mogu se delimično obraditi pomoću dopunskih standarda semantičkog veba. Prvi je Semantic Web Rule Language (SWRL), čija je radna specifikacija objavljena 2004. godine [116], koji kombinuje OWL DL i RuleML pravila. SWRL omogućava izražavanje pravila sa više uslova i izvođenje novih tvrdnji koje nisu neposredno zapisane u ontologiji. Praktična ograničenja SWRL-a uključuju dodatnu računsku složenost (SWRL bez ograničenja je neodlučiv, pa se često koristi bezbedan DL podskup) i činjenicu da ga ne podržavaju svi zaključivači podjednako.

Drugi standard je Shapes Constraint Language (SHACL), koji je W3C preporuka od 2017. godine [117]. SHACL se bavi drugom vrstom problema nego SWRL: dok SWRL proširuje izvođenje tvrdnji, SHACL proverava da li zadati RDF graf zadovoljava deklarirana ograničenja. Konkretno, SHACL omogućava da se nad RDF grafom definišu oblici (engl. *shapes*) i da se generišu izveštaji o povredama. Takva validacija pogodna je za proveru kardinalnosti, obaveznih vrednosti i formata identifikatora, koje se u UML-u uobičajeno izražavaju pomoću OCL ograničenja, a u relacionim bazama pomoću ograničenja CHECK i NOT NULL.

Metodologija ove disertacije koristi OWL kao primarni formalizam za PIM i HermiT za proveru logičke konzistentnosti PIM importnog zatvarača. SWRL i SHACL predstavljaju značajne dopunske standarde, ali nisu deo trenutnog toka generisanja. Uslovi kompletnosti i jednoznačnosti koje OWL otvoreni svet ne može da nametne proveravaju se namenskim validatorima napisanim u Javi nad modelom Jena⁸. Ova odluka o implementaciji ne menja opštu vrednost SWRL-a i SHACL-a, ali sprečava da se njihova moguća buduća primena prikaže kao već realizovana mogućnost.

⁸ Zvanična stranica biblioteke Apache Jena: <https://jena.apache.org/>

2.4.2 Ontologije u razvoju softvera: pregled pristupa

Ontološki vođeni informacijski sistemi

Primena ontoloških formalizama na razvoj informacionih sistema je zasebna oblast istraživanja koja je u poslednje dve decenije sistematizovana pod pojmom *Ontology-Driven Information Systems* (ODIS). Referentna monografija u ovoj oblasti je Sharman, Kishore i Ramesh (ur., 2007), *Ontologies, A Handbook of Principles, Concepts and Applications in Information Systems*. Autori obrazlažu da ontologija pokriva kako *strukturnu dimenziju* informacionog sistema (mehanizmi za generisanje i čuvanje generičkih komponenti IS-a i pristup tim komponentama), tako i *vremensku dimenziju* (smernice za razvoj novog informacionog sistema, pomoć u izboru odgovarajućih procesa, algoritama i softverskih komponenti).

Pionirski teorijski okvir za primenu ontologija na modelovanje informacionih sistema dao je Guarino [118]. Ovaj temeljni rad razmatra ključnu razliku između *ontologije* kao filozofske discipline i *ontologije* kao informatičkog artefakta i određuje trostruku ulogu ontologija u razvoju informacionih sistema: kao alat za analizu domena pre razvoja, kao sredstvo semantičke interoperabilnosti između sistema i kao komponenta izvršne infrastrukture tokom rada sistema.

Sistematski pregled uloga formalnih ontologija

Savremeni sistematski pregled uloge formalnih ontologija u razvoju informacionih sistema dali su Husáková i Bureš [55]. Na osnovu analize koja je počela sa 187 studija i bila sužena na 27 članaka koji direktno tretiraju vezu između ontologija i inženjeringa sistema, autori su identifikovali pet glavnih uloga formalnih ontologija u razvoju informacionih sistema: (1) interoperabilnost između heterogenih sistema, (2) upravljanje ljudskim resursima, (3) predstavljanje domenskog znanja, (4) semantiku u modelovanju zasnovanom na UML-u, i (5) upravljanje programskim kodom i dokumentacijom.

Posebno je značajna uloga ontologija u modelovanju zasnovanom na UML-u, koja neposredno povezuje ovo potpoglavlje sa raspravom iz potpoglavlja 2.3. Husáková i Bureš obrazlažu da ontologije mogu služiti kao spoljni formalni mehanizam za obogaćivanje semantike modela. Pregledni rad Mkhinini, Labbani-Narsis i Nicolle [80], analizira 66 radova koji povezuju ove dve vrste prikaza. Autori klasifikuju pristupe prema njihovoj motivaciji i posebno razmatraju transformacije iz UML-a u ontologije i iz ontologija u UML, njihove praktične primene i složenost pravila transformacije.

OntoUML i drugi ontološki profili UML-a

Razvijen pristup ontološkom proširenju UML-a predstavlja OntoUML, ontološki zasnovan profil UML-a za konceptualno modelovanje domenskih ontologija. Giancarlo Guizzardi razvio je OntoUML u okviru doktorske disertacije odbranjene na Univerzitetu u Twenteu [119], za koju je dobio nagradu CAiSE Best PhD Award. OntoUML je zasnovan na temeljnoj ontologiji UFO (*Unified Foundational Ontology*), koja razlikuje

klase prema njihovoj ontološkoj prirodi; na primer razlikuje supstancijalne kategorije (*kinds*), uloge (*roles*), faze životnog ciklusa (*phases*), i apstraktne kategorije (*mixins*). Ovakva ontološka disciplina ograničava ono što se može modelirati i omogućava otkrivanje pojedinih vrsta konceptualnih grešaka koje osnovni UML profil sam ne sprečava.

Kasniji razvoj OntoUML pristupa praćen je kroz niz radova istog autora, među kojima je sinteza Guizzardi, Wagner, Almeida i Guizzardi [120], koja postavlja UFO kao teorijsku osnovu OntoUML-a i drugih formalizama za konceptualno modelovanje. OntoUML uvodi ontološke razlike i ograničenja koja nisu neposredno dostupna u osnovnom UML profilu, ali zadržava UML notaciju i predstavlja njenu ontološki zasnovanu nadgradnju. Metodologija ove disertacije bira drugačije rešenje: OWL 2 DL ontologiju koristi kao primarni formalizam za PIM.

Ontologije za poslovne procese

Posebna grana primene ontologija odnosi se na formalizaciju notacija za modelovanje poslovnih procesa, pre svega BPMN-a (Business Process Model and Notation) i BPEL-a (Business Process Execution Language). De Nicola, Lezoche i Missikoff [54] u ranoj studiji razvijaju ontološki okvir za preslikavanje elemenata modelovanja poslovnih procesa u formalne ontološke aksiome. Annane, Aussenac-Gilles i Kamel [37] razvijaju OWL ontologiju čije jezgro predstavlja fragment metamodela BPMN 2.0 i proširuju ga taksonomijama i relacijama potrebnim za detaljno predstavljanje poslovnih procesa i odgovaranje na unapred definisana pitanja. Singer [36] predstavlja razrađenu OWL ontologiju BPMN-a, pokazuje da se BPMN modeli mogu serijalizovati u OWL-u i razmatra transformaciju u akterski prikaz procesa kada notacije imaju zajedničke osnovne pojmove.

U uporednoj studiji ontoloških pristupa poslovnim procesima, Annane, Kamel i Aussenac-Gilles [121] sistematski upoređuju sedam prikaza procesa i utvrđuju zajedničke obrasce i razlike. Ova grana istraživanja značajna je za procesni deo ontologije BAB Core, koji procese, zadatke, stanja i prelaze povezuje sa strukturom poslovne aplikacije.

Neurosimboličko povezivanje ontologija i savremenog mašinskog učenja

Ovo potpoglavlje ne bi bilo potpuno bez osvrta na jedan od novijih pravaca razvoja u oblasti, povezivanje ontologija sa mašinskim učenjem u paradigmi neurosimboličke veštačke inteligencije. Sarker, Zhou, Eberhart i Hitzler [38] sistematizuju načine povezivanja statističkog učenja sa simboličkim znanjem i zaključivanjem. Breit, Waltersdorfer, Ekaputra i saradnici [39] analiziraju 476 sistema koji kombinuju komponente mašinskog učenja i resurse semantičkog veba, opisuju njihove arhitektonske obrasce, nivo zrelosti, zadatke i domene primene.

U pogledu interakcije sa velikim jezičkim modelima, Neuhaus [62] obrazlaže da se ontologije i LLM-ovi međusobno dopunjuju, a ne nadmeću: LLM-ovi omogućavaju fleksibilno razumevanje prirodnog jezika, ali su skloni halucinacijama, dok ontologije

obezbeđuju stroge simboličke prikaze, ali zahtevaju ručnu izradu. Babaei Giglou, D'Souza i Auer [63] prikazuju primenu LLM-ova za automatsko učenje ontologija iz tekstualnih korpusa. Obrnuto, Palagin, Kaverinskiy, Litvin i Malakhov [64] koriste ontologije za strukturisanje upita i prikazuju rad sistema na zadacima izdvajanja entiteta, klasifikacije i generisanja odgovora. Ova dva pravca, LLM-ovi za učenje ontologija i ontologije za strukturisanje upita za LLM-ove, predstavljaju komplementarne istraživačke mogućnosti.

2.4.3 CodeOntology: ontološko modelovanje programskog koda

Motivacija za ontološko modelovanje koda

Pored modelovanja poslovnih domena i informacionih sistema na visokom nivou apstrakcije, ontološki formalizmi mogu se primeniti i na modelovanje samog programskog koda, čime nastaje formalni, mašinski čitljiv prikaz strukture, sintakse i semantike izvornih datoteka. Motivacija za ovaj pristup proizilazi iz porasta obima softvera otvorenog koda na platformama poput GitHuba, SourceForgea i GitLaba⁹, koji čini sve veći izvor znanja o softverskim obrascima, stilovima programiranja i problemskim rešenjima. Bez sistematskog formalnog modelovanja, ovo znanje ostaje skriveno u tekstualnim datotekama i nedostupno za napredno pretraživanje, analizu i ponovnu upotrebu.

Pregled pristupa ontološkom modelovanju koda

U oblasti ontološkog modelovanja programskog koda razvijeno je nekoliko značajnih inicijativa tokom poslednjih petnaest godina. Jedan od ranijih pristupa je *Source Code Representation Ontology* (SCRO), usmeren na odnose i zavisnosti između artefakata objektno-orijentisanog koda. Alnusair, Zhao i Bodden [122] opisuju SCRO kao OWL-DL model koji obuhvata klase i interfejsse, nasleđivanje, preopterećenje i redefinisane metode i njihove potpise, a koriste ga kao osnovu sistema za preporučivanje programskog koda. Kasniji pristup CodeOntology povezuje ontologiju objektno-orijentisanog koda sa raščlanjivačem koji izvorni kod ili bajtkod jezika Java serijalizuje u RDF trojke [123, 124]. De Aguiar, Falbo i Silva Souza [125] razvijaju referentnu ontologiju za objektno-orijentisani kod zasnovanu na OntoUML metodologiji i UFO temeljnoj ontologiji, čime uspostavljaju vezu sa širim pravcem ontoloških istraživanja opisanom u potpoglavlju 2.4.2.

CodeOntology

Dobro razrađen pristup ontološkom modelovanju koda je CodeOntology, koji su razvili Atzeni i Atzori. Referentan rad je Atzeni i Atzori [123]. Projekat se sastoji od dva glavna doprinosa: ontologije koja pruža formalni prikaz objektno-orijentisanih programskih jezika i raščlanjivača koji može da analizira izvorni kod na jeziku Java i serijalizuje ga u

⁹ Zvanične stranice platformi: GitHub: <https://github.com/>; SourceForge: <https://sourceforge.net/>; GitLab: <https://about.gitlab.com/>

RDF trojke. Raščlanjivač je uspješno primenjen na izvornom kodu OpenJDK 8, pri čemu je prikupljen strukturisan skup podataka sa više od dva miliona RDF trojki. CodeOntology omogućava generisanje povezanih podataka (engl. *Linked Data*) iz projekata na jeziku Java i izvršavanje SPARQL upita nad tako predstavljanim izvornim kodom.

CodeOntology je napisan u OWL 2 i prvenstveno je usmeren na programski jezik Java, ali je projektovan tako da se lako proširuje i prilagođava modelovanju drugih objektno-orijentisanih jezika. Proces modelovanja koji je bio osnova za izradu ontologije vođen je tipičnim *kompetencijskim pitanjima* (engl. *competency questions*) koja se postavljaju tokom razvoja softvera i inspirisan je reinženjeringom apstraktne sintakse jezika Java. Raščlanjivač izdvaja strukturne informacije koje su zajedničke svim objektno-orijentisanim jezicima (hijerarhiju klasa, metode i konstruktore), dok se po izboru mogu serijalizovati i svi izrazi i naredbe, čime se dobija potpuni prikaz izvornog koda u RDF-u. Pored toga, autori primenjuju tehnike semantičke analize poput razrešavanja imenovanih entiteta da bi analizirali komentare unutar koda i povezali entitete izdvojene iz izvornog koda sa odgovarajućim resursima baze DBpedia.

CodeOntology je dostupan u repozitorijumu Zenodo pod licencom Creative Commons Attribution 4.0 [126], čime je omogućena široka ponovna upotreba u istraživačkim i praktičnim projektima. Atzeni i Atzori [124] prikazuju konkretne primere izražajnih SPARQL upita nad izvornim kodom predstavljanim u RDF-u.

Primene i razvoj CodeOntology pristupa

Od svog objavljivanja 2017. godine, pristup CodeOntology proširen je u nekoliko pravaca. Atzeni i Atzori [127] prikazuju primenu ontologije CodeOntology kao posrednog sloja za automatsko prevođenje prirodnog jezika u programski kod, što nagoveštava savremene sisteme za generisanje koda zasnovane na LLM-ovima. Poozhithara, Asuncion i Lagesse [128] koriste CodeOntology za lakše otkrivanje obrazaca projektovanja u izvornom kodu pomoću upita nad njegovim semantičkim prikazom.

Srodan savremeni pravac povezuje modelom vođeno generisanje i velike jezičke modele. He i saradnici [129], u radu „A Hybrid Approach for EMF Code Generation“, koriste EMF i šablonski generator za izradu početne strukture koda, a LLM-ove za dopunjavanje i ispravljanje neimplementiranih metoda. Rad ne koristi CodeOntology, pa se veza sa njim u ovoj disertaciji posmatra kao arhitekturna analogija: formalni model određuje strukturni okvir, dok generativna komponenta dopunjava programsko ponašanje.

Značaj CodeOntology za ovu disertaciju

CodeOntology je značajan jer pokazuje da se struktura programskog koda može predstaviti i ispitivati kao RDF/OWL graf. U BAB metodologiji njegova uloga u trenutnoj implementaciji znatno je uža: BABj koristi termin `woc:hasFullyQualifiedName` za identifikovanje ciljnih tipova u jeziku Java. Generator

ne predstavlja generisani program u RDF-u, ne obilazi ceo graf WoC-a i ne koristi SPARQL kao korak izdvajanja za FreeMarker. CodeOntology zato predstavlja referentni i mogući dopunski pristup za buduću analizu i dvosmernu sledljivost, a ne jedinstveni formalizam predstavljanja u trenutnom lancu PIM-PSM-kod.

Ključna razlika između pristupa CodeOntology i metodologije ove disertacije jeste smer transformacije. CodeOntology modeluje kod naknadno: polazi od postojećeg izvornog koda na jeziku Java i serijalizuje ga u RDF. Metodologija ove disertacije ide u suprotnom smeru: polazi od ontologije kao merodavnog izvora specifikacije, a kod se generiše iz ontologije pomoću formalno definisanog lanca transformacija. Ontološki model u BAB metodologiji nije opis postojećeg koda, već specifikacija iz koje kod proističe. Različiti smerovi, od koda ka ontologiji u pristupu CodeOntology i od ontologije ka kodu u BAB metodologiji, odražavaju suštinsku razliku između obrnutog inženjerstva usmerenog ka semantičkom modelu i razvoja od modela ka kodu.

U narednim potpoglavljima razmatraju se preostale teorijske osnove metodološkog pristupa: teorija kategorija kao matematički formalizam za transformacije (potpoglavljje 2.5) i automatizacija generisanja koda (potpoglavljje 2.6), pre nego što se u potpoglavljju 2.7 odredi mesto ovog istraživanja u odnosu na prikazano stanje oblasti.

2.5 Teorija kategorija kao formalni okvir za transformacije modela

Teorija kategorija predstavlja četvrtu konstitutivnu osnovu metodološkog pristupa ove disertacije, pored paradigme inteligentnih procesno svesnih informacionih sistema (potpoglavljja 2.1 i 2.2), modelom vođenog razvoja softvera (potpoglavljje 2.3) i ontološkog modelovanja (potpoglavljje 2.4). Dok ontologije obezbeđuju semantičku strogost pri predstavljanju modela, teorija kategorija pruža formalni aparat za preslikavanja između modela. Funktori po definiciji čuvaju identitete i kompoziciju, dok svaka jača tvrdnja o očuvanju zahteva izričite pretpostavke i dokaz. Ovo potpoglavljje sistematski prikazuje istorijski razvoj teorije kategorija, njene osnovne pojmove relevantne za softversko inženjerstvo, a posebno primenu teorije kategorija u modelom vođenom inženjerstvu.

2.5.1 Osnovni pojmovi teorije kategorija

Istorijski nastanak i motivacija

Teorija kategorija nastala je 1945. godine u radu Ajlenberga i Mak Lejna [130]. Rad je predat Američkom matematičkom društvu 8. septembra 1942. godine, a primljen u časopis 15. maja 1945. godine i smatra se osnivačkim dokumentom oblasti. U njemu su prvi put formalno definisani pojmovi kategorije, funktora i prirodne transformacije, tri osnovna pojma na kojima je teorija kategorija zasnovana. Autori su bili motivisani konkretnim problemom iz algebarske topologije: kako formalno opisati pojavu da neke matematičke konstrukcije izgledaju „prirodno“ u smislu da su nezavisne od izabranog

načina predstavljanja; primer je odnos između vektorskog prostora i njegovog duala, koji je predmet uvodnog primera u izvornom radu.

Prvih petnaest godina nakon objavljivanja rada iz 1945, pojmovi teorije kategorija prvenstveno su služili kao pogodan jezik, bez statusa samostalne matematičke discipline. Situacija se značajno menja kroz dva ključna teksta: Ajlenberg i Stinrod [131] i Kartan i Ajlenberg [132]. Ovi radovi omogućili su generacijama matematičara da direktno uče topologiju i homološku algebru u kategorijalnom jeziku, čime su kategorijalne tehnike postale sastavni deo savremene matematike.

Standardni savremeni udžbenik teorije kategorija napisao je Mak Lejn [133], Graduate Texts in Mathematics vol. 5, Springer. Prvo izdanje zasnovano je na predavanjima koje je autor održao na Univerzitetu u Čikagu, Australijskom nacionalnom univerzitetu, koledžu Boudin (Bowdoin) i Univerzitetu Tulejn (Tulane) i smatra se *ključnim uvodom u oblast*. Drugo izdanje (1998) dodaje dva poglavlja o aktuelnim temama: simetričnim monoidalnim kategorijama i kategorijama višeg reda. Alternativna savremena referenca usmerena na primene u računarskim naukama jeste Barr i Wells [134].

Pojam kategorije

Kategorija je matematička struktura koja se sastoji od tri komponente. Prvo, kolekcije objekata (engl. objects), koje označavamo sa $Ob(\mathcal{C})$ za kategoriju $\mathcal{C}$. Drugo, za svaki par objekata $A, B \in Ob(\mathcal{C})$, kolekcije morfizama (engl. morphisms), odnosno strelica iz A u B , koje označavamo sa $Hom(A, B)$ ili $\mathcal{C}(A, B)$. Treće, operacije kompozicije (engl. composition), koja svakom paru morfizama $f: A \rightarrow B$ i $g: B \rightarrow C$ pridružuje morfizam $g \circ f: A \rightarrow C$.

Ove tri komponente moraju zadovoljavati dva aksioma. Prvi je asocijativnost kompozicije: za bilo koja tri morfizma $f: A \rightarrow B$, $g: B \rightarrow C$ i $h: C \rightarrow D$ važi $h \circ (g \circ f) = (h \circ g) \circ f$. Drugi je postojanje identitetnih morfizama: za svaki objekat A postoji morfizam $id_A: A \rightarrow A$ koji služi kao neutralni element kompozicije, tako da za svaki morfizam $f: A \rightarrow B$ važi $f \circ id_A = f$ i $id_B \circ f = f$. Ova dva aksioma su minimalna i u isto vreme dovoljna; beskrajno veliki broj matematičkih struktura i različitih oblasti formalnog zaključivanja zadovoljava upravo njih.

Tipični primeri kategorija su: **Set** (objekti su skupovi, morfizmi su funkcije), **Grp** (objekti su grupe, morfizmi su grupni homomorfizmi), **Top** (objekti su topološki prostori, morfizmi su neprekidne funkcije), **Vect_k** (objekti su vektorski prostori nad poljem k , morfizmi su linearna preslikavanja) i **Pos** (objekti su delimično uređeni skupovi, morfizmi su monotone funkcije). Ključni zaključak koji su Ajlenberg i Mak Lejn izveli iz ovog pregleda jeste da su neki strukturni obrasci zajednički ovim različitim kategorijama, što je upravo polazna motivacija kategorijalne apstrakcije.

Pojam funktora

Funktor je matematičko preslikavanje između dve kategorije koje očuvava strukturu. Preciznije, funktor $F: \mathcal{C} \rightarrow \mathcal{D}$ između kategorija $\mathcal{C}$ i $\mathcal{D}$ sastoji se od dve komponente: (1) preslikavanja objekata, koje svakom objektu $A \in Ob(\mathcal{C})$ pridružuje objekat $F(A) \in Ob(\mathcal{D})$, i (2) preslikavanja morfizama, koje svakom morfizmu $f: A \rightarrow B$ u $\mathcal{C}$ pridružuje morfizam $F(f): F(A) \rightarrow F(B)$ u $\mathcal{D}$.

Ova preslikavanja moraju zadovoljiti dva funktorska zakona koji preciziraju smisao očuvanja strukture na nivou kategorije. Prvi zakon je očuvanje kompozicije, $F(g \circ f) = F(g) \circ F(f)$, za bilo koje morfizme koji se mogu komponovati. Drugi zakon je očuvanje identiteta, $F(id_A) = id_{F(A)}$, za svaki objekat A . Ovi aksiomi garantuju očuvanje identiteta i kompozicije morfizama. Očuvanje dodatnih svojstava objekata i morfizama mora se iskazati i dokazati zasebno.

Upravo ova funktorska svojstva čine teoriju kategorija posebno značajnom za modelom vođeni razvoj softvera, što će biti detaljnije obrazloženo u potpoglavlju 2.5.2. Funktorska specifikacija po definiciji čuva kompoziciju morfizama i identitete. Saglasnost izvršne transformacije sa tom specifikacijom proverava se zasebno, testovima ili formalnom verifikacijom; to važi i kada je implementacija napisana imperativno, na primer u QVT-Operational ili jeziku Java. Mark Siman (Mark Seemann) [135] tu razliku pedagoški jasno izražava u seriji mrežnih članaka: „*Da bi neki objekat, na primer, bio funktor, mora poštovati određene jednostavne i intuitivne zakone. Zbog toga su pojmovi teorije kategorija određeniji i manje dvosmisleni od obrazaca projektovanja*“ (prevod autora).

Pojam prirodne transformacije

Treći osnovni pojam, i u izvesnom smislu ključni razlog za osnivanje teorije kategorija, jeste prirodna transformacija. Ako imamo dva funktora $F, G: \mathcal{C} \rightarrow \mathcal{D}$ koji oba preslikavaju iz iste izvorne kategorije u istu ciljnu kategoriju, prirodna transformacija $\eta: F \Rightarrow G$ jeste porodica morfizama $\eta_A: F(A) \rightarrow G(A)$, po jedan morfizam za svaki objekat A u izvornoj kategoriji, koja zadovoljava uslov prirodnosti (engl. naturality condition). Taj uslov zahteva da za svaki morfizam $f: A \rightarrow B$ u $\mathcal{C}$ odgovarajući dijagram u ciljnoj kategoriji komutira, odnosno, formalno zapisano, $\eta_B \circ F(f) = G(f) \circ \eta_A$.

Marquis [136] u pregledu objavljenom u Stanfordovoj enciklopediji filozofije navodi: „*Upravo da bi razjasnili kako se ove veze uspostavljaju i da bi ih međusobno uporedili, Ajlenberg i Mak Lejn su razvili teoriju kategorija*“ (prevod autora). Drugim rečima, pojam prirodne transformacije istorijski je prethodio ostalim osnovnim pojmovima i motivisao nastanak cele teorije; pojam funktora i pojam kategorije bili su formalni preduslovi za definiciju prirodne transformacije, koja je bila prvobitni cilj formalizacije. U kontekstu modelom vođenog razvoja, prirodne transformacije postaju relevantne kada se razmatra sinhronizacija između dva različita lanca transformacija.

Napredniji pojmovi

Pored tri osnovna pojma, za potpunu formalizaciju modelom vođenog razvoja neophodno je razmotriti i nekoliko naprednijih konstrukcija. Univerzalne konstrukcije (engl. *universal constructions*), poput proizvoda (product), koproizvoda (coproduct), pullback-a i pushout-a, obezbeđuju opšte obrasce koji se pojavljuju u raznim specifičnim matematičkim strukturama pod različitim imenima. U kategoriji Set proizvod je Dekartov proizvod skupova, a koproizvod je disjunktna unija; u kategoriji Grp proizvod je direktan proizvod grupa. Kategorijalna definicija apstrahuje tu zajedničku suštinu pomoću univerzalnog svojstva.

1 Izomorfizam u kategoriji jeste morfizam $f: A \rightarrow B$ za koji postoji inverzni morfizam $g: B \rightarrow A$ takav da važi $g \circ f = \text{id}_A$ i $f \circ g = \text{id}_B$. Objekti povezani izomorfizmom strukturno su nerazlučivi sa stanovišta posmatrane kategorije. Ako su $F, G: \mathcal{C} \rightarrow \mathcal{D}$ funktori, prirodna transformacija $\eta: F \Rightarrow G$ jeste prirodni izomorfizam kada je svaka njena komponenta $\eta_X: F(X) \rightarrow G(X)$ izomorfizam u $\mathcal{D}$. Tada su F i G izomorfni kao objekti funktorske kategorije $[\mathcal{C}, \mathcal{D}]$.

59 Posebno značajna konstrukcija za modelom vođeni razvoj jesu adjungovani funktori (engl. *adjoint functors*), koji formalizuju pojam slabe inverzne relacije između dve kategorije. Za funktore $I: \mathcal{C} \rightarrow \mathcal{D}$ i $J: \mathcal{D} \rightarrow \mathcal{C}$, adjunkcija $I \dashv J$ određena je prirodnim izomorfizmom

$$\text{Hom}_{\mathcal{D}}(I(X), Y) \cong \text{Hom}_{\mathcal{C}}(X, J(Y)).$$

63 Odgovarajuće prirodne transformacije $\eta: \text{Id}_{\mathcal{C}} \Rightarrow JI$, jedinica adjunkcije, i $\varepsilon: IJ \Rightarrow \text{Id}_{\mathcal{D}}$, kojedinica adjunkcije, zadovoljavaju trougaone identitete. Mak Lejn u svom udžbeniku poznatim iskazom izražava važnost ovog pojma: „*Adjungovani funktori pojavljuju se svuda*“ (prevod autora). Adjunkcije se pojavljuju kao formalni opis pojmova poput: za svaku povezujuću strukturu postoji najslabija ili najjača takva struktura koja je saglasna sa zadatim ograničenjima. U kontekstu MDE adjunkcije formalizuju odnose između različitih apstrakcionih nivoa modela. U ovoj disertaciji adjunkcija se ne tvrdi za trenutni jednosmerni tok generisanja; ona označava mogući formalni okvir za buduće dvosmerno softversko inženjerstvo.

Za modelovanje softvera posebno su značajni i bifunktori, odnosno funktori sa dve varijable, koji se tehnički definišu kao funktori $\mathcal{C} \times \mathcal{D} \rightarrow \mathcal{E}$. Bifunktori su ključni u formalizaciji tipičnih MDE scenarija u kojima transformacije zavise od dva ulaza, na primer, od izvornog modela i od konfiguracije preslikavanja (što je upravo formalizacija koju koristi metodologija ove disertacije u formalnom okviru iz Poglavlja 4).

2.5.2 Primena teorije kategorija u softverskom inženjerstvu

Rani pristupi: apstraktna specifikacija i semantika

Primena teorije kategorija u teorijskoj računarskoj nauci ima dugu istoriju, koja počinje početkom 1970-ih godina. Pregled ranih pristupa dao je Gogen (Goguen) [137]. Autor je

u tom programskom tekstu izdvojio niz oblasti računarske nauke u kojima se teorija kategorija pokazala korisnom: semantiku programskih jezika, teoriju tipova, specifikacije algebarskih struktura, semantiku paralelnih procesa i formalizaciju deduktivnih sistema. Gogen je posebno isticao prednost kategorijalnog aparata za kompozicionu semantiku, u kojoj se semantika složene strukture određuje na osnovu semantike njenih sastavnih delova i načina njihovog povezivanja.

Sistematski pristup prepoznavanju relevantnih primena za softverske inženjere dao je Isterbruk (Easterbrook) [138] u predavanju [138] (predavački materijal, Univerzitet u Torontu, Odeljenje za računarske nauke). Autor je izdvojio četiri glavne primene teorije kategorija u softverskom inženjerstvu: (1) kategoriju algebarskih specifikacija za predstavljanje kompozicije i profinjavanja, (2) kategoriju specifikacija temporalne logike za izgradnju modularnih specifikacija, (3) teoriju automata za poređenje automata i (4) logiku kao kategoriju za predstavljanje logičkih sistema kao kategorija i konstrukciju dokaza. Iako ovaj materijal nije recenziran u užem akademskom smislu, on služi kao korisna pedagoška polazna tačka za softverske inženjere koji se prvi put susreću sa oblastju.

Knjiga Barra i Wellsa [134] predstavlja sistematski udžbenik kategorijalnih pristupa namenjen računarskim naukama. Autori izričito usmeravaju materijal prema potrebama istraživača i praktičara u računarstvu, pa knjiga pruža neposrednu vezu između apstraktnih kategorijalnih pojmova i njihove primene u toj oblasti.

Kategorijalna formalizacija obrazaca projektovanja

Jedan od novijih pravaca u primeni teorije kategorija u softverskom inženjerstvu odnosi se na formalizaciju obrazaca projektovanja, pojma koji su uveli Gamma, Helm, Johnson i Vlassides [139]. Obrasci projektovanja su u izvornoj formulaciji neformalne pedagoške apstrakcije koje opisuju rešenja uobičajenih problema u objektno-orijentisanom projektovanju pogodna za ponovnu upotrebu. Međutim, njihova neformalna priroda ograničava mogućnost automatske analize i verifikacije.

U seriji članaka [135] Siman sistematski pokazuje kako se pojedini klasični obrasci projektovanja mogu povezati sa opštijim kategorijalnim strukturama, kao što su funktori, monoidi, monade i polugrupe. Milewski [140] daje širi pedagoški prikaz ovih struktura namenjen programerima funkcionalnih jezika. Prednost kategorijalnog tretmana je što su zakoni kojima podležu funktori, monade i srodne strukture matematički strogo formulisani, za razliku od neformalnih opisa obrazaca projektovanja.

Sistemska okvir Kovaljova za softversko inženjerstvo

Sistematsku primenu teorije kategorija u softverskom inženjerstvu dao je Kovaljov [99]. Autor primenjuje teoriju kategorija na problem predstavljanja heterogenih softverskih inženjerskih tehnologija u jedinstvenom obliku pogodnom za njihovu integraciju i koordinaciju u okviru zajedničkog ciklusa razvoja softverskih sistema. Posebna pažnja u

radu posvećena je savremenim pristupima kao što su modelom vođeno inženjerstvo i aspektno-orijentisano programiranje.

Kovaljov konstruiše univerzalne kategorijalno-semantičke modele ovih tehnologija i predlaže novu metodu razdvajanja aspekata, pri čemu izričito iskazuje aspektualnu strukturu formalnih modela programa. Posebno značajan rezultat jeste konstrukcija formalnih tehnologija (što autor naziva arhitektonskim školama) koje čine matematičku osnovu za modelom vođeno inženjerstvo. Centralni pojam u okviru Kovaljova jeste kategorija *c-DESC*, čiji su objekti formalni modeli programa, a morfizmi formalni opisi različitih aktivnosti njihove integracije. Kompozicija morfizama predstavlja izvršavanje višekoračnih procesa, a identitetni morfizmi predstavljaju „izostanak bilo kakve promene“.

Iz kategorije *c-DESC* Kovaljov izvodi funktor $\text{sig}: c\text{-DESC} \rightarrow \text{SIG}$, gde je *SIG* kategorija formalnih modela interfejsa (na primer deklaracije veb servisa u WSDL jeziku). Interfejs je „deo komponente koji je ‘vidljiv’ sistemu tokom integracije“, a profinjenje je „razvojni korak pojedinačne komponente, poput implementacije specifikacije veb servisa u konkretnom programskom jeziku“. Ovaj analitički okvir neposredno je relevantan za metodološki pristup ove disertacije; on pokazuje kako se tipične softverske inženjerske aktivnosti mogu formalizovati kao funktori između kategorija različitih nivoa apstrakcije.

Primena teorije kategorija na probleme softverskog inženjerstva, modelovanja i konceptualnih sistema pripada široj oblasti primenjene teorije kategorija (engl. *Applied Category Theory*, ACT). Ova oblast je tokom poslednje decenije institucionalno oblikovana kroz međunarodne radionice (konferencija *Applied Category Theory* održava se od 2018. godine), specijalizovani časopis *Compositionality*, osnovan 2019. godine, i sistematske preglede. Kategorija se u njoj koristi kao primarni jezik za zaključivanje o *kompozicijskim strukturama*, strukturama u kojima se složeni sistemi grade kombinovanjem jednostavnijih kroz dobro definisane operacije sa formalno garantovanim svojstvima. Fong i Spivak [141] u monografiji [141] sistematski izlažu sedam tematskih okvira primene (od resursnih teorija i baza podataka do dinamičkih sistema i logičkih semantika), pokazujući da je kategorijalni jezik prirodno sredstvo za formalizaciju problema u kojima se različiti formalizmi moraju kompozicijski povezivati.

Metodologija ove disertacije pripada tradiciji ACT-a po dva svojstva. Prvo, kategorijalni aparat koristi se kao primarni formalizam za opis lanca transformacija, a ne kao naknadni metateorijski sloj. Bifunktor F_s sa pripadajućom faktorizacijom predstavlja okvir specifikacije u kojem se tumače konkretni postupak izdvajanja PIM-a, mehanizam razrešavanja PSM-a i FreeMarker šabloni. Drugo, metodologija se bavi kompozicijom heterogenih formalizama: OWL modela, kategorijalnog aparata, unutrašnjih specifikacija artefakata i šablonskog jezika, što je klasa problema za koju Fong i Spivak [141, Poglavlje 1] teoriju kategorija predstavljaju kao prirodan jezik. Disertacija ne tvrdi da uvodi nove kategorijalne konstrukcije; ona ustaljene konstrukcije primenjuje na MDA lanac koji počinje OWL PIM-om.

2.5.3 Teorija kategorija u modelom vođenom inženjerstvu

Programski rad Diskina i Majbauma

Pravac posebno posvećen primeni teorije kategorija u modelom vođenom inženjerstvu uspostavljen je radom Diskina i Majbauma [98]. Autori u uvodu svog rada priznaju „skrivenu intrigu“ u naslovu: teorija kategorija je jedna od najapstraktnijih matematičkih disciplina, povremeno sa nadimkom „apstraktna glupost“ (*abstract nonsense*), dok je MDE noviji pravac u razvoju softvera, podržan u industriji standardima, alatima i statusom „novog srebrnog metka“. Iznenađujuće, kako autori ističu, kategorijalni obrasci pokazuju se neposredno primenljivim na matematičko modelovanje struktura koje se pojavljuju u svakodnevnoj MDE praksi.

Ključna tvrdnja Diskina i Majbauma jeste da se (1) operacije spajanja modela (engl. *model merging*), (2) transformacije modela, (3) sinhronizacije modela i (4) drugi važni scenariji upravljanja modelima mogu posmatrati kao *izvršavanja kategorijalnih specifikacija*. Pored toga, autori obrazlažu da je odnos između teorije kategorija i MDE-a složeniji i bogatiji nego što se obično pretpostavlja u „primenjenoj matematici“; teorija kategorija daje formalnu semantiku za MDE jezike i obezbeđuje skup *obrazaca projektovanja* i strukturnih principa od praktične vrednosti za MDE.

U srži pristupa Diskina i Majbauma jeste ideja da se različiti modeli posmatraju kao *objekti* u pogodnoj kategoriji, a preslikavanja između njih kao *morfizmi*. Transformacija modela (bilo da se radi o transformaciji iz PIM-a u PSM, sinhronizaciji između dva paralelna prikaza istog sistema ili integraciji više delimičnih modela u celovit model) formalizuje se kao funktor između odgovarajućih kategorija. Specifične kategorijalne konstrukcije kao što su *pushout* (za spajanje modela), *pullback* (za nalaženje zajedničkog podmodela) i *kolimit* (za objedinjavanje više modela) sistematski se koriste.

Trostruke grafovske gramatike i kategorijalna osnova

Jedan od konkretnih primera primene formalnih pristupa u MDE jesu trostruke grafovske gramatike (engl. *Triple Graph Grammars*, TGG), formalizam za dvosmernu transformaciju modela koji je razvio Schürr [142], a kasnije produbili Ehrig i saradnici. Anjorin, Schürr i Taentzer [143] razmatraju TGG kao postupak zasnovan na pravilima za zadavanje relacije konzistentnosti između izvornog, korespondentnog i ciljnog domena. Pokazuju kako se globalna ograničenja prevode u negativne uslove primene pravila i dokazuju da je tako izveden skup uslova potreban i dovoljan za usklađenost sa šemom, u granicama formalizma razmatranog u radu.

Savremeni razvoj: kategorijalna šema za softversko inženjerstvo

Nedavni razvoj u oblasti pokazuje kako se teorija kategorija može koristiti kao objedinjujući formalni jezik u sistemskom inženjerstvu. Breiner, Subrahmanian i Sriram [144] predstavljaju njene osnovne pojmove i naglašavaju precizne mehanizme za specifikaciju, transformaciju, kompoziciju i apstrakciju strukturiranih modela. Funktori

pri tome predstavljaju preslikavanja koja čuvaju strukturu, dok prirodne transformacije omogućavaju dosledno povezivanje takvih preslikavanja.

Posebno u pogledu transformacije modela, Vidalie, Batteux, Mhenni i Choley [145] pokazuju kako strukturni modeli koji se koriste u analizi bezbednosti sistema kritičnih po bezbednost (kao što je jezik S2ML) mogu biti formalizovani kao kategorije, a sinhronizacija između različitih prikaza modela istog sistema kao prirodne transformacije između odgovarajućih funktora. Autori izričito formulišu motivaciju za kategorijalni pristup: teorija kategorija poseduje „osobine koje obuhvataju i teoriju skupova i teoriju grafova“, kao i „suštinsko svojstvo kompozicije koje omogućava propagiranje relacija u modelima“ i „svojstva apstrakcije koja su od interesa u poređenju modela“.

Kategorijalna arhitektura za preduzeće zasnovano na modelima

Najnovije primene kategorijalnih pristupa u savremenom softverskom inženjerstvu obuhvataju i domen arhitektura preduzeća zasnovanih na modelima. Rad Mordekaja (Mordecai), Ferbenksa (Fairbanks) i Krolija (Crawley) [146] predstavlja kategorijalnu formulaciju ciklusa koncepta, modela, grafa i pogleda u projektovanju arhitekture sistema. Kategorijalne konstrukcije koriste se za povezivanje jezika za modelovanje, grafovskih prikaza i pogleda namenjenih zainteresovanim stranama.

U istom pravcu ide rad Kovaljova [147]. U njemu se kategorijalne konstrukcije koriste za sastavljanje heterogenih modela u preduzeću zasnovanom na modelima, sa naglaskom na kompoziciju i kolimite kao sredstvo objedinjavanja delimičnih prikaza sistema.

Značaj za metodologiju ove disertacije

Pregled literature o primeni teorije kategorija u modelom vođenom inženjerstvu pokazuje razvijenu istraživačku tradiciju. Za ovu disertaciju posebno su relevantni programski tekstovi Gogena [137] i Isterbruka [138], radovi Diskina i Majbauma [98] i Kovaljova [99], publikacija NIST-a [144] i rad Mordekaja, Ferbenksa i Krolija [146]. Zajedno, ovi izvori čine konceptualni okvir u koji se smešta metodološki doprinos ove disertacije.

Poseban doprinos disertacije u odnosu na postojeću literaturu jeste integracija ontološkog modelovanja i kategorijalne formalizacije u jedinstvenom okviru. Dok postojeća literatura pretežno koristi teoriju kategorija kao formalni jezik za transformacije modela (bez posebnog zahteva da ti modeli budu ontološki), metodologija ove disertacije kategorijalnu formalizaciju primenjuje upravo na modele izražene u OWL 2 DL-u, čime povezuje dve tradicije koje su se do sada razvijale paralelno: semantičku strogost ontoloških formalizama i strukturnu strogost kategorijalnih transformacija.

Formalna razrada ove integracije predstavlja predmet Poglavlja 4 (kategorijalna formalizacija), u kojem se uvodi porodica bifunktora F_s sa pridruženim konformansnim funktorima Φ_s koji zajedno formalizuju mehanizam preslikavanja implementacije (engl. *ImplementationMapping*). Ovaj formalni okvir povezuje dva zahteva: uslovnu, matematički zasnovanu garanciju očuvanja odabranog šematskog dela pod izričitim pretpostavkama verifikacije i semantičku strogost koju obezbeđuje osnova u OWL 2 DL-

u. Pre detaljnog izlaganja tog okvira, međutim, neophodno je razmotriti i poslednju konstitutivnu osnovu, automatizaciju generisanja koda, koja je predmet potpoglavlja 2.6.

2.6 Automatizacija generisanja koda

Automatizacija generisanja koda predstavlja poslednju od ključnih osnova metodološkog pristupa ove disertacije, pored paradigme inteligentnih procesno svesnih informacionih sistema (potpoglavlja 2.1 i 2.2), modelom vođenog razvoja softvera (potpoglavlje 2.3), ontološkog modelovanja (potpoglavlje 2.4) i teorije kategorija (potpoglavlje 2.5). Dok prethodna potpoglavlja pružaju teorijske osnove za modelovanje, predstavljanje znanja i formalizaciju transformacija, ovo potpoglavlje razmatra praktične mehanizme kojima se apstraktni modeli pretvaraju u izvršni kod. Razmatranje je organizovano oko tri pitanja: koje paradigme generisanja koda postoje i kako se klasifikuju, kakva je uloga šablonskog pristupa (TBCG) i odgovarajućih alata i na osnovu čega je u okviru BAB-a izabran FreeMarker kao šablonski mehanizam.

2.6.1 Paradigme generisanja koda u modelom vođenom inženjerstvu

Istorijski kontekst i tipologija transformacija

Automatsko generisanje koda ima dugu istoriju koja seže do ranih programa prevodilaca 1950-ih godina. Sistematski pregled koji daju Syriani, Luhunu i Sahraoui [148] eksplicitno postavlja istorijsku tačku: „*Generisanje koda postoji od pedesetih godina dvadesetog veka, a poreklo vodi od ranih programa prevodilaca*“ (prevod autora). Od tada su se softverske organizacije oslanjale na tehnike sinteze koda radi smanjenja vremena razvoja i povećanja produktivnosti. Automatsko generisanje koda predstavlja opšti pristup u kojem se isti generator može ponovo iskoristiti za proizvodnju različitih artefakata prema promenljivim ulaznim podacima.

U modelom vođenom inženjerstvu razlikuju se dve glavne paradigme transformacija koje su u neposrednoj vezi sa generisanjem koda. Prva je transformacija modela u model (engl. *model-to-model*, M2M), koja preslikava izvorni model u ciljni model u istoj ili drugoj modelarskoj notaciji. Druga je transformacija modela u tekst (engl. *model-to-text*, M2T), koja iz modela neposredno proizvodi tekstualni artefakt (izvorni kod, XML konfiguraciju, dokumentaciju, SQL skriptu ili slično). Sistematski pregled M2M transformacija dali su Czarnecki i Helsen [95], dok su Mens i Van Gorp [97] ponudili opštiju taksonomiju koja obuhvata obe paradigme.

Transformacija modela u tekst kao osnova generisanja koda

U okviru transformacije M2T ponovo se mogu razlikovati dve podvrste. Prva je *pristup zasnovan na programskom obilasku modela* (engl. *visitor-based* ili *code-based*), u kojem se izvorni model obilazi programskim kodom napisanim u jeziku opšte namene, poput Jave ili Pythona, a taj kod neposredno proizvodi ciljni tekstualni artefakt. Druga je *pristup putem šablona* (engl. *template-based*), u kojem se ciljni artefakt određuje kao šablon sa

mestima koja se popunjavaju podacima iz modela. Syriani, Luhunu i Sahraoui [148] na osnovu svoje sistematske mape literature zaključuju: „*Među različitim paradigmama transformacije modela u tekst, šablonsko generisanje koda (TBCG) najzastupljenije je u MDE-u*“ (prevod autora).

Pored paradigmi M2M i M2T postoji i treća grupa, pristupi neposredno zasnovani na programskom interfejsu (API-ju), u kojima alat obezbeđuje programski interfejs pomoću kojeg korisnik piše kod koji korak po korak gradi elemente ciljnog modela ili koda. Eclipse Modeling Framework (EMF), koji su opisali Steinberg, Budinsky, Paternostro i Merks [6], obezbeđuje takav API i istovremeno služi kao infrastruktura za druge generisane alate. EMF generiše Java klase iz Ecore metamodela, koje se zatim koriste u ručnom ili automatskom razvoju.

Sistematski pregled automatskih tehnika generisanja koda

Noviji sistematski pregled u oblasti dali su Alharbi i Alshayeb [149]. Autori su sistematski saželi i kritički razmotrili 76 primarnih studija povezanih sa automatskim generisanjem koda u oblasti softverskog inženjerstva. Izabrane studije analizirane su prema više kriterijuma: paradigmama, tehnikama, tipovima ulaza, međureprezentacijama, alatima za podršku, ciljnim programskim jezicima i metodama validacije. Autori beleže rast pristupa zasnovanih na mašinskom učenju, dok su paradigma model-u-kod i modelom vođene tehnike i dalje najzastupljenije.

Pregled iz ugla nabavke softverskih sistema dali su Klajn, Levinson i Marketi [150] iz Softverskog inženjerskog instituta Univerziteta Carnegie Mellon u tehničkoj noti [150]. Autori posebno ističu da MDE obuhvata više od alata za generisanje koda i da može uticati na ceo životni ciklus sistema, od prikupljanja zahteva do održavanja. U tehničkoj noti predstavljen je i upitnik namenjen stručnjacima za nabavku softvera radi prikupljanja podataka o alatima dobavljača i procene rizika njihove primene.

2.6.2 Šablonsko generisanje koda

Principi i arhitektonski okvir

Šablonsko generisanje koda (engl. *Template-Based Code Generation*, TBCG) jeste tehnika sinteze koja proizvodi kod na osnovu specifikacija visokog nivoa nazvanih *šablonima* (engl. *templates*). Prema Syriani, Luhunu i Sahraoui [148], TBCG je zastupljena tehnika u okviru MDE-a zato što oba pristupa naglašavaju apstrakciju i automatizaciju. Principi apstrakcije, kroz šablone koji opisuju obrasce koda na višem nivou od konkretnih realizacija, i automatizacije, kroz izvršavanje šablonskog mehanizma, čine TBCG i MDE pristupima koji se prirodno dopunjuju.

Sa arhitektonskog stanovišta, šablonski pristup zasniva se na tri komponente. Prva je metamodel, koji određuje strukturu ulaznog modela, odnosno entitete, atribute i relacije koji mogu biti sadržaj konkretnih modela. Druga je šablon, koji opisuje strukturu ciljnog teksta sa mestima za popunjavanje koja upućuju na elemente metamodela. Treća je

šablonski mehanizam (engl. *template engine*), alat koji prihvata konkretan model i šablon kao ulaz i proizvodi izlazni tekst. Prednost ovog pristupa, kako ističu Syriani i saradnici, jeste u tome što se ciljani tekst može neposredno pregledati u šablonu; programer vidi strukturu rezultata u samom šablonu i ne mora da je rekonstruiše iz postupaka koji generišu tekst.

Klasifikacija TBCG tehnika

Syriani, Luhunu i Sahraoui [148] u svom sistematskom pregledu pristupa TBCG uveli su klasifikacionu šemu sa nekoliko važnih dimenzija prema kojima se razmatraju karakteristike tehnika TBCG. Prva dimenzija je stil šablona (engl. *template style*), koji može biti zasnovan na izlazu (engl. *output-based*, kada je šablon blizak ciljnom jeziku i sadrži umetnute programske konstrukcije za dinamički sadržaj) ili zasnovan na postupcima (engl. *procedure-based*, kada je šablon ugnežđen u programsku logiku koja proizvodi izlaz). Autori su utvrdili da je stil zasnovan na izlazu dominantan u savremenim alatima MDE.

Druga dimenzija je tip ulaza, odnosno da li je ulaz modelarski jezik visokog nivoa, poput UML-a, EMF-a ili DSL-a, ili predstava nižeg nivoa, poput XML-a, JSON-a ili običnih struktura podataka. Treća dimenzija je tip izlaza. U velikoj većini slučajeva TBCG se koristi za generisanje izvornog koda, ali i drugih tekstualnih artefakata, poput dokumentacije, skripti za inicijalizaciju baze, konfiguracija ili izveštaja. Četvrta dimenzija je izražajnost šablonskog jezika, koja varira od jednostavnih šablona sa zamenom promenljivih do potpunih programskih jezika za šablone, poput Xtenda, koji ima mnoge karakteristike funkcionalnog programskog jezika.

Poređenje TBCG alata

Luhunu i Syriani [151] porede devet šablonskih alata koji pripadaju različitim tehnološkim pristupima: Acceleo, koji primenjuje standard MOFM2T i povezan je sa okvirom Eclipse Modeling Framework, Xpand, Xtend2, Epsilon Generation Language (EGL), Apache Velocity, JET (Java Emitter Templates), StringTemplate, T4 (Text Template Transformation Toolkit) i XSLT.

Ključni empirijski nalaz Luhunua i Syrianija [151] jeste da su alati zasnovani na modelima bili izražajni za ispitane obrasce metamodela, dok su alati zasnovani na kodu u njihovom eksperimentu uglavnom imali kraće vreme izvršavanja. JET je bio najbrži, a Xtend2 je ponudio najbolji odnos izražajnosti i učinka. Autori upozoravaju da je eksperimentalna postavka mogla pogodovati alatima zasnovanim na kodu, jer su izvršni mehanizmi i ulazni modeli bili predstavljeni u okruženju Jave. Takođe navode da se TBCG oslanja na više od 70 različitih alata, bilo da su zasnovani na modelima ili na programskom kodu.

Zrelost i industrijska prihvaćenost

Sousa, Syriani i Paquin [152] predstavljaju studiju industrijske upotrebe alata MDE pre uspostavljanja saradnje sa istraživačima. Autori opisuju kako su alati povezani sa internim softverom koji se koristi u proizvodnji, sa kojim izazovima su se korisnici susreli i koja su rešenja primenili. Ova studija pruža konkretan dokaz praktične upotrebe, dok širi pregled Syriani, Lahunua i Sahraoui [148] TBCG određuje kao zrelu tehniku koja se i dalje aktivno istražuje i za koju ostaju otvorena pitanja definisanja šablona i procene ispravnosti i kvaliteta generisanog koda.

Konkretan primer pristupa TBCG u oblasti ugrađenih sistema dali su Hu, Duan, Wang, Gao i Shang [153]. Autori razvijaju šablonski postupak za generisanje koda iz AADL modela za različite ciljne platforme i na primeru jedinice za obradu podataka pokazuju da se generisani kod može uspešno prevesti i izvršiti. Rad time potvrđuje da se TBCG primenjuje i u razvoju ugrađenih sistema za rad u realnom vremenu.

2.6.3 FreeMarker kao izabrani šablonski mehanizam

Karakteristike i mesto FreeMarkera među alatima

Apache FreeMarker je šablonski mehanizam za programski jezik Java, odnosno biblioteka koja generiše tekstualni izlaz (HTML veb strane, e-poštu, konfiguracione datoteke, izvorni kod i drugo) na osnovu šablona i promenljivih ulaznih podataka. Alat je dostupan pod licencom Apache 2.0, što omogućava njegovu upotrebu u projektima otvorenog i vlasničkog softvera. Zvanična dokumentacija [154] eksplicitno određuje alat kao šablonski mehanizam (engl. *template engine*) i naglašava da nije reč o jeziku opšte namene. Jezik za pisanje šablona, FreeMarker Template Language (FTL), projektovan je kao specijalizovani jezik usmeren na deklarativni pristup podacima i kontrolu toka, što ga čini pogodnim za generisanje koda u okviru procesa MDE.

Prema razlikovanju alata zasnovanih na modelima i alata zasnovanih na kodu koje koriste Syriani i saradnici [148] i Lahunua i Syriani [151], FreeMarker se u ovoj disertaciji razmatra kao alat zasnovan na kodu: nije izvorno razvijen u zajednici MDE, nema ugrađenu podršku za metamodel EMF-a i oslanja se na objekte koje priprema program na jeziku Java. FreeMarker nije bio među devet alata u eksperimentu Lahunua i Syriani, pa se njihovi rezultati o učinku ne pripisuju neposredno ovom alatu. Za potrebe implementacije okvira BAB korišćena je verzija 2.3.34, pod okriljem fondacije Apache Software Foundation.

Arhitektura i princip rada

Princip rada FreeMarkera, kako ga opisuju zvanična dokumentacija projekta Apache i pregledni članak Baeldunga [155], zasniva se na jasnom razdvajanju podataka od njihovog prikaza. Uobičajeno se programski jezik opšte namene, poput Java, koristi za pripremu podataka, izvršavanje upita nad bazom, poslovne proračune ili čitanje iz spoljnih izvora. FreeMarker zatim obrađuje pripremljene podatke pomoću šablona. U

šablonu se autor usredsređuje na to *kako* prikazati podatke, a izvan šablona na to *šta* prikazati. Ovaj pristup je u savremenoj terminologiji razvoja veb aplikacija poznat kao obrazac MVC (Model-View-Controller), strukturna paradigma koja omogućava razdvajanje posla dizajnera veb stranica (autora HTML-a) od posla programera (programera u Javi).

FTL šabloni sadrže tri vrste konstrukcija. Prvu čine interpolacije, izrazi oblika `${imeVarijable}` koji se zamenjuju vrednošću varijable tokom izvršavanja šablona. Drugu čine direktive, konstrukcije oblika `<#if ...>`, `<#list ...>` i `<#include ...>` koje određuju uslovno izvršavanje, iteracije i uključivanje drugih šablona. Treću čine makroi, korisnički definisane komponente koje se mogu pozivati sa više mesta i podržavaju parametre. Ulaz šablonu se obično obezbeđuje kao `Java Map<String, Object>` ili kao `JavaBean` čija svojstva postaju promenljive u kontekstu šablona.

Ključne funkcionalnosti relevantne za BAB metodologiju

Nekoliko konkretnih karakteristika FreeMarkera posebno je značajno za metodologiju razvoja informacionih sistema koju zahteva okvir BAB. Prva je slobodan izbor izlaznog formata. Za razliku od mehanizama koji su specifični za jedan format, poput JSP-a za HTML ili XSLT-a za XML, FreeMarker tretira svaki izlaz kao običan tekst. To znači da se isti šablonski mehanizam može koristiti za generisanje Java klasa, XML deskriptora, SQL skripti, JavaScript datoteka i bilo koje druge tekstualne predstave. Druga karakteristika je prilagodljivo učitavanje šablona. Alat podržava učitavanje šablona iz različitih izvora, odnosno iz sistema datoteka, JAR arhiva, baze podataka ili sa udaljenih URL adresa, što omogućava dinamičku konfiguraciju generatora.

Treća karakteristika je odsustvo zavisnosti od spoljnih biblioteka. FreeMarker 2.3.x ne zahteva druge biblioteke pored samog izvršnog okruženja Jave, što znatno smanjuje probleme kompatibilnosti sa drugim komponentama tipičnog poslovnog sistema u Javi. Četvrta je podrška za lokalizaciju. Alat podržava formatiranje brojeva i datuma prilagođeno regionalnim podešavanjima, kao i lokalizovane varijante šablona, što je važno za poslovne sisteme koji moraju podržati različita tržišta. Peta su mogućnosti obrade XML-a. FreeMarker može prihvatiti XML DOM strukturu i prolaziti kroz nju, što je značajno za razvoj sistema koji koriste obrasce Saga ili komuniciraju sa spoljnim programskim interfejsima zasnovanim na XML-u. Šesta karakteristika obuhvata ugrađene zaštitne mehanizme, poput blokova `attempt` i `recover`, koji olakšavaju obradu grešaka tokom generisanja.

Obrazloženje izbora FreeMarkera u okviru BAB-a

U kontekstu metodologije koja je predmet ove disertacije, izbor FreeMarkera kao šablonskog mehanizma za implementaciju sintaksne projekcije softverskog okvira BAB zasnovan je na nekoliko formalnih i praktičnih razloga. Prvi je sintaksna jednostavnost načina na koji se FreeMarker koristi u BAB generatoru. Iako FTL podržava uslove, iteracije, makroe i pozivanje funkcija koje mu izloži programski kod, šabloni generatora

dobijaju unapred pripremljen prikazni model i koriste se samo za oblikovanje ciljnog teksta. Sintaksna uloga zato nije ograničenje samog jezika FTL, već arhitektonsko ograničenje nametnuto realizacijom BAB generatora.

Drugi razlog je zrelost ekosistema Jave. Realizacija BABj zasniva se na skupu tehnologija Java, Jakarta EE i Vaadin, a FreeMarker se neposredno povezuje sa okruženjem Jave. Treći razlog je razdvajanje logike od prikaza: PIM ekstraktor i PSM razrešavač grade provereni prikazni model, dok FreeMarker taj model projektuje u sintaksu Jave. Šablon zato ne pristupa grafu Jena i ne donosi ontološke odluke ni odluke o semantici platforme.

Četvrti i metodološki najvažniji razlog jeste ograničenje FreeMarkera na sintaksnu ulogu. Konformanski sloj Φ_s u aktuелnoj implementaciji realizuju PIM ekstraktor i PSM razrešavač BABj-a. Oni iz modela Jena grade nepromenljivi ApplicationSpec, biraju realizacione profile i pripremaju sve vrednosti potrebne za generisanje. FreeMarker dobija samo prikazni model: ne sprovodi ontološko zaključivanje, ne izvršava SPARQL i ne odlučuje koje domenske klase ili relacije postoje. Time su semantičko izdvajanje i tekstualna projekcija jasno razdvojeni i mogu se zasebno proveravati.

Ovakvo svesno ograničenje FreeMarkera na čisto sintaksnu projekciju predstavlja metodološki doprinos. Ono čuva jasnu granicu između semantike, sadržane u ontologiji, i sintakse, sadržane u generisanom kodu. Narušavanje te granice karakteristična je slabost mnogih generatorskih okvira, u kojima se poslovna logika premešta u šablonske skripte. Detaljna razrada ovog argumenta, koja povezuje kategorijalnu formalizaciju iz poglavlja 3 sa inženjerskom implementacijom iz poglavlja 5, predmet je posebne sekcije 5.3.1 disertacije, ključne za obrazlaganje metodološke doslednosti celog pristupa.

Zaključno o izboru FreeMarkera

Iako Luhunu i Syriani [151] utvrđuju da alati specifični za MDE, poput Accelea i EGL-a, nude veću izražajnost za rad sa složenijim obrascima metamodela, poseban metodološki zahtev okvira BAB, prema kojem šablonski sloj treba da bude strogo sintaksno ograničen, a sva semantika da ostane u ontološkom sloju, čini izbor alata zasnovanog na kodu, poput FreeMarkera, doslednim formalnoj arhitekturi. Veća izražajnost alata specifičnih za MDE nije potrebna za tako ograničenu ulogu šablona i mogla bi da podstakne premeštanje dela semantičke logike u šablonske skripte. Slično obrazloženje odnosi se i na alternativne alate poput Velocityja ili StringTemplatea, koji bi načelno takođe mogli da posluže. FreeMarker je odabran zbog zrelosti dokumentacije, stabilnosti API-ja i široke podrške u razvoju poslovnih aplikacija u Javi, a ne zbog isključive metodološke pogodnosti.

Ovim je završen prikaz stanja oblasti koji obuhvata svih pet ključnih osnova metodološkog pristupa disertacije. U narednom potpoglavlju 2.7 objedinjuju se nalazi prethodnih potpoglavlja kako bi se jasno odredilo mesto doprinosa disertacije u odnosu na postojeću literaturu i utvrdila istraživačka praznina koja je podstakla nastanak metodološkog okvira predstavljenog u narednim poglavljima.

2.7 Pozicioniranje disertacije u odnosu na stanje oblasti

Prethodnih šest potpoglavlja, od paradigme PAIS (2.1) i ABPMS (2.2), preko modelom vođenog inženjerstva (2.3), ontološkog modelovanja (2.4), teorije kategorija (2.5) do automatizacije generisanja koda (2.6), pregledana su kao šest temeljnih oblasti na kojima počiva metodološki pristup ove disertacije. U svakom od tih potpoglavlja prikazana je zasebna istraživačka tradicija sa sopstvenim temeljnim radovima, razvijenom istraživačkom zajednicom i zrelim alatima. Ovo potpoglavlje ima drugačiju svrhu: umesto da pregleda novu oblast, ono objedinjuje nalaze svih prethodnih potpoglavlja, utvrđuje ključne istraživačke praznine na spojevima tih oblasti i jasno određuje mesto doprinosa ove disertacije u odnosu na postojeću literaturu.

Izlaganje je organizovano u tri koraka. U prvom (2.7.1) daje se pregled stanja oblasti i utvrđuju se četiri preseka između temeljnih oblasti na kojima su istraživačke praznine posebno vidljive. U drugom (2.7.2) formuliše se pet posebnih doprinosa koje disertacija nastoji da ostvari, svaki sa jasno određenom granicom onoga što jeste i nije rešeno. U trećem (2.7.3) istraživački okvir disertacije razgraničava se od srodnih pristupa koji na prvi pogled mogu izgledati slično, kako bi se izbeglo pogrešno poistovećivanje.

2.7.1 Objedinjavanje stanja oblasti i utvrđene istraživačke praznine

Zrelost posebnih oblasti

Pregled prethodnih potpoglavlja pokazuje da je svaka od šest temeljnih oblasti disertacije kao zasebna disciplina istraživački zrela. PAIS oblast ima stabilnu definiciju od monografije Dumasa, Van der Aalsta i Ter Hofstede [1], koja se i dalje koristi u literaturi. ABPMS paradigma, iako nastala tek 2023. godine kroz manifest Dumasa i saradnika, već ima jasno definisan cilj i pet osobina koje se mogu koristiti kao okvir za vrednovanje. MDE ima dve decenije iskustva, industrijski usvojene standarde i okvire, kao što su MDA i EMF, i obimnu literaturu o ograničenjima i uspešnim obrascima. Ontološko modelovanje, OWL 2 DL i deskriptivna logika SROIQ čine zrelo formalizam sa stabilnim zaključivačima i preko trideset godina teorijskog razvoja. Teorija kategorija, zasnovana 1945. godine, ima više od šezdeset godina primene u teorijskoj računarskoj nauci. Šablonsko generisanje koda dobro je utvrđen pristup u okviru MDE-a, a pregled Syrianija, Luhunua i Sahraouija [148] evidentira 70 alata koji ga podržavaju.

Iz ove zrelosti pojedinačnih oblasti sledi i oblik istraživačke praznine koji je značajan za ovu disertaciju. Praznina nije u tome što bilo koja od temeljnih oblasti ostaje nerazvijena, naprotiv, svaka je bogata teorijski i praktično. Praznina je u tome što povezivanje više ovih oblasti u jedinstvenu metodologiju za razvoj procesno svesnih informacionih sistema još nije dovoljno jasno definisano. Posebno su nedovoljno istraženi spojevi ovih oblasti na kojima se razvija metodološki doprinos ove disertacije.

Četiri posebne istraživačke praznine na spojevima temeljnih oblasti

Prva istraživačka praznina: PAIS i ontološko modelovanje. Formalne ontologije poslovnih procesa (pregled u potpoglavlju 2.4.2) razvijene su pretežno za predstavljanje, praćenje, analizu i poređenje procesnih modela i notacija, prvenstveno BPMN-a u ovde razmatranim radovima Annane i saradnika [37] i Singera [36]. Ove ontologije modeluju *proces*, ali ne i čitavu aplikativnu arhitekturu PAIS-a, koja obuhvata više od samih procesa: korisničke interfejse, modele podataka, poslovna pravila, integracije sa spoljnim sistemima i sigurnosna ograničenja. Obrnuto, u ovde analiziranim izvorima o ontologijama za razvoj informacionih sistema, uključujući ODIS paradigmu Sharmana, Kishorea i Ramesha [156], procesna komponenta nije razrađena kao prvoklasni element. U analiziranom korpusu nije pronađen jedinstven ontološki model koji istovremeno obuhvata procese i celokupnu aplikativnu strukturu PAIS-a.

Druga istraživačka praznina: MDE i ontološko modelovanje. Pregledni radovi Mkhinini, Labbani-Narsis i Nicolle [80] i Husáková i Bureš [55] pokazuju da su veze između UML-a i ontologija istraživane kroz više motiva i tehničkih obrazaca, uključujući ontološke profile, transformacije UML modela u ontologije, transformacije ontologija u UML i ontološku proveru modela. U takvom okruženju ova disertacija ispituje poseban slučaj: *OWL 2 DL ontologiju kao primarni PIM formalizam* i izvođenje transformacionog lanca iz nje bez posrednog UML modela.

Treća istraživačka praznina: teorija kategorija i ontološko modelovanje. Postojeća literatura o primeni teorije kategorija u softverskom inženjerstvu [98, 99, 146, 145] pretežno razmatra modele na opštem nivou, kao objekte u nekoj kategoriji, bez posebnih zahteva u pogledu onoga što ti modeli izražavaju. Obrnuto, literatura o ontološkom modelovanju retko koristi aparat teorije kategorija za formalizaciju transformacija. Iako postoje radovi o kategorijalnoj semantici logika [157], oni se ne primenjuju izričito na MDA lanac transformacija iz OWL ontologije. Praznina se ogleda u *kategorijalnoj formalizaciji MDA lanca transformacija koji počinje OWL 2 DL ontologijom* kao PIM-om.

Četvrta istraživačka praznina: ABPMS i determinističko generisanje. Manifest ABPMS-a [2] formuliše pet ključnih osobina sistema za upravljanje poslovnim procesima unapređenih veštačkom inteligencijom: autonomiju, sposobnost delovanja kroz razgovor, adaptivnost, samousavršavanje i objašnjivost. Postojeći pristupi koji teže realizaciji vizije ABPMS-a najčešće se oslanjaju na statističke komponente, uključujući velike jezičke modele i modele učenja iz toka posla, koje mogu unaprediti fleksibilnost i interakciju sa sistemom, ali same po sebi ne obezbeđuju formalnu garanciju očuvanja strukture, konzistentnosti i ograničenja sistema. S druge strane, deterministički MDE generatori obezbeđuju ponovljivost, kontrolu i formalno sledive transformacije, ali po pravilu nisu projektovani za sposobnost delovanja kroz razgovor, adaptivnost, autonomno delovanje niti samousavršavanje u smislu ABPMS-a. Zbog toga nedostaje metodološka veza između determinističkog, formalno proverljivog generisanja softvera i mehanizama zasnovanih na agentima potrebnih za realizaciju osobina ABPMS-a. U ovoj disertaciji ta

praznina razmatra se povezivanjem ontološkog modela, kategorijalno definisanog transformacionog lanca i dodatnog sloja softverskih agenata nad generisanim kodom.

Sažetak utvrđenih istraživačkih praznina

Navedene četiri istraživačke praznine zajedno određuju istraživački prostor u kojem se smestaju doprinosi disertacije. Nijedna od ovih praznina nije potpuno neistražena; svaka ima delimična rešenja u literaturi i predmet je aktivnog istraživanja. Doprinos ove disertacije nije u otkrivanju jedne od ovih pojedinačnih praznina, već u usklađenom rešavanju više njih istovremeno pomoću jedinstvene metodologije u kojoj jedno rešenje ne narušava drugo. Ovakav objedinjeni pristup relativno je redak u istraživačkoj literaturi upravo zato što zahteva istovremenu stručnost u više različitih istraživačkih tradicija.

2.7.2 Specifični doprinosi disertacije

Na osnovu utvrđenih istraživačkih praznina, doprinos disertacije može se prikazati kroz pet međusobno povezanih naučnih doprinosa. Važno je naglasiti razliku koja se provlači kroz celo istraživanje: metodologija kao univerzalna teorijska konstrukcija razdvaja se od softverskog okvira kao njene konkretne primene. Okvir BAB, zajedno sa realizacijom BABj i dodatnim slojem softverskih agenata, predstavlja konkretnu realizaciju metodologije, dok je sama metodologija apstraktna i prenosiva na druge skupove tehnologija.

Prvi doprinos: ontološki PIM za procesno svesne informacione sisteme

Prvi doprinos je modularni OWL PIM 2.0 za procesno svesne informacione sisteme. BAB Core čini stabilno platformski neutralno jezgro za aplikaciju, entitete, aktere, attribute i neutralne binarne relacije, dok zasebni moduli View, Behavior, Validation, Security, Service, Classification i Capability uvode samo one aspekte koje aplikacija koristi. Ekstenzije uvoze Core, a Core ne uvozi ekstenzije. Apstraktna hijerarhija mapping:ImplementationMapping fizički je izdvojena u BAB Mapping 2.0 rečnik, dok BABj sadrži konkretne izbore Java/Jakarta EE/Vaadin realizacije.

Ograničenja ovog doprinosa su dva. Prvo, BAB Core nije univerzalna ontologija PAIS-a, već je usmeren na tipične obrasce poslovnih aplikacija i njihovih procesa; ne obuhvata specijalizovane domene kao što su kritični sistemi rada u realnom vremenu, finansijski trgovački sistemi ili medicinski ekspertski sistemi. Drugo, doprinos je prvenstveno primenljiv na razvoj novih sistema sa jasnom domenskom specifikacijom; migracija postojećih sistema u ontološki model predstavlja zaseban istraživački problem.

Drugi doprinos: kategorijalna formalizacija MDA lanca iz OWL ontologije

Drugi doprinos je kategorijalna aparatura za MDA lanac koji počinje OWL PIM-om zasnovanim na BAB Core-u. Formalizam najpre izdvaja pune potkategorije $C_{Core}^{NonMapping}$ i $C_{Core}^{MappingOnly}$, a zatim nad preslikavajućim klasama gradi kategoriju dorada C_{Map} . Transformacioni lanac dopunjuju kategorija realizacionih specifikacija C_{BAB}^S i kategorija

generisanih artefakata C_{code}^S . Centralni rezultat odnosi se na kompoziciju $T_s = H_s \circ F_s$ i uslovno očuvanje eksplicitno odabranog šematskog dela. Ova formulacija ne podrazumeva semantičku ekvivalenciju izvršnog programa niti očuvanje podataka na nivou instanci.

Ključna razlika formalnog okvira jeste razdvajanje konformansnog preslikavanja Φ_s i sintaksičke projekcije H_s . U praktičnoj implementaciji ulazni deo realizacije u jeziku Java realizuju komponenta za izdvajanje PIM-a i komponenta za razrešavanje PSM-a u okviru BABj-a, koje model Jena prevode u nepromenljivi `ApplicationSpec` i razrešene specifikacije artefakata. Funktor H_s obuhvata programsku pripremu projektnog izlaza i generisanje teksta pomoću `FreeMarker`a. Konzola `SPARQL` služi pregledu grafa RDF, ali `SPARQL CONSTRUCT` upiti nisu mehanizam generatora. Ova podela omogućava da se odvoje uslovna šematska tvrdnja, potpunost realizacionog ugovora i tehnička izvršivost proizvedenog koda.

Ograničenja ovog doprinosa takođe moraju biti eksplicitno navedena. Prvo, `OWL DL` aksiomi kao što su `owl:inverseOf`, `owl:FunctionalProperty` i `rdfs:subPropertyOf` prvenstveno pripadaju `OWL 2 DL` semantici i ne postaju automatski jednačine kategorijalne prezentacije. Izvorna i kodna kategorija jesu slobodne, dok je ciljna kategorija C_{BAB}^S prezentovana eksplicitno izabranim skupom jednačina E_{BAB}^S . Drugo, prirodnost komponenti $\Phi_s(h)$ jeste deo formalne specifikacije, ali aktuelna implementacija još ne proverava automatski sve pripadajuće jednačine. Treće, Kanove ekstenzije nisu potrebne za korišćenu evaluacionu faktORIZACIJU i ostaju moguće sredstvo za složenije buduće scenarije.

Treći doprinos: reifikovana preslikavanja kao mehanizam platformске nezavisnosti

Treći doprinos je uvođenje reifikovanih preslikavanja kao eksplicitno modelovanih elemenata posebnog rečnika `BAB Mapping`. U matematičkom modelu kategorija C_{Core} posmatra se nad unijom šematskih grafova svih učitanih `BAB PIM 2.0` modula i rečnika `BAB Mapping`; zato se domenski i preslikavajući objekti i dalje izdvajaju punim potkategorijama, iako nisu fizički deklarirani u istoj ontološkoj datoteci. `BABj` sadrži konkretne individue sa ciljnim tipom, ulogom artefakta, šablonom i prioritetom. Formalni izraz je porodica bifunktora F_s sa konformansnim funktorima Φ_s . Promena skupa tehnologija zato ne zahteva promenu domenskog PIM-a, ali u praktičnoj realizaciji zahteva novi PSM profil, razrešavanje platformskih odluka, transformaciju modela u kod i testove.

Ograničenje doprinosa je da prenosivost na druge skupove tehnologija nije potvrđena drugom potpuno implementiranom realizacijom. Aktuelni generator podržava `BABj`, odnosno `Java/Jakarta EE/Vaadin` profil; ostali profili predstavljaju arhitektonsku mogućnost i pravac budućeg rada.

Četvrti doprinos: BAB okvir kao konkretna instanca metodologije

Četvrti doprinos nije metodološki u strogom smislu, već inženjerski: BAB okvir je razvijeni softverski okvir koji pokazuje izvodljivost predložene metodologije kroz konkretnu implementaciju u skupu tehnologija Java, Jakarta EE i Vaadin. Poglavlje 6 razdvaja automatizovanu primenu metodologije, proverenu tokom rada generatora za PostOffice, od njene ručne primene u aplikacijama WasteX, SoilLab i DentaScape. U ručnim primerima projektantski postupak sledio je razdvajanje domenskog modela, realizacionih odluka i programskih slojeva, ali konceptualni PIM i međustanja transformacije nisu tada formalizovani i sačuvani kao verzionisani ontološki artefakti. Zato se za njih može analizirati metodološka i arhitektonska saglasnost, ali se ne može naknadno sertifikovati ceo trag od PIM-a do koda.

Specifično, BAB okvir doprinosi demonstracijom sledećih stvari: (1) OWL 2 DL ontologija se može uspešno koristiti kao PIM za realni poslovni sistem; (2) deterministički lanac transformacija iz ontologije do izvršnog koda u potpunosti je tehnički izvodljiv sa savremenim alatima; (3) generator proizvodi izvršive softverske artefakte koji, pored strukturnih elemenata, uključuju osnovne slojeve aplikacije: perzistentni sloj, poslovnu logiku i komponente korisničkog interfejsa, u meri u kojoj su oni eksplicitno opisani ontološkim modelom i šablonima generisanja; (4) arhitektonski obrasci koji proizlaze iz ontološkog modelovanja kompatibilni su sa industrijski prihvaćenim Java poslovnim tehnologijama.

Sve tvrdnje o praktičnoj izvodljivosti u aktuelnoj verziji odnose se na BABj implementaciju. Eventualni BABp ili drugi profil može služiti samo kao ilustracija planirane prenosivosti dok ne dobije sopstveni PSM ugovor, generator, šablone i provereni izlaz. Ova rezerva je metodološki važna jer odvaja platformski neutralan PIM od empirijski potvrđene višeciljne generacije.

Peti doprinos: delimična realizacija osobina ABPMS-a u sloju softverskih agenata

Peti doprinos je razvoj sloja agenata iznad generisanog koda BAB okvira, koji kroz konkretne komponente delimično operativno realizuje elemente četiri ABPMS osobine iz manifesta Dumas i drugi [2]. Doprinos prevazilazi isključivo arhitektonsku tvrdnju o izvodljivosti jer obuhvata implementirane agente i izvršne mehanizme, ali se ne predstavlja kao potpuna realizacija tih osobina na nivou celog sistema.

Delimična realizacija obuhvata: objašnjivost kroz nadzor aktivnosti agenata i strukturisane izlaze jezičkih modela; adaptivnost kroz dinamičku registraciju i upravljanje agentima tokom izvršavanja; sposobnost delovanja kroz razgovor pomoću interfejsa SPARQL i agenta UserHelpDeskAgent, koji klasifikuje korisničke namere i koristi FIPA-ACL tipove poruka [158]; i autonomiju u zadatim okvirima pomoću agenta SqlErrorHealingAgent, postupka predlaganja ispravke i ograničenog kriterijuma zabranjenih potpisa. Detalji implementacije i ograničenja izloženi su u potpoglavlju 5.5.

Doprinos odgovara na četvrtu utvrđenu istraživačku prazninu, premošćavanje između determinističkog generisanja softvera i mehanizama zasnovanih na agentima potrebnih za realizaciju vizije ABPMS-a. Sloj softverskih agenata koristi generisanu, sledljivu strukturu aplikacije kao osnovu za izvršavanje, dok jasno definisani kriterijumi prihvatanja ograničavaju pojedine predložene radnje. Kategorijalna garancija iz Poglavlja 4 odnosi se na transformaciju odabranog šematskog dela ontologije obuhvaćenog transformacionom specifikacijom i ne proširuje se automatski na ponašanje agenata, ispravnost predloga jezičkog modela ili očuvanje invarijanti tokom izvršavanja.

Implementirano je pet konkretnih agenata koji pokazuju izvodljivost infrastrukture ABPMS-a na ograničenim klasama zadataka. Time su tokom rada sistema realizovani pojedini elementi četiri osobine, ali nisu potvrđeni njihovi puni ishodi na nivou sistema. Izvan obima ostaju sistematsko vrednovanje učinka agenata u produkciji, samousavršavanje zasnovano na povratnim spregama, autonomno prilagođavanje procesnog modela tokom izvršavanja i neprekidno učenje. To su glavni nerealizovani delovi vizije ABPMS-a i pravci daljeg istraživanja iz Poglavlja 8.

Povezivanje doprinosa disertacije sa utvrđenim istraživačkim prazninama sažeto je u Tabeli 2.1.

Doprinos	Odgovara na istraživačke praznine
D1 (BAB Core)	Praznina 1, Praznina 2
D2 (kategorijalna formalizacija)	Praznina 3
D3 (reifikovana preslikavanja)	Praznina 2, Praznina 4
D4 (BAB okvir)	Implicitno svi (validacija)
D5 (sloj softverskih agenata za ABPMS)	Praznina 4

Tabela 2.1. Povezivanje doprinosa sa istraživačkim prazninama.

2.7.3 Razgraničenje u odnosu na srodne pristupe

Razgraničenje od ontoloških profila UML-a

Prvi srodan pristup koji treba jasno razgraničiti jeste OntoUML i drugi ontološki profili UML-a [119]. OntoUML je ontološka nadgradnja UML-a; on UML proširuje disciplinovanim metajezikom konstrukcijama (kinds, roles, phases, mixins) koje nameću određenu ontološku strogost. Metodologija ove disertacije ne nadgrađuje UML; ona zamenjuje UML kao PIM formalizam OWL 2 DL ontologijom. Ova razlika je suštinska, a ne samo terminološka.

Posledica te razlike je da BAB PIM koristi standardno RDF/OWL okruženje: model se može ispitivati SPARQL upitima i proveravati zaključivačem HermiT u podržanom profilu. Aktuelni generator ipak ne koristi SPARQL kao transformacioni jezik; izdvajanje podataka i zatvorene ugovorne provere obavljaju namenske Java komponente.

Razgraničenje od platformi za razvoj uz malo programiranja ili bez programiranja

Važan industrijski srodnik metodologije ove disertacije jesu komercijalne platforme za razvoj uz malo programiranja ili bez programiranja (*low-code* i *no-code*): OutSystems, Mendix, Microsoft Power Platform, Appian¹⁰ i slične. One sa metodologijom BAB-a dele praktični cilj ubrzavanja razvoja poslovnih aplikacija i smanjivanja količine ručno pisanog koda, iako koriste drugačije modele, alate i mehanizme generisanja. Zbog toga predstavljaju važan industrijski kontekst u odnosu na koji se određuje mesto metodologije ove disertacije.

Razlika između metodologije BAB-a i platformi za razvoj uz malo programiranja odnosi se na tri povezane dimenzije. Prva je priroda merodavnog izvora specifikacije: BAB koristi otvoren RDF/OWL PIM nad kojim se mogu primeniti standardni upiti i zaključivanje. Druga je mogućnost uvida u transformaciju: BAB razdvaja izdvajanje PIM-a, BABj razrešavanje i šablone i za njih navodi proverljive ugovore. Treća je vlasništvo nad rezultatom: generator proizvodi standardan Java/Jakarta EE projekat. Generisana aplikacija ipak zavisi od biblioteke BAB navedene u Maven konfiguraciji, pa „nezavisnost” znači vlasništvo nad kodom i standardno izvršno okruženje, a ne odsustvo svih zavisnosti od okvira BAB.

Ova razgraničenja ne treba shvatiti kao tvrdnju da je metodologija BAB-a u svakom slučaju bolja od pristupa sa malo programiranja. Organizacijama bez sopstvenog razvojnog tima, sa relativno jednostavnim potrebama za aplikacijama i bez zahteva za formalnom verifikacijom, takve platforme nude brži put do funkcionalnog sistema. Metodologija BAB-a namenjena je drugačijem području primene: organizacijama sa složenijim domenskim modelima, zahtevima za formalnom konzistentnošću i sledljivošću od specifikacije do koda, potrebom za vlasništvom nad izvornim kodom i nezavisnošću od zatvorenog radnog okruženja, kao i istraživačkim ili regulatornim okruženjima u kojima su formalne osnove i mogućnost uvida u transformacije nužne. Sistem WasteX, predstavljen u poglavlju 6, pokazuje upravo takav kontekst. Reč je o industrijskoj primeni u domenu upravljanja otpadom sa složenim regulatornim zahtevima i potrebom za dugoročnim održavanjem izvornog koda nezavisno od bilo koje vlasničke platforme.

Razgraničenje od MDA transformacija iz UML-a

Drugi srodan pristup su klasične MDA metodologije koje počinju UML PIM-om i transformišu ga kroz QVT ili ATL u PSM pa u kod. BAB metodologija ostaje u okviru MDA paradigme, ali koristi modularni OWL PIM 2.0, apstraktnu hijerarhiju `mapping:ImplementationMapping` u posebnom rečniku preslikavanja i deterministički implementiran transformacioni tok. Njena trenutna realizacija nije SPARQL transformacija: semantičko izdvajanje i razrešavanje PSM-a obavljaju komponente u jeziku Java, a FreeMarker predstavlja završni sloj transformacije modela u tekst.

¹⁰ Zvanične stranice platformi: OutSystems: <https://www.outsystems.com/>; Mendix: <https://www.mendix.com/platform/>; Microsoft Power Platform: <https://www.microsoft.com/en-us/power-platform>; Appian: <https://appian.com/products/platform/low-code>

Razgraničenje od LLM-zasnovanog generisanja koda

Treći srodan pristup u intenzivnom razvoju čine sistemi za generisanje koda zasnovani na velikim jezičkim modelima, kao što su GitHub Copilot, ChatGPT i Cursor¹¹. Oni proizvode kod na osnovu statističkih obrazaca, bez formalne veze sa unapred zadatim modelom koja bi sama po sebi obezbedila njegovu saglasnost sa specifikacijom. BAB metodologija, nasuprot tome, proizvodi kod deterministički iz formalne specifikacije. Funktorska konstrukcija čuva identitete i kompoziciju, dok očuvanje odabranih šematskih razlika važi uslovno, pod pretpostavkama Propozicije 4.2; sintaksna ispravnost i izvršivost konkretnog koda zavise od implementacije generatora, šablona i odgovarajuće provere.

Ovi pristupi nisu međusobno isključujući. Radovi Breita i saradnika [39], Neuhausa [62] i Palagina i saradnika [64] pokazuju različite načine povezivanja formalnih ontoloških modela sa mašinskim učenjem i velikim jezičkim modelima. Metodologija BAB-a može poslužiti kao formalna osnova za takva buduća proširenja, ali njihova realizacija nije predmet ove disertacije.

Razgraničenje od kategorijalnih formalizacija sistemskog inženjerstva

Četvrti srodan pristup su kategorijalne formalizacije sistemskog inženjerstva iz pregleda Breinera, Subrahmaniana i Srirama [144] i rada Mordekaja, Ferbenksa i Krolija [146]. Ovi pristupi koriste teoriju kategorija za formalno predstavljanje, povezivanje i transformisanje strukturiranih modela u sistemskom inženjerstvu i model-zasnovanom preduzeću. Formalizam je sličan (objekti, morfizmi, funktori, prirodne transformacije), ali primena je drugačija: ova disertacija primenjuje kategorijalne konstrukcije na MDA lanac transformacija u razvoju softvera, a ne na razmenu između heterogenih sistemskih modela.

U izvesnoj meri, ova dva pristupa su komplementarna. Kategorijalni okvir ove disertacije fokusira se na vertikalnu transformaciju $PIM \rightarrow PSM \rightarrow \text{kod}$ u okviru jednog sistema; NIST-ov i MIT-ov okvir fokusiraju se na horizontalnu interoperabilnost između različitih modela različitih alata. Integracija ova dva perspektivna smera je moguć pravac budućeg istraživanja.

Razgraničenje od ontološkog modelovanja koda

Peti srodan pristup koji treba jasno razgraničiti jeste CodeOntology (Atzeni-Atzori 2017), pregledan u potpoglavlju 2.4.3. Kao što je tamo objašnjeno, CodeOntology ide u suprotnom smeru od metodologije BAB-a: on počinje sa postojećim izvornim Java kodom i serijalizuje ga u RDF/OWL za naknadnu analizu i izvršavanje upita. Metodologija BAB-a počinje sa ontologijom kao primarnim merodavnim izvorom specifikacije i generiše kod iz ontologije. Ova razlika u smeru, od koda do ontologije kod

¹¹ Zvanične stranice alata: GitHub Copilot: <https://github.com/features/copilot>; ChatGPT: <https://chatgpt.com/>; Cursor: <https://www.cursor.com/>

sistema CodeOntology i od ontologije do koda u metodologiji BAB-a, odražava suštinsku razliku između povratnog i direktnog inženjerstva (*reverse* i *forward engineering*).

Dva pristupa mogu biti komplementarna u budućim hibridnim metodologijama. CodeOntology može biti iskorišćen za analizu postojećih Java sistema i ekstrakciju ontoloških obrazaca iz njih; BAB metodologija može te obrasce zatim koristiti za ubrzan razvoj novih sistema. Ovakav hibridni pristup je eksplicitno predviđen u potpoglavlju 7 (Diskusija) kao smer budućeg istraživanja.

Sažetak poglavlja 2

Ovim je završen pregled stanja oblasti koji obuhvata šest konstitutivnih osnova metodološkog pristupa ove disertacije: procesno svesne informacione sisteme, procesno svesne sisteme upravljanja poslovnim procesima usmerene na autonomno delovanje, modelom vođeni razvoj, ontološko modelovanje, teoriju kategorija i automatizaciju generisanja koda, kao i njihove međusobne odnose. Utvrđene su četiri posebne istraživačke praznine koje zajedno određuju istraživačko područje kojem disertacija pripada; formulisano je pet posebnih doprinosa koje disertacija donosi; a pristup disertacije razgraničen je od šest srodnih istraživačkih pristupa sa kojima bi se mogao pogrešno poistovetiti.

Naredna poglavlja razrađuju formalne i praktične aspekte ovog pozicioniranja: Poglavlje 3 predstavlja modularni BAB PIM 2.0, zaseban rečnik BAB Mapping 2.0 sa hijerarhijom mapping:ImplementationMapping i BABj PSM; Poglavlje 4 uvodi formalizaciju lanca transformacija zasnovanu na teoriji kategorija; Poglavlje 5 opisuje BAB softverski okvir i generator kao realizaciju metodologije; Poglavlje 6 proverava pristup kroz studije slučaja; Poglavlje 7 kritički razmatra rezultate; a Poglavlje 8 iznosi zaključna razmatranja i pravce budućih istraživanja.

3. Ontologije kao osnova za specifikaciju modela

Ontološki sloj metodologije BAB razdvaja platformski nezavisan opis poslovne aplikacije, ugovor preslikavanja i odluke ciljne platforme. Aktuelna implementacija koristi modularni BAB PIM 2.0, zaseban rečnik BAB Mapping 2.0 i BABj 2.0 kao PSM profil za skup tehnologija Java/Jakarta EE/Vaadin. Strukturno jezgro PIM-a ne sadrži JPA, SQL, JPQL, Vaadin rute ni fizička imena. Takve odluke uvode se tek u BABj profilu i realizaciji generatora.

Ova organizacija prati osnovnu ideju MDA pristupa: model poslovnog sistema ne treba opteretiti detaljima konkretnog programskog jezika, biblioteke, korisničkog interfejsa ili sistema za perzistenciju (OMG, 2014a). PIM moduli opisuju nameru i strukturu aplikacije, rečnik preslikavanja opisuje apstraktni ugovor transformacije, a BABj navodi podatke o realizaciji za jedan ciljni skup tehnologija.

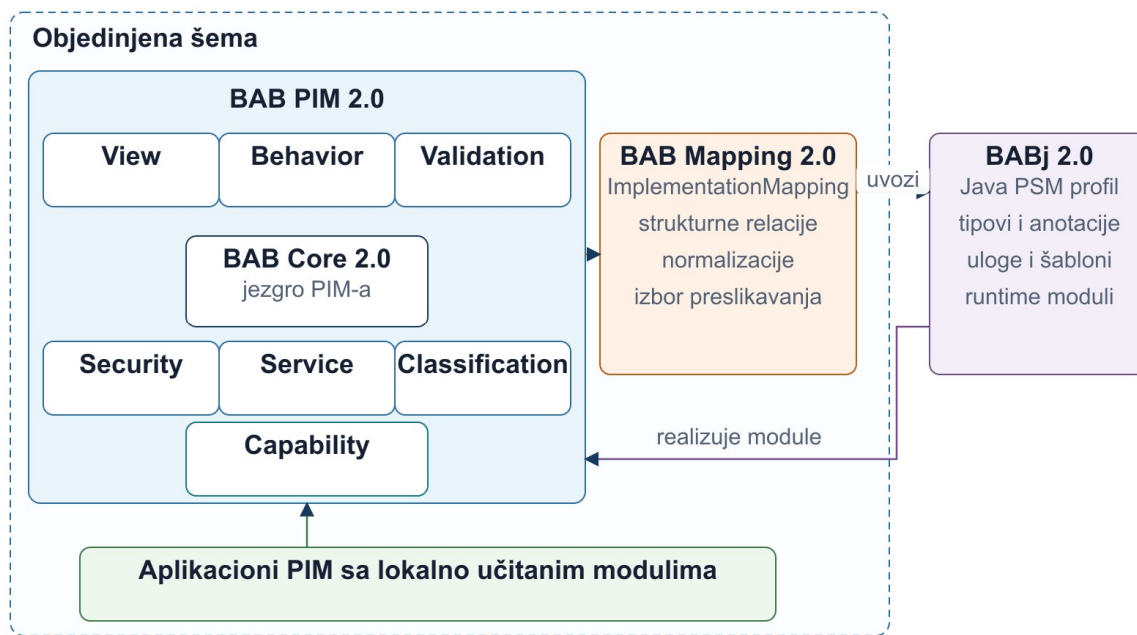
3.1 Arhitektura BAB ontološkog okvira

3.1.1 Modularni BAB PIM 2.0

Aktuelni PIM čine stabilno jezgro i neobavezna proširenja:

1. **BAB Core 2.0** definiše aplikaciju, entitet, aktera, atribut, neutralnu binarnu relaciju, kraj relacije i četiri kardinaliteta;
2. **BAB View 2.0** definiše poglede, forme, projekcije, kolone i funkcije prikaza;
3. **BAB Behavior 2.0** definiše procese, stanja, zadatke, prelaze i redosled izvršavanja;
4. **BAB Validation, Security, Service, Classification i Capability 2.0** uvode validaciona pravila, uloge i dozvole, neutralne servisne namere, kategorije klasifikacije i neobavezne semantičke sposobnosti elemenata modela.

Svako proširenje uvozi BAB Core, dok Core ne uvozi nijedno proširenje. Aplikaciona ontologija uvozi samo module koje koristi. Generator ipak pakuje i proverava sve rečnike, a lokalne deklaracije uvoza aplikacione ontologije razrešava rekurzivno bez mrežnog pristupa. Ovakva modularna organizacija prikazana je na Slici 3.1.



Slika 3.1. Modularna arhitektura BAB ontologija.

3.1.2 Objedinjena kategorijalna interpretacija jezgra

Modularizacija PIM-a omogućava da aplikacija uključi samo one aspekte koji su joj potrebni i da se pojedinačni rečnici razvijaju bez nepotrebnog povezivanja sa svim ostalim modulima. Transformacija, međutim, mora istovremeno da prati veze između domenskih entiteta, procesa, pogleda, pravila i izabranih preslikavanja. Kada bi se svaki fizički rečnik tumačio kao zaseban kategorijalni univerzum, njihove međusobne veze ne bi se mogle neposredno komponovati u jedinstvenom transformacionom lancu. Zbog toga matematička formalizacija objedinjuje učitane šematske grafove, uz očuvanje njihove fizičke i semantičke modularnosti.

Fizička modularnost ontologija ne menja matematički model iz Poglavlja 4. U tom modelu oznaka C_{Core} ne označava samo kategoriju izdvojenu iz datoteke BABCore-2.0.ttl. Ona označava kategoriju izvedenu iz unije šematskih grafova svih učitanih BAB PIM 2.0 modula i BAB Mapping 2.0 rečnika.

Zato se u objedinjenom kategorijalnom univerzumu i dalje razlikuju puna potkategorija domenskih objekata $C_{Core}^{NonMapping}$ i puna potkategorija preslikavajućih objekata $C_{Core}^{MappingOnly}$. Njihova pripadnost istoj kategoriji predstavlja matematičku konstrukciju nad unijom učitanih rečnika, a ne tvrdnju da su svi pojmovi deklarirani u istoj RDF datoteci ili imenskom prostoru.

3.1.3 BAB Mapping 2.0

BAB Mapping 2.0 definiše apstraktni transformacioni ugovor. Korenska klasa `mapping:ImplementationMapping` ima specijalizacije za entitete, atribut, home, forme, poglede, servise, procese i zadatke. Svojstvo `mapping:hasImplementationMapping`

omogućava eksplicitan izbor realizacije individue, dok `mapping:targetsStack`, `mapping:mapsToTypeSpec`, `mapping:emitsAnnotation`, `mapping:implementsTypeSpec`, `mapping:artifactRole`, `mapping:templateName` i `mapping:mappingPriority` opisuju profil realizacije. Klasa `mapping:RuntimeModule` i svojstva `mapping:requiresRuntimeModule`, `mapping:moduleGroupId` i `mapping:moduleArtifactId` opisuju modularne zavisnosti potrebne tokom izvršavanja za ciljni skup tehnologija bez unošenja Maven koordinata u PIM.

Rečnik sadrži `mapping:StructuralRelationMapping` za vezu izabrane relacije sa ciljnom strukturnom oznakom i `mapping:SourceNormalization` za tipiziranu normalizaciju izvornog zapisa. Pravilo normalizacije navodi `canonicalRelation`, `reversedRelation`, `sourceKind` i `targetKind`; inverzni zapis dobija kanonski smer zamenom mesta argumentima. Raniji naziv `NormalizationEquation` zadržan je radi kompatibilnosti i označava ovu normalizaciju, bez poistovećivanja sa proizvoljnom jednačinom kategorijalnih staza. Svojstvo `hasImplementationMapping` ima deklarisan domen `core:ModelElement`. BAB Behavior eksplicitno tipizira inverzne veze procesa, a BAB Security izjednačava dva predikata zahteva za ulogom preko `owl:equivalentProperty`.

Kategorija dorada C_{Map} iz Poglavlja 4 gradi se nad klasama ove hijerarhije i njihovim odnosom specijalizacije. Ona nije ontologija, već kategorijalna konstrukcija. Fizička datoteka BAB Mapping 2.0 jeste rečnik iz koga se izdvajaju njeni objekti i relevantne relacije.

3.1.4 BABj 2.0 kao platformski specifična realizacija

BABj 2.0 uvozi BAB Mapping 2.0. Individua `babj:JavaJakartaVaadin` navodi da realizuje svih osam PIM modula i bira skup generišućih relacija koje transformacija koristi. BABj zatim daje njihove označene strukturne slike, tipizirana pravila normalizacije izvornog zapisa, preslikavanja tipova podataka, osnovne BAB tipove, anotacije, interfejse, uloge artefakata i imena šablona. Isti profil obavezno zahteva modul `bab-core` i sadrži Maven koordinate podržanih BAB 2 modula. Realizacije sposobnosti mogu svojstvom `mapping:requiresRuntimeModule` zahtevati dodatni modul, pa generator iz PIM-a izvodi najmanji potreban skup modula za izvršavanje.

BABj koristi termin `woc:hasFullyQualifiedName` za identifikovanje ciljnih Java tipova, ali ne uvozi niti učitava kompletnu WoC ontologiju kao zaseban rečnik koji generator koristi u radu. Upotreba tog svojstva ne znači da je svaka klasa, metoda, polje i telo metoda generisanog projekta predstavljeno ontološki.

3.2 Konceptualni model PIM-a

3.2.1 Aplikacija, entitet, akter i atribut

`core:BusinessApplication` predstavlja aplikacionu celinu, dok `core:Entity` predstavlja tip domenskog objekta čije se instance pojavljuju tokom izvršavanja sistema. Svojstvo `core:isAbstract` razlikuje apstraktne od konkretnih entiteta. Tek konkretni entiteti mogu neposredno postati artefakti ciljne aplikacije.

Entitet sadrži attribute preko svojstva `core:hasAttribute`, čije je inverzno svojstvo `core:isAttributeOf`. Ime atributa zadato je svojstvom `core:attributeName`, dok se XSD tip navodi svojstvom `core:attributeType`. Standardni XSD tipovi ostaju platformski neutralni, a BABj svojstvom `mapping:realizesDatatype` povezuje podržane tipove sa Java realizacijama.

`core:Actor` je specijalizacija entiteta i predstavlja tip perzistentnog i autentifikovanog operatora aplikacije. Konkretni korisnički nalozi nisu individue PIM-a, već podaci koji nastaju tokom izvršavanja generisane aplikacije. Izabrani BABj profil trenutno zahteva tačno jedan konkretan primarni tip aktera zbog prijavljivanja, sesije i opšteg povezivanja korisničkog interfejsa.

Akter se obrađuje istim postupkom generisanja artefakata `Entity`, `DTO` i `Home`, vođenim sposobnostima, kao i ostali entiteti. Može imati dodatne PIM attribute, relacije i kompatibilne sposobnosti. BABj njegov obavezni operatorski ugovor realizuje interfejsima `OperaterEntityInterface` i `OperaterHomeInterface`, dok posebno ostaje samo povezivanje delova aplikacije za prijavljivanje, sesiju i razvojni početni nalog. Ugovori za agente i Web Push više se ne dodaju svakom akteru implicitno. Sposobnost `AgentAssignable` bira interfejs `HasAgents` i modul `bab-ai`, a `WebPushSubscribable` bira `VapidInterface` i modul `bab-webpush`. Validator odbija sposobnost koja bi duplirala obavezni operatorski član i ne dozvoljava da se sposobnost namenjena akteru dodeli običnom domenskom entitetu.

3.2.2 Neutralni model binarnih relacija

`core:Relation` sadrži tačno dva resursa tipa `core:RelationEnd`. Svaki kraj eksplicitno navodi entitet koji ga deklariše (`core:declaringEntity`), ciljni entitet (`core:targetEntity`), ime uloge (`core:roleName`), suprotni kraj (`core:oppositeEnd`), navigabilnost i jednu od četiri kontrolisane kardinalnosti: `core:ZeroOrOne`, `core:ExactlyOne`, `core:ZeroOrMore` ili `core:OneOrMore`.

PIM ne navodi `OneToOne`, `OneToMany`, `ManyToOne`, `ManyToMany`, `mappedBy`, kolonu spajanja (`join`), tabelu spajanja (`join`), način učitavanja (`fetch`) ni fizičko ime tabele. BABj iz para suprotnih kardinalnosti izvodi JPA vrstu relacije, a iz stabilnih pravila i PSM profila određuje vlasničku stranu i fizička imena. PIM 2.0 dopušta i nenavigabilne krajeve, ali aktuelni BABj profil za perzistentne relacije zahteva da su oba kraja navigabilna. Podržani kardinaliteti prepoznaju se prema punom IRI-ju. Za obavezan kraj

emituje se `@NotNull`, a za kolekciju kardinaliteta `OneOrMore` i `@Size(min = 1)`. Time se čuva deklaracija odgovarajućeg ograničenja; njegovo sprovođenje tokom izvršavanja proverava se zasebno.

3.2.3 Pogledi, forme i projekcije

Izdvajanje pogleda, formi i projekcija u poseban PIM modul motivisano je razlikom između domenske strukture podatka i načina na koji se taj podatak prikazuje ili menja. Isti entitet može učestvovati u više korisničkih scenarija, sa različitim skupovima kolona, oznakama, redosledom i dozvoljenim operacijama. Kada bi se te odluke unosile neposredno u Core entitet, domenski model bi postao vezan za jedan način prezentacije i teže bi se ponovo koristio. BAB View zato opisuje nameru prikaza i potrebne projekcije, ali ne propisuje konkretne Vaadin komponente niti raspored elemenata na ekranu.

Pojmovi prikaza pripadaju modulu BAB View. Svaki generisani pogled eksplicitno upućuje na konkretan domenski entitet i tačno jednu projekciju. Funkcije kreiranja i izmene zahtevaju eksplicitnu formu.

Kolona projekcije može neposredno da uputi na Core atribut svojstvom `view:fromAttribute` ili da opiše putanju preko jednog singularnog kraja relacije svojstvima `view:throughRelationEnd` i `view:subAttribute`. Alternativni naziv (*alias*), oznaka i redni broj čine platformski neutralnu specifikaciju prikaza. Jedan validirani plan projekcije pokreće konstruktor DTO-a, JPQL `select` i `join` izraze i anotacije kolona pogleda. Prolazak kroz koleksijski kraj relacije odbija se dok se ne uvede posebno agregaciono proširenje.

3.2.4 Neobavezni aspekti i ugovori sposobnosti

Modul Behavior opisuje procese, stanja, zadatke i prelaze. Generator iz validnog grafa priprema izvršivi skelet mašine stanja, dok poslovna tela zadataka i uslova ostaju eksplicitne tačke proširenja. Validation uvodi obaveznost, opseg, regularni izraz i jedinstvenost. Security uvodi uloge i dozvole, Service neutralne operacione namere Query, Create, Update i Delete, a Classification uređene kategorije evaluacije.

Modul Capability izražava neobavezne sposobnosti entiteta, pogleda i prozora bez navođenja Java tipova. BABj ih pomoću `mapping:realizesCapability` vezuje za konkretne BAB interfejse, anotacije, obavezna polja i metode. Podržane sposobnosti obuhvataju, između ostalog, aktivnost, naziv, evidenciju izmena, izvršioca, validaciju, isplatu, storno, vremenske i prostorne opsege, količinu, datoteke, generisanu oznaku, lokaciju, REST sinhronizaciju, uloge i barkod, kao i funkcije prikaza i prozora.

Ugovori sposobnosti mogu se kombinovati samo kada su ugovori modularne biblioteke BAB 2 kompatibilni. Validator, na primer, odbija kombinaciju `DateRanged` i `DurationTracked` zbog nekompatibilnih potpisa metode `getDuration()`. `GeneratedCode` zahteva parametar `generatedCodePrefix`, a `FileBacked` zahteva

`fileStorageLocation`. Sposobnosti koje pripadaju samo akteru imaju zaseban tip `ActorCapability`, čime se njihova dozvoljena upotreba proverava pre generisanja.

3.3 BABj 2.0 kao izvršni PSM profil

Kada su domenski pojmovi i apstraktne vrste preslikavanja određeni, generator još nema dovoljno podataka da napravi konkretne programske artefakte. Potrebno je odrediti koje tipove, interfejse, anotacije, module i šablone koristi izabrani skup tehnologija, a da se te odluke ne unesu u PIM. PSM profil predstavlja kontrolisanu vezu između ova dva nivoa. On apstraktnom transformacionom ugovoru pridružuje proverljive realizacione podatke i time omogućava da se isti PIM razreši prema različitim skupovima tehnologija kada za njih postoje odgovarajući profili i generatorska podrška.

BABj 2.0 je takav profil realizacije za Java/Jakarta EE/Vaadin okruženje. Njegova svrha nije da ontološki opiše celokupnu sintaksu programskog jezika niti svaku liniju budućeg izvornog koda. Profil beleži samo odluke koje moraju biti eksplicitne, proverljive i zamenljive pre generisanja. Podaci koji zavise od konkretnog aplikacionog konteksta, kao što su izvedena imena, generički argumenti i vlasništvo nad relacijom, izračunavaju se kasnije u generatoru. Ova podela održava PSM dovoljno izražajnim za automatizaciju, a dovoljno ograničenim da ostane pregledan i deterministički razrešiv.

3.3.1 Deklaracija ciljnog skupa tehnologija

`babj:JavaJakartaVaadin` predstavlja izabrani skup tehnologija i svojstvom `mapping:realizesModule` navodi `Core`, `View`, `Behavior`, `Validation`, `Security`, `Service`, `Classification` i `Capability`. Svojstvom `mapping:selectsGeneratingRelation` bira relacije šeme koje ulaze u sertifikovanu transformacionu specifikaciju, a svojstvom `mapping:requiresRuntimeModule` određuje `bab-core` kao obaveznu osnovu svakog generisanog projekta. Konkretne realizacije sposobnosti istim svojstvom dodaju neobavezne module `bab-ai`, `bab-webpush` i `bab-webauthn` samo kada su odgovarajuće sposobnosti izabrane u PIM-u.

BABj nije celovita ontologija Java sintakse niti ontološka kopija BAB biblioteke. Njegova uloga je da pruži realizacioni opis koji generator može da proveri i upotrebi.

3.3.2 Tipovi podataka i BAB ugovori

BABj preslikava standardne XSD tipove na Java tipove i identifikuje osnovne tipove `AbstractEntity`, `AbstractHome`, `GenericView`, `GenericWindow`, `JAX-RS Application`, `EJB Stateless`, `Java Runnable` i potrebne anotacije. Potpuno kvalifikovana imena zapisana su terminom `woc:hasFullyQualifiedName`.

PSM profil ne određuje sam celokupan sadržaj Java datoteke. Generator izračunava imena klasa, paketa, tabela i kolona, putanje ruta, vlasničke strane relacija, generičke argumente i druge kontekstualne podatke, a zatim priprema nepromenljivu strukturu podataka koja se prosleđuje šablonima.

3.3.3 Profili preslikavanja i uloge artefakata

Podrazumevane BABj individue pokrivaju entitet, perzistentni entitet, atribut, home, formu, pogled, servis, proces i korisnički i servisni zadatak. Svaka navodi ciljni skup tehnologija, specifikaciju ciljnog tipa, ulogu artefakta, šablon i prioritet, a po potrebi i anotacije ili interfejse.

Operatorski ugovor aktera predstavljen je Core tipom Actor i realizuje se istim mehanizmom sposobnosti koji se koristi za ostale neobavezne ugovore. Zajednička entitetska preslikavanja primenjuju se na njegove Entity, DTO i Home artefakte.

3.3.4 Izbor i objedinjavanje preslikavanja

Aplikacioni element može svojstvom `mapping:hasImplementationMapping` eksplicitno da uputi na jednu ili više BABj realizacionih individua. Generator kandidate filtrira prema ulozi artefakta i ciljnom skupu tehnologija, a zatim ih stabilno uređuje prema prioritetu i IRI-ju. Kada eksplicitna veza nije navedena, koristi se odgovarajuće podrazumevano BABj preslikavanje.

Više kompatibilnih preslikavanja i realizacija sposobnosti može doprineti različitim aspektima istog artefakta. `PsmResolver` i `CapabilityCodeGenerator` objedinjuju njihove podatke u deterministički profil koji predstavlja ulaz za FreeMarker.

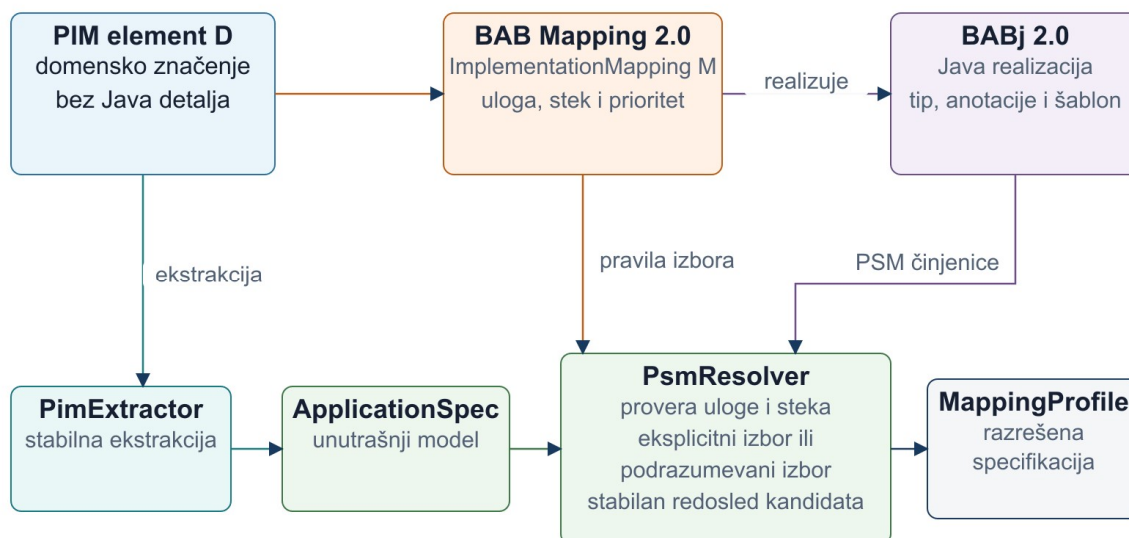
3.4 Veza PIM-a, preslikavanja i PSM-a

3.4.1 Tri odgovornosti u fizički odvojenim rečnicima

Razdvajanje odgovornosti između PIM modula, rečnika BAB Mapping i rečnika BABj nije samo način organizovanja ontoloških datoteka. Njime se uspostavlja jasna granica između opisa poslovnog domena, apstraktnih pravila preslikavanja i konkretnih odluka vezanih za izabrani skup tehnologija. Zahvaljujući toj granici, domenski model može ostati nezavisan od programskog jezika i biblioteka, dok se realizacioni detalji mogu menjati ili proširivati bez unošenja platformskih pojmova u PIM.

Transformacija započinje elementom PIM-a, na primer entitetom, atributom, servisom ili procesnim zadatkom. BAB Mapping određuje kakva je realizaciona strategija dopuštena za taj element, dok BABj obezbeđuje njenu konkretnu realizaciju za Java/Jakarta EE/Vaadin okruženje. Generator tokom razrešavanja povezuje ova tri nivoa, primenjuje eksplicitno izabrano ili podrazumevano preslikavanje i formira specifikaciju ciljnog artefakta. Odnos ovih odgovornosti prikazan je na Slici 3.2 i može se sažeti izrazom:

PIM element + strategija preslikavanja + BABj realizacija → specifikacija artefakta.



Slika 3.2. Veza PIM pojma, BAB Mapping profila i BABj realizacije.

Rezultat ovog povezivanja nije nova monolitna ontologija, već razrešena specifikacija koju generator gradi za konkretni ciljani skup tehnologija. Time se čuva poreklo svake odluke: domensko značenje potiče iz PIM-a, transformacioni ugovor iz BAB Mapping-a, a izvršna realizacija iz BABj-a.

Fizička odvojenost ne menja konstrukciju iz Poglavlja 4. Tamo se C_{Core} izvodi iz unije šematskih grafova svih učitanih PIM modula i BAB Mapping rečnika, pa njegove pune potkategorije i poprečne relacije ostaju definisane u istom objedinjenom kategorijalnom univerzumu.

3.4.2 Granice platformске nezavisnosti

Platformska nezavisnost znači da PIM moduli ne sadrže Java, JPA, Vaadin, HTTP ili fizičke SQL odluke. Ona ne znači da se proizvoljan novi skup tehnologija dobija samo dodavanjem ontološke datoteke.

Za novi skup tehnologija potrebno je obezbediti realizacije modula i ugovora sposobnosti, strukturne slike generišućih relacija, pravila normalizacije i eventualne jednačine ciljne prezentacije, preslikavanja tipova, transformaciju modela u kod, šablone, validatore i testove ciljnog projekta. Aktuelna implementacija potvrđuje samo realizaciju Java/Jakarta EE/Vaadin.

3.5 Validacija ontološkog i realizacionog ugovora

3.5.1 Granica OWL zaključivanja

HermiT se primenjuje na aplikacioni PIM i učitane BAB PIM module. Pre rezonovanja uklanjaju se deklaracije uvoza, jer su odgovarajući grafovi već lokalno učitani. Iskazi iz BAB Mapping, BABj i WoC imenskog prostora isključuju se iz rezonovanog prikaza,

pošto izvršni PSM graf koristi metamodeliranje koje ne pripada semantičkoj granici aplikacionog PIM-a.

Zasebna ugovorna provera zato proverava PSM rečnike, podrazumevana preslikavanja, PIM module koje skup tehnologija realizuje, deklaracije BAB 2 modula potrebnih tokom izvršavanja, jedinstvenost Maven koordinata, obavezni bab-core, tipove podataka, realizacije sposobnosti, izabrane generišuće relacije i tipizirana pravila normalizacije. Uspešna HeriT provera sama ne potvrđuje da je transformacioni ugovor potpun.

3.5.2 Provere zatvorenog sveta

OWL koristi pretpostavku otvorenog sveta, pa odsustvo podatka ne znači da je iskaz netačan. Generator, međutim, mora odbiti nepotpun ulaz pre pisanja datoteka. Namenski Java validatori proveravaju, između ostalog:

- postojanje tačno jedne poslovne aplikacije i tačno jednog konkretnog primarnog aktera za izabrani BABj profil;
- ispravnost i jednoznačnost imena i Java identifikatora;
- pripadnost atributa entitetima i postojanje podržanog tipa;
- tačno dva potpuna, suprotna i za BABj navigabilna kraja binarne relacije;
- ispravnost projekcionih kolona, formi i relacionih putanja;
- konzistentnost pogleda, uloga, servisnih namera, procesa i klasifikacija;
- kompatibilnost ugovora sposobnosti i prisustvo njihovih obaveznih parametara;
- pripadnost eksplicitnih preslikavanja odgovarajućoj hijerarhiji, ulozi i ciljnom skupu tehnologija.

Te provere nisu zamena za OWL rezonovanje. HeriT proverava semantičku konzistentnost, dok zatvoreni validatori proveravaju izvršivost konkretnog generatorskog ugovora.

3.5.3 TBox i ABox način modelovanja

Zajednička pravila prepoznavanja podržavaju elemente deklarisanе kao instance odgovarajućih PIM klasa i aplikacione OWL klase koje specijalizuju PIM tipove. Oba oblika normalizuju se u isti nepromenljiv ApplicationSpec sa stabilnim redosledom elemenata. Aktuelni referentni primeri prvenstveno koriste TBox način modelovanja za entitetske tipove, uz ABox resurse za deskriptore, attribute i druge elemente modela.

Ova tehnička podrška ne proširuje granicu formalne tvrdnje na očuvanje poslovnih instanci. Konkretni korisnici, domenski zapisi i operativni događaji nastaju u ciljnom sistemu i nisu obuhvaćeni šematskim sertifikatom transformacije.

3.5.4 Značenje uspešne validacije

Uspešna validacija znači da je PIM logički konzistentan u proveravanom OWL fragmentu, da ulaz ispunjava generatorski ugovor utvrđen proverama zatvorenog sveta i da BABj sadrži realizacione podatke potrebne za izabrani model. Sertifikat transformacije

je dodatna, odvojena provera ugrađene transformacione specifikacije i pokreće se na zahtev.

Nijedan od ovih nivoa sam ne dokazuje poslovnu ispravnost aplikacije, korektnost korisnički dopisanih tela zadataka, konfiguraciju za puštanje u rad ili semantičku ekvivalentnost izvršavanja.

3.6 Sažetak i prelaz na kategorijalnu formalizaciju

Aktuelni ontološki okvir čine modularni BAB PIM 2.0, BAB Mapping 2.0 i BABj 2.0. Core je neutralno strukturno jezgro, neobavezni PIM moduli izdvajaju pojedinačne aspekte, rečnik preslikavanja čuva apstraktni i sertifikacioni ugovor, a BABj sadrži realizacije za skup tehnologija Java/Jakarta EE/Vaadin.

U matematičkom modelu iz narednog poglavlja oznaka C_{Core} predstavlja kategoriju nad unijom šematskih grafova svih učitanih PIM modula i BAB Mapping rečnika. Ona sadrži domensku i preslikavajuću punu potkategoriju, dok kategorija dorada C_{Map} , bifunktor F_S , funktor H_S i kompozicija T_S određuju transformacioni aparat.

4. Teorija kategorija kao osnova za transformacije modela

Formalizacija u ovom poglavlju preuzima kategorijalni aparat iz rada Bogdanović i saradnici [15], koji predstavlja neposrednu teorijsku osnovu disertacije, i povezuje ga sa sadašnjom implementacijom generatora. Njegovi standardni kategorijalni pojmovi provereni su prema Mac Laneu [133] i Barru i Wellsu [134], dok primenu teorije kategorija u transformacijama modela i projektovanju softverskih sistema nezavisno obrazlažu Diskin i Maibaum [98] i Kovalyov [99]. Matematička konstrukcija se ne menja zbog podele ontologija na zasebne module. Kategorija C_{Core} posmatra se nad unijom šematskih elemenata svih učitanih modula BAB PIM 2.0 i zasebnog rečnika BAB Mapping 2.0. Domenski i preslikavajući objekti zato se i dalje izdvajaju kao dve pune potkategorije iste kategorije C_{Core} . Nad preslikavajućim klasama zatim se uvodi kategorijalna konstrukcija C_{Map} , čiji morfizmi predstavljaju dorade određene hijerarhijom `rdfs:subClassOf`. Zasebnost ontologije BAB Mapping ne menja ovu objedinjenu kategorijalnu interpretaciju.

Formalni rezultat odnosi se na šematsku strukturu transformacionog lanca. Funktorska svojstva slede iz konstrukcije kada su preslikavanja dobro definisana, dok vernost i injektivnost na objektima važe samo pod eksplicitnim uslovima verifikacije. Ne tvrde se očuvanje podataka na nivou instanci, semantička ekvivalencija ontologije i programa niti mogućnost rekonstrukcije ontologije iz generisanog koda.

Iako se posredna kategorija može izostaviti iz zapisa kompozitnog funktora $T_s = H_s \circ F_s$, kategorija C_{BAB}^s ne može se izostaviti iz formalnog aparata. Na njenom nivou definišu se bifunktor F_s , parametrizovan ciljnim skupom tehnologija, jednačine prezentacije E_{BAB}^s i prirodne transformacije dorada, a kroz nju se T_s faktoriše. Zbog toga ona ostaje neophodna za specifikaciju, dokaz dobro definisanosti i verifikaciju transformacije.

4.1 Konstrukcija kategorija iz ontoloških skeleta

4.1.1 Ontološki skelet i slobodna kategorija

Neka je O OWL ontologija, $Cls(O)$ skup njenih imenovanih klasa, a $Prop(O)$ skup imenovanih objektnih svojstava koja se koriste u posmatranoj transformaciji.

Definicija 4.1 (ontološki skelet). Ontološki skelet $G(O)$ jeste usmereni multigraf čiji su čvorovi imenovane klase iz $Cls(O)$. Svako izabrano objektno svojstvo p , sa deklarisanim klasnim domenom D_p i opsegom R_p , određuje generišuću granu

$$e_p: D_p \rightarrow R_p,$$

označenu IRI-jem svojstva p . Kada domen predstavlja imenovanu uniju dopuštenih klasa, transformaciona specifikacija razmatra odgovarajuću granu za svaki dopušteni imenovani domen. Svojstva tipova podataka i anotaciona svojstva nose vrednosti, ali ne generišu navigacione šematske morfizme. Dva različita objektna svojstva sa istim domenom i opsegom ostaju dve različite paralelne grane.

Definicija 4.2 (slobodna kategorija). Za usmereni multigraf G , slobodna kategorija $\text{Free}(G)$ određena je na sledeći način [133, Poglavlje II.7]:

- objekti su čvorovi grafa: $\text{Ob}(\text{Free}(G)) = V(G)$;
- morfizmi su konačne, uključujući i prazne, sekvence kompozabilnih grana;
- identitet id_X jeste prazna staza u čvoru X ;
- kompozicija je nadovezivanje staza.

Asocijativnost i zakoni identiteta važe po konstrukciji. Prirodno utapanje $\iota_G: G \rightarrow \text{Free}(G)$ svakom čvoru pridružuje istoimeni objekat, a svakoj grani jednočlanu stazu.

Lema 4.1 (univerzalno svojstvo). Za svaki morfizam grafova $\varphi: G \rightarrow U(\mathcal{D})$, gde $U(\mathcal{D})$ označava graf objekata i morfizama kategorije $\mathcal{D}$, postoji jedinstven funktor $F_\varphi: \text{Free}(G) \rightarrow \mathcal{D}$ takav da je $\varphi = U(F_\varphi) \circ \iota_G$. Dovoljno je, dakle, zadati sliku čvorova i generišućih grana; slika proizvoljne staze jednoznačno se dobija kompozicijom.

4.1.2 Objedinjena kategorija jezgra i njene pune potkategorije

Neka O_{Core} označava uniju šematskih elemenata svih učitanih ontologija BAB PIM 2.0, BAB Core, View, Behavior, Validation, Security, Service, Classification i Capability, zajedno sa rečnikom BAB Mapping 2.0. Neka G_{Core} označava usmereni multigraf izdvojen iz šematskog sloja te unije. Njegovi čvorovi su deklarirane OWL klase, a generišuće grane izabrana objektna svojstva, kao što su `core:hasAttribute`, `core:hasRelation`, `behavior:hasProcess` i `mapping:hasImplementationMapping`. Definiše se

$$C_{\text{Core}} := \text{Free}(G_{\text{Core}}).$$

Objekti kategorije C_{Core} obuhvataju i domenske klase i klase preslikavanja. Zbog toga se njen skup objekata razlaže na dva disjunktna skupa:

$$\text{Ob}(C_{\text{Core}}^{\text{NonMapping}}) = \{X \in \text{Ob}(C_{\text{Core}}) \mid X \text{ nije mapping:ImplementationMapping niti njena potklasa}\},$$

$$\text{Ob}(C_{\text{Core}}^{\text{MappingOnly}}) = \{X \in \text{Ob}(C_{\text{Core}}) \mid X = \text{mapping:ImplementationMapping ili je njena potklasa}\}.$$

Ovi skupovi određuju pune potkategorije

$$C_{\text{Core}}^{\text{NonMapping}} \hookrightarrow C_{\text{Core}}, C_{\text{Core}}^{\text{MappingOnly}} \hookrightarrow C_{\text{Core}}.$$

Punost znači da svaka potkategorija sadrži sve morfizme kategorije C_{Core} između svojih objekata. Kategorijalni aksiomi se nasleđuju automatski.

U objedinjenom skeletu generišuće grane svojstva `mapping:hasImplementationMapping` usmerene su od domenskih objekata ka `mapping:ImplementationMapping`. Nijedna generišuća grana nije usmerena iz preslikavajućeg dela nazad ka domenskom delu. Zbog toga staza između dva objekta iz

22

22

$C_{Core}^{NonMapping}$ ne može proći kroz preslikavajući objekat, a staza između dva objekta iz $C_{Core}^{MappingOnly}$ ne može napustiti preslikavajući deo i vratiti se u njega. Otuda su obe pune potkategorije ujedno slobodno generisane svojim indukovanim grafovima:

$$C_{Core}^{NonMapping} = Free(G_{Core}^{NonMapping}), C_{Core}^{MappingOnly} = Free(G_{Core}^{MappingOnly}).$$

Zapažanje 4.1 (uloga mapping:hasImplementationMapping). Morfizmi generisani svojstvom `mapping:hasImplementationMapping` pripadaju celom C_{Core} , ali ne pripadaju nijednoj od navedene dve pune potkategorije, jer povezuju njihove objekte. Na nivou šeme predstavljaju prelaz od domenskog pojma ka apstraktnoj vrsti preslikavanja. ABox tvrdnja

$D \text{ mapping:hasImplementationMapping } \mu$

određuje arhitektonski značajan par (D, M) , gde je M tip individue μ . Ove poprečne grane ne pripadaju proizvodu kategorija koji predstavlja domen bifunktoru, ali određuju parove na kojima transformacija ima nameravano arhitektonsko značenje.

4.1.3 Kategorija dorada preslikavanja

Puna potkategorija $C_{Core}^{MappingOnly}$ sadrži morfizme objedinjenog skeleta između preslikavajućih objekata. Međutim, dorada strategija u radu se ne zasniva na običnim objektnim svojstvima, već na hijerarhiji `rdfs:subClassOf`. Zbog toga se nad istim skupom objekata uvodi druga kategorija.

Definicija 4.3 (kategorija C_{Map}). Objekti kategorije C_{Map} jesu `mapping:ImplementationMapping` i sve njene potklase. Za $M, M' \in Ob(C_{Map})$ postoji najviše jedan morfizam

$$h: M \rightarrow M'$$

i on postoji tačno kada je M' specijalizacija klase M , odnosno kada važi

$$M' \text{rdfs:subClassOf}^* M.$$

Strelice su usmerene od opštije ka specifičnijoj strategiji. Identiteti slede iz refleksivnosti, a kompozicija iz tranzitivnosti relacije potklase. Prema tome, C_{Map} je tanka, odnosno preduređena kategorija. Ona ima isti skup objekata kao $C_{Core}^{MappingOnly}$, ali u opštem slučaju nema isti skup morfizama. Ova razlika je suštinska: $C_{Core}^{MappingOnly}$ jeste puna potkategorija C_{Core} , dok C_{Map} formalizuje isključivo poredak dorada.

4.1.4 Prezentovana kategorija realizacionog potpisa

Neka $s \in S$ označava ciljni skup tehnologija, O_{BAB}^s njegovu ontologiju specifičnu za platformu, a V^s rečnik termina kojima profil opisuje strukturu koda. Iz njihovog šematskog sloja izdvaja se usmereni multigraf G_{BAB}^s . Njegovi čvorovi predstavljaju tipove

i realizacione obrasce specifične za ciljni skup tehnologija, a grane izabrane strukturne, navigacione i doradne relacije ciljnog potpisa.

Realizacioni obrazac određuje kako se domenski pojam određene vrste, zajedno sa izabranom klasom preslikavanja, može predstaviti u ciljnom skupu tehnologija. Obrazac može obuhvatiti ciljni tip, anotacije, nasleđivanje, kompoziciju, interfejse, ulogu šablona, deklaracije uvoza, polja i metode. Konkretno klase aplikacije nisu unapred objekti grafa G_{BAB}^S ; one ulaze kada konformansni funktor primeni realizacioni obrazac na domenski objekat.

Neka je E_{BAB}^S eksplicitno zadat skup parova paralelnih staza (u, v) u $\text{Free}(G_{BAB}^S)$. On može sadržati:

1. jednačine konformnosti, naročito jednačine prirodnosti iz potpoglavlja 4.2.2;
2. jednačine koherencije dorade, na primer izjednačavanje neposredne dorade sa odgovarajućom složenom stazom;
3. izričito predviđene normalizacije, kada se u ciljnom skupu tehnologija dve alternativne strukturne staze namerno tretiraju kao isti realizacioni obrazac.

Treća vrsta je neobavezna i mora biti izričito deklarirana. Ona nije automatski prevod proizvoljnih OWL aksioma niti tvrdnja o semantičkoj ekvivalenciji programa. Neka je $\equiv_E$ najmanja kongruencija nad $\text{Free}(G_{BAB}^S)$ koja sadrži E_{BAB}^S i kompatibilna je sa kompozicijom.

Definicija 4.4 (kategorija C_{BAB}^S). Realizaciona kategorija specifična za ciljni skup tehnologija jeste prezentovana kategorija određena generatorima i relacijama [134]:

$$C_{BAB}^S = \text{Free}(G_{BAB}^S) / \equiv_E = \langle G_{BAB}^S \mid E_{BAB}^S \rangle.$$

Njeni morfizmi su klase ekvivalencije $[u]$ staza, a kompozicija je $[v] \circ [u] = [v \circ u]$. Konstrukcija je dobro definisana jer je $\equiv_E$ kongruencija.

Ova konstrukcija nije kružna. Najpre se na nivou grafa zadaju realizacioni obrasci, slike generišućih grana i staze koje predstavljaju dorade u $\text{Free}(G_{BAB}^S)$. Obe strane svake jednačine prirodnosti tada su već određene kao paralelne staze slobodne kategorije. Te jednačine se potom uključuju u E_{BAB}^S , a kvocijent po $\equiv_E$ obezbeđuje njihovu jednakost u C_{BAB}^S . Funktori ka kvocijentu i iz njega smatraju se dobro definisanim tek nakon provere odgovarajuće kompatibilnosti sa ovom kongruencijom.

Za slučaj Jave, $s = j$, ontologija O_{BAB}^j jeste BABj 2.0. Ona uvozi BAB Mapping 2.0 i za potpuno kvalifikovano ime ciljnog tipa koristi termin `woc:hasFullyQualifiedName`, ali sadašnji profil ne učitava celu WoC ontologiju. Formalni ciljni graf može sadržati i druge strukturne relacije ciljnog jezika kada su deo izabrane transformacione specifikacije. Objektno-orijentisano nasleđivanje ne poistovećuje se sa dorađivanjem preslikavanja. Za potonje se u formalnom potpisu koristi izabrana relacija dorade, u radu označena sa `babj:refinedBy`; ta oznaka ne mora biti deklarirana kao svojstvo u sadašnjem BABj RDF fajlu.

4.1.5 Kategorija generisanog koda

Prethodne kategorije opisuju strukturu izvornog modela, preslikavanja i realizacione obrasce cilnog skupa tehnologija, ali transformacioni lanac se završava konkretnim izvornim artefaktima. Zato je potrebna posebna ciljna kategorija koja predstavlja rezultat generisanja i daje kodomen funktoru H_s . Njome se razlikuje specifikacija načina realizacije od strukture zaista generisanih klasa i njihovih zavisnosti. Ova kategorija ne formalizuje potpunu semantiku cilnog programskog jezika, već samo one strukturne odnose koji su relevantni za proveru generatorskog izlaza i kompoziciju transformacije.

Za izabrani ciljni skup tehnologija s , neka G_{code}^s bude usmereni multigraf čiji su čvorovi specifikacije generisanih izvornih artefakata, a grane eksplicitne strukturne zavisnosti koda: deklaracije uvoza, nasleđivanje, implementiranje interfejsa, tipovi polja i zavisnosti potpisa metoda. Definiše se

$$C_{\text{code}}^s := \text{Free}(G_{\text{code}}^s).$$

Objekat ove kategorije nije samo putanja datoteke. Njegov identitet obuhvata ulogu artefakta, potpuno kvalifikovano ime i generisani strukturni sadržaj, uključujući superklasu, interfejse, anotacije, polja, metode, deklaracije uvoza i ulogu šablona. Kod se ne kvotira po semantičkoj ekvivalenciji cilnog jezika. Ako dva realizaciona puta pokušaju da proizvedu isti identifikator sa različitim sadržajem, to je verifikaciona greška, a ne jednačina u C_{code}^s .

4.2 Konformansni funktor i bifunktor transformacije modela u model

4.2.1 Realizaciono preslikavanje

Za svaki ciljni skup tehnologija $s \in S$, ontologija O_{BAB}^s za svaku podržanu klasu $M \in \text{Ob}(C_{\text{Map}})$ obezbeđuje označenu imenovanu individuu

$$\mu_s(M) \in O_{\text{BAB}}^s,$$

koja je instanca klase M i nosi vezivanja specifična za ciljni skup tehnologija. Za skup tehnologija zasnovan na Javi, na primer,

$$\mu_j(\text{mapping:EntityMapping}) = \text{babj:DefaultEntityMapping}_j,$$

$$\mu_j(\text{mapping:PersistentEntityMapping}) = \text{babj:DefaultPersistentEntityMapping}_j.$$

Realizaciona individua nije domenski objekat. Ona predstavlja profil koji može sadržati ciljni tip, anotacije, interfejse, superklasu, strukturnu ulogu, šablon i druge parametre generisanja. Kategorija C_{Map} ostaje nezavisna od cilnog skupa tehnologija; ta zavisnost sadržana je u izboru μ_s i u konformansnom funktoru.

4.2.2 Konformansni funktor i prirodnost dorada

Za izabrani ciljni skup tehnologija definiše se konformansni funktor

$$\Phi_s: C_{\text{Map}} \rightarrow [C_{\text{Core}}^{\text{NonMapping}}, C_{\text{BAB}}^s],$$

gde je cilj funktorska kategorija čiji su objekti funktori $C_{\text{Core}}^{\text{NonMapping}} \rightarrow C_{\text{BAB}}^s$, a morfizmi prirodne transformacije.

Na objektima, svakoj klasi preslikavanja M pridružuje se funktor

$$\Phi_s(M): C_{\text{Core}}^{\text{NonMapping}} \rightarrow C_{\text{BAB}}^s.$$

Za domenski objekat D , vrednost $\Phi_s(M)(D)$ jeste realizaciona specifikacija dobijena primenom profila $\mu_s(M)$ na D . Ona istovremeno zavisi od identiteta domenskog objekta i izabrane strategije preslikavanja.

Pošto je izvorna potkategorija slobodno generisana, dovoljno je zadati sliku generišućih grana. Ontologija specifična za ciljni skup tehnologija i rečnik strukture koda određuju preslikavanje oznaka

$$\lambda_s: \text{Edge}(G_{\text{Core}}^{\text{NonMapping}}) \rightarrow \text{Lab}(G_{\text{BAB}}^s).$$

Za generišuću granu $e: D \rightarrow D'$, morfizam $\Phi_s(M)(e)$ predstavljen je ciljnom strukturnom relacijom sa oznakom $\lambda_s(e)$, iz $\Phi_s(M)(D)$ u $\Phi_s(M)(D')$. Preslikavanje λ_s ne zavisi od M , dok izvorni i ciljni realizacioni objekti zavise od profila $\mu_s(M)$. Za stazu $f = e_n \circ \dots \circ e_1$ postavlja se

$$\Phi_s(M)(f) = \Phi_s(M)(e_n) \circ \dots \circ \Phi_s(M)(e_1),$$

a identiteti se šalju u identitete. Ako je λ_s definisana i kompatibilna sa domenima i opsezima na svim izabranim generatorima, univerzalno svojstvo slobodne kategorije daje jedinstven funktor $\Phi_s(M)$.

Na morfizmima kategorije C_{Map} , doradi $h: M \rightarrow M'$ pridružuje se prirodna transformacija

$$\Phi_s(h): \Phi_s(M) \Rightarrow \Phi_s(M').$$

Njena komponenta u domenskom objektu D jeste morfizam

$$\Phi_s(h)_D = \rho_s(D, h): \Phi_s(M)(D) \rightarrow \Phi_s(M')(D).$$

Morfizam $\rho_s(D, h)$ predstavlja ciljnu doradu realizacionog profila, na primer promenu anotacije, superklase, interfejsa, uloge ili šablona. Za svaki generišući morfizam $e: D \rightarrow D'$, skup E_{BAB}^s sadrži jednačinu prirodnosti

$$\rho_s(D', h) \circ \Phi_s(M)(e) = \Phi_s(M')(e) \circ \rho_s(D, h).$$

Odgovarajući komutativni dijagram prirodnosti dorade prikazan je na Slici 4.1.

$$\begin{array}{ccc}
\Phi_s(M)(D) & \xrightarrow{\Phi_s(M)(f)} & \Phi_s(M)(D') \\
\Phi_s(h)_D \downarrow & & \downarrow \Phi_s(h)_{D'} \\
\Phi_s(M')(D) & \xrightarrow{\Phi_s(M')(f)} & \Phi_s(M')(D')
\end{array}$$

Slika 4.1. Komutativni dijagram prirodnosti dorade, prema Bogdanović i saradnici [15].

Pošto je $C_{\text{Core}}^{\text{NonMapping}}$ slobodno generisana, prirodnost za proizvoljne staze sledi indukcijom po dužini staze i kompatibilnošću kongruencije sa kompozicijom.

Primer preuzet iz rada Bogdanović i saradnici [15] razmatra doradu

$$h:\text{mapping:EntityMapping} \rightarrow \text{mapping:PersistentEntityMapping}$$

i domenski morfizam

$$\text{core:hasAttribute:core:Entity} \rightarrow \text{core:Attribute}.$$

U Java realizaciji oba profila prikazuju `hasAttribute` vezom polja, formalno označenom kao `woc:hasField`. Dodavanje perzistentne realizacije menja profil klase, ali ne sme narušiti pripadnost polja klasi. Dve staze, najpre veza polja pa dorada, ili najpre dorada pa odgovarajuća veza polja, identifikuju se jednačinom prirodnosti u E_{BAB}^J .

Prirodnost ima dve uloge. Prvo, ona obezbeđuje da je $\Phi_s(h)$ zaista morfizam u funktorskoj kategoriji. Drugo, ona izjednačava dve kanonske faktORIZACIJE morfizma (f, h) u proizvodnoj kategoriji:

$$(f, h) = (f, \text{id}_{M'}) \circ (\text{id}_D, h) = (\text{id}_{D'}, h) \circ (f, \text{id}_M).$$

Time je omogućena dobro definisana slika para morfizama pod bifunktorom.

Pošto je C_{Map} preduređena kategorija, dorada je određena izvorom i ciljem. Komponente moraju dodatno zadovoljiti koherenciju identiteta i kompozicije:

$$\rho_s(D; M, M) = \text{id}_{\Phi_s(M)(D)},$$

$$\rho_s(D; M, M'') = \rho_s(D; M', M'') \circ \rho_s(D; M, M').$$

Pod tim uslovima važi $\Phi_s(\text{id}_M) = \text{id}_{\Phi_s(M)}$ i $\Phi_s(k \circ h) = \Phi_s(k) \circ \Phi_s(h)$, pa je Φ_s funktor.

4.2.3 Evaluacioni bifunktor

Za svaki ciljni skup tehnologija s definiše se

$$F_s: C_{\text{Core}}^{\text{NonMapping}} \times C_{\text{Map}} \rightarrow C_{\text{BAB}}^s.$$

Na objektima i morfizmima važi

$$F_s(D, M) = \Phi_s(M)(D),$$

$$F_s(f, h) = \Phi_s(h)_{D'} \circ \Phi_s(M)(f), (f, h): (D, M) \rightarrow (D', M').$$

Po prirodnosti je isti morfizam moguće zapisati i kao

$$F_s(f, h) = \Phi_s(M')(f) \circ \Phi_s(h)_D.$$

Svojstvo `mapping:hasImplementationMapping` nije morfizam domena ovog bifunktora. Ono ostaje poprečni morfizam u C_{Core} i semantički određuje parove (D, M) koji imaju nameravano arhitektonsko značenje. Formalno je F_s totalan na celom proizvodu kategorija; u toku rada generatora primenjuje se na eksplicitno ili podrazumevano odabrane, tipski kompatibilne parove.

Propozicija 4.1 (bifunktor F_s). Za svaki $s \in S$, preslikavanje F_s jeste funktor iz proizvoda kategorija $C_{\text{Core}}^{\text{NonMapping}} \times C_{\text{Map}}^s$ u C_{BAB}^s , pa je bifunktor u svoja dva argumenta.

Dokaz. Posmatrane kategorije su male, pa se konstrukcija odvija u kategoriji **Cat**, koja je kartezijanski zatvorena [133, Poglavlje IV.6]. Za male kategorije C , M i B postoji prirodna korespondencija

$$\text{Hom}(C \times M, B) \cong \text{Hom}(M, [C, B]).$$

U konkretnom slučaju evaluacioni funktor je

$$\text{ev}_s: [C_{\text{Core}}^{\text{NonMapping}}, C_{\text{BAB}}^s] \times C_{\text{Core}}^{\text{NonMapping}} \rightarrow C_{\text{BAB}}^s.$$

Na objektima $\text{ev}_s(G, D) = G(D)$, a za prirodnu transformaciju $\eta: G \Rightarrow G'$ i morfizam $f: D \rightarrow D'$ važi

$$\text{ev}_s(\eta, f) = \eta_{D'} \circ G(f) = G'(f) \circ \eta_D.$$

Neka su π_1 i π_2 projekcije iz proizvodne kategorije na prvi i drugi faktor. Tada se F_s faktoriše kao

$$F_s = \text{ev}_s \circ \langle \Phi_s \circ \pi_2, \pi_1 \rangle.$$

Funktor Φ_s , projekcije, njihovo uparivanje i evaluacija jesu funktori. Njihova kompozicija je zato funktor. Pošto joj je domen proizvodna kategorija, F_s je bifunktor. $\square$

Ova faktORIZACIJA odvajja domenski faktor od razrešavanja specifičnog za ciljni skup tehnologija. Pri promeni ciljnog skupa tehnologija zadržavaju se $C_{\text{Core}}^{\text{NonMapping}}$, C_{Map} i oblik evaluacione konstrukcije, dok se menjaju μ_s , Φ_s , C_{BAB}^s i kasniji funktor transformacije modela u kod. To je formalna osnova višeciljne generacije, ali ne i dokaz da je novi skup tehnologija implementiran samim dodavanjem ontoloških iskaza.

4.3 Funktor transformacije modela u kod i kompozicija

4.3.1 Funktor transformacije platformski specifičnog modela u kod

Drugu fazu transformacije predstavlja

$$H_s: C_{BAB}^s \rightarrow C_{code}^s.$$

Na objektima, H_s realizacionu specifikaciju šalje u specifikaciju izvorne datoteke ili fragmenta koji je realizuje. Ulaz sadrži sve informacije relevantne za izabrani profil: ciljni tip, superklasu, anotacije, interfejsa, deklaracije uvoza, polja, metode i šablonsku ulogu. Na morfizmima, ciljna strukturna relacija preslikava se u odgovarajuću zavisnost generisanog koda.

Pošto je C_{BAB}^s prezentovana kategorija, H_s se konstruiše u dva koraka. Najpre se zadaje preslikavanje generatora G_{BAB}^s u C_{code}^s , koje se po univerzalnom svojstvu slobodne kategorije jedinstveno proširuje na

$$\tilde{H}_s: \text{Free}(G_{BAB}^s) \rightarrow C_{code}^s.$$

Zatim se proverava kompatibilnost sa kvocijentom:

$$u \equiv_E v \Rightarrow \tilde{H}_s(u) = \tilde{H}_s(v).$$

Ako je $\tilde{H}_s$ kompatibilan sa kvocijentom, odnosno ako iz $u \equiv_E v$ sledi $\tilde{H}_s(u) = \tilde{H}_s(v)$, onda $\tilde{H}_s$ faktoriše kroz kvocijent i određuje dobro definisan funktor

$$H_s([u]) = \tilde{H}_s(u).$$

Java instanca funktora H_j obuhvata programsku pripremu projektnih struktura iz razrešene realizacione specifikacije, generisanje pomoću FreeMarker šablona i formiranje artefakata, uz bezbedno upisivanje preko pripremnog direktorijuma i manifesta. Učitavanje pomoću biblioteke Jena, validacija i izdvajanje podataka prethode ovoj fazi, dok razrešavanje BAB_j profila pripada realizaciji preslikavanja ka PSM-u u radu generatora. Prema tome, H_j označava fazu od razrešene realizacione specifikacije do artefakata koda, a ne celokupan tok rada generatora niti samo poziv FreeMarker šablona.

4.3.2 Kompozitna transformacija

Celokupan lanac za ciljni skup tehnologija s glasi

$$C_{Core}^{\text{NonMapping}} \times C_{Map} \xrightarrow{F_s} C_{BAB}^s \xrightarrow{H_s} C_{code}^s.$$

Njegova kompozicija

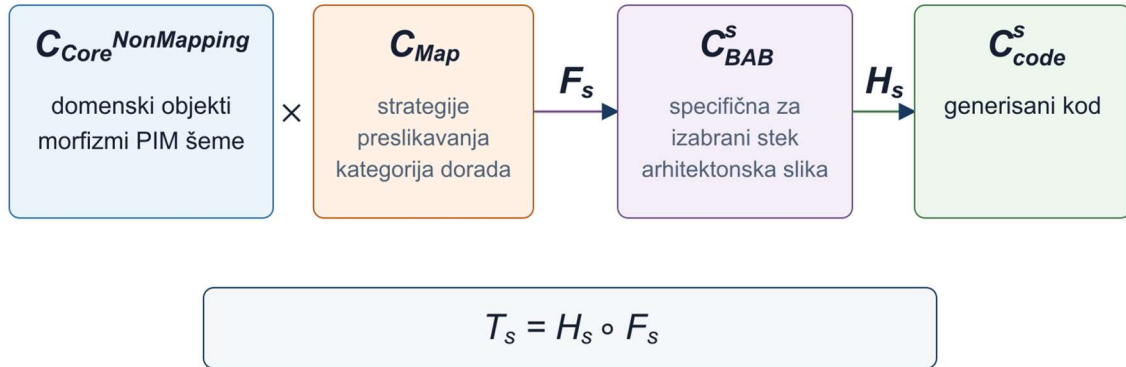
$$T_s = H_s \circ F_s$$

jeste funktor kada su F_s i H_s dobro definisani. Za objekte i morfizme važi

$$T_s(D, M) = H_s(F_s(D, M)),$$

9

Kompozicija kategorijskog transformacionog lanca od PIM šeme do koda prikazana je na Slici 4.2.



Slika 4.2. Kategorijski transformacioni lanac od PIM šeme do koda.

Kompozicija automatski čuva identitete i kompoziciju. Iz te činjenice, međutim, ne slede automatski vernost, injektivnost na objektima niti odsustvo neželjenih identifikacija u kvocijentu.

4.4 Uslovno očuvanje šematske strukture

4.4.1 Centralni rezultat

Propozicija 4.2 (uslovno strukturno očuvanje). Neka su F_s i H_s bifunktor transformacije modela u model i funktor transformacije modela u kod za izabrani ciljni skup tehnologija s , a $T_s = H_s \circ F_s$ njihova kompozicija, posmatrana na parovima koje bira `mapping:hasImplementationMapping` ili odgovarajuća politika podrazumevanog profila. Pretpostavimo:

1. grafovsko preslikavanje koje određuje F_s injektivno je na odabranim parovima izvornih morfizmama, osim identifikacija uvedenih eksplicitno nameravanim normalizacionim jednačinama u E_{BAB}^s , a kongruencija $\equiv_E$ ne uvodi dodatne identifikacije njihovih F_s -slika;
2. grafovsko preslikavanje koje određuje H_s konstantno je na klasama kongruencije $\equiv_E$, tako da je H_s dobro definisan na C_{BAB}^s , i injektivno je na relevantnim ciljnim klasama ekvivalencije;
3. šema imenovanja injektivna je na razmatranim domenskim klasama;
4. različite odabrane strategije preslikavanja određuju različite profile generisanja, tako da ih identitet artefakta razlikuje prema generisanom strukturnom sadržaju.

Tada važe sledeća svojstva.

(a) **Totalnost.** Za svaki odabrani par (D, M) postoji jedinstvena specifikacija generisanog artefakta

$$T_s(D, M) \in \text{Ob}(C_{\text{code}}^s).$$

(b) **Strukturna koherentnost.** Za svaki odabrani par morfizama $(f, h): (D, M) \rightarrow (D', M')$ postoji jedinstven morfizam

$$T_s(f, h): T_s(D, M) \rightarrow T_s(D', M').$$

Kompozitni funktor čuva identitete i komponentnu kompoziciju:

$$T_s(\text{id}_D, \text{id}_M) = \text{id}_{T_s(D, M)},$$

$$T_s(f' \circ f, h' \circ h) = T_s(f', h') \circ T_s(f, h).$$

(c) **Vernost modulo nameravane normalizacije.** Ako parovi morfizama (f, h) i (g, k) imaju F_s -slike koje nisu identifikovane eksplicitnim normalizacionim jednačinama, tada

$$T_s(f, h) \neq T_s(g, k).$$

Ako E_{BAB}^s ne sadrži takve normalizacije na razmatranom šematskom delu, iskaz se svodi na strogu vernost.

(d) **Injektivnost na objektima.** Ako odabrani parovi (D, M) i (D', M') nisu jednaki niti identifikovani normalizacijom profila generisanja, tada

$$T_s(D, M) \neq T_s(D', M').$$

Dokaz. Totalnost i strukturna koherentnost slede iz činjenice da je T_s kompozicija funktora. Za vernost je potrebno da F_s različite izvorne parove pošalje u različite klase ekvivalencije, osim eksplicitno dopuštenih normalizacija, i da H_s očuva te razlike. Injektivnost na objektima sledi iz injektivne šeme imenovanja za različite domenske objekte i iz pretpostavke da različite strategije za isti domenski objekat daju različite strukturne profile. Ako dva deklarativno različita preslikavanja daju isti profil, ona se za potrebe ove propozicije tretiraju kao ekvivalentna, a ne kao dva različita očuvana ulaza. $\square$

4.4.2 Posledice

Posledica 4.1 (očuvanje informacija na šematskom nivou). Pod pretpostavkama Propozicije 4.2, svakoj preslikanoj domenskoj klasi i strukturnoj relaciji, zajedno sa odabranom strategijom, odgovara generisana specifikacija u C_{code}^s . Razlike koje nisu eksplicitno normalizovane ne gube se niti stapaju tokom transformacije.

Posledica 4.2 (slikovna potkategorija). Slika $\text{Im}(T_s)$ čini potkategoriju kategorije C_{code}^s . Nakon kvotiranja odabranih parova objekata i morfizama po eksplicitno nameravanim normalizacijama, indukovana restrikcija T_s jeste izomorfizam na svoju sliku. Generisani kod zato sadrži potkategoriju izomorfnu odabranom i kvotiranom preslikavanom šematskom delu, a ne nužno celom proizvodu $C_{\text{Core}}^{\text{NonMapping}} \times C_{\text{Map}}$.

4.4.3 Granice formalne tvrdnje

Propozicija 4.2 ne obuhvata sledeće tvrdnje.

Očuvanje na nivou instanci. Ne dokazuje se da se ABox podaci očuvavaju u operativnom ponašanju generisanog programa. Formalizam se odnosi na TBox i izabranu šematsku strukturu.

Semantička ekvivalencija. Ne tvrdi se da generisani kod ima istu denotacionu ili operativnu semantiku kao OWL ontologija. Takav rezultat zahtevao bi formalnu semantiku ciljnog jezika i morfizam koji čuva semantiku između dva nivoa.

Dvosmerna ekvivalencija, odnosno ekvivalencija sa povratnom rekonstrukcijom. Ne tvrdi se da se početna ontologija može rekonstruisati iz koda. Za T_s nije konstruisan levi niti desni adjungovani funktor.

Bezuslovna korektnost implementacije. Kategorijalna prezentacija određuje šta implementacija treba da zadovolji. Ona sama po sebi ne dokazuje da konkretan generator ispunjava sve pretpostavke Propozicije 4.2.

4.5 Validacija modela i verifikacija transformacije

4.5.1 Validacija domenske ontologije

Validacija prethodi transformaciji. U sadašnjem generatoru HermiT proverava PIM aplikacije zajedno sa učitanim BAB PIM 2.0 modulima, dok namenski validatori nad Jena modelom proveravaju uslove zatvorenog sveta: obavezno prisustvo i jednoznačnost elemenata, ispravne tipove, relacije, projekcije putanje, procese, servise, uloge, sposobnosti i Java identifikatore. BAB Mapping 2.0 i BABj 2.0 proveravaju se zasebno kao PSM ugovor koji se koristi u radu generatora; puna WoC ontologija nije deo tog grafa.

Validacija modela nije kategorijalno svojstvo. Ona obezbeđuje da je ulaz transformacije dobro formiran. Kategorijalni aparat zatim opisuje strukturu transformacije i uslove pod kojima ona čuva odabrani šematski deo.

4.5.2 Pet obaveza verifikacije

Da bi se Propozicija 4.2 primenila na konkretnu implementaciju, potrebno je proveriti pet uslova.

1. **Kompletnost preslikavanja.** Svaka izabrana generišuća grana grafa $G_{Core}^{NonMapping}$ koja se pojavljuje u transformaciji mora pripadati domenu funkcije λ_s .
2. **Injektivnost preslikavanja.** Različite izabrane relacije objedinjenog PIM skeleta ne smeju biti preslikane na istu ciljnu oznaku, osim kada je identifikacija izričito deklarirana u E_{BAB}^S .

3. **Determinizam generisanja.** Ista realizaciona specifikacija mora dati isti strukturni izlaz, bez nedeterminističkih izbora i nekontrolisanog spoljašnjeg stanja.
4. **Kompatibilnost sa kvocijentom.** Ako $u \equiv_E v$, tada $\tilde{H}_s(u) = \tilde{H}_s(v)$. Šabloni i programska priprema ne smeju razlikovati staze koje je ciljna prezentovana kategorija izjednačila.
5. **Odsustvo neželjenog sažimanja.** Kongruencija ne sme identifikovati F_s -slike različitih odabranih izvornih morfizama osim kada je takva normalizacija izričito nameravana.

Tek kada su svih pet uslova proverena, implementacija je saglasna sa kategorijalnom specifikacijom na izabranom šematskom delu i mogu se primeniti garancije Propozicije 4.2.

4.5.3 Status sadašnje implementacije

Aktuelna implementacija zadržava faktORIZACIJU $T_j = H_j \circ F_j$ i bifunktorski specifikacioni okvir razvijen u radu Bogdanović i saradnici [15] i izložen u ovom poglavlju. Veza sa generisanim kodom ispituje se izvršivom proverom određenog ulaza, razrešenog BABj profila i stvarnog izlaza. Time se inženjerski dokazi odnose na konkretan fragment, uz zadržavanje uslovnog karaktera opšteg rezultata.

Sertifikat ugrađene transformacione specifikacije proverava pokrivenost izabranih relacija, njihove ciljne oznake, deklarisanu normalizaciju i ponovljivost fiksiranog korpusa. Normalizacija prethodi građenju konkretnog strukturnog grafa: zahteva kanonski predikat, odgovarajuće tipove krajeva i eksplicitnu inverznu ili ekvivalentnu OWL deklaraciju. Za inverzni zapis menjaju se mesta argumentima. Provera obuhvata normalizaciju RDF zapisa, a ne proizvoljne jednačine paralelnih staza u E_{BAB}^j . Obim proverene specifikacije prikazan je u Poglavlju 6.

Provera konkretnog izlaza zasniva se na zasebnom ugovoru entitetske strukture. Iz normalizovanog PIM-a izdvaja se konačan graf G_P entitetskih deklaracija, atributskih i relacionih mesta, vlasništva i ciljnih entiteta. Mesto atributa ili kraja relacije identifikuje se parom koji čine IRI vlasnika i IRI člana. Ovaj graf predstavlja konkretno instanciranje izabranog šematskog potpisa; ne poistovećuje se sa celom kategorijom svih ontoloških klasa ili svih dorada preslikavanja.

Graf G_A dobija se nezavisnim čitanjem Java sintaksnog stabla aplikacionih izvora: vlasništvo sledi iz pripadnosti polja klasi, a referenca iz deklarisanog ciljnog tipa. Proverivač ne prihvata generatorske metapodatke kao dokaz postojanja polja ili veze. Dodatno proverava tipove, odabrane anotacije, strukturu pristupnih metoda, razrešeni nadtip i direktne zavisnosti od BAB modula. Razrešavanje imena ograničeno je na podržane generatorske obrasce; potpuna kompilacija ostaje zasebna provera.

Dopunski rezultat za fiksirani profil. Ako proverena korespondencija $j: G_P \rightarrow G_A$ predstavlja morfizam grafova koji je injektivan na čvorovima i granama, indukovani

funktor $Free(j)$ veran je i injektivan na objektima. Njegova slika predstavlja potkategoriju izomorfnu izvornoj slobodnoj kategoriji, pri čemu slika ne mora biti puna. Zaista, staze se preslikavaju granu po granu, pa se čuvaju identiteti i nadovezivanje. Jednake slikovne staze imaju jednaku dužinu i, zbog injektivnosti na granama, jednake izvorne grane. Injektivnost na čvorovima obezbeđuje da kompozabilne slikovne staze imaju kompozabilne originale. Ovo je primena standardne konstrukcije slobodne kategorije [134].

Implementirani proverivač grafova proverava konačne dovoljne uslove tog rezultata: ukupnost korespondencije, tipove, krajeve i injektivnost. Veza matematičkog rezultata sa Java implementacijom ostaje uslovljena ispravnošću izdvajanja grafa i samog proverivača; programski kod nije mašinski verifikovan u dokaznom asistentu. Ova provera ne ustanovljava prirodnost svih dorada strategija, opštu kompatibilnost kvocijentnih prezentacija, očuvanje poslovnih ABox podataka, semantičku ekvivalenciju izvršavanja ni povratnu rekonstrukciju. Primeri i namerno izmenjeni izlazi iz Poglavlja 6 daju zasebnu empirijsku proveru realizovanog ugovora.

4.6 Sažetak

Formalni lanac metodologije glasi

$$C_{Core}^{NonMapping} \times C_{Map} \xrightarrow{F_s} C_{BAB}^s \xrightarrow{H_s} C_{code}^s.$$

Obe klase objekata pripadaju objedinjenoj kategoriji C_{Core} , konstruisanoj nad unijom svih učitanih BAB PIM 2.0 modula i BAB Mapping 2.0: domenski objekti određuju punu potkategoriju $C_{Core}^{NonMapping}$, a `mapping:ImplementationMapping` hijerarhija punu potkategoriju $C_{Core}^{MappingOnly}$. Nad istim preslikavajućim objektima gradi se preduređena kategorija dorada C_{Map} . Poprečni morfizmi `mapping:hasImplementationMapping` pripadaju C_{Core} i biraju arhitektonski relevantne parove.

Konformansni funktor Φ_s , njegove prirodne transformacije i evaluaciona faktORIZACIJA određuju bifunktor F_s . Funktor H_s dobro je definisan tek kada poštuje kongruenciju prezentovane kategorije C_{BAB}^s . Kompozicija $T_s = H_s \circ F_s$ uslovno čuva odabranu šematsku strukturu pod pretpostavkama kompletnosti, injektivnosti, determinizma, kompatibilnosti sa kvocijentom i odsustva neželjenog sažimanja.

5. BAB softverski okvir: implementacija metodologije

5.1 Arhitektura BAB softverskog okvira

BAB okvir je predstavljen kao proširiva softverska platforma za procesno svesne poslovne informacione sisteme [12]. U ovom potpoglavlju razlikuju se ontološki profil generatora i izvršna biblioteka. Trenutno proverena verzija generatora koristi modularni BAB PIM 2.0, BAB Mapping 2.0 i BABj 2.0 kao ugovor transformacije, a generisani projekti u Javi koriste modularnu biblioteku BAB 2.0. Potpoglavlja od 5.1.1 do 5.1.4 opisuju programske ugovore te biblioteke koje BABj profil bira, dok potpoglavlje 5.1.5 objašnjava raspodelu između obaveznog jezgra i neobaveznih modula. Rezultati u Poglavlju 6 vezani su za tačno identifikovanu verziju generatora.

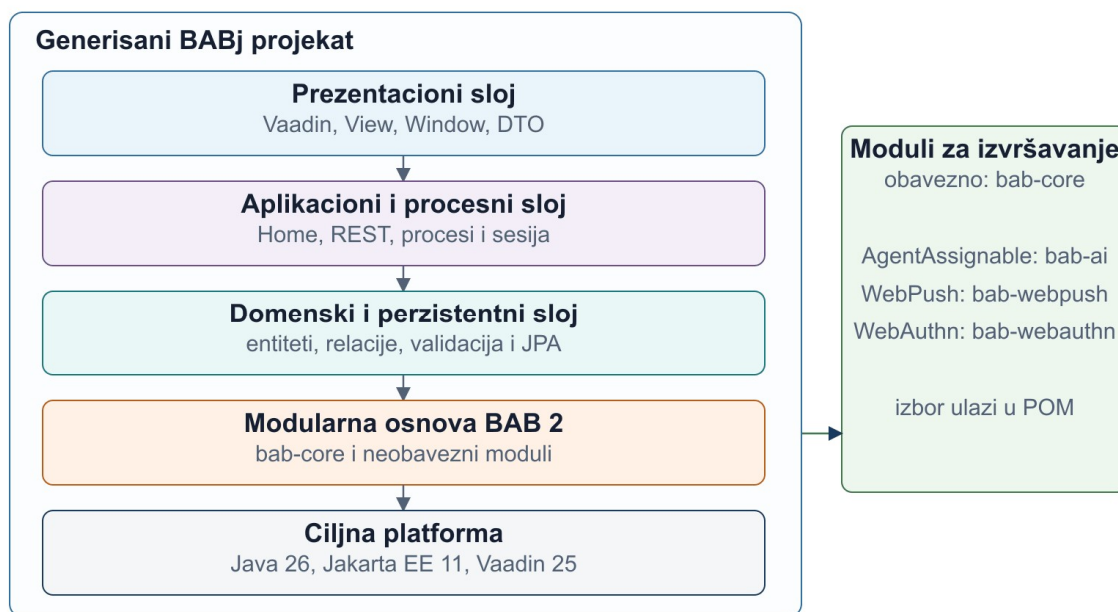
5.1.1 Slojevita arhitektura poslovnog sistema

BAB okvir organizovan je kao slojevita arhitektura poslovnog sistema. Ovaj obrazac razdvaja odgovornosti na prezentacioni, poslovni i podatkovni sloj kako bi se posledice promena ograničile na jasno određene delove sistema. Iako savremeni poslovni sistemi mogu koristiti i mikroservisne arhitekture ili arhitekture bez stalno aktivnih servera (engl. *serverless*), slojeviti pristup odgovara cilju BAB okvira: izgradnji poslovnih aplikacija srednje veličine sa jasno razgraničenim odgovornostima.

Jakarta EE izabrana je zbog nekoliko svojstava relevantnih za BAB okvir. Reč je o standardizovanoj platformi koja objedinjuje specifikacije JPA, EJB, CDI, JTA, Bean Validation i JAX-RS. Njihova međusobna usklađenost smanjuje potrebu za povezivanjem većeg broja nezavisnih biblioteka. Aplikacioni serveri, kao što su WildFly, Payara i Open Liberty¹², preuzimaju upravljanje životnim ciklusom komponenta, transakcijama, bezbednošću i resursima. Time se deo infrastrukturne složenosti izmešta iz generatora koda u standardizovano izvršno okruženje. U kategorijalnom modelu standardizovani konstrukti Jakarta EE predstavljaju strukturne elemente ciljnog PSM sloja.

Slojevita arhitektura BAB okvira obuhvata šest celina. Infrastrukturni sloj uspostavlja vezu sa relacionom bazom, određuje JTA izvor podataka u prostoru JNDI i podešava jedinicu perzistencije u datoteci persistence.xml. Perzistentni sloj čine entiteti izvedeni iz klase AbstractEntity i metapodaci njihovog JPA preslikavanja. Sloj domenskih repozitorijuma zasniva se na hijerarhiji AbstractHome i obuhvata operacije CRUD, upite i transakcione granice. Sloj orkestracije povezuje više repozitorijuma u poslovne tokove i komunicira sa slojem agenata. Prezentacioni sloj realizovan je komponentama Vaadin Flow i apstrakcijama GenericView, GenericTable i GenericWindow. Sloj agenata deluje poprečno u odnosu na ostale slojeve i sa njima komunicira putem sabirnice događaja. Ovi slojevi i izbor izvršnih modula prikazani su na Slici 5.1.

¹² Zvanične stranice aplikacionih servera: WildFly: <https://www.wildfly.org/>; Payara: <https://www.payara.fish/>; Open Liberty: <https://openliberty.io/>



Slika 5.1. Slojevita arhitektura generisane BAB 2 aplikacije i izbor izvršnih modula.

Transakcione granice u Jakarta EE deklarativno se određuju anotacijom `@TransactionAttribute` na metodama komponenata EJB. U BAB okviru transakcija se po pravilu započinje pozivom metode repozitorijumske klase i završava po izlasku iz nje. Za posebne slučajeve, kao što je beleženje događaja koje mora opstati i nakon poništavanja glavne transakcije, koristi se režim `REQUIRES_NEW`. Prezentacioni sloj zato ne otvara niti zatvara transakcije neposredno.

Životni ciklus komponenata određen je pravilima CDI. Generičke klase prikaza uglavnom imaju opseg `@Dependent` ili `@ViewScoped`, dok su klase `AbstractHome` komponente EJB označene anotacijama `@Stateless` i `@LocalBean`. Komponente sloja agenata po pravilu imaju opseg `@ApplicationScoped`, jer treba da postoje tokom čitavog životnog ciklusa aplikacije. Zavisnosti se razrešavaju pomoću kontejnera `BeanManager`, a prilagođena komponenta `GenericCdiBeanProvider` omogućava pribavljanje upravljanih objekata iz konteksta koji nisu deo uobičajenog grafa CDI.

Slojevita organizacija olakšava održavanje sistema. Promene šeme baze i preslikavanja entiteta prvenstveno se zadržavaju u perzistentnom sloju, dok prezentacione komponente koriste DTO objekte. Izmene poslovne logike u klasama `AbstractHome` ne zahtevaju promenu prikaza sve dok se poštuju ugovori CRUD interfejsa. Pojedinačni slojevi mogu se proveravati alatima Arquillian i Weld JUnit¹³ uz zamenske implementacije zavisnosti. U formalnom modelu relevantne su strukturne zavisnosti između slojeva; njihovo očuvanje zavisi od uslova transformacije utvrđenih u Poglavlju 4.

¹³ Zvanične stranice alata za proveru: Arquillian: <https://arquillian.org/>; Weld: <https://weld.cdi-spec.org/>; JUnit: <https://junit.org/junit5/>

5.1.2 AbstractEntity, AbstractHome, AbstractDTO: apstrakcije jezgra

Jezgro BAB okvira čine tri povezane apstrakcije: entitetska klasa AbstractEntity, repozitorijumska klasa AbstractHome i objekat za prenos podataka AbstractDTO. Ove klase odgovaraju obrascima domenskog modela, repozitorijuma i objekta za prenos podataka, uz parametrizovane tipove koji omogućavaju statičku proveru tipske ispravnosti.

Klasa AbstractEntity je apstraktna i sadrži metapodatke zajedničke svim entitetima sistema. Anotacija @MappedSuperclass određuje da JPA ne pravi posebnu tabelu za ovu klasu, već njena polja uključuje u tabelu svakog konkretnog naslednika. Klasa implementira Serializable radi serijalizacije u okruženju Jakarta EE i Comparable<AbstractEntity> radi uređivanja entiteta u kolekcijama i komponentama korisničkog interfejsa na osnovu vrednosti metode getCompareValue().

Identifikator entiteta je polje tipa Long označeno anotacijama @Id, @GeneratedValue(strategy = GenerationType.IDENTITY) i @Column. Strategija IDENTITY prepušta generisanje primarnog ključa bazi podataka i primenljiva je na dijalekte MySQL, MariaDB i SQL Server koje BAB podržava. Polje version, označeno anotacijom @Version, realizuje optimističko zaključavanje (engl. *optimistic locking*). Pri ažuriranju Hibernate¹⁴ proverava da li se vrednost verzije promenila nakon učitavanja entiteta. Ako jeste, izaziva se OptimisticLockException, koji repozitorijumske klase obrađuju tokom grupnih operacija.

Posebnu pažnju zahtevaju metode equals i hashCode. Pre prvog čuvanja entitet nema vrednost polja id, a Hibernate može koristiti posredničke objekte koji otežavaju neposredno poređenje tipova. Zato implementacija prikazana u Listingu 5.1 najpre poziva Hibernate.unproxy(o), a zatim poredi konkretne tipove i vrednosti identifikatora.

```
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null) return false;
    Object other = Hibernate.unproxy(o);
    if (!(other instanceof AbstractEntity that)) return false;
    if (this.getClass() != other.getClass()) return false;
    return id != null && id.equals(that.getId());
}
```

Listing 5.1. Implementacija metode equals u klasi AbstractEntity.

Sa kategorijalnog stanovišta, AbstractEntity je ciljna bazna specifikacija za perzistentne entitete u kategoriji C_BABj. Funktor F_s, čije je dejstvo opisano u Poglavlju 4, preslikava konkretan PIM element klase core:Entity, zajedno sa izabranim preslikavanjem, u realizacionu specifikaciju čiji Java artefakt nasleđuje AbstractEntity.

Klasa AbstractSettings, koja nasleđuje AbstractEntity, ilustruje proširenje bazne klase domenskim poljima. Polja backupLocation, pluginsLocation i aktivan označena su

¹⁴ Zvanične stranice: MySQL: <https://www.mysql.com/>; MariaDB: <https://mariadb.org/>; Microsoft SQL Server: <https://www.microsoft.com/en-us/sql-server>; Hibernate ORM: <https://hibernate.org/orm/>

anotacijama validacije `@NotNull` i `@NotEmpty`, kao i odgovarajućim anotacijama `@Column`. Za tekstualna polja koristi se podrazumevani tip `nvarchar(500)`. Primeri koji zahtevaju drugačiju dužinu ili tip kolone moraju to navesti posebnim preslikavanjem.

Klasa `AbstractAgentConfig` implementira `CheckAktivanInterface`, što repozitorijumska klasa prepoznaje i koristi za dodavanje uslova `WHERE x.aktivan = true` u upite. Polje `agentClass` tipa `Class<? extends Agent>` čuva se kao potpuno kvalifikovano ime klase pomoću konvertora `AgentClassConverter` i anotacije `@Convert`. Pri pokretanju sistema odgovarajuća klasa pribavlja se refleksijom.

Klasa `AbstractHome` je centralna apstrakcija poslovnog sloja. Njena deklaracija prikazana je u Listingu 5.2.

```
public abstract class AbstractHome<
    T extends AbstractEntity,
    D extends AbstractDTO<T>>
    implements Serializable,
        CheckUniqueFieldHomeInterface,
        AbstractHomeCRUDInterface<T, D> {
    @PersistenceContext
    protected EntityManager entityManager;

    private final Class<T> entityClass;
    private final Class<D> gridClass;
    protected String name;
    protected String shortName;
    protected boolean unique;

    protected AbstractHome(Class<T> entityClass, Class<D> gridClass) {
        this.entityClass = entityClass;
        this.gridClass = gridClass;
    }
    // ...
}
```

Listing 5.2. Deklaracija generičke klase `AbstractHome`.

Dvojni tipski parametar `T` i `D` zaslužuje pažnju. `T` je tip entiteta, koji se koristi za sve operacije na perzistentnom modelu (`find`, `save`, `remove`, `merge`). `D` je tip DTO objekta koji se koristi za projekciju u listama i tabelama na korisničkom interfejsu. Ova jasna podela rešava klasičan problem `N+1` upita i nepotrebnog učitavanja celokupnih grafova objekata kada je potrebno prikazati samo nekoliko atributa u tabeli. `AbstractDTO` je svojevrсна fasada nad entitetom, sadrži konstruktor koji prima izabrane attribute i koristi se u JPA konstrukciji `SELECT new`.

Osnovne operacije CRUD realizuju se pomoću interfejsa `EntityManager`. Metoda `save` je polimorfna: ako je vrednost `id` jednaka `null`, poziva se `persist`, a u suprotnom `merge`. Metoda `saveInNewTrans` označena je anotacijom `@Transactional(Transaction.TxType.REQUIRES_NEW)`, čime se obezbeđuje da se snimanje izvrši u izdvojenoj transakciji. To je korisno pri beleženju događaja ili revizijskih podataka koji treba da ostanu sačuvani i kada se nadređena transakcija poništi.

Metoda `saveBatch`, prikazana u Listingu 5.3, pokazuje kako se ručno upravlja konkurentnošću.

```
public void saveBatch(List<T> items, Function<T, T> beforeSave) {  
    int batch = 200;  
    int counter = 0;  
    for (T item : items) {  
        try {  
            T prepared = (beforeSave != null) ? beforeSave.apply(item) : item;  
            save(prepared);  
            if (++counter % batch == 0) {  
                entityManager.flush();  
                entityManager.clear();  
            }  
        } catch (OptimisticLockException ole) {  
            LOGGER.warn("Optimistic lock skipped entity {}", item, ole);  
        }  
    }  
}
```

Listing 5.3. Grupno čuvanje entiteta u metodi `saveBatch`.

Veličina grupe od 200 entiteta izabrana je empirijski kao ravnoteža između efikasnog grupisanja JDBC operacija i ograničavanja rasta keša prvog nivoa. Periodični pozivi `flush` i `clear` oslobađaju taj keš, dok obrada izuzetka `OptimisticLockException` unutar petlje sprečava da pojedinačni sukob prekine čitavu grupnu operaciju.

Metoda `getFullObject` rešava problem odloženog učitavanja u okruženju Vaadin, u kojem transakcija ne traje duže od obrade korisničkog događaja. Metoda pomoću `java.beans.Introspector` pribavlja `BeanInfo` ciljne klase, a zatim za svako svojstvo `ManyToOne` poziva `Hibernate.initialize`. Tako DTO objekat ili komponenta korisničkog interfejsa dobija inicijalizovan entitet bez izuzetka `LazyInitializationException`.

Generisanje JPQL upita za `findAll` još je jedan primer načina na koji je okvir projektovan. Metoda `getSelect` podrazumevano vraća tekst u obliku prikazanom u Listingu 5.4.

```
SELECT new fqn.of.DTOClass(x.field1, x.field2, ...) FROM Entity x
```

Listing 5.4. Podrazumevani oblik JPQL projekcionog upita.

Konkretno repozitorijumske klase mogu proširiti ili redefinisati ovaj metod i proslediti parametre pomoću `Map<String, Object>`. Metode `getJoin` i `getGroupBy` predstavljaju tačke proširenja koje se popunjavaju kada su potrebna spajanja ili agregacije. Okvir takođe automatski prepoznaje određene interfejse: za `CheckAktivanInterface` dodaje uslov `WHERE x.aktivan = true`, a za `CheckStornoEntityInterface` uslov `WHERE x.storniran = false`. Time se pravila filtriranja ne ponavljaju u svakoj konkretnoj implementaciji.

U kategorijalnoj specifikaciji za `AbstractHome` relevantne su strukturne veze između entitetskih specifikacija, repozitorijumskih operacija i zavisnosti prema sloju perzistencije. Njihovo očuvanje pripada šematskom obimu transformacije iz Poglavlja 4; ono ne predstavlja tvrdnju o operativnoj semantičkoj ekvivalenciji Java i SQL operacija.

Klasa `AbstractDTO` zaokružuje ovu grupu od tri apstrakcije. Ona je takođe parametrizovana po tipu `T` i sadrži najmanji potreban skup podataka o entitetu, obično `id` i nekoliko atributa potrebnih za prikaz. Pored toga sadrži metodu `fromEntity`, koja iz potpunog entiteta proizvodi `DTO`, i metode `getCompareValue` i `toString` koje koriste komponente korisničkog interfejsa.

5.1.3 `GenericView`, `GenericWindow`, `GenericTable`: prezentacione apstrakcije

Prezentacioni sloj BAB okvira organizovan je oko komponentata `GenericView`, `GenericTable` i `GenericWindow`, koje zajedno realizuju obrazac „pregled i dijalog” u poslovnim aplikacijama. Sve tri su zasnovane na platformi Vaadin Flow, čiji se model komponentata izvršava na serveru. Stanje se čuva i ažurira na serverskoj strani, dok se sa pregledačem razmenjuje putem veze `WebSocket` ili postupka dugotrajnog čekanja na odgovor (engl. *long polling*). Zbog toga se korisnički interfejs može generisati kao Java kod iz istog ontološkog modela koji opisuje i ostale slojeve aplikacije.

Klasa `GenericView` nasleđuje kontejnersku komponentu `Div` biblioteke Vaadin i implementira skup interfejsa koji opisuju njene odgovornosti, kao što je prikazano u Listingu 5.5.

```
public class GenericView<
    T extends AbstractEntity,
    V extends AbstractHome<T, D> & CheckUniqueFieldHomeInterface,
    D extends AbstractDTO<T>,
    K extends AbstractEntity & OperaterEntityInterface>
    extends Div
    implements PopulateEvent,
               RefreshGridInterface,
               AfterNavigationObserver,
               NameConversionInterface,
               GenerateResourceInterface,
               AsyncHandlerInterface,
               UsesRenderersInterface {

    @Inject @Primary
    protected Provider<V> serviceEJBProvider;

    @Inject @Primary
    protected Provider<GenericWindow<T, V, D, K>> editWindowProvider;

    protected D selekcija;
    protected D previus;
    protected MenuBar menu;
    protected ContextMenu contextMenu;

    protected static final String QUERY_PARAM_ID = "id";
    // ...
}
```

Listing 5.5. Deklaracija generičke klase `GenericView`.

Četiri tipska parametra, `T`, `V`, `D` i `K`, odražavaju odgovornosti generičkog korisničkog interfejsa. `T` je tip entiteta, `V` je repozitorijumska klasa koja upravlja tim entitetom i implementira `CheckUniqueFieldHomeInterface`, `D` je `DTO` objekat za prikaz u tabeli, a `K`

je tip korisničkog entiteta koji implementira `OperaterEntityInterface`. Poslednji parametar omogućava evidentiranje korisnika koji menja podatke.

Upotreba `Provider<V>` i `Provider<GenericWindow<T, V, D, K>>` umesto neposrednog ubrizgavanja zavisnosti omogućava odloženo i ponovljeno pribavljanje instanci. Komponenta korisničkog interfejsa zato može otvoriti novu instancu dijaloga pri svakom izboru komande „Novi”. Ako postoji više implementacija iste apstrakcije, anotacija `@Primary` određuje podrazumevanu.

Životni ciklus `GenericView` započinje metodom `afterNavigation` interfejsa `AfterNavigationObserver`, koju Vaadin poziva nakon završetka navigacije. Tada se učitava izbor iz parametra upita, na primer identifikator iz adrese `/faktura?id=42`, i inicijalizuju se meni i kontekstne komande. Metoda `populate` zatim pribavlja podatke iz repozitorijumske klase i popunjava tabelu. Asinhrono ažuriranje interfejsa podržano je pomoću `AsynHandlerInterface` i obrasca `UI.getCurrent().access(...)`, koji omogućava bezbedno objavljivanje promena iz pozadinske niti.

Klasa `GenericTable` prikazuje listu podataka. Ona nasleđuje `VerticalLayout` i sadrži komponentu `Grid<D>`, skup dugmadi za osnovne operacije i komponentu `MenuBar` za dodatne komande. Elementi `cssButton1` i `cssButton2` služe kao tačke za primenu pravila CSS, pa se njihov izgled može menjati bez izmene Java koda.

`GenericTable` sadrži strukture za upravljanje metapodacima kolona. Mapa `mapiranjeKolona` povezuje ime atributa sa oznakom kolone, `mapiranjeHeadera` čuva zaglavlja, a skup `iskljuceneKolone` navodi attribute koji se ne prikazuju. Generisani kod ove podatke dobija iz eksplicitnih kolona PIM projekcije: atributa, oznake, rednog broja, aliasa i podatka o podrazumevanom sortiranju. Tok generatora ih preuzima iz `ApplicationSpec` modela, a ne iz SPARQL rezultata u šablonu.

Komponenta `vlHideColumns` otvara meni za izbor vidljivih kolona. Izbor se čuva u sesiji, a po potrebi i u korisničkom profilu. Višestruko sortiranje uključuje se pozivom `grid.setMultiSort(true)`. Za velike liste koristi se `DataProvider` sa odloženim učitavanjem: metode `findAllLazy` i `findSizeLazy` iz repozitorijumske klase pretvaraju se u funkcije koje `Grid` poziva prema potrebi tokom pomeranja kroz sadržaj. Time se ograničava rast zauzeća memorije.

Klasa `GenericWindow`, čija je deklaracija prikazana u Listingu 5.6, nasleđuje komponentu `Dialog` biblioteke Vaadin i predstavlja modalni dijalog za unos i uređivanje pojedinačnog entiteta. Centralni mehanizam povezivanja polja korisničkog interfejsa i entitetskih atributa je `BeanValidationBinder<T>`, koji koristi Jakarta Bean Validation za automatsko sprovođenje validacionih anotacija (`@NotNull`, `@Size`, `@Pattern`, `@Min`, `@Max`) postavljenih na entitetu. Time se izbegava dvostruko deklarisanje validacionih pravila, jednom u entitetu, a drugi put u korisničkom interfejsu, što je tipičan izvor neusklađenosti u okvirima u kojima pravila nisu centralizovana.

```
public class GenericWindow<T extends AbstractEntity, V extends AbstractHome<T, D>,
    D extends AbstractDTO<T>, K extends AbstractEntity &
    OperaterEntityInterface>
    extends Dialog {

    protected BeanValidationBinder<T> binder;
    protected VerticalLayout vl;
    protected Button btnOk;
    protected Button btnCancel;
    protected Checkbox cbCU; // continue editing
    protected SoundProfile sounds;
    protected T selekcija;
    protected T unos;
    protected final Set<Component> addButtons = new HashSet<>();

    @Inject
    protected Provider<SystemBroadcaster> systemBroadcaster;

    // ...
}
```

Listing 5.6. Deklaracija generičke klase GenericWindow.

Polje `skok` omogućava navigaciju do povezane forme, dok `putanja` označava aktivni položaj u sistemu kartica. Instanca `SoundProfile` određuje zvučne signale za uspeh, grešku, upozorenje i obaveštenje. Polje `cbCU` omogućava da dijalog nakon čuvanja ostane otvoren radi unosa sledećeg entiteta, što ubrzava grupni unos podataka.

Mapa `entities` sadrži pomoćne povezane entitete potrebne komponentama `ComboBox` i poljima sa automatskim dovršavanjem. Skup `addButtons` je proširiv: konkretne implementacije mogu registrovati dodatnu dugmad za posebne radnje, kao što su „Štampaj”, „Pošalji e-poštom” i „Generiši izveštaj”.

Podrška za kartice u `GenericWindow` važna je zbog složenosti tipičnih poslovnih entiteta. Faktura, na primer, ima opšte podatke, stavke, plaćanja, beleške i prilog. Ovih pet logičkih celina prikazuje se na zasebnim karticama istog dijaloga, dok se entitet kao logička celina čuva u jednoj transakciji.

U transformacionom lancu generičke prezentacione komponente nastaju iz iste ontološke specifikacije. Dodavanje atributa domenskoj klasi zato može proizvesti odgovarajuću kolonu u tabeli i polje u dijalogu, pod uslovom da preslikavanja i šabloni zadovolje uslove verifikacije iz Poglavlja 4.

5.1.4 Sposobnosti kao ugovori između PIM-a i biblioteke

Aktuelna arhitektura ne modeluje ponovljive osobine entiteta i korisničkog interfejsa nizom posebnih vrsta artefakata. Modul `BAB Capability 2.0` uvodi sposobnosti kao ugovore nezavisne od platforme, a `BABj 2.0` ih svojstvom `mapping:realizesCapability` povezuje sa interfejsima, poljima i obrascima ponašanja modularne `BAB 2` biblioteke. Generator proverava zavisnosti, parametre, vlasništvo i sukobe sposobnosti pre pripreme prikaznog modela. Sposobnosti `AgentAssignable`, `WebPushSubscribable` i `WebAuthnAware` dodatno određuju da generisani POM uključi

module `bab-ai`, `bab-webpush` i `bab-webauthn`; njihov izostanak ostavlja projekat na obaveznom `bab-core` modulu.

Ovaj mehanizam povezuje ontološku specifikaciju sa generičkim klasama biblioteke bez unošenja Java detalja u PIM. Isti osnovni postupak obrade entiteta koristi se i za Actor: tip aktera može imati dodatne attribute, relacije i kompatibilne sposobnosti, dok se konkretni korisnici stvaraju i čuvaju kao podaci nastali tokom izvršavanja. Time se razdvaja šematski ugovor generatora od skupa konkretnih korisničkih instanci.

Domet ove tvrdnje je važan. Realizovan je skup sposobnosti deklarisan u aktuelnom ugovoru modula BAB Capability i BABj i pokriven testovima generatora; iz toga ne sledi da proizvoljan Java interfejs ili proizvoljna kombinacija osobina mogu automatski da se izvedu iz ontologije. Dodavanje nove sposobnosti zahteva usklađenu PIM deklaraciju, BABj realizaciju, generator i testove.

5.1.5 Modularna realizacija biblioteke BAB 2

BAB 2 je modularna izvršna biblioteka. Njena uloga ne menja matematički aparat niti osnovnu strukturu BAB PIM/BAB Mapping/BABj ontologija: reč je o ciljnoj realizaciji programskih ugovora, a ne o novoj domenskoj ontologiji ili promeni značenja kategorije C_{Core} . Aktuelni BABj profil neposredno koristi ovu biblioteku.

Maven reaktor BAB 2 obuhvata katalog zavisnosti `bab-bom`, obavezni modul `bab-core` i neobavezne module `bab-mail`, `bab-sms`, `bab-reporting`, `bab-rules`, `bab-ai`, `bab-saas`, `bab-webauthn`, `bab-signing` i `bab-webpush`. Jezgro zadržava standardni obrazac poslovne aplikacije prikazan u Listingu 5.7.

```
model -> projection -> home -> view/window
```

Listing 5.7. Osnovni sled slojeva u BAB poslovnoj aplikaciji.

Biblioteka sadrži i zajedničke ugovore za konfiguraciju, skladištenje, pokretanje, proveru spremnosti i razrešavanje javne adrese aplikacije. Neobavezni moduli zavise od jezgra i komuniciraju posredstvom usko definisanih SPI ugovora; jezgro ne zavisi od njihovih konkretnih implementacija. Time se funkcije kao što su elektronska pošta, izveštavanje, pravila, veštačka inteligencija, višekorisnička SaaS podrška, WebAuthn, potpisivanje i Web Push mogu uključivati prema potrebama aplikacije, bez njihovog obaveznog prisustva u svakoj BAB aplikaciji.

Proverena razvojna grana BAB 2 koristi Javu 26, Vaadin 25.2.6 i standardnu distribuciju WildFly 41.0.1.Final sa Jakarta EE 11. Podrška za baze podataka ograničena je na SQL Server i Azure SQL, dok su rezervne kopije i obnavljanje podataka odgovornost infrastrukture u kojoj je aplikacija postavljena. Konfiguracija je nezavisna od načina raspoređivanja i učitava se iz sistemskih svojstava, promenljivih okruženja i odgovarajućih datoteka sa poverljivim podacima, spoljne konfiguracione datoteke i bezbednih podrazumevanih vrednosti.

Aktuelni profil generatora realizuje ovu modularizaciju kroz BABj. BAB Mapping 2.0 uvodi pojam `RuntimeModule` i svojstva koja opisuju zavisnosti i Maven koordinate,

BABj navodi raspoložive BAB 2 module, a PsmResolver iz obaveznog jezgra i izabranih realizacija sposobnosti izračunava najmanji potreban skup zavisnosti. Generisani POM uvozi bab-bom, uvek uključuje bab-core i dodaje neobavezne module samo kada ih PIM zahteva. Minimalna aplikacija, primer sa sposobnostima i PostOffice izgrađeni su kao WAR projekti za Javu 26, Vaadin 25.2.6 i Jakarta EE 11. Ova provera potvrđuje tok generatora za BAB 2, ali ne predstavlja dokaz postavljanja u stvarno radno okruženje ili izvršenja poslovnih scenarija na serveru.

5.2 BABj: Java/Jakarta EE/Vaadin realizacija

5.2.1 BABj 2.0 kao profil realizacije modularnog PIM-a

BABj 2.0 je platformski specifična ontologija kojom se elementi modularnog BAB PIM-a 2.0 realizuju u skupu tehnologija Java/Jakarta EE/Vaadin. Ona uvozi BAB Mapping 2.0, a centralna individua babj:JavaJakartaVaadin svojstvom mapping:realizesModule navodi osam PIM modula koje profil podržava: Core, View, Behavior, Validation, Security, Service, Classification i Capability. Time se fizička podela PIM-a na module povezuje sa jednim proverljivim ugovorom realizacije.

BABj sadrži instance klase rečnika preslikavanja i navodi činjenice koje generator može da proveri i upotrebi: preslikavanja tipova podataka, ciljne bazne tipove, anotacije, interfejse, uloge artefakata, prioritete, imena šablona i podržane sposobnosti. Svaka realizaciona individua svojstvom mapping:targetsStack navodi ciljni skup tehnologija. PIM zato može biti logički konzistentan, a da mu ipak nedostaje podržano vezivanje specifično za dati skup tehnologija; generator odvojeno proverava PIM semantiku i potpunost BABj ugovora.

Modularne zavisnosti potrebne za izvršavanje predstavljene su individuama klase mapping:RuntimeModule. Njihove Maven koordinate pripadaju BABj profilu, a ne platformski neutralnom PIM-u. Skup tehnologija babj:JavaJakartaVaadin obavezno zahteva bab-core; realizacije sposobnosti AgentAssignable, WebPushSubscribable i WebAuthnAware zahtevaju bab-ai, bab-webpush i bab-webauthn. PsmResolver iz tih veza izračunava najmanji potreban skup modula i uređuje ga u unapred utvrđenom redosledu pre pripreme šablona za Maven.

Tipovi podataka vezuju se za standardne XSD tipove. xsd:string realizuje se kao java.lang.String, xsd:boolean kao java.lang.Boolean, xsd:decimal kao java.math.BigDecimal, datumi i vreme kao odgovarajući tipovi paketa java.time, a celobrojni tipovi kao Long ili Integer. Ova tabela nije ugrađena u PIM niti u FreeMarker šablone, već se razrešava iz grafa PSM-a.

Profil sadrži i specifikacije ključnih tipova BAB i Jakarta okruženja: AbstractEntity, AbstractHome, GenericView, GenericWindow, JAX-RS Application, EJB Stateless, Java Runnable i JPA anotacije. Za potpuno kvalifikovano ime koristi se termin woc:hasFullyQualifiedNamed. BABj koristi taj IRI, ali ne uvozi niti učitava celu WoC

ontologiju, pa se ne tvrdi da profil ontološki opisuje sva polja, metode i tela metoda generisanog projekta.

PIM element može svojstvom `mapping:hasImplementationMapping` izričito da izabere jednu ili više realizacionih individua. Kada izbor nije naveden, koristi se podrazumevano preslikavanje odgovarajuće uloge. Kandidati se uređuju prema `mapping:mappingPriority`, a zatim prema IRI-ju, dok se kompatibilne anotacije i interfejsi mogu objediniti u profil na determinističan način.

5.2.2 Perzistentni sloj, neutralne relacije i sposobnosti entiteta

Modul BAB Core 2.0 relaciju modeluje klasom `core:Relation` koja ima tačno dva kraja `core:RelationEnd`. Svaki kraj navodi entitet koji ga deklariše, ciljni entitet, naziv uloge, kardinalnost, suprotni kraj i navigabilnost. PIM ne sadrži tip veze u JPA, `mappedBy`, spojnu kolonu, strategiju učitavanja niti druge fizičke nazive zavisne od platforme. Generator te vrednosti izvodi deterministički iz dva validirana kraja i tek ih u BABj fazi prevodi u `OneToOne`, `OneToMany`, `ManyToOne` ili `ManyToMany` ugovor.

Za svaki konkretan domenski entitet generiše se Java klasa koja nasleđuje `AbstractEntity`. Pored atributa i relacija, klasa dobija samo one anotacije i interfejse koje nalažu izabrani profil i sposobnosti. JPA anotacije se postavljaju u skladu sa pristupom preko svojstava, odnosno na pristupne metode. Validaciona pravila prevode se u odgovarajuće metapodatke samo za podržani skup pravila; proizvoljan OWL aksiom ne postaje automatski Java validacija.

Modul Capability razdvaja ugovore za višekratnu upotrebu od posebnosti pojedinačnih šablona. Sposobnosti entiteta, pogleda i prozora vezuju se sa BABj realizacijama preko `mapping:realizesCapability`. Generator proverava njihove zavisnosti, sukobe i parametre, a zatim generiše potrebna polja, interfejse i fragmente ponašanja. Na primer, `GeneratedCode` zahteva `generatedCodePrefix`, `FileBacked` zahteva `fileStorageLocation`, dok se `DateRanged` i `DurationTracked` ne mogu kombinovati zbog nekompatibilnih povratnih tipova metode `getDuration()` u ciljnoj BAB biblioteci.

`core:Actor` prolazi kroz isti tok vođen sposobnostima kao ostali entiteti. Konkretan primarni akter može imati dodatne attribute, relacije i kompatibilne sposobnosti, a BABj profil zahteva tačno jednog takvog aktera. Obavezni operatorski ugovor daje `OperaterEntityInterface` i `OperaterHomeInterface`, dok se funkcionalnosti agenata i tehnologije Web Push dodaju samo preko sposobnosti `AgentAssignable` i `WebPushSubscribable`. Time običan akter više ne zahteva uključivanje modula `bab-ai` i `bab-webpush`. Actor je tip entiteta u šemi, dok su konkretni korisnici podaci dostupni tokom izvršavanja i nisu ugrađeni u generisani kod kao ontološke individue.

Projekcije određuju sadržaj DTO-a i upita home sloja. Direktna projekciona kolona može upućivati na atribut posmatranog entiteta, dok relaciona kolona navodi imenovanu putanju kroz navigabilne krajeve relacija. Generator proverava kontinuitet putanje, izvodi potrebne klauzule JOIN i priprema Java izraz pre primene šablona.

5.2.3 Aplikacioni, servisni i procesni sloj

Home sloj nasleđuje `AbstractHome<Entity, DTO>` i povezuje perzistentne entitete sa projekcionim DTO objektima. Generator priprema JPQL izraz `SELECT`, konstruktor DTO-a, potrebne klauzule `JOIN` i stabilan redosled kolona. Time šablon ostaje mehanizam za oblikovanje izlaza, a odluke o strukturi upita donose se u razrešivaču.

Servisni modul opisuje operacije i krajnje tačke nezavisne od platforme. BABj ih realizuje kao zaštićene JAX-RS resurse i aplikacionu klasu. Putanje, Java imena, HTTP metode i zavisnosti proveravaju se pre generisanja. Za poslovna pravila koja prevazilaze podržane generičke operacije ostavljaju se jasno definisana mesta za proširenje.

Behavior modul opisuje procese, stanja, prelaze i zadatke. U Java profilu iz njega nastaje osnovna procesna struktura sa uslovima i pozivima zadataka koji se mogu neposredno izvesti iz grafa PIM-a. Korisnički i servisni zadaci dobijaju jasno označena mesta za dopunu domenskog ponašanja. Generator time obezbeđuje izvršivu strukturu, ali ne tvrdi da iz strukturnog modela automatski izvodi celokupnu poslovnu logiku.

Classification modul podržava zatvorenu klasifikaciju tokom izvršavanja za ograničeni skup OWL `equivalentClass` izraza. Generisani klasifikator koristi kod kategorije, prioritet i proveru potpunosti ulaznih podataka i vraća objašnjenje ispunjenih uslova. Ovo ponašanje pripada generisanoj aplikaciji; ne treba ga poistovetiti sa samostalnim zaključivanjem pomoću rezonera HermiT niti proširivati na proizvoljne OWL konstrukcije.

5.2.4 Prezantacioni i bezbednosni sloj i omotač aplikacije

View modul definiše poglede, projekcije, forme, funkcije i raspored. BABj ih realizuje klasama izvedenim iz `GenericView` i `GenericWindow`, dok generator priprema rute, tipske parametre, kolone, komande i autorizacione podatke. Sposobnosti pogleda i prozora dodaju samo kompatibilne interfejsa i obrasce ponašanja, čime se izbegava množenje posebnih šablona za svaku kombinaciju osobina.

Security modul definiše uloge i dozvole, koje se prevode u konstante i autorizacione anotacije generisanog projekta. Generator pravi i minimalni omotač aplikacije: infrastrukturu za prijavljivanje, početnu navigaciju, sloj operatera i sesije, kao i pomoćne adaptere za odjavljivanje, portal, profil i registre vidžeta. Ako PIM nema domenske poglede, neutralni početni pogled obezbeđuje početnu rutu.

Profil Acceptance pri prvom raspoređivanju može da napravi početne razvojne naloge. Lozinka se uzima iz promenljive okruženja `OWLBAB_BOOTSTRAP_PASSWORD`, dok je `changeit` samo rezervna vrednost za razvojno okruženje i ne predstavlja politiku radnog okruženja. Ova pogodnost za početno postavljanje mora se razlikovati od bezbednosne konfiguracije ciljnog sistema.

Generisani profil je potpun Maven WAR projekat za Java 26, modularni BAB 2.0.0, Vaadin 25.2.6 i Jakarta EE Web 11. Maven POM uvozi `bab-bom`, uključuje obavezni

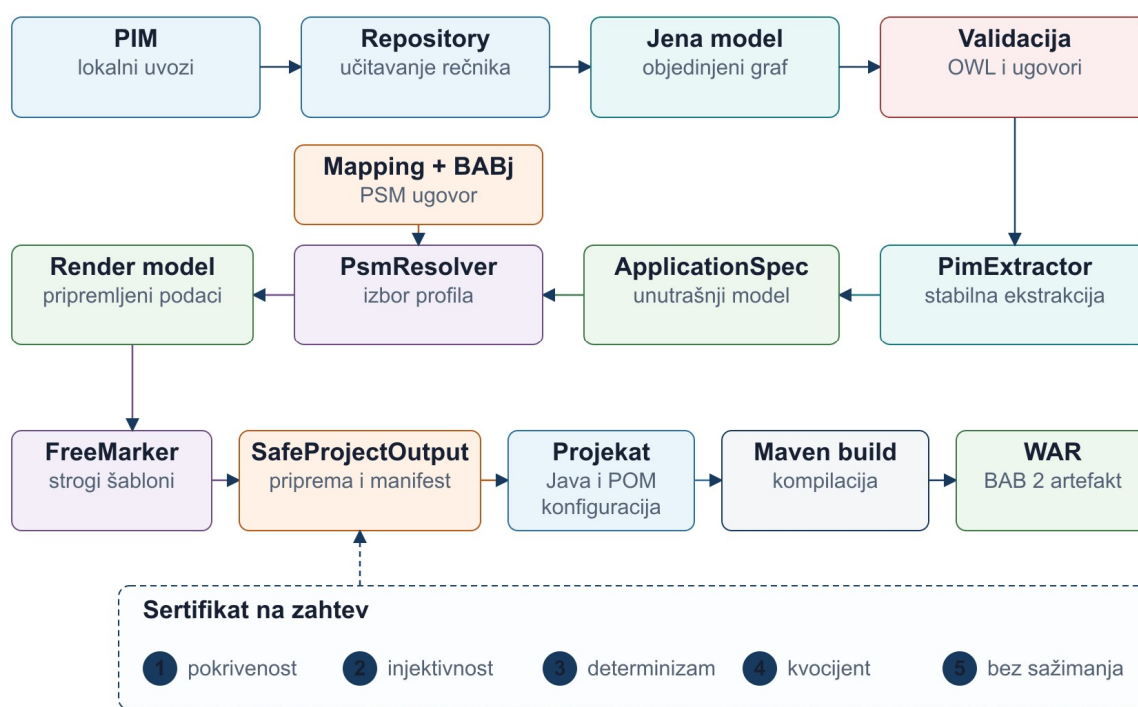
bab-core i dodaje samo neobavezne module koje zahtevaju izabrane sposobnosti. Ove verzije predstavljaju trenutni ugovor o izlazu generatora u posmatranoj razvojnoj verziji, a ne opštu obavezu formalnog modela. Drugi skup tehnologija zahtevao bi sopstvenu PSM ontologiju, preslikavanja, razrešavanje, šablone, validatore i regresione testove.

5.3 BAB generator: transformator modela

5.3.1 Operativni transformacioni lanac

BAB generator realizuje transformaciju od aplikacionog PIM-a zasnovanog na modularnom BAB PIM-u 2.0 do potpunog Maven WAR projekta. Operativni tok prikazan je na Slici 5.2, a može se sažeti sledećom kompozicijom:

PIM + moduli BAB PIM 2.0 → validirani ApplicationSpec → BAB Mapping 2.0 + BABj 2.0 → priprema artefakata + FreeMarker → privremeni izlaz + manifest.



Slika 5.2. Operativni tok generatora od modularnog PIM-a do WAR artefakta.

Ovaj inženjerski tok odgovara formalnoj kompoziciji iz Poglavlja 4, ali je ne treba poistovetiti sa njom. Kategorijalni model određuje strukturu i uslove očuvanja, dok implementacija sadrži konkretne odluke o učitavanju RDF-a, proveru ugovora, imenovanju, izboru šablona i pisanju datoteka.

Komandni i grafički interfejs koriste isti transformacioni servis. Time se validacija i generisanje sprovode po istim pravilima, nezavisno od načina pokretanja.

Prvi korak je učitavanje aplikacione ontologije i njenih lokalnih uvoza, zajedno sa ugrađenim modulima BAB PIM 2.0. BAB Core sadrži osnovne domenske pojmove, dok su View, Behavior, Validation, Security, Service, Classification i Capability zasebni PIM moduli. BAB Mapping 2.0 sadrži apstraktni rečnik preslikavanja, a BABj 2.0 njegove Java realizacije individue. BABj uvozi BAB Mapping; za potpuno kvalifikovano ime koristi `woc:hasFullyQualifiedName`, ali ne učitava celu WoC ontologiju.

Drugi korak čine odvojene provere. Hermit proverava aplikacioni PIM i njegovo zatvorenje po uvozima. Zatim namenski Java validatori proveravaju uslove kompletnosti i jednoznačnosti koje OWL otvoreni svet ne može da nametne. BABj se proverava zasebno kao PSM ugovor. Na taj način neuspeh precizno pokazuje da li je problem u logičkoj konzistentnosti, strukturi aplikacionog modela ili nedostajućoj realizaciji ciljnog skupa tehnologija.

Treći korak je ekstrakcija u unutrašnji model `ApplicationSpec`. On sadrži uređene i nepromenljive specifikacije aplikacije, entiteta, atributa, dvostranih relacija, pogleda, formi, servisa, uloga, procesa, klasifikacija, sposobnosti i eksplicitnih preslikavanja. Uređivanje se obavlja prema stabilnim ključevima, a ne prema redosledu RDF trojki. Zbog toga serijalizacija istog grafa drugačijim redosledom ne menja generisani rezultat.

Četvrti korak obavlja PSM razrešavač. On iz BAB Mapping/BABj ugovora bira preslikavanje, ciljni tip, anotacije, interfejse, ulogu artefakta, sposobnosti i šablon. Za modularni BAB 2 odvojeno računa najmanji potreban skup modula potrebnih tokom izvršavanja: polazi od obaveznog `bab-core` i dodaje module koje zahtevaju realizacije sposobnosti iz PIM-a. Zatim iz platformski nezavisnog PIM-a izvodi fizička imena, Java identifikatore, JPA vrste i vlasništvo relacija, rute, REST metode, projekcione upite i druge vrednosti koje moraju biti potpuno pripremljene pre prikaza šablona.

Peti korak prevodi `ApplicationSpec` i razrešeni PSM u prikazni model sastavljen od običnih, bezbedno obrađenih vrednosti. FreeMarker ne dobija Jena graf niti objekte ontologije i ne izvršava upite. Time su ekstrakcija značenja i prikaz izvornog koda razdvojeni, što olakšava testiranje oba koraka.

Poslednji korak generiše celokupan projekat u privremenom radnom direktorijumu. Tek kada su svi šabloni uspešno obrađeni, rezultat može da zameni prethodno upravljane datoteke. Ovaj redosled sprečava da neuspeh usred generisanja ostavi izlazni projekat u polovičnom stanju.

5.3.2 Ekstrakcija pomoću Apache Jena i validacija ugovora

Apache Jena koristi se za učitavanje RDF/OWL grafa, razrešavanje resursa i navigaciju kroz njegove veze. Generatorski tok ne zasniva se na nizu SPARQL upita. Namenske komponente neposredno obilaze model, proveravaju tipove i svojstva i grade `ApplicationSpec`. Grafički korisnički interfejs sadrži posebnu SPARQL konzolu za pregled učitanih modela, ali rezultat takvog upita nije ulaz u FreeMarker niti uslov za generisanje.

Ekstraktor podržava ABox i TBox stil modelovanja. U prvom slučaju entiteti, atributi i ostali elementi predstavljeni su individuama odgovarajućih PIM klasa. U drugom slučaju aplikacioni tipovi mogu biti klase izvedene iz BAB tipova. Podaci oba oblika normalizuju se u isti unutrašnji model. Time se dozvoljava različit način uređivanja PIM-a bez udvostručavanja narednih faza generatora.

Validacija se sprovodi pre ekstrakcije vrednosti potrebnih šablonima. Minimalni ugovor zahteva tačno jednu *BusinessApplication* i tačno jednog konkretnog primarnog Actor elementa. Dodatne provere obuhvataju ispravnost identifikatora, podržane tipove podataka, jedinstvena imena, tačno dva uzajamno povezana kraja svake relacije, projekcione putanje, redosled kolona, postojanje podrazumevanog pogleda, servisne operacije, procesna stanja, usklađenost sposobnosti i dozvoljenog vlasnika svake sposobnosti, PSM preslikavanja, deklaracije modula potrebnih tokom izvršavanja i jednoznačnost njihovih Maven koordinata.

Takav pristup je namerno stroži od same OWL konzistentnosti. Na primer, izostanak kraja relacije u otvorenom svetu ne znači da taj kraj ne postoji, ali generator ne može da izvede JPA anotaciju dok oba kraja nisu eksplicitno poznata. Slično tome, više jednakih rednih brojeva projekcionih kolona ne mora učiniti ontologiju logički nekonzistentnom, ali daje dvosmislen prikaz. Zato se ovi uslovi tretiraju kao ugovor izvršive transformacije.

HermiT se pokreće nad aplikacionim PIM-om i zatvorenjem njegovih uvoza iz BAB PIM-a 2.0. BAB Mapping 2.0 i BABj 2.0 ne uključuju se u isti graf nad kojim je izvedeno zaključivanje, već se kao PSM ugovor proveravaju posebnim proverama tipova, modula, sposobnosti, podrazumevanih preslikavanja, ciljnih skupova tehnologija i šablona. Puna WoC ontologija nije deo ovog toka. Time se izbegava pogrešan zaključak da OWL konzistentnost objedinjene PIM/PSM strukture predstavlja dokaz izvršivosti generatora.

5.3.3 PSM razrešavanje i FreeMarker šabloni

PSM razrešavač predstavlja operativni deo Java instance formalnog preslikavanja F_S . Za svaki artefakt obrađuje odgovarajuće realizacione individue. Eksplicitni izbor preko `mapping:hasImplementationMapping` mora da označava poznatu deklaraciju sa odgovarajućim ciljnim skupom tehnologija. Nepoznata ili nekompatibilna deklaracija izaziva grešku. Podrazumevani profil koristi se kada nema eksplicitnog kandidata za potrebnu ulogu artefakta. Prioritet i IRI daju stabilan redosled; kompatibilne anotacije, interfejsi i realizacije sposobnosti mogu se objediniti, dok se konfliktni ciljni tipovi, šabloni i višeznačna preslikavanja tipova podataka odbijaju. CLI i desktop tok koriste i aplikaciono deklarirana preslikavanja.

Razrešeni profil sadrži ime šablona i potpuno kvalifikovano ime ciljnog tipa. Pored toga može sadržati anotacije i interfejse koje treba generisati. PSM razrešavač dopunjava profil izračunatim podacima. Tako se, na primer, Java tip atributa čita iz `realizesDatatype`, dok se ime polja, ime tabele i JPA parametri izvode iz modela po determinističkim pravilima. Maven koordinate obaveznog i opcionih BAB 2 modula čitaju se iz individua `RuntimeModule`; one se ne određuju u FreeMarker šablonu.

Generator čuva normalizovani aplikacioni model i razrešeni strukturni deo PSM-a kao zasebne međuartefakte. PSM zapis obuhvata ugovor entiteta, atributskih i relacionih mesta, profila i potrebnih modula; njime nisu obuhvaćene sve odluke generatora. Identifikatori stanja, zadataka i prelaza ostaju sačuvani u aplikacionom modelu, što omogućava praćenje njihovog porekla. Šablon entiteta koristi nadtip određen razrešenim profilom.

FreeMarker šabloni podeljeni su na projektne i Java šablone. Projektni deo sadrži Maven POM, konfiguraciju perzistencije i CDI-ja, README i metapodatke. Java deo obuhvata entitete, DTO i home klase, poglede, forme, operatorski i sesijski sloj, REST resurse, procese, klasifikaciju i pomoćne infrastrukturne adaptore.

Šabloni rade u strogom režimu. Nedostajući šablon ili promenljiva prekidaju generisanje pre zamene postojećeg projekta. Oni ne sadrže poslovno zaključivanje niti pristup ontologiji; očekuju da su sva imena, liste, uvozi, anotacije i fragmenti izraza već pripremljeni. Ova podela omogućava zasebne testove prikaznog modela i šablona i sprečava da se ista odluka donosi na više mesta.

U terminima formalizacije, H_s obuhvata programsku pripremu projektnih struktura, izbor i strogu obradu šablona i formiranje konačnih artefakata. FreeMarker je njegov završni deo za generisanje tekstualnog izlaza, a ne ceo funktor. Ni FreeMarker ni uspešna generacija sami po sebi ne dokazuju da H_s čuva sve jednačine prezentovane kategorije; za to su potrebne provere kompletnosti, injektivnosti, kompatibilnosti sa kvocijentom i odsustva neželjenog sažimanja.

5.3.4 Generisani projekat i bezbedna ponovna generacija

Generator proizvodi potpun Maven WAR projekat. Ciljnu platformu čine Java, Jakarta EE, Vaadin i modularna biblioteka BAB 2. Projekat uključuje obavezni modul bab-core i opcione module koje zahtevaju izabrane sposobnosti. Pored izvornog koda nastaju konfiguracija perzistencije, konfiguracija komponenti i metapodaci o aplikaciji i njenim zavisnostima. Verzije korišćene u evaluaciji navedene su u Prilogu G.

Pri ponovnom generisanju potrebno je razlikovati datoteke kojima upravlja generator od ručno dodatih delova aplikacije. Generator vodi evidenciju sopstvenog izlaza i otkriva naknadne izmene sadržaja. Ručno izmenjenu upravljaju datoteku zamenjuje samo na izričit zahtev korisnika, dok datoteke izvan te evidencije ostavlja netaknutim. Ovaj mehanizam omogućava kontrolisano dopunjavanje generisanog projekta.

Celokupan novi izlaz priprema se pre zamene prethodnog. Ako obrada nekog šablona ne uspe, prethodni projekat ostaje raspoloživ. Evidencija upravljanih datoteka i odložena zamena zajedno sprečavaju delimično ažuriranje projekta i gubitak ručnih dopuna.

Skup izlaznih artefakata prilagođava se konkretnom modelu. REST komponente, na primer, nastaju kada PIM sadrži servisne specifikacije, dok se nepotrebne pomoćne komponente i prazne projekcije ekstenzije izostavljaju.

Izgradnja generisanog projekta proverava njegovu kompilaciju i pakovanje. Za izvršavanje aplikacije potrebni su odgovarajući Jakarta EE server, izvor podataka i podešavanje ciljnog okruženja. Primena u stvarnom radu dodatno zahteva upravljanje promenama šeme baze, bezbednosnu konfiguraciju i proveru poslovnih scenarija. Ovi zadaci pripadaju integraciji aplikacije i nisu obuhvaćeni samom generacijom.

5.3.5 Ponovljivost, testiranje i granice prenosivosti

Ponovljivost se zasniva na sortiranom unutrašnjem modelu, stabilnom izboru preslikavanja, determinističkom imenovanju i proveru sadržaja izlaza. Skup testova proverava ontološke rečnike, OWL konzistentnost i zadovoljivost, ABox i TBox ulaz, relacije, projekcije putanje, procese, klasifikacije, stabilnu ekstrakciju, ponovljenu generaciju, zaštitu lokalnih izmena, BAB 2 koordinate modula, obavezni profil samo sa modulom bab-core, selektivno uključivanje opcionih modula i zabranu sposobnosti namenjenih samo akteru na običnom entitetu.

Referentni PIM-ovi proveravaju domenske entitete, neutralne relacije, projekcije, home i UI sloj, operatorsku infrastrukturu, servise, klasifikacije, sposobnosti i procesni skelet. Aplikaciona ontologija može biti podeljena u više datoteka i uvoziti samo potrebne PIM module. Evaluacija u Poglavlju 6 razlikuje osobine potvrđene postojećim testovima i sertifikatom od osobina koje tek treba empirijski proveriti na dodatnim studijama slučaja.

Aktuelna implementacija ne sadrži alternativni TypeScript ili Python generator. Arhitektura PSM profila pokazuje gde bi se novi skup tehnologija uključio, ali za njegovu stvarnu podršku morali bi se obezbediti odgovarajući tipovi, preslikavanja, PSM razrešavanje, šabloni, validatori i testovi ciljnog projekta. Zbog toga se u radu razlikuju arhitektonska proširivost i empirijski potvrđena prenosivost. Prva je svojstvo dizajna, dok je druga za sada potvrđena samo za BABj profil.

5.3.6 Sertifikat transformacione specifikacije

Pored redovnog skupa testova, generator na zahtev izdaje sertifikat ugrađene transformacione specifikacije. On povezuje rezultat provere sa tačno određenim ontologijama, preslikavanjima, šablonima, testnim korpusom i implementacijom generatora. Time se omogućava ponavljanje postupka nad istim skupom artefakata.

Sertifikat beleži rezultate pet provera iz Poglavlja 4: pokrivenost izabranih generišućih relacija, injektivnost ciljnih oznaka uz deklarisanu normalizaciju, determinizam dva pokretanja nad fiksiranim korpusom, tipsku kompatibilnost normalizacija izvornog zapisa i odsustvo neželjenog sažimanja identiteta artefakata. Svako ponavljanje korpusa uključuje i nezavisnu strukturnu proveru stvarnog Java izlaza.

Obuhvat provere deklarisan je u rečniku BAB Mapping 2.0. Profil bira posmatrani šematski deo ontologije, strukturna preslikavanja određuju ciljne oznake, a pravila normalizacije određuju kanonsku orijentaciju alternativnih izvornih zapisa i tipove njihovih krajeva. Time se formalne obaveze povezuju sa izvršivim proverama.

Sertifikat nije dokaz semantičke ekvivalencije OWL modela i Java programa. Njegova važnost je uža i proverljiva: za identifikovane ulaze, verziju generatora i izabrani šematski deo daje ponovljiv zapis da su konkretne inženjerske obaveze zadovoljene. Promena relevantne ontologije, profila, šablona ili generatora zahteva ponovno izdavanje sertifikata.

Zasebna provera postojećeg izlaza poredi konkretan PIM i razrešeni profil sa deklaracijama pročitanim iz generisanog Java koda. Obuhvata entitete i njihove članove, strukturu pristupnih metoda, ciljeve relacija, izabrane anotacije i direktne zavisnosti od BAB modula. Nalaz razlikuje prihvaćen izlaz, utvrđenu neusaglašenost i konstrukciju izvan podržanog ugovora. Sačuvana korespondencija modelskih elemenata i programskih deklaracija omogućava da se utvrdi mesto odstupanja. Poslovna tela metoda, izvršavanje procesa, ponašanje biblioteka i puna prirodost dorada nisu obuhvaćeni ovom proverom.

Format zapisa, oznaka strukturnog ugovora i podaci potrebni za ponavljanje provera navedeni su u Prilogu G. Njihova uloga je identifikacija i reproduktivnost eksperimenta; granice zaključka određuje obuhvat same provere.

5.4 Operativni PSM ugovor i sledljivost realizacije

5.4.1 Preslikavanje ključnih BAB apstrakcija

Ključne apstrakcije BAB biblioteke nisu u BABj 2.0 predstavljene potpunim ontološkim opisom svih polja i metoda. Umesto toga, BABj za njih navodi podatak o ciljnom tipu i preko BAB Mapping 2.0 povezuje ga sa ulogom artefakta, sposobnošću i šablonom koji taj ugovor realizuje. Takav opis je uži od potpunog ontološkog opisa izvornog koda, ali neposredno odgovara informacijama koje trenutni generator može da proveriti i upotrebi.

Za domenski entitet podrazumevano preslikavanje bira `rs.co.bora5.programs.bab.model.AbstractEntity`. Perzistentna varijanta dodatno zahteva JPA anotacije `jakarta.persistence.Entity` i `jakarta.persistence.Table`. Akter koristi isti zajednički tok entiteta, DTO-a i home sloja, dok operatorski ugovor proizlazi iz njegovih sposobnosti. Za home sloj cilj je `rs.co.bora5.programs.bab.session.AbstractHome`, za pogled `rs.co.bora5.programs.bab.front.views.GenericView`, a za formu `rs.co.bora5.programs.bab.front.windowes.GenericWindow`.

Servisno preslikavanje koristi JAX-RS Application kao platformsku specifikaciju servisnog sloja, procesno preslikavanje EJB anotaciju `Stateless`, a preslikavanja zadataka Java tip `Runnable`. Ove veze ne znače da PIM klase nasleđuju Java klase. One određuju realizacioni profil koji generator primenjuje pri generisanju ciljnog artefakta.

`AbstractHome` je dobar primer podele odgovornosti. PIM projekcija određuje koje vrednosti treba prikazati i preko kojih relacionih putanja se do njih dolazi. BABj preslikavanje određuje baznu klasu i šablon home artefakta. PSM razrešavač izračunava tipove entiteta i DTO-a, JPQL select izraz i potrebne join delove. FreeMarker zatim

generiše konačnu Java klasu. Nijedan od ova četiri sloja sam ne sadrži potpunu realizaciju; ona nastaje njihovom kontrolisanom kompozicijom.

Isti obrazac važi za korisnički interfejs. Moduli View, Security i Capability određuju pogled, projekciju, formu, funkcije, uloge i sposobnosti. BABj bira `GenericView` ili `GenericWindow` i pripadajući šablon. Generator priprema rutu, parametre Java tipova, kolone i podatke o ovlašćenjima. Na taj način se poslovna specifikacija ne opterećuje zavisnostima od Vaadin klase, dok PSM ostaje dovoljno konkretan da se može proveriti pre generisanja.

Ovakva organizacija podržava i specijalizovana preslikavanja. Element aplikacije može svojstvom `mapping:hasImplementationMapping` izričito da upućuje na individuu klase `mapping:ImplementationMapping` ili njene potklase. Individua mora pripadati odgovarajućoj hijerarhiji, odnositi se na izabrani skup tehnologija i imati ulogu artefakta koja se poklapa sa kontekstom. Generator ne prihvata proizvoljnu vezu između PIM elementa i šablona, jer bi time preslikavanje izgubilo značenje na nivou tipova i realizacije.

5.4.2 Uloga WoC rečnika i granice ontološkog opisa koda

U BABj ontologiji termin `woc:hasFullyQualifiedName` koristi se za povezivanje PSM specifikacije sa potpuno kvalifikovanim imenom Java tipa. BABj koristi taj termin, ali ne uvozi niti učitava celu WoC ontologiju. Aktuelni generator iz PSM grafa čita samo podatke potrebne za razrešavanje profila i ne predstavlja svaki paket, konstruktor, metod, parametar i telo metoda kao poseban element generisane aplikacije.

Odluka da se ne modeluje celokupan ciljani kod zasniva se na odnosu troška i koristi. Potpuni ontološki prikaz Java programa bio bi znatno veći od PIM-a, zahtevao bi usklađivanje sa svakom izmenom šablona i otvorio bi dodatna pitanja pripadnosti profilima OWL-a. Za potrebe generatora dovoljni su podatak o ciljnom tipu, realizaciona uloga, anotacije, interfejsi, šablon i izračunate vrednosti. Struktura generisanog koda proverava se testovima i kompilacijom, a ne ponovnim pretvaranjem čitavog Java programa u OWL graf.

Ovo ne umanjuje značaj `CodeOntology` i srodnih rečnika kao osnove za analizu koda. Njihova moguća buduća primena obuhvata dvosmernu sledljivost, upite nad postojećim kodom i poređenje generisanih i ručno izmenjenih struktura. Međutim, takva primena nije deo trenutnog lanca transformacija i zato se u disertaciji ne prikazuje kao podržana mogućnost.

Trenutna sledljivost postiže se neposrednijim mehanizmima. Unutrašnji model čuva IRI-je izvornih PIM elemenata, sposobnosti i izabranih individua preslikavanja. Generisani `application-metadata.json` beleži metapodatke o procesima, klasifikacijama i PSM poreklu, dok `.owlbab-manifest` evidentira datoteke kojima upravlja generator i njihove sažetke. Sertifikat transformacije dodatno čuva otiske ulaznih komponenti i rezultate pet

provera šematskog preslikavanja. Zajedno oni omogućavaju da se utvrde poreklo artefakta, primenjeni profil i stanje u odnosu na poslednje generisanje.

Važno je razlikovati tri vrste sledljivosti. Sledljivost modela i profila povezuje PIM element sa izabranim preslikavanjem. Sledljivost profila i artefakta povezuje preslikavanje sa ulogom i šablonom. Izlazna sledljivost povezuje generisanu datoteku sa manifestom i metapodacima. Trenutni generator podržava ove veze, ali ne tvrdi da može iz proizvoljno izmenjenog Java koda rekonstruisati PIM.

U formalnom okviru iz Poglavlja 4 ciljna kategorija koda zato se izvodi iz specifikacija generisanih artefakata i njihovih strukturnih zavisnosti. Ona nije ontologija svake sintaksne konstrukcije Java programa. Ovim ograničenjem zadržava se preciznost tvrdnje o šematskom očuvanju: posmatraju se odabrani elementi i veze koje transformacioni ugovor izričito prenosi, a ne potpuna operativna semantika generisane aplikacije.

5.5 Mehanizam unapređenja pomoću veštačke inteligencije

5.5.1 AbstractAI: ontološki okvir

Sloj agenata u okviru BAB-a opisan je posebnim modelom AbstractAI, namenjenim autonomnim komponentama. Ovaj model nije deo važećeg ugovora generatora BAB Core/BABj i generator ga trenutno ne obrađuje. U nastavku se zato prikazuje kao zaseban sloj u okviru BAB-a i pravac njegovog povezivanja sa generisanim aplikacijama. Hijerarhija klasa u AbstractAI prikazana je u Listingu 5.8.

```
bab:Agent, apstraktna klasa entiteta sa ulogom autonomnog dejstvenika
    bab:StatefulAgent, agent sa nekim oblikom internog stanja
bab:MemoryAwareAgent, agent sa pamćenjem razgovora (LangChain4j ChatMemory)
    bab:RagAwareAgent, agent sa RAG (retrieval-augmented generation)
sposobnošću

bab:AgentEvent, događaj koji agenti razmenjuju
    bab:LogEvent
    bab:EntityEvent
    bab:AgentErrorEvent
    bab:AgentMessageEvent

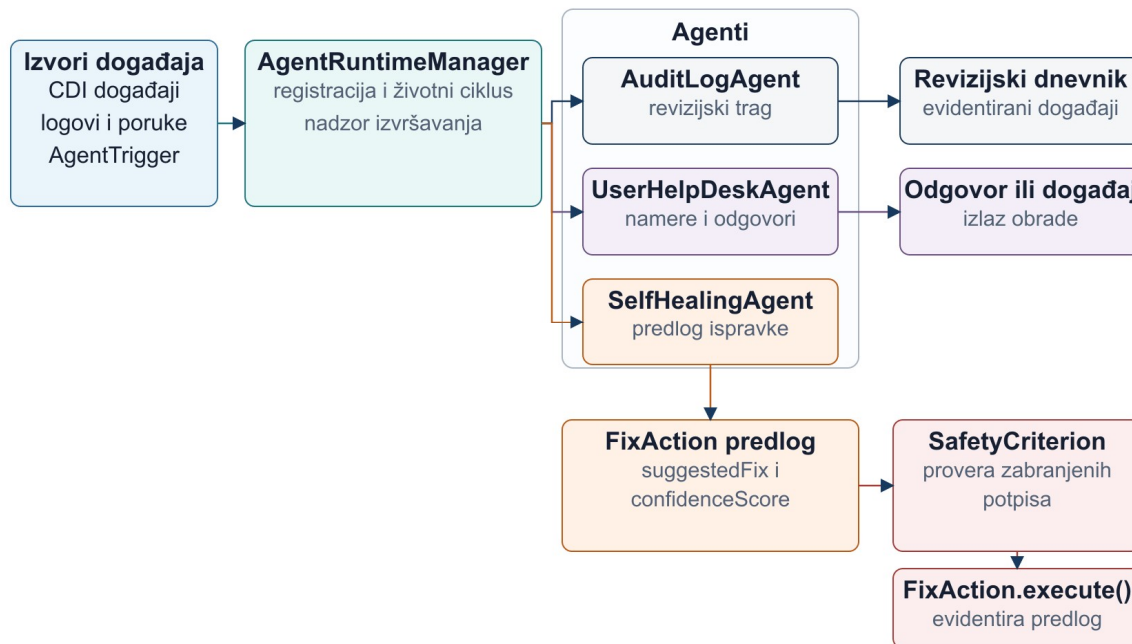
bab:AgentAction, radnja koju agent može izvršiti
    bab:CompositeAction
    bab:ConditionalAction
    bab:FireEventAction
    bab:LogAction
    bab:SendEmailAction
    bab:FixAction

bab:SafetyCriterion, bezbednosni kriterijum koji se primenjuje pre izvršavanja radnje
    bab:ForbiddenSignatureCriterion
    bab:CompositeSafetyCriterion
    bab:EntityTypeCriterion
    bab:AlwaysSafeCriterion
    bab:NeverSafeCriterion
```

Listing 5.8. Hijerarhija klasa modela AbstractAI.

9

Kontrolisani tok delovanja ovih agenata prikazan je na Slici 5.3.



Slika 5.3. Kontrolisani tok delovanja agenata u okviru BAB-a.

23

Komunikativne namere modelovane su kao skup individua izveden iz specifikacije FIPA ACL (Foundation for Intelligent Physical Agents, Agent Communication Language). FIPA ACL definiše govorne činove za komunikaciju u sistemima sa više agenata, dok BAB koristi uži podskup prilagođen poslovnim aplikacijama, prikazan u Listingu 5.9 [158].

```
bab:Performative_REQUEST , zahtev da primalac obavi neku akciju
bab:Performative_INFORM , informacija o činjenici, bez očekivanja odgovora
bab:Performative_PROPOSE , predlog koji primalac može prihvatiti ili odbiti
bab:Performative_REFUSE , odbijanje predloga ili zahteva
bab:Performative_FAILURE , obaveštenje o neuspehu obavljanja akcije
```

Listing 5.9. Podskup komunikativnih namera FIPA ACL u okviru BAB-a.

Ovakva klasifikacija omogućava da svaki komunikacioni događaj ima izričito navedenu nameru. Time se stvaraju uslovi za analizu komunikacije između agenata i buduće uvođenje protokola po uzoru na okvire BDI (engl. *belief-desire-intention*) [2].

U realizaciji u Javi, interfejs Agent definiše minimalni skup operacija prikazan u Listingu 5.10.

```
public interface Agent {
    String getName();
    boolean supports(AgentEvent event);
    void handle(AgentEvent event);
    default void initialize(AgentContext context) {}
    void setContext(AgentContext context);
    AgentContext getContext();
    default AgentScope getScope() { return null; }
    default ChatMemory getChatMemory() { return null; }
}
```

Listing 5.10. Interfejs Agent.

AbstractAgent implementira osnovne metode (ime i kontekst), a AbstractAgentWithMemory<T extends AbstractAgentMemory> dodaje sloj pamćenja pomoću klase MessageWindowChatMemory iz biblioteke LangChain4j¹⁵. Mehanizam pamćenja, prikazan u Listingu 5.11, važan je za agente koji vode duže razgovore, poput agenta korisničke podrške, jer im omogućava da očuvaju kontekst tokom više interakcija.

```
public abstract class AbstractAgentWithMemory<T extends AbstractAgentMemory>
    extends AbstractAgent {

    protected MessageWindowChatMemory chatMemory;
    protected AbstractBabChatMemoryStore<T> memoryStore;
    protected int maxMessages = 20;

    @Override
    public void initialize(AgentContext context) {
        super.initialize(context);
        this.chatMemory = MessageWindowChatMemory.builder()
            .maxMessages(maxMessages)
            .chatMemoryStore(memoryStore)
            .id(getName())
            .build();
    }
}
```

Listing 5.11. Klasa AbstractAgentWithMemory.

5.5.2 Preslikavanje osobina ABPMS na arhitekturu BAB

Manifest sistema za upravljanje poslovnim procesima proširenih veštačkom inteligencijom (ABPMS), koji su Dumas i saradnici objavili 2023. godine, razmatra autonomiju u zadatim okvirima, sposobnost delovanja kroz razgovor, prilagodljivost, samousavršavanje i objašnjivost. U nastavku su situaciona svesnost, autonomno odlučivanje, proaktivnost, memorija i konverzacija upotrebljeni kao nazivi mehanizama za analizu BAB arhitekture, a ne kao zamenska lista osobina manifesta. Njihovo povezivanje sa pet ABPMS osobina i granice provere dati su u Tabeli 6.7.

Situaciona svesnost.

¹⁵ Zvanična dokumentacija biblioteke LangChain4j: <https://docs.langchain4j.dev/>

Radno značenje u analizi arhitekture: sposobnost sistema da prati stanje procesa, događaje, izuzetke i kontekst relevantan za poslovanje.

Realizacija u okviru BAB-a: sabirnica događaja CDI, životni ciklus entiteta JPA i komponenta AgentMonitorRegistry beleže promene nad entitetima, pojavu izuzetaka i stanje komponenta agenata. AgentMonitorRegistry vodi podatke AgentStats za svakog agenta, kao što je prikazano u Listingu 5.12.

```
public static final class AgentStats {  
    public volatile boolean registered;  
    public volatile boolean currentlyProcessing;  
    public final AtomicLong totalEvents = new AtomicLong();  
    public final AtomicLong failedEvents = new AtomicLong();  
    public volatile Instant lastStarted;  
    public volatile Instant lastFinished;  
}
```

Listing 5.12. Podaci o stanju agenta u klasi AgentStats.

Na primer, AuditLogAgent prati događaje EntityEvent tipa UPDATE i DELETE i beleži ih u revizijski dnevnik. Zajedno sa komponentom EntityEventBridge iz potpoglavlja 5.2.3, ovaj mehanizam pruža sled promena koje su obuhvaćene odgovarajućim događajima. On ne predstavlja potpunu situacionu svesnost, jer ne obuhvata događaje koje sistem ne objavljuje niti prediktivnu analizu stanja.

Ograničenja trenutne implementacije: situaciona svesnost je trenutno reaktivna (zasnovana na događajima), bez prediktivnog modula. Buduća istraživanja mogu uvesti modul za prepoznavanje obrazaca koji bi predviđao verovatne događaje na osnovu istorijskih podataka.

Autonomno donošenje odluka.

Definicija: sposobnost sistema da donosi i sprovodi odluke bez ljudske intervencije kada to procena situacije nalaže.

Realizacija u okviru BAB-a: hijerarhija SelfHealingAgent. Apstraktna klasa SelfHealingAgent<T> koristi veliki jezički model putem unutrašnjeg interfejsa LogAnalyst, prikazanog u Listingu 5.13.

```
interface LogAnalyst {  
    @UserMessage("Analyze the following system log and extract the structured data:  
    {{it}}")  
    LogAnalysisResult analyze(String logLine);  
}
```

Listing 5.13. Interfejs LogAnalyst za strukturisanu analizu dnevnika.

LogAnalysisResult vraća strukturisani objekat sa isCritical, summary, suggestedFix i confidenceScore. Ako je log kritičan i confidenceScore iznad 0.85, agent postupkom iz Listinga 5.14 samostalno predlaže FixAction.

```
protected AgentAction proposeFix(LogEvent event) {
    LogAnalyst analyst = context.getLlmRouter().createService(LogAnalyst.class,
getChatMemory());
    LogAnalysisResult r = analyst.analyze(event.getLogLine());
    if (r.isCritical() && r.getConfidenceScore() > 0.85) {
        return new FixAction(r.getSuggestedFix());
    }
    return null;
}
```

Listing 5.14. Predlaganje korektivne radnje na osnovu analize dnevnika.

Kao konkretan primer, `SqlErrorHealingAgent` prepoznaje `SQLException` u dnevnicima i obrađuje predlog SQL ispravke. Kriterijum `ForbiddenSignatureCriterion` proverava zadati spisak zabranjenih potpisa (`DROP TABLE`, `TRUNCATE`, `DELETE FROM`, `GRANT`, `REVOKE` i `DROP DATABASE`). Ova provera nije formalni dokaz bezbednosti. Pregled iz potpoglavlja 6.5.3 pokazuje da `FixAction.execute()` trenutno beleži predlog u dnevnik, dok stvarno izvršenje ispravke ostaje predmet dalje realizacije.

Ograničenja: odluke se trenutno zasnivaju na odgovoru velikog jezičkog modela, bez formalne provere prema modelu domena. Budući rad može uključiti simbolički zaključivač koji bi proveravao da predložena radnja ne narušava ontološke invarijante.

Proaktivnost.

Definicija: sposobnost da agent ne čeka da bude pitan, već da samostalno inicira akciju kada prepozna potrebu.

Realizacija u okviru BAB-a: interfejs `AgentTrigger` i mehanizam `fireEvent`. Agent se može povezati sa okidačem koji periodično proverava stanje, na primer postojanje nezavršenih faktura starijih od 30 dana, i zatim objavljuje `AgentMessageEvent` za odgovornog korisnika. Obrada događaja u klasi `EmailNotificationAgent` prikazana je u Listingu 5.15.

```
@Override
public void handle(AgentEvent event) {
    if (event instanceof AgentErrorEvent err) {
        String adminEmail = systemUtils.getSystem().getEmail();
        emailUtils.sendErrorMessageFromAgent(adminEmail, err);
    }
}
```

Listing 5.15. Obrada greške u klasi `EmailNotificationAgent`.

Ograničenja: proaktivno ponašanje je trenutno zadato programskim kodom agenata. Dalja istraživanja mogu uvesti deklarativna pravila (npr. SWRL) koja korisnik može menjati bez izmene koda.

Učenje na osnovu iskustva.

Definicija: sposobnost da agent prilagođava svoje ponašanje na osnovu prethodnih ishoda.

Realizacija u okviru BAB-a: `AbstractAgentMemory` i `AbstractBabChatMemoryStore` čuvaju razgovore agenata u polju `chatMessagesJson`. Posle ponovnog pokretanja sistema

agent zato može obnoviti deo prethodnog konverzacionog konteksta. UserHelpDeskAgent koristi MessageWindowChatMemory sa najviše 20 poruka u okviru korisničke sesije.

Ograničenja: memorija služi za očuvanje konverzacionog konteksta, a ne za autonomno učenje ili prilagođavanje parametara modela. Budući rad može beležiti ishode radnji i ocene korisnika i koristiti ih kao primere u kontekstu upita.

Prirodnost razgovora.

Definicija: sposobnost agenta da komunicira prirodnim jezikom uz uvažavanje konteksta, namere i emocionalnog tona poruke.

Realizacija u okviru BAB-a: UserHelpDeskAgent. Agent koristi IntentClassifier, servis LangChain4j sa upitom koji zahteva izdvajanje namere, emocionalnog tona i ciljnog entiteta iz poruke korisnika. Na osnovu dobijene klasifikacije bira odgovarajuću granu obrade, kao što je prikazano u Listingu 5.16.

```
@Override
public void handle(AgentEvent event) {
    if (event instanceof AgentMessageEvent msg
        && msg.getReceiver().equals(getName())
        && msg.getPerformative() == Performative.REQUEST) {

        UserIntentResult intent = classifier.classify((String) msg.getPayload());
        switch (intent.getIntent()) {
            case TECHNICAL_ISSUE -> handleTechnicalIssue(msg, intent);
            case NAVIGATION      -> sendReply(msg, "Navigacija do entiteta: " +
intent.getTargetEntity());
            case FAQ, UNKNOWN    -> handleChat(msg);
        }
    }
}
```

Listing 5.16. Obrada korisničke poruke u klasi UserHelpDeskAgent.

Tehničke greške prevode se u AgentErrorEvent sa prefiksom „UserReported:”, čime se uključuje EmailNotificationAgent. Pitanja o navigaciji dobijaju kratak odgovor sa imenom entiteta, dok se opšta pitanja prosleđuju interfejsu Assistant sa opštom sistemskom porukom.

Ograničenja: kvalitet konverzacije zavisi od izabranog modela i jezika. Razlike između modela na srpskom jeziku nisu sistematski ispitane u ovoj disertaciji. Budući rad može obuhvatiti prilagođavanje modela srpskom korpusu i generisanje potpomognuto pretraživanjem dokumentacije organizacije.

Prikazani mehanizmi posredstvom konkretnih komponenata tokom rada sistema delimično realizuju elemente objašnjivosti, prilagodljivosti, sposobnosti delovanja kroz razgovor i autonomije u zadatim okvirima. Njihov doprinos pojedinačnim ABPMS osobinama ocenjuje se odvojeno u potpoglavlju 6.5; izvršavanje ograničene funkcije agenta nije dokaz potpune realizacije osobine na nivou celog sistema.

5.5.3 Povezivanje sa spoljnim uslugama veštačke inteligencije

Veliki jezički modeli povezani su sa BAB okvirom pomoću biblioteke LangChain4j. Njeni osnovni mehanizmi obuhvataju apstrakciju ChatModel, AiServices za strukturisane pozive, ChatMemory za očuvanje konteksta i prilagođavanja za različite pružaoce modela.

Klasa LlmRouter posreduje između klase repozitorijuma za konfiguraciju velikog jezičkog modela, AbstractLLMHome, i konkretnog objekta ChatModel, kao što je prikazano u Listingu 5.17.

```
public class LlmRouter<T extends AbstractLLM> {

    @Inject LlmFactory factory;
    @Inject @Primary AbstractLLMHome<T, ?> serviceEJB;

    public String chat(String prompt, ChatMemory memory) {
        T cfg = (T) serviceEJB.findActiveDefault();
        ChatModel model = factory.build(cfg);
        if (memory != null) {
            memory.add(UserMessage.from(prompt));
            ChatResponse response = model.chat(memory.messages());
            memory.add(response.aiMessage());
            return response.aiMessage().text();
        } else {
            return model.chat(prompt);
        }
    }

    public <S> S createService(Class<S> serviceClass, ChatMemory memory) {
        T cfg = (T) serviceEJB.findActiveDefault();
        if (cfg == null) throw new IllegalStateException("Nije konfigurisan podrazumevani LLM.");
        ChatModel model = factory.build(cfg);
        return
        AiServices.builder(serviceClass).chatModel(model).chatMemory(memory).build();
    }
}
```

Listing 5.17. Usmeravanje poziva velikom jezičkom modelu u klasi LlmRouter.

LlmFactory omogućava pristup podržanim pružiocima modela posredstvom jedinstvenog fabričkog interfejsa prikazanog u Listingu 5.18.

```
@ApplicationScoped
public class LlmFactory {

    public ChatModel build(AbstractLLM cfg) {
        return switch (cfg.getModel()) {
            case OPENAI -> OpenAiChatModel.builder()
                .apiKey(cfg.getKey())
                .modelName(cfg.getModelName())
                .temperature(cfg.getTemperature())
                .build();
            case OLLAMA -> OllamaChatModel.builder()
                .baseUrl(cfg.getUrl() != null ? cfg.getUrl() :
"http://localhost:11434")
                .modelName(cfg.getModelName())
                .temperature(cfg.getTemperature())
                .build();
            case HUGGINGFACE, MISTRAL, GOOGLE, CUSTOM ->
                throw new IllegalArgumentException("Provajder " + cfg.getModel() +
" još nije implementiran.");
        };
    }
}
```

Listing 5.18. Izbor pružaoca modela u klasi LlmFactory.

Integracija sa uslugom OpenAI realizuje se pomoću programskog interfejsa dostupnog preko protokola HTTPS. Upotreba spoljašnje usluge podrazumeva troškove po pozivu i zahteva procenu zaštite i poverljivosti podataka koji se šalju izvan lokalnog sistema.

Integracija sa sistemom Ollama omogućava lokalno izvršavanje podržanih otvorenih modela. U tom slučaju podaci mogu ostati u lokalnoj mreži, sistem može raditi bez pristupa internetu, a verzija modela nalazi se pod neposrednom kontrolom organizacije. Ograničenja se odnose na zahteve lokalnog hardvera, brzinu odziva i kvalitet rezultata manjih modela. Konkretno zahteve i razlike treba utvrditi merenjem za izabrani model i radno opterećenje.

Integracija sa Hugging Face predviđena je za budući rad. Njena moguća uloga jeste pristup specijalizovanim modelima putem usluge Inference Endpoints.

Integracija sa modelima Anthropic Claude¹⁶ trenutno nije deo nabiranja LlmType. Mogla bi se dodati pomoću tipa CUSTOM i odgovarajućeg proširenja klase LlmFactory. Takvo proširenje nije deo postojeće implementacije.

Oblikovanje upita u BAB-u zasniva se na tri načela: jasno određenoj ulozi u sistemskoj poruci, primerima očekivanih ulaza i izlaza i utvrđenoj strukturi odgovora opisanoj anotacijama @Description na klasama DTO. Primer je prikazan u Listingu 5.19.

¹⁶ Zvanična dokumentacija i stranice usluga: OpenAI API: <https://developers.openai.com/api/docs/>; Ollama: <https://ollama.com/>; Hugging Face Inference Endpoints: <https://endpoints.huggingface.co/>; Anthropic Claude: <https://platform.claude.com/docs/en/home>

```
public class LogAnalysisResult {

    @Description("True ako log označava kritičnu grešku koja zahteva intervenciju.")
    private boolean isCritical;

    @Description("Kratak sažetak greške u jednoj rečenici.")
    private String summary;

    @Description("Predlog SQL ili Java koda koji bi rešio problem.")
    private String suggestedFix;

    @Description("Verovatnoća da je predlog ispravan, od 0.0 do 1.0.")
    private double confidenceScore;

}
```

Listing 5.19. Struktura rezultata analize dnevnika.

Na osnovu anotacija `@Description`, `LangChain4j` formira deo upita koji opisuje očekivanu strukturu odgovora u formatu JSON, a zatim pokušava da taj odgovor pretvori u Java objekat. Uspešnost ovog postupka zavisi od izabranog modela i mora se utvrditi proverom dobijenog izlaza.

Generisanje potpomognuto pretraživanjem (engl. *retrieval-augmented generation*, RAG) nije potpuno implementirano, ali je predviđeno klasom `RagAwareAgent`. Zamisao je da sistem pre poziva modela pretraži vektorsko skladište dokumentacije i u upit uključi najrelevantnije odlomke [159]. `LangChain4j` podržava više vektorskih skladišta, kao i lokalno skladište u memoriji za manje korpuse.

Pozivanje alata omogućava modelu da zatraži izvršavanje unapred definisane Java metode i zatim upotrebi njen rezultat pri formiranju odgovora. U biblioteci `LangChain4j` takve metode označavaju se anotacijom `@Tool`. Ova mogućnost još nije deo opisane funkcionalnosti `UserHelpDeskAgent` i predstavlja pravac daljeg razvoja.

Upravljanje kontekstnim prozorom važno je tokom dužih razgovora. `MessageWindowChatMemory` čuva poslednjih N poruka, podrazumevano 20, pa starije poruke izlaze iz aktivnog konteksta. Za zahtevnije slučajeve mogu se koristiti `TokenWindowChatMemory`, koji prati broj tokena, ili trajno skladište iz kojeg se biraju poruke relevantne za novi upit.

Upotreba tokena prati se pomoću objekta `ChatResponse.tokenUsage`. BAB može beležiti agenta, model, broj ulaznih i izlaznih tokena, procenjeni trošak i vremensku oznaku u tabeli `LLMTokenUsage`. Ti podaci predstavljaju osnovu za praćenje troškova i kasniju analizu korišćenja modela.

Rezervni model može se koristiti kada primarni model ne odgovori u zadatom vremenu ili vrati grešku. U `LlmRouter`-u je ova logika realizovana obradom izuzetaka, uz najviše tri ponovna pokušaja.

Ograničavanje učestalosti poziva sprečava da jedan agent iscrpi raspoloživu kvotu modela. Sprovodi se algoritmom „kante sa žetonima” (engl. *token bucket*), koji ograničava broj poziva po agentu u jedinici vremena.

Zaštita komunikacije obuhvata čuvanje ključeva programskog interfejsa u šifrovanom obliku, upotrebu zaštićenog transportnog kanala prema spoljašnjim uslugama i mogućnost lokalnog pristupa sistemu Ollama. Komponenta DataAnonymizer predviđena je za zamenu podataka koji mogu identifikovati lice generičkim oznakama pre slanja spoljašnjem modelu. Delotvornost ovih mera mora se proveriti posebnom bezbednosnom analizom.

Pozivi velikim jezičkim modelima nisu obuhvaćeni kategorijalnim rezultatom iz Poglavlja 4, koji se odnosi na determinističnu transformaciju šematskih artefakata. Formalizovanje nedeterminističnog ponašanja modela zahtevalo bi dodatni matematički okvir i ostaje izvan obima ove disertacije.

Peto poglavlje prikazalo je slojevitú arhitekturu BAB okvira, njegove osnovne apstrakcije, modularni BAB PIM 2.0, BAB Mapping 2.0, BABj 2.0 profil, ugovore zasnovane na sposobnostima i generator zasnovan na bibliotekama Jena i FreeMarker. Sledljivost obezbeđuju metapodaci, manifest i sertifikat transformacione specifikacije. Zasebni sloj agenata AbstractAI nije deo ugovora generatora. Veza sa Poglavljima 3 i 4 odnosi se na šematsku strukturu i proverljive uslove transformacije, dok stepen praktične primenljivosti treba utvrditi proverom.

Poglavlje 6 proverava ove arhitektonske odluke na primeru PostOffice, analizama sistema WasteX, SoilLab i DentaScape, rezultatima objavljene studije i inženjerskom proverom generatora i biblioteka. Nalazi potvrđuju pojedine realizacione i strukturne osobine, dok prednost u ukupnom vremenu razvoja i tehničkom dugu i potpuna podrška ABPMS osobinama tokom rada sistema ostaju hipoteze za posebno ispitivanje.

6. Validacija i evaluacija

Validacija obuhvata proveru formalnog modela, realizacije transformacionog lanca i primenljivosti BAB okvira u poslovnim aplikacijama. Jedinice analize su PIM aplikacije, izabrana transformaciona specifikacija, generisani projekat i konkretne komponente informacionog sistema. Uspeh na jednom od ovih nivoa ne podrazumeva uspeh na ostalima: logički konzistentan PIM može biti nepotpun za generisanje, projekat koji se kompajlira može zahtevati dopunu poslovne logike, a postojanje komponente zasnovane na agentima samo po sebi ne potvrđuje autonomno unapređenje procesa.

Poglavlje objedinjuje više međusobno dopunjujućih vrsta dokaza. Slučaj PostOffice služi za pregled konkretnog puta od modularnog PIM-a do koda koji koristi BAB 2. WasteX pokazuje primenu BAB 2 apstrakcija, poslovnih procesa i integracija u razvijenom poslovnom sistemu. SoilLab predstavlja proverenu upotrebu osnovnog modula BAB 2 u istraživačkoj aplikaciji za senzorska merenja i pripremu prediktivnih modela, dok DentaScape pokazuje upotrebu šireg skupa opcionih modula u aplikaciji za stomatološku ordinaciju. Objavljena studija statičke klasifikacije daje zasebne empirijske rezultate za dva domena. Automatizovani testovi i sertifikat proveravaju identifikovanu BAB 2 verziju generatora i deklarirani šematski deo ontologije, a zasebna provera Maven reaktora ispituje samu modularnu biblioteku. Njihov domet prikazan je u Tabeli 6.1.

Predmet	Postupak	Vrsta zaključka
PostOffice PIM i izlaz	Pregled ontologije, validacija, generisanje, provera izgradnje i ručna funkcionalna provera na serveru WildFly 41.0.1	Izvodljivost transformacije i rada generisane aplikacije za konkretan model; ponovljivost se proverava sertifikacionim korpusom
WasteX	Arhitektonska analiza domenskih, servisnih, integracionih komponenti i komponenti zasnovanih na agentima	Ručna primena metodologije i modularnih BAB 2 ugovora u složenom poslovnom sistemu
SoilLab	Pregled izvornog koda, dokumentovanog protokola modelovanja i automatizovanih testova	Ručna primena metodologije i proverena upotreba bab-core ugovora u senzorskom domenu
DentaScape	Pregled izvornog koda, zavisnosti i automatizovanih testova	Ručna primena metodologije i modularnih BAB 2 ugovora u stomatološkom domenu
Statička klasifikacija	Rezultati objavljenog protokola za dva domena	Saglasnost klasifikacija i ponašanje pri odabranim greškama
Generator i sertifikat	Automatizovani testovi i pet provera transformacione specifikacije	Inženjerska verifikacija izabranog korpusa i šematskog dela ontologije obuhvaćenog transformacionom specifikacijom
BAB 2 biblioteka	Maven reaktorska izgradnja i testovi modula	Ispravnost proverene razvojne revizije na nivou jedinica, odvojeno od provere generatora i generisanih aplikacija
ABPMS osobine	Povezivanje zahteva sa izvršnim komponentama i granicama njihove realizacije	Stepen delimične operativne realizacije, uz razdvajanje od potvrde osobine na nivou celog sistema

Tabela 6.1. Predmeti validacije, postupci i domet zaključaka.

Verifikacija generatora, provere biblioteke i provere aplikacija odnose se na zasebno identifikovane skupove artefakata. Njihove verzije i evidencija potrebna za ponavljanje postupka objedinjene su u Prilogu G. Vreme razvoja, tehnički dug i pokrivenost izvršenog koda predstavljaju posebne merne veličine. Tamo gde odgovarajuća merenja nisu sprovedena, navode se granice nalaza i uslovi za dalje ispitivanje, bez zaključivanja o veličini pretpostavljene prednosti metodologije.

Metodologija se u ovom poglavlju razmatra kroz dva načina primene. U automatizovanoj primeni PIM, realizacioni profil, razrešena specifikacija i generisani artefakti postoje kao eksplicitna i proverljiva međustanja. U ručnoj primeni projektant sledi isto razdvajanje domenskih pojmova, platformskih odluka i programskih slojeva, ali pojedina međustanja mogu ostati implicitna i neposredno se preneti u kod. Odsustvo sačuvanog OWL PIM-a zato ne znači da metodologija nije korišćena. Ono znači da se njen transformacioni trag ne može naknadno u potpunosti reprodukovati, automatski proveriti ili obuhvatiti sertifikatom iz Poglavlja 4.

6.1 Potvrda koncepta 1: PostOffice informacioni sistem

6.1.1 Opis domena i ontološki model

PostOffice predstavlja ograničen model evidencije korisnika poštanskih usluga, pošiljaka i toka njihove isporuke. Izabran je zato što na malom broju domenskih pojmova povezuje perzistentne podatke, binarnu relaciju, projekciju za prikaz, uređivanje podataka, ovlašćenja i proces. Njegova svrha je da omogući praćenje transformacije pojedinačnih elemenata, a ne da obuhvati sve funkcije poštanskog poslovanja.

Ulazna datoteka `postoffice-pim.ttl` uvozi module `Core`, `View`, `Behavior`, `Validation`, `Security` i `Capability` verzije 2.0 i pet lokalnih ontologija: domenski model, prikaze, ponašanje, validaciju i bezbednost. BAB Mapping i BABj učitavaju se kao zaseban transformacioni ugovor. U domenskom delu postoje jedna aplikacija `PostOffice`, dva entiteta `Customer` i `Parcel` i konkretan tip aktera `PostalClerk`. Ova tri tipa međusobno su disjunktna.

`Customer` ima attribute `name` i `email`, dok `Parcel` ima `trackingNumber`, `recipientName`, `weightGrams` i `status`. Tipovi su platformski neutralni: tekst se opisuje sa `xsd:string`, a masa u gramima sa `xsd:int`. Modul `Validation` zahteva prisustvo imena, broja za praćenje, primaoca i statusa, zadaje obrazac broja za praćenje i adrese elektronske pošte i donju granicu mase od jednog grama. To su deklarirani uslovi modela; njihova prisutnost u generisanom kodu razmatra se odvojeno od ponašanja aplikacije tokom izvršavanja.

Relacija `CustomerParcelRelation` sadrži dva uzajamno povezana kraja. Kraj `ParcelSenderEnd` pripada pošiljci, cilja korisnika i ima kardinalitet `ExactlyOne`; suprotni kraj `CustomerParcelsEnd` pripada korisniku, cilja pošiljke i ima kardinalitet `ZeroOrMore`. Time se izražava da svaka pošiljka ima jednog pošiljaoca, a korisnik može

poslati više pošiljaka. PIM ne navodi JPA anotacije ni strani ključ. Njih izvodi BABj realizacija, koja ovaj par krajeva pretvara u veze ManyToOne i OneToMany.

Prikaz ParcelView povezan je sa formom ParcelEditForm i projekcijom od pet kolona: brojem za praćenje, imenom pošiljaoca, primaocem, masom i statusom. Ime pošiljaoca dobija se preko relacionog kraja do atributa CustomerName; ostale kolone potiču neposredno iz pošiljke. Ovaj primer proverava da se ista relacija dosledno upotrebljava u modelu entiteta, projekcionom DTO-u i upitu home sloja. Za korisnike postoji zaseban prikaz sa dve kolone i pripadajućom formom.

Pošiljci su dodeljene sposobnosti GeneratedCode, Audited, Validatable, Storable i Geolocated, uz prefiks generisane oznake PO. Prikaz dodatno zahteva izvoz, automatsko prilagođavanje mreže i obradu teme, a forma uzastopni unos i osvežavanje izbora. Ovi zahtevi ostaju odvojeni od konkretnih Java interfejsa. PostalClerk predstavlja tip izvršioca i izvor operatorskog ugovora, dok su pojedinačni nalozi podaci aplikacije. Uloga PostalClerkRole određuje pristup prikazima i zadatku otpreme; ona se takođe razlikuje od tipa aktera i konkretnog korisničkog naloga.

Proces ParcelDelivery sadrži stanja Accepted, InTransit i Delivered, četiri zadatka i dva prelaza. Zadaci su početak prijema, korisnička otprema, servisni zadatak isporuke i završetak. Deklarisani uslovi prelaza predstavljaju imenovane tačke za povezivanje sa poslovnom logikom. Tekst „delivery confirmation received” sam po sebi ne proverava potvrdu isporuke: odgovarajući izvršni predikat mora biti realizovan. Jednako tako, atribut Parcel.status i procesno stanje nisu automatski sinhronizovani samo zato što su oba opisana u ontologiji.

6.1.2 Generisanje koda i rezultati

Generator učitava aplikacioni PIM, rekurzivno razrešava njegove lokalne uvoze, proverava ontološki i operativni ugovor i izdvaja uređeni ApplicationSpec. BABj profil zatim razrešava ciljne tipove, sposobnosti, module potrebne tokom izvršavanja i šablone, a pripremljeni podaci prosleđuju se FreeMarker-u. Dobijeni projekat koristi Java 26, modularni BAB 2.0.0, Vaadin 25.2.6 i Jakarta EE Web 11. PostOffice ne zahteva sposobnost zasnovanu na agentima, Web Push ili WebAuthn, pa njegov POM uvozi bab-bom i kao izvršnu BAB zavisnost uključuje samo bab-core. Tabela 6.2 prikazuje karakteristične veze između ulaza i izlaza.

PIM element	Realizaciona odluka	Izlazni artefakt ili član
Parcel	Perzistentni BAB entitet	model/Parcel.java, JPA anotacije i nasleđivanje AbstractEntity
Dva kraja CustomerParcelRelation	Par ManyToOne/OneToMany	Veze pošiljaoca i kolekcije pošiljaka u entitetima
Projekcija ParcelListProjection	DTO i relacioni projekcioni upit	ParcelDTO.java i ParcelHome.java
ParcelView i ParcelEditForm	BAB prikaz i prozor za uređivanje	ParcelView.java i EditParcelWindow.java
PostalClerk	Primarni operatorski ugovor	Model, DTO, home i aplikaciono povezivanje prijave i sesije
ParcelDelivery	Procesni skelet sa tačkama proširenja	Procesne klase i metapodaci o stanjima, zadacima i prelazima

Tabela 6.2. Izabrani tragovi transformacije PostOffice modela.

Entitet, DTO, home, prikaz i forma nastaju kao povezani artefakti. Na primer, izmena projekcije pošiljke zahteva da naredna generacija uskladi tip DTO-a, njegov konstruktor, projekcioni upit i kolone prikaza. Ponovljivost se proverava poređenjem sastava i sadržaja celog upravljanog izlaza u dva nezavisna generisanja. Takva provera obuhvata međusobno povezane slojeve aplikacije, a ne samo broj datoteka ili jedan izabrani Java izvor.

PostOffice se razlikuje od prihvatnog modela Catalog, čiji su rezultati navedeni u potpoglavlju 6.6.1. Broj njegovih datoteka i naziv WAR-a ne prenose se iz Catalog provere. Detaljni rezultati ponovljenog postupka i njihove granice prikazani su u istom potpoglavlju.

Generisani projekat predstavlja aplikacionu osnovu u obimu eksplicitno modelovanih ugovora. Procesni zadaci, uslovi prelaza i posebna poslovna pravila mogu zahtevati dopunu. Pored kompilacije i pakovanja, generisana aplikacija PostOffice raspoređena je i pokrenuta na serveru WildFly 41.0.1. Ručnom funkcionalnom proverom potvrđeni su prijavljivanje korisnika, CRUD operacije, navigacija, filtriranje i modelovano ponašanje. Ova provera potvrđuje izvršivost navedenih funkcija konkretnog generisanog sistema; ne predstavlja automatizovan funkcionalni test, ispitivanje performansi, dugotrajan rad ili proveru svih mogućih poslovnih scenarija.

6.2 Potvrda koncepta 2: WasteX: kooperativni IS za sekundarne sirovine

6.2.1 Procesni model: sakupljanje, reciklaža, cirkulacija

WasteX je poslovni informacioni sistem za evidenciju i koordinaciju poslova povezanih sa otpadom i sekundarnim sirovinama. Sistem koristi modularne BAB 2 ugovore i obuhvata širi skup funkcija od ovde razmatranih. Izdvojeni su sakupljanje, obrada materijala i cirkulacija ambalaže jer pokazuju vezu poslovnih podataka, izvršilaca, stanja i ponovljivih BAB apstrakcija.

Za razliku od PostOffice-a, WasteX predstavlja ručnu primenu BAB metodologije. Domenski pojmovi, odgovornosti slojeva i realizacioni ugovori preneti su u aplikaciju

projektantskim postupkom, bez sačuvanog formalnog niza ontoloških međustanja. Zato tabela povezivanja sa BAB PIM modulima predstavlja rekonstrukciju metodološke strukture vidljive u kodu, a ne rezultat ponovnog pokretanja generatora.

Sakupljanje počinje evidentiranjem najave i povezivanjem učesnika, prevoza i materijala. Klase Najava, NajavaStavka, PutniNalog, Firma, Vozilo i OtpadInfo predstavljaju ključne podatke tog dela sistema. Nabranje NajavaStatus razlikuje kreiranu najavu, potvrdu logistike, potvrdu prevoznika, kreiranu dispoziciju i stornirana stanja. To omogućava razdvajanje odgovornosti učesnika i praćenje statusa dokumenta. Merenja su predstavljena klasom Odvaga, koja čuva izmerene količine, vremenske oznake, podatke o ambalaži i veze sa pratećim dokumentima.

Obrada materijala obuhvata evidenciju proizvodnog naloga i količina koje ulaze u obračun. U razmatranom fragmentu klasa NalogProizvodnjaOJU povezana je sa odvagom, izvornim i odredišnim magacinskim mestom, izvršiocem, potvrdom i storniranjem. Njeni atributi obuhvataju bruto količinu, ambalažu, talog i neto količinu. Ovaj fragment dokumentuje informacionu podršku delu procesa obrade i reciklaže; iz njega se ne izvodi da sistem formalno modeluje sve fizičke postupke reciklaže ili automatski proverava ukupan materijalni bilans.

Cirkulacija je naročito vidljiva na modelu povratne ambalaže. Ambalaza je specijalizacija artikla sa barkodom i evidencijom zaduženja. AmbalazaIstorija beleži ambalažu, izvršioca, vreme i vrstu radnje. Vrste radnji obuhvataju unos, promenu lokacije, zaduživanje, izlazak i razduživanje. Time se poslovni tok predstavlja istorijom promena nad trajno identifikovanim objektom, umesto samo njegovim trenutnim stanjem.

Odabrani poslovni fragmenti sistema WasteX i njihova veza sa pojmovima BAB PIM-a prikazani su u Tabeli 6.3.

Poslovni fragment	Artefakti WasteX-a	Odgovarajući pojmovi BAB PIM-a
Sakupljanje i prevoz	Najava, NajavaStatus, PutniNalog, Odvaga	Core entiteti i relacije; Behavior stanja i zadaci; Security uloge
Obrada materijala	NalogProizvodnjaOJU, odvaga, magacinska mesta	Core atributi i relacije; Validation ograničenja; Capability potvrda, izvršilac i storniranje
Cirkulacija ambalaže	Ambalaza, AmbalazaZaduzen, AmbalazaIstorija	Core identitet i veze; Behavior radnje; Capability barkod i evidencija izvršioca
Prikaz i razmena podataka	DTO, home, prikazi, forme i REST krajnje tačke	View projekcije i forme; Service operacije; realizacija kroz BABj

Tabela 6.3. Analitičko povezivanje odabranih WasteX fragmenata sa BAB PIM pojmovima.

Zajednički obrasci vidljivi su i neposredno u deklaracijama klasa. Najava nasleđuje AbstractEntity i realizuje ugovore za izvršioca, potvrdu, storniranje, lokacije i REST razmenu, dok NajavaHome nasleđuje AbstractHome<Najava, NajavaDTO>. Sličan obrazac postoji kod odvage, otkupnog lista i proizvodnog naloga. Ovi primeri podržavaju

izbor ponovljivih apstrakcija koje BAB PIM Capability i BABj kasnije eksplicitno razdvajaju na semantičku nameru i tehničku realizaciju.

6.2.2 Evolucija od centralizovanog ka međuorganizacionom sistemu

Centralizovana aplikacija raspolaže lokalnim identifikatorima i neposrednim pristupom sopstvenim servisima. U međuorganizacionoj razmeni potrebno je povezati zapise koji potiču iz različitih sistema, identifikovati učesnika i odrediti ko sme da potvrdi ili izmeni dokument. WasteX proširuje postojeći domenski i home sloj komponentama za razmenu, pri čemu REST operacije ponovo koriste poslovne servise aplikacije.

JAX-RS aplikacija postavlja osnovnu putanju /api. NajjavaEndpoint izlaže operacije na /v1/najjava, a PutniNalogEndpoint na /v1/putniNalog. U obradi potvrde najave ulazni DTO povezuje se sa prevoznikom i putnim nalogom, zatim se preko javnog identifikatora traži postojeći dokument. U zavisnosti od ishoda, dokument se kreira ili dopunjava, čuva kroz home sloj i prosleđuje njegovoj operaciji potvrde. Ovaj sled pokazuje ponovno korišćenje poslovne realizacije pri dodavanju drugog kanala pristupa.

Podrška nije ograničena na REST klasu. Entitet Najjava sadrži publicId, isExternal i isSync, njegov DTO prenosi podatke razmene, a home sloj realizuje RestResourceHomeInterface. Servlet filter povezuje API pristup sa korisničkim ugovorom BAB biblioteke. Posebna integracija sa spoljašnjim poslovnim sistemom vidi se u metodi insertDOK0InPantheon klase NajjavaHome.¹⁷ Takva integracija sadrži domenske i realizacione odluke koje se ne mogu izvesti samo iz činjenice da je element označen kao servis.

Prelazak ka kooperativnom sistemu zato zahvata identitet zapisa, prenos podataka, granice ovlašćenja, poslovne operacije i sinhronizaciju. BAB apstrakcije omogućavaju da se deo postojeće realizacije ponovo upotrebi, ali broj novih krajnjih tačaka nije mera ukupnog napora integracije. Pregled koda potvrđuje navedene mehanizme; ne potvrđuje njihove performanse pod opterećenjem, isporuku bez dupliranja niti oporavak posle prekida komunikacije. Za ove osobine potrebni su integracioni scenariji sa ponovljenim, zakasnelim i neuspelim zahtevima.

6.2.3 Složenost integracije i stabilnost arhitektonskih ugovora

WasteX objedinjuje perzistentni domenski model, poslovne servise, korisnički interfejs, REST komunikaciju, upravljanje dokumentima i komponente zasnovane na agentima. Zajedničke BAB 2 apstrakcije daju stabilne ugovore između ovih slojeva, dok specifična pravila sakupljanja, prerade, transporta i cirkulacije ambalaže ostaju u aplikacionom kodu. Primer je zato značajan zbog obima i raznovrsnosti integracije, a ne zbog pojedinačne verzije tehnološke zavisnosti.

Modularnost BAB 2 omogućava da aplikacija zadrži obavezni osnovni modul i uključi samo proširenja koja odgovaraju njenim sposobnostima. Time se razlikuju zajednički

¹⁷ Zvanična stranica poslovno-informacionog sistema PANTHEON: <https://www.datalab.rs/>

ugovori okvira od domenske logike i opcionalnih tehničkih funkcija. WasteX pokazuje da isti arhitektonski obrazac može obuhvatiti veliki broj povezanih poslovnih celina, bez tvrdnje da je čitava postojeća aplikacija automatski izvedena iz jednog OWL PIM-a.

6.2.4 Objavljena uporedna evaluacija

U radu [160] prikazano je sekvencijalno poređenje dve realizacije istog informacionog sistema za poslovne procese u upravljanju otpadom. Isti tim od dva programera najpre je realizovao sistem neposredno na platformi Jakarta EE, a zatim ponovio razvoj sa istim funkcionalnim zahtevima koristeći BAB okvir. Objavljeni rezultati sažeti su u Tabeli 6.4.

Pokazatelj	Neposredna realizacija	Realizacija pomoću BAB-a
Broj klasa	856	684
Broj linija koda	136.632	54.799
Razvojni napor u čovek-danima	120	50
Broj prijavljenih grešaka tokom provere kvaliteta	52	13
Vreme uključivanja novog čvora	Nije posebno mereno	Jedan radni dan

Tabela 6.4. Objavljeni rezultati poređenja neposredne realizacije i realizacije pomoću BAB okvira.

Ovi rezultati daju empirijsku podršku tvrdnji da zajedničke apstrakcije, generisanje ponovljivih delova i standardizovani REST ugovori mogu smanjiti razvojni napor i broj grešaka u posmatranom sistemu. Njihov domet ipak određuje nacrt studije: faze su izvedene sekvencijalno, pa se uticaj BAB okvira ne može potpuno razdvojiti od iskustva stečenog u prvoj realizaciji, efekta redosleda i drugih uslova konkretnog projekta. Rezultati zato predstavljaju objavljenu evaluaciju jedne studije slučaja, a ne regresioni test aktuelnog generatora niti opštu procenu za svaki domen.

6.3 Dodatne aplikacije zasnovane na BAB bibliotekama

6.3.1 Primena BAB ugovora u istraživačkoj aplikaciji

SoilLab je istraživačka platforma za evidenciju merenja zemljišta električnim senzorima i pripremu predikcije fizičko-hemijskih parametara. Aplikacija je realizovana na platformi Java/Jakarta EE sa korisničkim interfejsom Vaadin. Koristi osnovni modul bab-core, bez opcionalnih BAB proširenja, pa omogućava posmatranje primene zajedničkih apstrakcija u istraživačkom domenu.

Domenski sloj razdvaja eksperiment, fizički senzor i njegov materijal, stanje uzorka, merenje, tačku spektra i referentno opažanje. Posebni entiteti PredictionModelVersion, PredictionModelTarget i Prediction čuvaju identitet modela, uređeni skup ciljeva i dobijene predikcije. Ove klase nasleđuju AbstractEntity, dok odgovarajući servisi nasleđuju AbstractHome. Prikazi i dijalozi koriste generičke BAB apstrakcije GenericView i GenericWindow, a projekcije su izvedene iz AbstractDTO. Time se isti obrazac, koji povezuje model, projekciju, home sloj i prikaz ili prozor, primenjuje u domenu koji se razlikuje i od PostOffice-a i od WasteX-a.

SoilLab predstavlja ručnu primenu metodologije: domenski model, projekcije, home sloj i prikazi oblikovani su prema BAB razdvajanju odgovornosti, a zatim neposredno realizovani nad osnovnim modulom BAB 2. Konceptualni PIM i transformaciona međustanja nisu sačuvani kao OWL i Mapping artefakti, pa ovaj primer potvrđuje arhitektonsku primenu metodologije i izvršnu saglasnost aplikacije, ali ne omogućava naknadno ponavljanje automatskog toka od PIM-a do koda.

6.3.2 Granica istraživačke evaluacije i predviđanja u stvarnoj primeni

SoilLab sadrži zajednički pripremljen postupak za poređenje više regresionih kandidata različite složenosti. Za arhitektonsku analizu nije presudan matematički oblik pojedinačnog kandidata, već jedinstven ugovor evaluacije: isti izvor podataka, identifikovan skup za obučavanje, ciljni parametar, frekvencijska mreža, ulazni kanali, način razdvajanja podataka i definicije metrika. Postupak evaluacije razdvaja ponovljena merenja tako da merenja istog uzorka ili senzora ne budu istovremeno uključena u skup za obučavanje i skup za testiranje. Način grupisanja zavisi od toga da li se ispituje prenos na novo stanje zemljišta ili na prethodno neviđen senzor.

Ovaj protokol predstavlja plan i izvršivo istraživačko okruženje, a ne objavljen empirijski rezultat. Zato se na njegovoj osnovi ne tvrdi da je neki kandidat precizniji od ostalih. Uvođenje u stvarnu primenu zahteva obučavanje na odobrenim podacima, zabeleženu verziju modela, podatke o ciljevima i evaluaciji, povezivanje modela sa aplikacijom i proveru na zamrznutom referentnom skupu. Dok ti uslovi nisu ispunjeni, aplikacija prijavljuje da prediktivni model nije spreman za upotrebu.

U proveru SoilLab-a uspešno je izvršeno svih 111 testova. Oni obuhvataju domenske ugovore, uvoz merenja, kvalitet spektra, korisnički interfejs, granice BAB modula, pripremu podataka i predikcione adaptore. Rezultat potvrđuje ispitane programske ugovore, dok postavljanje u radno okruženje, rad sa bazom, obučavanje svih kandidata i naučna tačnost predikcija nisu provereni ovim postupkom.

6.3.3 DentaScape 2: primena modularne biblioteke BAB 2

DentaScape 2 je poslovna aplikacija za rad stomatološke ordinacije, realizovana pomoću jezika Java, korisničkog interfejsa Vaadin i modularne biblioteke BAB 2. Pored osnovnog modula, uključuje module za elektronsku poštu, SMS, izveštavanje, SaaS, potpisivanje i Web Push. U odnosu na SoilLab, ovaj primer pokazuje upotrebu šireg skupa zajedničkih izvršnih sposobnosti.

Domenski izvori obuhvataju dijagnoze, intervencije, posete, termine, sale, korisnike i finansijske zapise, dok aplikacioni sloj koristi apstrakcije modularne BAB 2 biblioteke. DentaScape je ručno projektovan prema istom razdvajanju domenskog modela, aplikacionih ugovora i platformskih realizacija koje metodologija formalizuje. Time daje konkretan dokaz njene ručne primene i upotrebe BAB 2 ugovora izvan same biblioteke. Pošto konceptualni PIM i međustanja nisu sačuvani kao OWL/TTL artefakti, nije moguće naknadno potvrditi njegov transformacioni trag sertifikatom. Sposobnost

OwlBabGenerator-a da proizvede BAB 2 projekat proverena je odvojeno na PostOffice i ciljanim prihvatnim primerima.

DentaScape je uspešno kompajliran i svih pet postojećih testova završeno je uspešno. Korišćena BAB biblioteka proverena je zasebno, sa 176 uspešnih testova u sklopu izgradnje 12 modula. Ovi rezultati podržavaju kompilacionu i testnu saglasnost aplikacije sa proverenim bibliotekama. Serversko raspoređivanje, veza sa bazom podataka, korisnički scenariji u pregledaču i pokrivenost koda nisu obuhvaćeni proverom.

6.4 Objavljena validaciona studija: statička klasifikacija u dva domena

6.4.1 Domet studije

U radu [16] ispitana je mogućnost da se model podataka, pravila klasifikacije i aplikaciona infrastruktura, koji se u konvencionalnim sistemima često održavaju kao tri nezavisna artefakta, objedine u ontologijom vođenom toku razvoja. Studija obuhvata domen vrednovanja istraživačkih rezultata i domen prenatalne dijagnostike. U oba slučaja razmatrana su pravila statičke klasifikacije nad strukturisanim zapisima, odnosno ona klasa problema koja se može neposredno predstaviti OWL definisanim klasama.

Iz studije se preuzimaju istraživački protokol, rezultati zaključivanja, rezultati klasifikacije, testovi namerno unetih grešaka i precizno navedene granice merenja. Ti rezultati se ne koriste kao opis implementacije generatora. Aktuelnu implementaciju određuju modularni BAB PIM 2.0, BAB Mapping 2.0, BABj 2.0, ApplicationSpec, ugovori zasnovani na sposobnostima i deterministički generatorski tok opisani u poglavljima od 3. do 5.

6.4.2 Domen vrednovanja istraživačkih rezultata

Ontologija ResearchEvaluation u objavljenoj studiji obuhvatila je šest domenskih entiteta: konferenciju, zbornik radova, publikaciju, organizaciju, osobu i pravilnik vrednovanja. Pravila su formalizovala, između ostalog, kategorije međunarodne konferencije (M30), nacionalne konferencije (M60) i rada u celini objavljenog u zborniku međunarodne konferencije (M33). HermiT je potvrdio konzistentnost ontologije i zadovoljivost svih posmatranih klasa, a zaključivanjem je izvedeno da je InternationalConference potklasa klase NationalConference.

Poslednji nalaz je metodološki važan. Ako jedan zapis istovremeno zadovoljava uslove više kategorija, samo klasno zaključivanje ne može da sprovede pravilo „primeni najvišu odgovarajuću kategoriju”. Zbog toga je izbor konačne kategorije u generisanoj aplikaciji sproveden zasebnim, determinističkim korakom zasnovanim na prioritetu. Na podacima studije slučaja ICIST 2014 generisani sistem proizveo je očekivane klasifikacije M30 i M33. Time je potvrđena saglasnost rezultata za ispitane slučajeve, a ne opšta semantička ekvivalencija različitih formalizama za pravila.

Komplementarnost ontološke provere pod otvorenom pretpostavkom sveta i izvršne provere kompletnog zapisa pod zatvorenom pretpostavkom sveta dodatno je ispitana namernim unošenjem grešaka. Rezultati su sažeti u narednoj tabeli.

Namerno unete greške i način njihovog otkrivanja prikazani su u Tabeli 6.5.

Oznaka	Namerno uneta promena	Ishod i mehanizam otkrivanja
F1	Vrednost totalPapers nije navedena	HermiT ne izvodi pripadnost klasi; Java validacija odbija zapis kao nepotpun.
F2	totalPapers = 5, ispod praga 10	Zapis je potpun, ali uslov klasifikacije nije ispunjen; aplikacija vraća EXCLUDED.
F3	Tekstualna vrednost upisana u celobrojno polje	HermiT prijavljuje nekonzistentnost zbog povrede opsega tipa podatka.
F4	Ista individua deklarirana i kao konferencija i kao publikacija	HermiT prijavljuje nekonzistentnost zbog aksioma disjunktnosti.
F5	Definisana klasa sa protivrečnim pragovima	HermiT utvrđuje da je klasa nezadovoljiva.

Tabela 6.5. Namerno unete greške u objavljenoj studiji [16].

Rezultati pokazuju da se logička konzistentnost, zadovoljivost klasa i izvođenje klasne hijerarhije proveravaju na ontološkom nivou, dok se potpunost poslovnog zapisa i konačno vrednovanje predikata proveravaju u generisanoj aplikaciji. Odsustvo podatka pod otvorenom pretpostavkom sveta nije isto što i negativna vrednost uslova; zato su oba nivoa provere potrebna.

6.4.3 Provera prenosivosti u prenatalnom domenu

Drugi eksperiment primenio je isti ontologijom vođeni tok na statička pravila prenatalne dijagnostike. Korišćeno je šest sintetičkih zapisa: slučaj preeklampsije, slučaj eklampsije i četiri negativna ili granična slučaja. HermiT je potvrdio konzistentnost kombinovane ontologije, zadovoljivost posmatranih dijagnostičkih klasa i odnos u kome je eklampsija potklasa preeklampsije. Generisana aplikacija je za svih šest zapisa proizvela klasifikacije saglasne sa očekivanim ishodima i ontološkim izvođenjima, bez izmene generatora specifične za prenatalni domen [16].

Ovaj rezultat pruža ograničen dokaz prenosivosti na drugi domen statičke klasifikacije. Ne obuhvata računanje aritmetičkog rizika niti obradu vremenskih tokova događaja. Aritmetička pravila mogu zahtevati komplementarni formalizam, poput SWRL-a ili DMN-a, dok je za kontinualnu obradu događaja prikladnija hibridna arhitektura sa proceduralnom ili CEP komponentom. Stoga rezultat ne podržava tvrdnju da je OWL pristup univerzalna zamena za sve vrste sistema zasnovanih na pravilima.

6.4.4 Delimična merenja vremena izvršavanja

U objavljenoj studiji sprovedeno je po 30 merenja nakon pet zagrevanja, na računaru sa operativnim sistemom Windows 11, procesorom Intel Core Ultra 7 155H, 32 GB

memorije i okruženjem Eclipse Temurin OpenJDK 24.0.2. Dobijeni rezultati prikazani su u Tabeli 6.6.

Ontologija	Operacija	Broj ponavljanja	Srednja vrednost $\pm$ SD (ms)	Medijana (ms)	Minimum do maksimum (ms)
ResearchEvaluation	HermiT zaključivanje	30	1465,684 $\pm$ 243,356	1405,392	od 1184,527 do 2240,665
PrenatalDiagnosis	HermiT zaključivanje	30	1369,898 $\pm$ 258,580	1293,901	od 1116,119 do 2091,937
ResearchEvaluation	BAB generisanje koda	30	100,208 $\pm$ 66,049	89,467	od 36,126 do 309,954
PrenatalDiagnosis	BAB generisanje koda	30	106,654 $\pm$ 109,054	70,265	od 19,347 do 433,398

Tabela 6.6. Delimična merenja HermiT zaključivanja i BAB generisanja u objavljenoj studiji [16].

Granice merenja moraju ostati izričite. HermiT je pokretan preko Owlready2 0.51¹⁸, pri čemu je svako ponavljanje koristilo novi World i novi proces Java virtuelne mašine, a učitavanje ontologije nije bilo uključeno u primarno merenje. Merenje generisanja obuhvatilo je samo poziv metode BabProjectGenerator.generate() nakon što je ontologija već bila učitana, proverena i izdvojena u aplikacioni model. Provera generisanog projekta i uklanjanje izlaznih datoteka obavljani su izvan merenog intervala. Ove vrednosti zato ne predstavljaju vreme klasifikacije u generisanoj aplikaciji, potpuno vreme celog toka niti merenje aktuelne implementacije. One su opisni podaci za dve male ontologije i ne dokazuju skalabilnost niti prednost u pogledu učinka u odnosu na Jess i Drools¹⁹ ili druge sisteme za obradu pravila.

6.5 Evaluacija delimične operativne realizacije ABPMS osobina

ABPMS se ovde razmatra kroz pet osobina iz manifesta: autonomiju u zadatim okvirima, sposobnost delovanja kroz razgovor, adaptivnost, samousavršavanje i objašnjivost [2]. Evaluacija razdvaja dostupnost komponente, proveru njenog ponašanja i realizaciju osobine na nivou celog sistema. Ove kategorije su potrebne jer, na primer, čuvanje razgovora jeste mehanizam memorije, ali ne dokazuje samousavršavanje procesa.

6.5.1 Objašnjivost: upiti nad modelom i transparentnost transformacija

Objašnjivost transformacije zasniva se na tome da je moguće utvrditi izvor elementa, izabranu realizaciju i nastali artefakt. Za PostOffice to znači praćenje veze Parcel $\rightarrow$ preslikavanje entiteta u BABj $\rightarrow$ uloga entity $\rightarrow$ šablon entiteta $\rightarrow$ model/Parcel.java. Kod projekcionog atributa potrebno je pratiti i putanju ParcelSenderEnd $\rightarrow$ CustomerName, iz koje nastaju kolona DTO-a i deo upita. Takav trag objašnjava zašto je artefakt određene vrste nastao i odakle potiču izabrani strukturni podaci.

¹⁸ Zvanična dokumentacija paketa Owlready2: <https://owlready2.readthedocs.io/en/latest/>

¹⁹ Zvanične stranice sistema za obradu pravila: Jess: <https://ip.sandia.gov/software/jess/>; Drools: <https://www.drools.org/>

Dokazna osnova ovog traga raspoređena je između PIM-a, ApplicationSpec modela, BABj profila, PSM razrešavača, šablona i izlaznih metapodataka. application-metadata.json daje aplikacioni pregled, dok .owlbab-manifest povezuje upravljane putanje sa sažecima sadržaja. Manifest proverava identitet i promenu datoteke; sam po sebi ne objašnjava poslovno značenje svih njenih naredbi. Sertifikat dodatno identifikuje transformacionu specifikaciju, ali ne rekonstruiše ceo PIM iz izlaznog koda.

SPARQL konzola omogućava pregled modela i odgovore na formalno postavljena pitanja, na primer koji entiteti imaju određenu sposobnost ili koje uloge pripadaju zadatku. Ona je pomoćni analitički interfejs. Generatorski tok opisan u potpoglavlju 5.3 koristi neposrednu ekstrakciju nad Jena grafom, a FreeMarker dobija pripremljene vrednosti.

Na nivou agenata, pregledani izvori BAB biblioteke sadrže AgentMonitorRegistry, koji prati registrovanost, obradu događaja, brojače ishoda i vremenske oznake. Takvi podaci podržavaju operativnu inspekciju. Za dokaz da korisnik razume odluku sistema ipak je potrebna korisnička evaluacija: tačan trag porekla koda i razumljivo obrazloženje poslovne odluke predstavljaju različite zahteve.

6.5.2 Adaptivnost: promena skupa tehnologija i ponašanja

Adaptivnost ima razvojni i operativni aspekt. Razvojni aspekt obuhvata promenu PIM-a, izbor profila realizacije i novu generaciju aplikacije. Operativni aspekt odnosi se na prilagođavanje tokom rada sistema. Izbor BAB 2 modula potrebnih tokom izvršavanja na osnovu sposobnosti pripada prvom aspektu, dok kontrole agenata iz potpoglavlja 5.5 pripadaju drugom.

U razvojnem toku promena domenskog atributa, projekcije ili sposobnosti unosi se u odgovarajući PIM modul, a posledice se prenose kroz zajednički generator. Promena platforme može zahtevati izmene profila, razrešavača, šablona i BAB biblioteke. Za procenu napora potrebno je beležiti rad u svakom od tih slojeva, uključujući ručne dorade i regresionu proveru. Sama razlika broja redova pre i posle promene nije pouzdana zamena za taj rad.

Manifest pruža kontrolu nad ponovnom generacijom: lokalno izmenjena upravljana datoteka ne prepisuje se bez izričite politike zamene. To je zaštita integriteta izlaza, ali i praktično ograničenje adaptacije. Ako je razvojni tim menjao generisane datoteke, mora razrešiti njihov odnos prema novom izlazu; generator ne obavlja automatsko semantičko spajanje promena.

WasteX sadrži aplikacioni AgentRuntimeManager, izveden iz generičkog BAB upravljača. U pregledanim izvorima biblioteke postoje registracija, pauziranje, nastavak i zaustavljanje agenata i podešavanje izvršnog skupa niti. To su konkretni mehanizmi upravljanja komponentama. Oni ne predstavljaju sami po sebi izbor boljeg poslovnog procesa na osnovu cilja, događaja i izmerenih posledica. Za tu tvrdnju potreban je

scenario u kome se dokumentuju uzrok promene, odluka, primenjena adaptacija i rezultat u odnosu na prethodno ponašanje.

6.5.3 Sposobnost delovanja kroz razgovor i formalne osnove za autonomiju

Konverzaciona podrška zahteva više od mogućnosti izvršavanja SPARQL upita. U pregledanom kodu agenata `UserHelpDeskAgent` prima poruku, klasifikuje nameru i bira granu za tehničku prijavu, navigacioni odgovor ili razgovor. Ovaj mehanizam predstavlja implementacionu osnovu konverzacije, ali pregled ne obuhvata kontrolisani skup korisničkih zadataka i očekivanih odgovora. Zato se ne navode stopa uspeha razgovora, tačnost prevođenja prirodnog jezika u upit niti uspešnost izvršavanja poslovnih radnji na osnovu razgovora.

Tok predlaganja ispravke razdvaja prepoznavanje problema, predlog, proveru i radnju. `SelfHealingAgent` koristi strukturisani odgovor jezičkog modela, dok `SqlErrorHealingAgent` proverava predlog preko kriterijuma zabranjenih potpisa. U pregledanoj realizaciji `ForbiddenSignatureCriterion` proverava prisustvo zadatih podniski; to nije dokaz očuvanja ontoloških ili baznih invarijanti. Vrednost `confidenceScore`, koju vraća model, takođe nije empirijski kalibrisana verovatnoća ispravnosti.

Pregledana metoda `FixAction.execute()` beleži predlog u dnevnik i ostavlja stvarno izvršenje kao tačku za dalju realizaciju. Taj kod potvrđuje postojanje prototipa toka predloga i provere, ali ne potvrđuje automatsku popravku aplikacije. Zato se ovaj mehanizam koristi kao primer proširivosti sloja agenata, a ne kao dokaz potpuno autonomnog samopopravljanja sistema.

Formalni rezultat iz Poglavlja 4 odnosi se na determinističku transformaciju odabranih šematskih artefakata. On ne proširuje svoje garancije na odgovor jezičkog modela, sadržaj predložene ispravke ili ishod njenog izvršavanja. Povezivanje takve radnje sa formalnim modelom zahtevalo bi eksplicitne preduslove i postuslove i zasebnu proveru njihovog očuvanja.

ABPMS osobina	Raspoloživa podrška	Status u ovoj evaluaciji
Objašnjivost	Trag od PIM-a, preko PSM-a, do artefakta i nadzor događaja	Utvrđena podrška za strukturnu analizu i uvid u rad sistema; razumljivost objašnjenja korisniku nije merena
Adaptivnost	Razdvojene realizacione odluke i kontrole agenata	Provereno postojanje realizacionih izbora i kontrola; autonomna adaptacija procesa nije proverena
Sposobnost delovanja kroz razgovor	UserHelpDeskAgent, klasifikacija namere i poruke	Pregledana komponenta; nema sistematske evaluacije izvršenja zadataka razgovorom
Autonomija u zadatim okvirima	Predlog ispravke i kriterijum prihvatanja	Prototip toka sa ograničenom proverom; stvarna automatska popravka nije potvrđena
Samousavršavanje	Memorija razgovora i operativni brojači	Infrastrukturalna osnova; poboljšanje procesa na osnovu prethodnih ishoda nije izmereno

Tabela 6.7. Podrška za ABPMS osobine i granice raspoložive evaluacije.

Nalazi potvrđuju delimičnu operativnu realizaciju elemenata četiri ABPMS osobine kroz konkretne komponente, uz infrastrukturnu podršku za petu. Ne opravdavaju zbirnu tvrdnju da su četiri ili svih pet osobina potpuno empirijski potvrđene na nivou sistema. Za takav zaključak potrebno je odvojeno ispitati objašnjenja, razgovore, upravljanje agentima i izvršne radnje, uz poznate ulaze, očekivane ishode i evidentirane neuspehe.

6.6 Inženjerska verifikacija i kvantitativni pokazatelji

6.6.1 Aktuelna inženjerska verifikacija

Verifikacija ispituje tri povezana pitanja: da li ulazni modeli zadovoljavaju ontološke i operativne uslove generatora, da li je generisanje ponovljivo i da li stvarni programski izlaz odgovara izabranoj strukturi modela. Kompilacija i pakovanje projekta čine zaseban završni korak. Predmet provere je istraživačka verzija generatora sa strukturnom proverom izlaza, koja zadržava bifunktorski okvir i međukategoriju PSM-a opisane u radu Bogdanović i saradnici [15] i definisane u Poglavlju 4. Tačan sastav ove verzije i njena razlika u odnosu na prethodnu osnovu zabeleženi su u Prilogu G.

Test paket obuhvata 60 testova u 12 testnih klasa; svi su završeni bez neuspeha, grešaka i preskočenih testova. Provere pokrivaju ontološke i generatorske ugovore, ekstrakciju modela, razrešavanje profila, obradu šablona, ponovljeno generisanje, zaštitu ručnih izmena i strukturnu saglasnost izlaza. Broj testova opisuje obim ovog paketa, a ne broj nezavisnih domenskih studija ili procenat pokrivenosti koda.

Referentni modeli PostOffice i Catalog prošli su ontološku i ugovornu validaciju. Sertifikat ugrađene transformacione specifikacije potvrdio je svih pet implementiranih provera za 34 izabrane šematske relacije, pet tipiziranih normalizacija i 27 šablona. Dva generisanja fiksnog sertifikacionog korpusa proizvela su isti skup od 38 artefakata, sa

jednakim sadržajem. Ponovljivost se ovde odnosi na upravljane izvorne i konfiguracione datoteke; binarna jednakost WAR arhiva nije predmet ove provere.

Nezavisna strukturna provera prihvatila je oba referentna izlaza. Posmatrani PostOffice graf obuhvata 11 čvorova i 10 grana, a Catalog graf 22 čvora i 23 grane. To su grafovi izabranih entitetskih deklaracija i njihovih atributskih i relacionih mesta, definisani u potpoglavlju 4.5.3. Brojevi ne opisuju ceo ontološki ili programski graf. Sačuvani normalizovani PIM i razrešeni strukturni PSM omogućavaju praćenje odluka transformacije, dok se postojanje odgovarajućih programskih deklaracija utvrđuje neposrednim čitanjem Java izvora.

PostOffice i Catalog uspešno su kompajlirani i upakovani kao WAR projekti. Time je potvrđena mogućnost izgradnje konkretnih generisanih aplikacionih osnova nad BAB 2 bibliotekom. PostOffice je dodatno raspoređen i pokrenut na serveru WildFly 41.0.1, uz ručnu proveru prijavljivanja, CRUD operacija, navigacije, filtriranja i modelovanog ponašanja. Ova funkcionalna provera nije bila deo automatizovanog test paketa, a PostOffice projekat nije sadržao aplikacione testove. Serversko izvršavanje sistema Catalog nije provereno.

Poseban eksperiment ispituje da li strukturna provera otkriva odstupanje koje kompilacija dopušta. U kopiji Catalog izlaza metoda `Product.getName()` izmenjena je tako da vraća polje sku umesto polja name. Oba polja imaju tip `String`, a sačuvani PIM, PSM i aplikacioni metapodaci ostali su neizmenjeni. Ispravan i izmenjen projekat uspešno su kompajlirani, ali je strukturna provera odbila izmenjen izlaz i označila pogrešnu pristupnu metodu. Nalaz pokazuje dodatnu vrednost neposrednog poređenja modela i koda za ispitivanu vrstu odstupanja.

Regresioni i mutacioni testovi dodatno ispituju promene tipova, ciljeva relacija, anotacija i zavisnosti, uklanjanje minimalne kardinalnosti kolekcije, pogrešne normalizacione tipove i konflikte profila. Proverava se i da testna klasa ne može da zameni nedostajuću aplikacionu klasu. Kontrolna izmena komentara u Java izvoru ne menja ishod strukturne provere, što pokazuje da se prihvatanje izlaza ne svodi na tekstualnu jednakost sa očekivanom datotekom. Ovi primeri predstavljaju ciljne provere podržanog ugovora; ne daju iscrpnu evaluaciju svih programa niti formalni dokaz ispravnosti samog proverivača.

Rezultati aktuelne inženjerske verifikacije sažeti su u Tabeli 6.8.

Predmet provere	Rezultat	Obuhvat
Test paket generatora	60 uspešnih testova u 12 testnih klasa	Ontološki, generatorski i strukturni ugovori
Validacija PIM-a	PostOffice i Catalog prihvaćeni	Logička konzistentnost i operativni uslovi ulaza
Sertifikat transformacione specifikacije	Svih pet provera uspešno	34 relacije, pet tipiziranih normalizacija i 27 šablona
Ponovljivost	Jednak sadržaj 38 artefakata u dva generisanja	Fiksirani sertifikacioni korpus

Predmet provere	Rezultat	Obuhvat
PostOffice izlaz	31 upravljana datoteka, od toga 24 Java izvora; uspešna izgradnja i ručna funkcionalna provera na serveru WildFly 41.0.1	Prihvaćen strukturni graf sa 11 čvorova i 10 grana; potvrđeni prijavljivanje, CRUD, navigacija, filtriranje i modelovano ponašanje
Catalog izlaz	38 upravljanih datoteka, od toga 31 Java izvor; uspešna izgradnja	Prihvaćen strukturni graf sa 22 čvora i 23 grane
Namerno izmenjen Catalog izlaz	Kompilacija uspešna; strukturna provera odbija izlaz	Pristupna metoda vraća pogrešno polje istog tipa

Tabela 6.8. Rezultati verifikacije generatora i referentnih izlaza.

Zasebni ranije zabeleženi rezultati dopunjuju ovaj nalaz. Minimalni projekat uspešno je izgrađen samo sa osnovnim BAB modulom, dok je primer sa sposobnostima za agente, Web Push i WebAuthn uspešno izgrađen sa odgovarajućim opcionim modulima. Time je potvrđen izbor zavisnosti za ispitane kombinacije. Nezavisne provere BAB biblioteke, SoilLab-a i DentaScape-a dale su redom 176, 111 i pet uspešnih testova. Te provere odnose se na verzije navedene u Prilogu G i nisu ponovljene u okviru opisane verifikacije istraživačke verzije generatora. Ne sabiraju se sa njegovim testovima niti potvrđuju sve kombinacije modula. Ručna provera serverskog izvršavanja odnosi se samo na generisanu aplikaciju PostOffice i navedeni skup funkcija.

Za WasteX je sproveden pregled izvornog koda, bez nove izgradnje, funkcionalnih testova ili raspoređivanja. Pregledani korpus sadrži 890 Java datoteka i 91.184 fizička reda, uključujući komentare i prazne redove, uz isključivanje spoljnih biblioteka i izlaza izgradnje. Ove veličine opisuju razmeru aplikacije i ne predstavljaju broj poslovnih funkcija, broj generisanih klasa ili meru kvaliteta.

Rezultati podržavaju uslovno očuvanje izabrane šematske strukture i izvodljivost transformacije za proverene modele i BABj profil. Njihov domet ostaje na TBox/schema nivou: iz njih ne slede očuvanje poslovnih ABox podataka, semantička ekvivalencija izvršavanja ili povratna rekonstrukcija modela. Promena ontologija, generatora, šablona ili ciljne biblioteke zahteva ponavljanje relevantnih provera.

6.6.2 Metrike: tehnički dug, pokrivenost koda testovima i vreme razvoja

Broj uspešnih testova, broj artefakata i ponovljivost njihovog sadržaja neposredno opisuju provereni korpus. Oni ne predstavljaju meru tehničkog duga, pokrivenosti izvršenih instrukcija ili radnog vremena razvojnog tima. Merenje tih veličina zahteva različite instrumente i jedinice analize, kako je prikazano u Tabeli 6.9.

Veličina	Postupak i jedinica analize	Raspoloživi rezultat
Regresiona uspešnost	Ishodi identifikovanih test paketa	Brojevi testova, neuspeha i grešaka generatora, SoilLab-a, BAB 2 i DentaScape-a iz potpoglavlja 6.6.1
Obim izlaza	Broj upravljanih datoteka i Java izvornih datoteka konkretnog PIM-a	Zasebni rezultati za PostOffice i Catalog
Ponovljivost	Jednak sastav i sadržaj upravljanih izlaznih datoteka	Dva izvršavanja fiksiranog sertifikacionog korpusa

Veličina	Postupak i jedinica analize	Raspoloživi rezultat
Tehnički dug	Statička analiza uz fiksiranu verziju alata, profil pravila i obuhvat	Nema uporedivog izveštaja za ovde proverene izlaze i kontrolnu realizaciju
Pokrivenost koda	Instrumentacija instrukcija i grana; zasebno generator i aplikacija	Nema izveštaja iz kog bi se izračunao procenat pokrivenosti
Vreme razvoja	Evidencija rada od zahteva do istih kriterijuma prihvatanja	Nema uparenih merenja modelom vođenog i konvencionalnog razvoja
Trajanje generatorskog toka	Odvojeno učitavanje, validacija, ekstrakcija, PSM, generisanje i izgradnja	Objavljene delimične vrednosti iz 6.4.4 nisu merenja ove verzije toka

Tabela 6.9. Razgraničenje pokazatelja i status raspoloživih podataka.

Za statičku analizu potrebno je zasebno posmatrati generator, generisani aplikacioni kod, ručne dopune i ciljnu biblioteku. Ako se najveći deo ponašanja nasleđuje iz BAB biblioteke, nizak broj upozorenja u kratkim aplikacionim klasama ne opisuje kvalitet svih izvršnih putanja. Procena vremena otklanjanja upozorenja predstavlja model konkretnog alata, pa tehnički dug treba tumačiti uz korišćena pravila i potvrđene nalaze. Bez takvog izveštaja nije opravdano navesti nultu vrednost ili ocenu kvaliteta.

Pokrivenost instrukcija može se izraziti odnosom izvršenih i instrumentiranih instrukcija, a pokrivenost grana odnosom izvršenih i instrumentiranih grana. Test paket generatora proverava ekstrakciju, ugovore i sadržaj izlaza, dok bi testovi aplikacije proveravali njeno ponašanje. Uspeh svih izvršenih testova zato ne podrazumeva stoprocentnu pokrivenost koda.

Za vreme razvoja relevantan je ukupan napor do prihvaćene funkcionalnosti. U modelom vođenom postupku on obuhvata modelovanje, ispravljanje validacionih problema, eventualnu dopunu preslikavanja ili šablona, dopunu poslovne logike, integraciju i testiranje. U konvencionalnom postupku obuhvata ručno povezivanje istih slojeva i istu završnu proveru. Vreme rada mašine i vreme aktivnog rada čoveka beleže se odvojeno. Istorija Git²⁰ izmena može pomoći u rekonstrukciji redosleda promena, ali razmak između datuma izmena ne predstavlja broj radnih sati.

6.6.3 Poređenje sa konvencionalnim razvojem

Raspoložive studije omogućavaju kvalitativno poređenje mesta na kome se odluke održavaju. U ontologijom vođenom postupku strukturne odluke i podržani ugovori unose se u PIM i prenose kroz generator. U konvencionalnom postupku odgovarajuće odluke unose se u entitete, DTO, servisni i UI sloj i povezuju ručno. Dobit koja je neposredno pokazana jeste automatizovano stvaranje međusobno povezanih artefakata. Veličina uštede ukupnog rada nije utvrđena kontrolisanim poređenjem.

WasteX nije kontrolna grupa za PostOffice. Sistemi se razlikuju po obimu, istoriji, domenu i količini poslovne logike. Iz odnosa njihovih veličina ne može se izračunati ubrzanje generisanja niti procenat ručno izbegnutog rada. Jednako tako, istorijska procena

²⁰ Zvanična stranica sistema Git: <https://git-scm.com/>

autora o tipičnom razvoju nema istu težinu kao vremenska evidencija istog zadatka izvedenog pod uporedivim uslovima.

Kontrolisano poređenje može se organizovati kao upareni zadatak. Početni zahtev obuhvatio bi isti skup entiteta, relaciju, projekciju, ovlašćenja i procesni prelaz; zatim bi sledila unapred određena promena, na primer dodavanje atributa i pripadajuće kolone ili izmena uslova prelaza. Obe realizacije morale bi da zadovolje iste funkcionalne testove. Za izdvajanje doprinosa generatora ručna realizacija koristila bi istu BAB biblioteku i isti ciljni skup tehnologija. Poređenje sa razvojem bez BAB-a merilo bi zajednički doprinos biblioteke i generatora i predstavljalo drugi eksperiment.

Pri uključivanju više učesnika potrebno je uravnotežiti iskustvo i redosled rada, kako učenje domena u prvom zadatku ne bi pogodovalo drugom. Beležili bi se aktivni sati, ručne intervencije, greške utvrđene kriterijumima prihvatanja i napor u zadatku izmene. Trošak izrade zajedničkog generatora i šablona iskazivao bi se odvojeno od njihove primene u pojedinačnoj aplikaciji. Ovaj postupak određuje uslove za proveru tvrdnje o produktivnosti; u ovoj evaluaciji nije sproveden, pa se ne iskazuju uporedne vrednosti ili statistički zaključci.

6.6.4 Ispunjenje istraživačkih pitanja

Odgovori na istraživačka pitanja povezuju formalni rezultat, analizu implementacije i empirijsku evidenciju. Izraz „potvrđeno” odnosi se na konkretan ispitani fragment, dok „delimično” označava da je deo pitanja odgovoren, ali šira tvrdnja zahteva dodatnu proveru. Ova razlika sprečava da se implementacioni primer proglasi dokazom svih delova istraživačke hipoteze.

Veza rezultata evaluacije sa istraživačkim pitanjima prikazana je u Tabeli 6.10.

Pitanje	Odgovor zasnovan na rezultatima	Granica zaključka
RQ1 - Ontologija kao PIM za PAIS	Modularni PIM povezuje entitete, neutralne relacije, prikaze, uloge, sposobnosti i procesni skelet; PostOffice pokazuje automatizovani generatorski tok, dok WasteX, SoilLab i DentaScape pokazuju ručnu primenu istog metodološkog razdvajanja u dodatnim domenima.	Za ručne primere nisu sačuvani formalni PIM i međustanja, pa se njihov ceo transformacioni trag ne može reprodukovati ili sertifikovati; nije pokazana ni opšta potpunost za sve procesne semantike.
RQ2 - Kategorijalna formalizacija	Poglavlje 4 određuje uslovno očuvanje odabrane šematske strukture; provera konkretnog izlaza dodatno ustanovljava konačne uslove ugrađivanja izabranog grafa i otkriva ispitane promene koje prolaze kompilaciju.	Ne slede ABox očuvanje, semantička ekvivalencija izvršavanja, opšti mašinski dokaz teorije ili povratna rekonstrukcija.
RQ3 - Reifikacija i platformska nezavisnost	BAB Mapping i BABj eksplicitno razdvajaju nameru od realizacije, dok modularni BAB 2 ugovori razdvajaju osnovne i opcione izvršne sposobnosti.	Potpuno je realizovan jedan ciljni profil. Dodatni skup tehnologija i trošak prelaska ostaju neprovereni.

Pitanje	Odgovor zasnovan na rezultatima	Granica zaključka
RQ4 - ABPMS osobine	Sledljivost, kontrole agenata, UserHelpDeskAgent i tok predloga ispravke potvrđuju delimičnu realizaciju elemenata četiri osobine tokom rada sistema.	Nema dovoljno podataka o izvršavanju i ispitivanju korisnika za potvrdu četiri pune osobine na nivou sistema.
RQ5 - Razlika u odnosu na konvencionalni razvoj	Pokazani su generisanje povezanih slojeva, ponovljivost, rana validacija i primena BAB apstrakcija u WasteX-u, SoilLab-u i DentaScape-u; studija iz 6.2.4 daje zasebne pokazatelje razvojnog napora i grešaka, a studija statičke klasifikacije potvrđuje očekivane klasifikacije.	Za aktuelni generator kvantitativna prednost u radnim satima, tehničkom dugu i održavanju nije utvrđena uparenim poređenjem.

Tabela 6.10. Veza rezultata evaluacije sa istraživačkim pitanjima.

Istraživačka hipoteza podržana je u delu formalnog modelovanja, uslovnog šematskog očuvanja i izvodljivosti determinističkog generisanja za BABj. Identifikovana razvojna revizija generatora proizvodi modularne BAB 2 projekte i selektivno uključuje proverene opcione module. Ručna funkcionalna provera PostOffice-a na serveru WildFly 41.0.1 dodatno potvrđuje izvršivost generisanog sistema za prijavljivanje, CRUD operacije, navigaciju, filtriranje i modelovano ponašanje. WasteX, SoilLab i DentaScape dopunjuju taj nalaz primerima ručne primene metodologije i zajedničkih BAB 2 apstrakcija u dodatnim domenima. SoilLab i DentaScape imaju proverenu kompilacionu i testnu saglasnost sa odgovarajućim modulima, dok WasteX pokazuje arhitekturu složenog poslovnog sistema. Pošto formalni PIM i međustanja ovih ručnih primena nisu sačuvani, njihov transformacioni trag ne može se naknadno potpuno reprodukovati ili sertifikovati. Formalni aparat podržava višeciljnu konstrukciju, dok praktična realizacija dodatnog ciljnog profila ostaje otvorena za posebno ispitivanje. Puna operativna podrška ABPMS osobinama takođe ostaje predmet daljeg rada. Objavljena studija iz potpoglavlja 6.2.4 daje empirijsku podršku merljivom smanjenju razvojnog napora i broja prijavljenih grešaka u posmatranom slučaju, bez opšteg zaključka o veličini dobitka u svakom projektu.

7. Diskusija rezultata

7.1 Doprinos u odnosu na stanje oblasti

Rezultati na kojima se zasniva disertacija pripadaju trima slojevima dokaza. Prvi sloj je kategorijalna formalizacija transformacionog lanca, razvijena u radu Bogdanović i saradnici [15] i potpunije obrazložena u Poglavlju 4. Njome se razdvajaju funktorska svojstva specifikacije, koja važe po konstrukciji, od uslovne tvrdnje o očuvanju odabranog šematskog dela ontologije obuhvaćenog transformacionom specifikacijom, koja zahteva kompletnost i injektivnost preslikavanja, kompatibilnost sa kvocijentom i odsustvo neželjenog sažimanja elemenata. Drugi sloj je empirijska studija statičke klasifikacije [16], u kojoj su provereni ontološko zaključivanje, reprodukcija očekivanih klasifikacija, ponašanje pri namerno unetim greškama i ograničena prenosivost između dva domena. Treći sloj čini trenutna implementacija generatora: modularni BAB PIM 2.0, BAB Mapping 2.0, BABj 2.0 profil, nepromenljivi ApplicationSpec, razrešavanje vođeno sposobnostima, zaštita datoteka pod kontrolom generatora i sertifikat transformacione specifikacije.

Nadogradnja opisana u 6.6.1 jača treći sloj dokaza nezavisnim čitanjem konkretnog Java izlaza i proverom konačne korespondencije izabranog šematskog fragmenta. Primer gettera koji vraća pogrešno polje pokazuje razliku između uspešne kompilacije i saglasnosti sa izvornom specifikacijom. Taj rezultat dopunjuje inženjersku realizaciju opisanu u radu Bogdanović i saradnici [15] i formalizovanu u Poglavlju 4. Ne proširuje automatski garanciju na sve dorade preslikavanja, kategorijalne jednačine ili izvršno ponašanje sistema.

Ovakva podela je važna zato što formalni rezultat, empirijski rezultat i programska implementacija ne potvrđuju istu vrstu tvrdnje. Kategorijalni okvir određuje pod kojim se uslovima čuva izabrana struktura na nivou šeme, ali sam po sebi ne dokazuje semantičku ekvivalenciju izvršnog programa. Objavljena studija validacije potvrđuje saglasnost rezultata za konkretne skupove podataka i dve ontologije statičke klasifikacije, ali ne uspostavlja opštu prenosivost na proizvoljne domene. Trenutni generator pokazuje da se metodologija može realizovati determinističkim tokom i sa strožim operativnim ugovorima, ali njegovi testovi ne zamenjuju novu proveru u drugim domenima i ocenu performansi.

Rad [16] prepoznaje obrazac tri nezavisno održavana artefakta: model podataka, bazu pravila i aplikacionu infrastrukturu. Njihovo objedinjavanje u toku vođenom ontologijom smanjuje rizik od neusaglašenog ručnog održavanja. Trenutna arhitektura povezuje aplikacioni PIM, njegove PIM module, BAB Mapping, profil BABj, šablone i generisane artefakte eksplicitnim ugovorima i determinističkim transformacionim tokom. Apstraktne klase preslikavanja nalaze se u BAB Mapping 2.0, dok su konkretne realizacije u jeziku Java u BABj-u. Matematička kategorija C_{Core} ipak se posmatra nad unijom tih učitanih šema; fizička modularnost zato ne menja formalnu konstrukciju. Cilj nije svođenje svih

tehničkih datoteka na jednu datoteku, već uklanjanje više nezavisnih merodavnih izvora domenskog znanja.

U odnosu na klasične sisteme zasnovane na pravilima, prednost ontološkog pristupa najjasnije se ispoljava kod statičke klasifikacije nad strukturisanim zapisima. Logička konzistentnost, zadovoljivost klasa i subsumpcija mogu se proveriti pre generisanja koda, dok namenski validatori i generisana aplikacija proveravaju kompletnost zapisa i izvršavaju poslovne predikate pod zatvorenim pretpostavkom sveta. Testovi sa namerno unetim greškama F1-F5 pokazuju da ova dva nivoa nisu zamene jedan za drugi, već međusobno dopunjujući mehanizmi.

7.2 Ograničenja pristupa

Prvo ograničenje odnosi se na izražajni domet OWL 2 DL. Statička klasifikacija zasnovana na kvalitativnim uslovima dobro se preslikava u definisane klase i koristi mogućnosti zaključivača. Aritmetički proračuni, agregacije i složeno bodovanje samo su delimično pogodni za takav pristup i mogu zahtevati SWRL, DMN ili namenski proceduralni sloj. Obrada vremenskih tokova i složenih događaja nije prirodan zadatak za trenutni postupak, u kojem je OWL primarni formalizam; za nju je primerenija hibridna arhitektura sa CEP mehanizmom. Izbor ontologije zato mora zavisiti od strukture problema, a ne od pretpostavke da je jedan formalizam univerzalno nadmoćan.

Drugo ograničenje proističe iz otvorene pretpostavke sveta. Zaključivač ne tumači odsustvo podatka kao njegovu negaciju, pa se poslovna potpunost ne može pouzdano proveravati samo OWL zaključivanjem. Trenutna implementacija zato koristi HermiT nad zatvaračem uvoza PIM-a za logičke provere, a namenske validatore u jeziku Java za zatvorene ugovore, kompletnost i jednoznačnost podataka potrebnih generatoru. Ova podela povećava pouzdanost, ali zahteva disciplinovano održavanje granice između ontološke semantike i operativnih pravila.

Treće ograničenje odnosi se na platformsku nezavisnost. PIM i formalna porodica transformacija projektovani su tako da podrže više ciljnih skupova tehnologija, ali je BABj jedina potpuno realizovana i proveravana realizacija na konkretnoj platformi. Novi skup tehnologija ne zahteva promenu domenskog PIM-a, ali zahteva PSM profil, komponentu za razrešavanje platformskih odluka, šablone, validatore i testove. Zbog toga se podrška za više ciljnih platformi trenutno tumači kao arhitektonska mogućnost, a ne kao empirijski potvrđena jednakost više implementacija.

Četvrto ograničenje je obim empirijske provere trenutnog generatora. Objavljeni formalni i empirijski nalazi ostaju deo dokazne osnove samo u granicama svojih istraživačkih protokola. Poglavlje 6 dopunjuje ih proverom PostOffice PIM-a, generisanih BAB 2 projekata i referentnog korpusa, kao i analizom ručne primene metodologije u aplikacijama WasteX, SoilLab i DentaScape. Generisani PostOffice dodatno je raspoređen na serveru WildFly 41.0.1, gde su ručno provereni prijavljivanje, CRUD operacije, navigacija, filtriranje i modelovano ponašanje. Ova provera potvrđuje rad konkretnog izlaza, ali nije automatizovana korisnička ili dugotrajna operativna evaluacija.

U ručno razvijenim aplikacijama domenski i realizacioni koraci neposredno su pretočeni u kod, bez sačuvanog formalnog PIM-a i svih međustanja. SoilLab i DentaScape potvrđuju da je uz odgovarajuće module BAB 2 prevođenje uspešno i da testovi prolaze, dok WasteX pokazuje primenu arhitekture u složenom poslovnim sistemu. Odsustvo formalnog traga ograničava reprodukciju i sertifikaciju procesa, ali nije dokaz da metodologija nije korišćena. Za aktuelni generator nije sprovedeno kontrolisano poređenje radnih sati, tehničkog duga i korisničkih ishoda. Objavljena sekvencijalna studija iz potpoglavlja 6.2.4 daje zasebne pokazatelje razvojnog napora i prijavljenih grešaka za BAB okvir, uz ograničenja efekta učenja i redosleda realizacija; ona ne izdvaja doprinos sadašnjeg generatora.

Peto ograničenje je složenost usvajanja. Izrada kvalitetnog OWL PIM-a zahteva poznavanje ontološkog modelovanja, otvorene pretpostavke sveta i razlike između aksioma za zaključivanje i ograničenja za zatvorenu validaciju. Grafički alati ili alati prilagođeni domenu mogu smanjiti ovu prepreku, ali nisu deo postupka koji je u trenutnoj verziji rada proveren u celini.

7.3 Pretnje validnosti

Pretne internog validnosti proizlaze iz toga što su autori sistema učestvovali i u izradi ontologija, generatora i testova. Poznati slučajevi mogu nenamerno pogodovati podržanom skupu konstrukcija. Ovaj rizik je delimično ublažen namernim unošenjem različitih klasa grešaka i prenosom na drugi domen, ali su za snažniji zaključak potrebni nezavisno ponavljanje istraživanja i skup modela koji nisu nastali tokom razvoja generatora.

Pretne eksternog validnosti odnose se na mali broj domena. Objavljena studija obuhvata dve male ontologije statičke klasifikacije, dok se PostOffice, WasteX, SoilLab i DentaScape odnose na ograničen skup poslovnih i istraživačkih aplikacija. SoilLab proširuje domen na senzorska merenja, ali njegovih 111 testova aplikacije nije naučna validacija kandidata za predikciju. DentaScape dodaje stomatološki domen i proverenu integraciju sa BAB-om 2, ali njegovih pet testova ne predstavljaju potpunu funkcionalnu ili korisničku evaluaciju. WasteX doprinosi većim obimom i složenijim poslovnim vezama, ali u ovoj proveru nema zaseban korpus izvršenih testova. Rezultati se ne mogu bez dodatne provere preneti na velike ontologije, sisteme sa intenzivnom aritmetikom, obradu događaja u realnom vremenu ili sisteme kritične po bezbednost.

Pretne konstruktivnog validnosti naročito su važne kod merenja. Vreme samostalnog HermiT zaključivanja preko Owlready2 nije vreme klasifikacije u generisanoj aplikaciji. Isto tako, vreme poziva metode za generisanje, nakon prethodnog učitavanja, validacije i ekstrakcije modela, nije potpuno vreme od ontologije do proverene aplikacije. Ove veličine se zato prikazuju odvojeno i opisno, bez tvrdnje o prednosti u pogledu performansi. Buduća evaluacija trenutnog generatora treba unapred da definiše granice svakog merenja i da odvojeno prati validaciju ontologije, ekstrakciju PIM-a, PSM razrešavanje, generisanje, kompilaciju i klasifikaciju tokom izvršavanja.

Pretnja pouzdanosti potiče iz ograničenog korpusa prihvatnih testova. Trenutni testovi i sertifikat potvrđuju konkretne ontologije, očekivane ishode i testove sa namerno unetim greškama obuhvaćene tim korpusom, ali ne i proizvoljne domene ili modele. Za širu regresionu osnovu potrebno je uskladiti dodatne ontologije i očekivane ishode sa ugovorom BAB PIM 2.0/BAB Mapping 2.0/BABj 2.0 i ponoviti proveru nad trenutnim generatorom.

8. Zaključak i pravci daljih istraživanja

8.1 Zaključak

Disertacija predlaže metodologiju po kojoj je razvoj procesno svesnih informacionih sistema vođen ontologijama, pri čemu modularni BAB PIM 2.0 ima ulogu platformski nezavisne osnove, BAB Mapping 2.0 određuje apstraktne strategije realizacije, BABj 2.0 sadrži njihove Java/Jakarta EE/Vaadin realizacije, a deterministički generator proizvodi programske artefakte za ciljni skup tehnologija. Kategorijski okvir formalizuje ovaj tok kao kompoziciju transformacija i precizira uslove pod kojima se čuva odabrani šematski deo. Time se funktorska svojstva formalne specifikacije jasno odvajaju od ispravnosti rada generatora, koja se proverava validatorima, testovima, sertifikatom transformacione specifikacije i izgradnjom generisanog projekta.

Rad objavljen u časopisu *Computers* pokazao je da pristup vođen ontologijama može reprodukovati očekivane klasifikacije u domenu vrednovanja istraživačkih rezultata i u šest sintetičkih slučajeva prenatalne dijagnostike [16]. Testovi namerno unetih grešaka dodatno su pokazali uzajamno dopunjavanje OWL zaključivanja pod otvorenom pretpostavkom sveta i provere tokom izvršavanja pod zatvorenom pretpostavkom sveta. Ovi rezultati podržavaju izvodljivost pristupa za sisteme statičke klasifikacije, ali ne predstavljaju dokaz univerzalne primenljivosti na aritmetička pravila, vremensku obradu događaja ili proizvoljno velike ontologije.

Arhitekturu čine BAB Core sa zasebnim PIM modulima, rečnik BAB Mapping i BABj kao eksplicitan PSM profil. Generator razrešava platformsku semantiku i sposobnosti pre generisanja i koristi FreeMarker kao završnu projekciju prethodno pripremljene specifikacije. U matematičkom modelu C_{Core} je jedna kategorija nad unijom svih učitanih PIM modula i BAB Mapping šeme, sa domenskom i preslikavajućom punom potkategorijom. Poglavlja 3-5 daju merodavan opis implementacije generatora koja je predmet disertacije.

Rezultati podržavaju osnovnu istraživačku hipotezu u delu koji se odnosi na formalno modelovanje, determinističku transformaciju i izvodljivost generisanja za BABj. Validacija referentnih modela, testovi generatora, sertifikat transformacione specifikacije i uspešna izgradnja PostOffice i Catalog projekata potvrđuju ove osobine za ispitani korpus. Generisana aplikacija PostOffice dodatno je raspoređena i pokrenuta na serveru WildFly 41.0.1, gde su ručno provereni prijavljivanje, CRUD operacije, navigacija, filtriranje i modelovano ponašanje. Nezavisna provera generisanog koda otkrila je namerno odstupanje koje prolazi kompilaciju. Time je pokazana mogućnost povezivanja formalno određenog strukturnog ugovora sa proverom konkretnih programskih artefakata i radom generisanog sistema. Ručna provera PostOffice-a ne predstavlja automatizovanu proveru svih poslovnih scenarija niti se prenosi na druge generisane aplikacije.

WasteX, SoilLab i DentaScape dopunjuju nalaz primerima ručne primene metodologije u različitim domenima. Njihova arhitektura pokazuje upotrebu zajedničkih BAB

apstrakcija, dok zasebni testovi biblioteke, SoilLab-a i DentaScape-a potvrđuju ispitane programske ugovore. Pošto konceptualni PIM i transformaciona međustanja ovih primena nisu formalno sačuvani, za njih se ne tvrdi potpuna reproduktivnost ili sertifikacija toka od PIM-a do koda.

Evaluacija podržava delimičnu realizaciju elemenata četiri ABPMS osobine kroz konkretne komponente zasnovane na agentima. Potvrda punih osobina na nivou sistema, merljivo ubrzanje ukupnog razvoja i smanjenje tehničkog duga zahtevaju dodatno kontrolisano ispitivanje. Objavljena sekvencijalna studija iz potpoglavlja 6.2.4 daje pokazatelje razvojnog napora i prijavljenih grešaka, ali ne izdvaja doprinos aktuelnog generatora od efekta učenja i redosleda realizacija. Delimična merenja iz potpoglavlja 6.4.4 odnose se na pojedinačne faze obrade i ne zamenjuju poređenje ukupnog razvoja.

8.2 Pravci daljih istraživanja

8.2.1 Dvosmerni inženjering

Aktuelni tok je jednosmeran: od PIM-a, preko realizacionog profila, ka generisanom kodu. Dalji rad treba da razmotri kontrolisano vraćanje dozvoljenih izmena iz koda u model, razlikovanje generisanih delova od delova koji se ručno održavaju i formalne uslove pod kojima sinhronizacija ostaje dosledna. Jedan mogući pravac jeste formalizacija para apstrakcije i konkretizacije pomoću Galoisove veze, u smislu odnosa koji se koristi u teoriji apstraktne interpretacije [161], uz izričito definisanje informacija koje se pri svakoj transformaciji mogu izgubiti. Za praktičnu primenu biće neophodni mehanizmi otkrivanja sukoba i pravila prioriteta između modela i ručno izmenjenog koda.

8.2.2 Puna realizacija ABPMS osobina

Postojeći ontološki model, sledljivost generisanih artefakata i sloj zasnovan na agentima pružaju tehničku osnovu za objašnjivost, sposobnost delovanja kroz razgovor, prilagodljivost i autonomiju u zadatim okvirima. Dalji rad treba da ispita ove osobine kroz zadatke sa stvarnim korisnicima i dugotrajnim izvršavanjem sistema. Posebno su potrebni pouzdano povezivanje prirodnog jezika sa formalnim upitima, kontrolisano prilagođavanje procesa na osnovu događaja i merila, kao i mehanizam samousavršavanja sa proverljivim bezbednosnim ograničenjima. Statističke komponente ne treba da menjaju PIM ili izvršni sistem bez prolaska kroz izričitu proveru i pregled promene.

8.2.3 Realizacija alternativnih skupova platformskih tehnologija

Platformska nezavisnost treba da bude proverena najmanje jednom dodatnom potpunom realizacijom. Novi skup tehnologija mora obuhvatiti sopstveni PSM profil, razrešavač ciljnih tipova i ugovora, skup šablona, validatore, testove i proveru izvršivog izlaza. Poređenje sa BABj treba da obuhvati broj generisanih datoteka, očuvanje istog izabranog PIM fragmenta, ponašanje pri nepodržanim konstrukcijama i obim izmena specifičnih za

dati skup tehnologija. Tek takva paralelna implementacija može empirijski potvrditi praktičnu višeciljnu prirodu metodologije.

8.2.4 Domenski specifični jezici i grafički alati za ontološko modelovanje

Jedna od glavnih prepreka usvajanju pristupa jeste zahtev da softverski inženjeri neposredno rade sa RDF/OWL sintaksom i semantikom otvorenog sveta. Domenski specifičan jezik ili grafički alat mogao bi da ponudi pojmove bliske razvoju poslovnih aplikacija, dok bi u pozadini proizvodio validan PIM zasnovan na odgovarajućim modulima BAB PIM-a 2.0. Takav alat treba da prikazuje poreklo greške, razlikuje logičke aksiome od zatvorenih generatorskih ugovora i omogućava pregled veze između PIM elementa, izabranog PSM preslikavanja i generisanog artefakta.

8.2.5 Proširenje regresionog korpusa

Neposredni pravac daljeg rada jeste uključivanje ontologija ResearchEvaluation i PrenatalDiagnosis, očekivanih klasifikacija i testova grešaka F1-F5 u regresioni korpus aktuelnog BAB PIM 2.0/BAB Mapping 2.0/BABj 2.0 ugovora. Eksperiment treba sprovesti sa unapred definisanim granicama merenja za HermiT, izdvajanje PIM-a, PSM razrešavanje, generisanje, kompilaciju i izvršnu klasifikaciju. Time bi se proširila proverljiva regresiona osnova i dobio pouzdan odgovor o uticaju izmena generatora na ispravnost i učinak sistema.

Realizovana strukturna provera povezuje tipiziranu normalizaciju, sačuvani PSM i deklaracije pročitane iz Java koda. Dalji rad treba da proširi ovaj postupak na ostale konstrukcije i dorade, uz zasebno određivanje njihovih formalnih obaveza i empirijsku proveru. Bifunktorski okvir razvijen u radu Bogdanović i saradnici [15] i izložen u Poglavlju 4 ostaje osnova tog proširenja.

Literatura

- [1] Dumas, M., van der Aalst, W. M. P. i ter Hofstede, A. H. M. (ur.). (2005). *Process-Aware Information Systems: Bridging People and Software Through Process Technology*. John Wiley & Sons. ISBN 978-0-471-66306-5. DOI: 10.1002/0471741442.
- [2] Dumas, M., Fournier, F., Limonad, L., Marrella, A., Montali, M., Rehse, J.-R., Accorsi, R., Calvanese, D., De Giacomo, G., Fahland, D., Gal, A., La Rosa, M., Völzer, H. i Weber, I. (2023). „AI-Augmented Business Process Management Systems: A Research Manifesto”. *ACM Transactions on Management Information Systems*, 14(1), članak 11, 1-19. DOI: 10.1145/3576047. arXiv:2201.12855.
- [3] da Silva, A. R. (2015). „Model-Driven Engineering: A Survey Supported by the Unified Conceptual Model”. *Computer Languages, Systems & Structures*, 43, 139-155. DOI: 10.1016/j.cl.2015.06.001.
- [4] Selic, B. (2003). „The Pragmatics of Model-Driven Development”. *IEEE Software*, 20(5), 19-25. DOI: 10.1109/MS.2003.1231146.
- [5] Object Management Group. (2014a). *MDA Guide, Version 2.0*. Dokument ormsc/2014-06-01. <https://www.omg.org/cgi-bin/doc?ormsc/14-06-01>
- [6] Steinberg, D., Budinsky, F., Paternostro, M. i Merks, E. (2008). *EMF: Eclipse Modeling Framework* (2. izd.). Addison-Wesley Professional. ISBN 978-0-321-33188-5.
- [7] Elekes, M., Molnár, V. i Micskei, Z. (2023). „Assessing the Specification of Modelling Language Semantics: A Study on UML PSSM”. *Software Quality Journal*, 31(2), 575-617. DOI: 10.1007/s11219-023-09617-5.
- [8] Broy, M. i Cengarle, M. V. (2011). „UML Formal Semantics: Lessons Learned”. *Software and Systems Modeling*, 10(4), 441-446. DOI: 10.1007/s10270-011-0207-y.
- [9] Kurtev, I. (2008). „State of the Art of QVT: A Model Transformation Language Standard”. U *Applications of Graph Transformations with Industrial Relevance*, LNCS 5088, str. 377-393. Springer. DOI: 10.1007/978-3-540-89020-1_26.
- [10] van Amstel, M., Bosems, S., Kurtev, I. i Ferreira Pires, L. (2011). „Performance in Model Transformations: Experiments with ATL and QVT”. U *Theory and Practice of Model Transformations*, LNCS 6707. Springer. DOI: 10.1007/978-3-642-21732-6_14.
- [11] Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F. i Rudolph, S. (ur.). (2012). *OWL 2 Web Ontology Language Primer (Second Edition)*. W3C Recommendation. <https://www.w3.org/TR/2012/REC-owl2-primer-20121211/>
- [12] Bogdanović, B., Obradović, Đ., Segedinac, M. i Konjović, Z. (2024). „BAB Framework - Towards an Extensible Software Platform for AI-Augmented Process Aware Business Information Systems”. U Trajanović, Filipović i Zdravković (ur.), *Disruptive Information Technologies for a Smart Society*, LNNS 860, str. 197-212. Springer. DOI: 10.1007/978-3-031-71419-1_18.
- [13] Cyganiak, R., Wood, D. i Lanthaler, M. (ur.). (2014). *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. <https://www.w3.org/TR/rdf11-concepts/>
- [14] Harris, S. i Seaborne, A. (ur.). (2013). *SPARQL 1.1 Query Language*. W3C Recommendation. <https://www.w3.org/TR/sparql11-query/>

- [15] Bogdanović, B., Marković, V., Segedinac, M. i Konjović, Z. (2026b). „Methodology for rapid development of PAIS based on ontological modeling and category theory”. *Computer Science and Information Systems*, Online-First, br. 00, str. 44-44. DOI: 10.2298/CSIS260815044B.
- [16] Bogdanović, B. i Nikolić, S. (2026). „From Rule Engines to Ontologies: An OWL 2 DL Approach for Domain-Specific Evaluation Information Systems”. *Computers*, 15, članak 544. DOI: 10.3390/computers15080544.
- [17] van der Aalst, W. M. P. (2009). „Process-Aware Information Systems: Lessons to Be Learned from Process Mining”. U *Transactions on Petri Nets and Other Models of Concurrency II*, LNCS 5460, 1-26. DOI: 10.1007/978-3-642-00899-3_1.
- [18] Scheer, A.-W. (1999). *ARIS - Business Process Frameworks* (3. izd.). Springer Berlin, Heidelberg. DOI: 10.1007/978-3-642-58529-6.
- [19] Brown, C. (2005). „Workflow Management in Staffware”. U Dumas, M., van der Aalst, W. M. P. i ter Hofstede, A. H. M. (ur.), *Process-Aware Information Systems: Bridging People and Software through Process Technology*. Wiley. DOI: 10.1002/0471741442.ch14.
- [20] Andrews, T., Curbera, F., Dholakia, H., Golland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I. i Weerawarana, S. (2003). *Business Process Execution Language for Web Services, Version 1.1*. BEA Systems, IBM, Microsoft, SAP AG i Siebel Systems. <https://www.oasis-open.org/committees/download.php/2046/BPEL%20V1-1%20May%205%202003%20Final.pdf>
- [21] Dumas, M., La Rosa, M., Mendling, J. i Reijers, H. A. (2018). *Fundamentals of Business Process Management* (2. izd.). Springer. DOI: 10.1007/978-3-662-56509-4.
- [22] van der Aalst, W. M. P. i van Hee, K. M. (2002). *Workflow Management: Models, Methods, and Systems*. MIT Press. ISBN 978-0-262-01189-1.
- [23] Georgakopoulos, D., Hornick, M. i Sheth, A. (1995). „An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure”. *Distributed and Parallel Databases*, 3(2), 119-153. DOI: 10.1007/BF01277643.
- [24] van der Aalst, W. M. P. (2013). „Business Process Management: A Comprehensive Survey”. *ISRN Software Engineering*. DOI: 10.1155/2013/507984.
- [25] Weske, M. (2019). *Business Process Management: Concepts, Languages, Architectures* (3. izd.). Springer. ISBN 978-3-662-59431-5. DOI: 10.1007/978-3-662-59432-2.
- [26] Reichert, M. i Weber, B. (2012). *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer. ISBN 978-3-642-30408-8. DOI: 10.1007/978-3-642-30409-5.
- [27] Adams, M., ter Hofstede, A. H. M., Russell, N. i van der Aalst, W. M. P. (2010). „Dynamic and Context-Aware Process Adaptation”. U Wang i Sun (ur.), *Handbook of Research on Complex Dynamic Process Management*, str. 104-136. IGI Global. DOI: 10.4018/978-1-60566-669-3.ch005.
- [28] Hepp, M., Leymann, F., Domingue, J., Wahler, A. i Fensel, D. (2005). „Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management”. U *Proceedings of the IEEE International Conference on e-Business Engineering*, str. 535-540. DOI: 10.1109/ICEBE.2005.110.

- [29] van der Aalst, W. M. P. (2016). *Process Mining: Data Science in Action* (2. izd.). Springer. ISBN 978-3-662-49850-7. DOI: 10.1007/978-3-662-49851-4.
- [30] Camunda Platform Team. (2022). „Camunda Platform 8.0 Released: What's New”. *Camunda Blog*, 12. april 2022. <https://camunda.com/blog/2022/04/camunda-platform-8-0-released-whats-new/>
- [31] Freund, J. (2022). „Camunda Platform 8: A 10-Year Journey”. *Camunda Blog*, 14. april 2022. <https://camunda.com/blog/2022/04/camunda-platform-8-a-10-year-journey/>
- [32] IEEE Task Force on Process Mining. (2012). „Process Mining Manifesto”. U Daniel, Barkaoui i Dustdar (ur.), *Business Process Management Workshops*, LNBIP 99, str. 169-194. Springer. DOI: 10.1007/978-3-642-28108-2_19.
- [33] Kabicher-Fuchs, S., Rinderle-Ma, S., Recker, J., Indulska, M., Charoy, F., Christiaanse, E., Dunkl, R., Grambow, G., Kolb, J., Leopold, H. i Mendling, J. (2012a). „Human-Centric Process-Aware Information Systems (HC-PAIS)” [tehnički izveštaj]. arXiv:1211.4986. <https://arxiv.org/abs/1211.4986>
- [34] Kabicher-Fuchs, S. i Rinderle-Ma, S. (2012b). „Work Experience in PAIS - Concepts, Measurements and Potentials”. U *Advanced Information Systems Engineering - CAiSE 2012*, LNCS 7328, str. 678-694. Springer. DOI: 10.1007/978-3-642-31095-9_44.
- [35] Grambow, G., Oberhauser, R. i Reichert, M. (ur.). (2017). *Advances in Intelligent Process-Aware Information Systems: Concepts, Methods, and Technologies*. Intelligent Systems Reference Library, 123. Springer. ISBN 978-3-319-52179-4. DOI: 10.1007/978-3-319-52181-7.
- [36] Singer, G. (2019). „An Ontological Analysis of Business Process Modeling and Execution”. arXiv:1905.00499. <https://arxiv.org/abs/1905.00499>
- [37] Annane, A., Aussenac-Gilles, N. i Kamel, M. (2019). „BBO: BPMN 2.0 Based Ontology for Business Process Representation”. U *Proceedings of the 20th European Conference on Knowledge Management (ECKM 2019)*, sv. 1, str. 49-59. DOI: 10.34190/KM.19.113.
- [38] Sarker, M. K., Zhou, L., Eberhart, A. i Hitzler, P. (2021). „Neuro-Symbolic Artificial Intelligence - Current Trends”. *AI Communications*, 34(3), 197-209. DOI: 10.3233/AIC-210084.
- [39] Breit, A., Waltersdorfer, L., Ekaputra, F. J., Sabou, M., Ekelhart, A., Iana, A., Paulheim, H., Portisch, J., Revenko, A., ten Teije, A. i van Harmelen, F. (2023). „Combining Machine Learning and Semantic Web: A Systematic Mapping Study”. *ACM Computing Surveys*, 55(14s), članak 313, 1-41. DOI: 10.1145/3586163.
- [40] Russell, S. i Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4. izd.). Pearson. ISBN 978-0-13-461099-3.
- [41] Minsky, M. (1975). „A Framework for Representing Knowledge”. U P. H. Winston (ur.), *The Psychology of Computer Vision*, str. 211-277. McGraw-Hill. ISBN 978-0-07-071048-1.
- [42] Mayer, R. C., Davis, J. H. i Schoorman, F. D. (1995). „An Integrative Model of Organizational Trust”. *Academy of Management Review*, 20(3), 709-734. DOI: 10.5465/amr.1995.9508080335.
- [43] Toreini, E., Aitken, M., Coopamootoo, K., Elliott, K., Zelaya, C. G. i van Moorsel, A. (2020). „The Relationship Between Trust in AI and Trustworthy Machine Learning

- Technologies". U *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, str. 272-283. ACM. DOI: 10.1145/3351095.3372834.
- [44] Esser, S. i Fahland, D. (2021). „Multi-Dimensional Event Data in Graph Databases". *Journal on Data Semantics*, 10(1-2), 109-141. DOI: 10.1007/s13740-021-00122-1.
- [45] van der Aalst, W. M. P. (2021). „Hybrid Intelligence: To Automate or Not to Automate, That Is the Question". *International Journal of Information Systems and Project Management*, 9(2), 5-20. DOI: 10.12821/ijispm090201.
- [46] Casati, F., Ceri, S., Paraboschi, S. i Pozzi, G. (1999). „Specification and Implementation of Exceptions in Workflow Management Systems". *ACM Transactions on Database Systems*, 24(3), 405-451. DOI: 10.1145/328939.328996.
- [47] Reichert, M. i Dadam, P. (1998). „ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control". *Journal of Intelligent Information Systems*, 10(2), 93-129. DOI: 10.1023/A:1008604709862.
- [48] Reichert, M., Rinderle, S. i Dadam, P. (2003). „ADEPT Workflow Management System". U *Business Process Management - BPM 2003*, str. 370-379. Springer. DOI: 10.1007/3-540-44895-0_25.
- [49] Marrella, A., Mecella, M. i Sardiña, S. (2017). „Intelligent Process Adaptation in the SmartPM System". *ACM Transactions on Intelligent Systems and Technology*, 8(2), članak 25, 1-43. DOI: 10.1145/2948071.
- [50] Marrella, A., Mecella, M. i Sardiña, S. (2018). „Supporting Adaptiveness of Cyber-Physical Processes Through Action-Based Formalisms". *AI Communications*, 31(1), 47-74. DOI: 10.3233/AIC-170748.
- [51] Klein, M. i Dellarocas, C. (2000). „A Knowledge-Based Approach to Handling Exceptions in Workflow Systems". *Computer Supported Cooperative Work*, 9(3-4), 399-412. DOI: 10.1023/A:1008759413689.
- [52] Kearney, P. (2002). „Integrating AI Planning Techniques with Workflow Management System". *Knowledge-Based Systems*, 15(5-6), 285-291. DOI: 10.1016/S0950-7051(01)00167-8.
- [53] Cao, J., Yang, J., Chan, A. T. S. i Xu, Y. (2005). „Exception Handling in Distributed Workflow Systems Using Mobile Agents". U *IEEE International Conference on e-Business Engineering*, str. 48-55. DOI: 10.1109/ICEBE.2005.65.
- [54] De Nicola, A., Lezoche, M. i Missikoff, M. (2007). „An Ontological Approach to Business Process Modeling". U *Proceedings of the 3rd Indian International Conference on Artificial Intelligence*. ISBN 978-0-9727412-2-4.
- [55] Husáková, M. i Bureš, V. (2020). „Formal Ontologies in Information Systems Development: A Systematic Review". *Information*, 11(2), članak 66. DOI: 10.3390/info11020066.
- [56] Abonyi, J., Nagy, L. i Ruppert, T. (2024). *Ontology-Based Development of Industry 4.0 and 5.0 Solutions for Smart Manufacturing and Production*. Springer. DOI: 10.1007/978-3-031-47444-6.
- [57] Yang, C., Zheng, Y., Tu, X., Ala-Laurinaho, R., Autiosalo, J., Seppänen, O. i Tammi, K. (2023). „Ontology-Based Knowledge Representation of Industrial Production Workflow". *Advanced Engineering Informatics*, 58, članak 102185. DOI: 10.1016/j.aei.2023.102185.

- [58] Kratsch, W., Manderscheid, J., Röglinger, M. i Seyfried, J. (2021). „Machine Learning in Business Process Monitoring: A Comparison of Deep Learning and Classical Approaches Used for Outcome Prediction”. *Business & Information Systems Engineering*, 63, 261-276. DOI: 10.1007/s12599-020-00645-0.
- [59] Bharadiya, J. (2023). „Machine Learning and AI in Business Intelligence: Trends and Opportunities”. *International Journal of Computer*, 48(1), 123-134.
<https://www.ijcjournal.org/index.php/InternationalJournalOfComputer/article/view/2087>
- [60] Teubner, T., Flath, C. M., Weinhardt, C., van der Aalst, W. i Hinz, O. (2023). „Welcome to the Era of ChatGPT et al.” *Business & Information Systems Engineering*, 65(2), 95-101. DOI: 10.1007/s12599-023-00795-x.
- [61] Busch, K., Rochlitzer, A., Sola, D. i Leopold, H. (2023). „Just Tell Me: Prompt Engineering in Business Process Management”. U *International Conference on Business Process Modeling, Development and Support*, str. 3-11. Springer. DOI: 10.1007/978-3-031-34241-7_1.
- [62] Neuhaus, F. (2023). „Ontologies in the Era of Large Language Models - A Perspective”. *Applied Ontology*, 18(4), 399-407. DOI: 10.3233/AO-230072.
- [63] Babaei Giglou, H., D'Souza, J. i Auer, S. (2023). „LLMs4OL: Large Language Models for Ontology Learning”. U *The Semantic Web - ISWC 2023*, LNCS 14265, str. 408-427. Springer. DOI: 10.1007/978-3-031-47240-4_22.
- [64] Palagin, O., Kaverinskiy, V., Litvin, A. i Malakhov, K. (2023). „OntoChatGPT Information System: Ontology-Driven Structured Prompts for ChatGPT Meta-Learning”. *International Journal of Computing*, 22(2), 170-183. DOI: 10.47839/ijc.22.2.3086.
- [65] Nasiri, S., Adadi, A. i Lahmer, M. (2023). „Automatic Generation of Business Process Models from User Stories”. *International Journal of Electrical and Computer Engineering*, 13(1), 809-822. DOI: 10.11591/ijece.v13i1.pp809-822.
- [66] Schüler, P. i Alpers, S. (2024). „State of the Art: Automatic Generation of Business Process Models”. U *Business Process Management Workshops - BPM 2023*, LNBIP 492, str. 161-173. Springer. DOI: 10.1007/978-3-031-50974-2_13.
- [67] Greenfield, J., Short, K., Cook, S., Kent, S. i Crupi, J. (2004). *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley. ISBN 978-0-471-20284-4.
- [68] Miller, J. i Mukerji, J. (2003). *MDA Guide, Version 1.0.1*. Object Management Group, dokument omg/2003-06-01. <https://www.omg.org/cgi-bin/doc?omg/03-06-01>
- [69] Object Management Group. (2017b). *Unified Modeling Language (UML), Version 2.5.1*. Dokument formal/2017-12-05. <https://www.omg.org/spec/UML/2.5.1>
- [70] Object Management Group. (2016). *Meta Object Facility (MOF), Version 2.5.1*. Dokument formal/19-10-01. <https://www.omg.org/spec/MOF/2.5.1>
- [71] Object Management Group. (bez datuma). *MDA FAQ*.
https://www.omg.org/mda/faq_mda.htm (pristupljeno 10. septembra 2026).
- [72] Fuentes, L., Quintana, V., Llorens, J., Génova, G. i Prieto-Díaz, R. (2003). „Errors in the UML Metamodel?” *ACM SIGSOFT Software Engineering Notes*, 28(6), 3. DOI: 10.1145/966221.966236.

- [73] Fecher, H., Schönborn, J., Kvas, M. i de Roeper, W.-P. (2005). „29 New Uncertainties in the Semantics of UML 2.0 State Machines”. U Lau i Banach (ur.), *Formal Methods and Software Engineering*, LNCS 3785, str. 52-65. Springer. DOI: 10.1007/11576280_5.
- [74] Rączkowska, A. i Walkowiak-Gall, A. (2019). „15 Interpretation Problems in the Semantics of UML 2.5.1 Activities”. U *Information Systems Architecture and Technology*, AISC 852. Springer. DOI: 10.1007/978-3-319-99981-4_25.
- [75] Padi, S., Breiner, S. J., Subrahmanian, E. i Sriram, R. D. (2021). „Deconstructing UML, Part 1: Modeling Classes with Categories”. NISTIR 8358. DOI: 10.6028/NIST.IR.8358.
- [76] Object Management Group. (2021). *Semantics of a Foundational Subset for Executable UML Models (fUML), Version 1.5*. <https://www.omg.org/spec/FUML/1.5>
- [77] Object Management Group. (2019). *Precise Semantics of UML State Machines (PSSM), Version 1.0*. <https://www.omg.org/spec/PSSM/1.0>
- [78] Object Management Group. (2017a). *Action Language for Foundational UML (Alf), Version 1.1*. <https://www.omg.org/spec/ALF/1.1>
- [79] Broy, M., Crane, M. L., Dingel, J., Hartman, A., Rumpe, B. i Selic, B. (2007). „2nd UML 2 Semantics Symposium: Formal Semantics for UML”. U *Models in Software Engineering*, LNCS 4364, str. 318-323. Springer. DOI: 10.1007/978-3-540-69489-2_39.
- [80] Mkhinini, M., Labbani-Narsis, O. i Nicolle, C. (2020). „Combining UML and Ontology: An Exploratory Survey”. *Computer Science Review*. DOI: 10.1016/j.cosrev.2019.100223.
- [81] Object Management Group. (2014b). *Object Constraint Language (OCL), Version 2.4*. <https://www.omg.org/spec/OCL/2.4>
- [82] Wilke, C. i Demuth, B. (2011). „UML Is Still Inconsistent! How to Improve OCL Constraints in the UML 2.3 Superstructure”. U *Proceedings of the Workshop on OCL and Textual Modelling*, Electronic Communications of the EASST, 44. DOI: 10.14279/tuj.eceasst.44.669.
- [83] Damus, C., Sánchez-Barbudo Herrera, A., Uhl, A., Willink, E. i saradnici. (2018). *Eclipse OCL Documentation*, verzija 6.6.0. Eclipse Foundation. <https://download.eclipse.org/ocl/doc/6.6.0/ocl.pdf>
- [84] Kahani, N., Bagherzadeh, M., Cordy, J. R., Dingel, J. i Varró, D. (2019). „Survey and Classification of Model Transformation Tools”. *Software and Systems Modeling*, 18(4), 2361-2397. DOI: 10.1007/s10270-018-0665-6.
- [85] Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I. i Valduriez, P. (2006). „ATL: A QVT-like Transformation Language”. U *Companion to OOPSLA 2006*, str. 719-720. ACM. DOI: 10.1145/1176617.1176691.
- [86] Abdelmalek, H., Khriess, I. i Jakimi, A. (2024). „Towards an Effective Approach for Composition of Model Transformations”. *Frontiers in Computer Science*, 6, članak 1357845. DOI: 10.3389/fcomp.2024.1357845.
- [87] Object Management Group. (2015). *XML Metadata Interchange (XMI), Version 2.5.1*. Dokument formal/15-06-07. <https://www.omg.org/spec/XMI/2.5.1>
- [88] Schwichtenberg, S., Jovanovikj, I., Gerth, C. i Engels, G. (2018). „Poster: CrossEcore: An Extendible Framework to Use Ecore and OCL across Platforms”. U *Proceedings of the 40th*

- International Conference on Software Engineering: Companion Proceedings*, str. 292-293. ACM. DOI: 10.1145/3183440.3194976.
- [89] Hinkel, G. (2018). „NMF: A Multi-platform Modeling Framework”. U *Theory and Practice of Model Transformation, ICMT 2018*, LNCS 10888, str. 184-194. Springer. DOI: 10.1007/978-3-319-93317-7_10.
- [90] Hermannsdoerfer, M., Benz, S. i Juergens, E. (2008). „Automatability of Coupled Evolution of Metamodels and Models in Practice”. U *Proceedings of MODELS 2008*, LNCS 5301, str. 645-659. Springer. DOI: 10.1007/978-3-540-87875-9_45.
- [91] Rose, L. M., Kolovos, D. S., Paige, R. F., Polack, F. A. C. i Poulding, S. M. (2014). „Epsilon Flock: A Model Migration Language”. *Software & Systems Modeling*, 13(2), 735-755. DOI: 10.1007/s10270-012-0296-2.
- [92] Bettini, L. (2016). *Implementing Domain-Specific Languages with Xtext and Xtend* (2. izd.). Packt Publishing. ISBN 978-1-78646-496-5.
- [93] Eclipse Foundation. *Epsilon Documentation*. <https://eclipse.dev/epsilon/>
- [94] Object Management Group. (2008). *MOF Model to Text Transformation Language (MOFM2T), Version 1.0*. <https://www.omg.org/spec/MOFM2T/1.0>
- [95] Czarnecki, K. i Helsen, S. (2006). „Feature-Based Survey of Model Transformation Approaches”. *IBM Systems Journal*, 45(3), 621-645. DOI: 10.1147/sj.453.0621.
- [96] Rose, L. M., Paige, R. F., Kolovos, D. S. i Polack, F. A. C. (2008). „The Epsilon Generation Language”. U *Proceedings of ECMDA-FA 2008*, LNCS 5095, str. 1-16. Springer. DOI: 10.1007/978-3-540-69100-6_1.
- [97] Mens, T. i Van Gorp, P. (2006). „A Taxonomy of Model Transformation”. *Electronic Notes in Theoretical Computer Science*, 152, 125-142. DOI: 10.1016/j.entcs.2005.10.021.
- [98] Diskin, Z. i Maibaum, T. (2012). „Category Theory and Model-Driven Engineering: From Formal Semantics to Design Patterns and Beyond”. *Electronic Proceedings in Theoretical Computer Science*, 93, 1-21. DOI: 10.4204/EPTCS.93.1.
- [99] Kovalyov, S. P. (2016). „Category-Theoretic Approach to Software Systems Design”. *Journal of Mathematical Sciences*, 214(6), 814-853. DOI: 10.1007/s10958-016-2814-1.
- [100] Gruber, T. R. (1993). „A Translation Approach to Portable Ontology Specifications”. *Knowledge Acquisition*, 5(2), 199-220. DOI: 10.1006/knac.1993.1008.
- [101] Gruber, T. R. (1995). „Toward Principles for the Design of Ontologies Used for Knowledge Sharing”. *International Journal of Human-Computer Studies*, 43(5-6), 907-928. DOI: 10.1006/ijhc.1995.1081.
- [102] Borst, W. N. (1997). *Construction of Engineering Ontologies for Knowledge Sharing and Reuse* [doktorska disertacija]. University of Twente. DOI: 10.3990/1.9789036509886.
- [103] Studer, R., Benjamins, V. R. i Fensel, D. (1998). „Knowledge Engineering: Principles and Methods”. *Data & Knowledge Engineering*, 25(1-2), 161-197. DOI: 10.1016/S0169-023X(97)00056-6.
- [104] Guarino, N., Oberle, D. i Staab, S. (2009). „What Is an Ontology?” U *Handbook on Ontologies* (2. izd.), str. 1-17. Springer. DOI: 10.1007/978-3-540-92673-3_0.

- [105] Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F. i Stein, L. A. (2001). *DAML+OIL (March 2001) Reference Description*. W3C Note, 18. decembar 2001. <https://www.w3.org/TR/daml+oil-reference/>
- [106] Smith, M. K., Welty, C. i McGuinness, D. L. (ur.). (2004). *OWL Web Ontology Language Guide*. W3C Recommendation. <https://www.w3.org/TR/owl-guide/>
- [107] Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A. i Lutz, C. (ur.). (2012a). *OWL 2 Web Ontology Language Profiles (Second Edition)*. W3C Recommendation. <https://www.w3.org/TR/owl2-profiles/>
- [108] Artale, A., Calvanese, D., Kontchakov, R. i Zakharyashev, M. (2007). „DL-Lite in the Light of First-Order Logic”. U *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI 2007)*, str. 361-366. AAAI Press. <https://cdn.aaai.org/AAAI/2007/AAAI07-056.pdf>
- [109] Jackson, D. (2002). „Alloy: A Lightweight Object Modelling Notation”. *ACM Transactions on Software Engineering and Methodology*, 11(2), 256-290. DOI: 10.1145/505145.505149.
- [110] Bąk, K., Diskin, Z., Antkiewicz, M., Czarnecki, K. i Wąsowski, A. (2016). „Clafer: Unifying Class and Feature Modeling”. *Software & Systems Modeling*, 15, 811-845. DOI: 10.1007/s10270-014-0441-1.
- [111] Horrocks, I., Kutz, O. i Sattler, U. (2006). „The Even More Irresistible SROIQ”. U *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*, str. 57-67. AAAI Press. <https://cdn.aaai.org/KR/2006/KR06-009.pdf>
- [112] Motik, B., Patel-Schneider, P. F. i Cuenca Grau, B. (ur.). (2012b). *OWL 2 Web Ontology Language Direct Semantics (Second Edition)*. W3C Recommendation. <https://www.w3.org/TR/owl2-direct-semantics/>
- [113] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D. i Patel-Schneider, P. F. (ur.). (2007). *The Description Logic Handbook: Theory, Implementation, and Applications* (2. izd.). Cambridge University Press. ISBN 978-0-521-78176-3. DOI: 10.1017/CBO9780511711787. [Prvo izdanje: 2003.]
- [114] Hitzler, P., Krötzsch, M. i Rudolph, S. (2009). *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC. ISBN 978-1-4200-9050-5.
- [115] Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A. i Katz, Y. (2007). „Pellet: A Practical OWL-DL Reasoner”. *Journal of Web Semantics*, 5(2), 51-53. DOI: 10.1016/j.websem.2007.03.004.
- [116] Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosz, B. i Dean, M. (2004). *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C Member Submission. <https://www.w3.org/Submission/SWRL/>
- [117] Knublauch, H. i Kontokostas, D. (ur.). (2017). *Shapes Constraint Language (SHACL)*. W3C Recommendation. <https://www.w3.org/TR/shacl/>
- [118] Guarino, N. (1998). „Formal Ontology and Information Systems”. U Guarino (ur.), *Formal Ontology in Information Systems: Proceedings of FOIS'98*, str. 3-15. IOS Press. <https://www.loa.istc.cnr.it/old/Papers/FOIS98.pdf>

- [119] Guizzardi, G. (2005). *Ontological Foundations for Structural Conceptual Models*. Telematica Instituut Fundamental Research Series, 15. Twente University Press. ISBN 978-90-75176-81-3.
- [120] Guizzardi, G., Wagner, G., Almeida, J. P. A. i Guizzardi, R. S. S. (2015). „Towards Ontological Foundations for Conceptual Modeling: The Unified Foundational Ontology (UFO) Story”. *Applied Ontology*, 10(3-4), 259-271. DOI: 10.3233/AO-150157.
- [121] Annane, A., Kamel, M. i Aussenac-Gilles, N. (2020). „Comparing Business Process Ontologies for Task Monitoring”. U *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART 2020)*, str. 634-643. SCITEPRESS. DOI: 10.5220/0008978706340643.
- [122] Alnusair, A., Zhao, T. i Bodden, E. (2010). „Effective API Navigation and Reuse”. U *2010 IEEE International Conference on Information Reuse and Integration*, str. 7-12. IEEE. DOI: 10.1109/IRI.2010.5558972.
- [123] Atzeni, M. i Atzori, M. (2017a). „CodeOntology: RDF-ization of Source Code”. U d'Amato i drugi (ur.), *The Semantic Web - ISWC 2017*, LNCS 10588, str. 20-28. Springer. DOI: 10.1007/978-3-319-68204-4_2.
- [124] Atzeni, M. i Atzori, M. (2017b). „CodeOntology: Querying Source Code in a Semantic Framework”. U *Proceedings of the ISWC 2017 Posters & Demonstrations Track*, CEUR Workshop Proceedings 1963. <https://ceur-ws.org/Vol-1963/paper543.pdf>
- [125] de Aguiar, C. Z., Falbo, R. A. i Souza, V. E. S. (2019). „OOC-O: A Reference Ontology on Object-Oriented Code”. U *Conceptual Modeling - ER 2019*, LNCS 11788, str. 13-27. Springer. DOI: 10.1007/978-3-030-33223-5_3.
- [126] Atzeni, M. i Atzori, M. (2017c). *CodeOntology* [skup podataka]. Zenodo. DOI: 10.5281/zenodo.577939.
- [127] Atzeni, M. i Atzori, M. (2018). „Translating Natural Language to Code: An Unsupervised Ontology-Based Approach”. U *Proceedings of the 2018 IEEE International Conference on Artificial Intelligence and Knowledge Engineering (AIKE 2018)*, str. 1-8. DOI: 10.1109/AIKE.2018.00009.
- [128] Poozhithara, A., Asuncion, H. U. i Lagesse, B. (2022). „Towards Lightweight Detection of Design Patterns in Source Code”. U *Proceedings of the 34th International Conference on Software Engineering and Knowledge Engineering (SEKE 2022)*. DOI: 10.18293/seke2022-167.
- [129] He, X., Chen, R., Zhang, Z., Wang, Y. i Dong, Q. (2025). „A Hybrid Approach for EMF Code Generation: Code Templates Meet Large Language Models”. arXiv:2512.05498. <https://arxiv.org/abs/2512.05498>
- [130] Eilenberg, S. i Mac Lane, S. (1945). „General Theory of Natural Equivalences”. *Transactions of the American Mathematical Society*, 58(2), 231-294. DOI: 10.1090/S0002-9947-1945-0013131-6.
- [131] Eilenberg, S. i Steenrod, N. (1952). *Foundations of Algebraic Topology*. Princeton University Press.
- [132] Cartan, H. i Eilenberg, S. (1956). *Homological Algebra*. Princeton University Press.

- [133] Mac Lane, S. (1971/1998). *Categories for the Working Mathematician* (2. izd.). Graduate Texts in Mathematics, 5. Springer. ISBN 978-0-387-98403-2. DOI: 10.1007/978-1-4757-4721-8.
- [134] Barr, M. i Wells, C. (1990/1999). *Category Theory for Computing Science* (3. izd. 1999). Centre de recherches mathématiques. ISBN 978-2-921120-35-0.
- [135] Seemann, M. (2017-2018). „From Design Patterns to Category Theory” [serija mrežnih članaka]. <https://blog.ploeh.dk/2017/10/04/from-design-patterns-to-category-theory/> (pristupljeno 17. septembra 2026).
- [136] Marquis, J.-P. (2019). „Category Theory”. *Stanford Encyclopedia of Philosophy*. <https://plato.stanford.edu/entries/category-theory/>
- [137] Goguen, J. A. (1991). „A Categorical Manifesto”. *Mathematical Structures in Computer Science*, 1(1), 49-67. DOI: 10.1017/S0960129500000050.
- [138] Easterbrook, S. (1999). „An Introduction to Category Theory for Software Engineers” [predavački materijal]. University of Toronto, Department of Computer Science. <https://www.cs.toronto.edu/~sme/presentations/cat101.pdf>
- [139] Gamma, E., Helm, R., Johnson, R. i Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. ISBN 978-0-201-63361-0.
- [140] Milewski, B. (2018). *Category Theory for Programmers*. Blurb. ISBN 978-1-5184-0350-7. <https://bartoszmilewski.com/2014/10/28/category-theory-for-programmers-the-preface/>
- [141] Fong, B. i Spivak, D. I. (2019). *Seven Sketches in Compositionality: An Invitation to Applied Category Theory*. Cambridge University Press. DOI: 10.1017/9781108668804.
- [142] Schürr, A. (1994). „Specification of Graph Translators with Triple Graph Grammars”. U *Graph-Theoretic Concepts in Computer Science - WG 1994*, LNCS 903, str. 151-163. Springer. DOI: 10.1007/3-540-59071-4_45.
- [143] Anjorin, A., Schürr, A. i Taentzer, G. (2012). „Construction of Integrity Preserving Triple Graph Grammars”. U *Graph Transformations - 6th International Conference, ICGT 2012*, LNCS 7562, str. 356-370. Springer. DOI: 10.1007/978-3-642-33654-6_24.
- [144] Breiner, S., Subrahmanian, E. i Sriram, R. D. (2023). „Category Theory”. U Madni, Augustine i Sievers (ur.), *Handbook of Model-Based Systems Engineering*, str. 1259-1299. Springer. DOI: 10.1007/978-3-030-93582-5_85.
- [145] Vidalie, J., Batteux, M., Mhenni, F. i Choley, J.-Y. (2022). „Category Theory Framework for System Engineering and Safety Assessment Model Synchronization Methodologies”. *Applied Sciences*, 12(12), članak 5880. DOI: 10.3390/app12125880.
- [146] Mordecai, Y., Fairbanks, J. P. i Crawley, E. F. (2021). „Category-Theoretic Formulation of the Model-Based Systems Architecting Cognitive-Computational Cycle”. *Applied Sciences*, 11(4), članak 1945. DOI: 10.3390/app11041945.
- [147] Kovalyov, S. P. (2020). „Leveraging Category Theory in Model Based Enterprise”. *Advances in Systems Science and Applications*, 20(1), 50-65. DOI: 10.25728/assa.2020.20.1.781.
- [148] Syriani, E., Luhunu, L. i Sahraoui, H. (2018). „Systematic Mapping Study of Template-Based Code Generation”. *Computer Languages, Systems & Structures*, 52, 43-62. DOI: 10.1016/j.cl.2017.11.003. arXiv:1703.06353.

- [149] Alharbi, M. i Alshayeb, M. (2026). „Automatic Code Generation Techniques: A Systematic Literature Review”. *Automated Software Engineering*, 33(1), članak 4. DOI: 10.1007/s10515-025-00551-3.
- [150] Klein, J., Levinson, H. L. i Marchetti, J. (2015). *Model-Driven Engineering: Automatic Code Generation and Beyond*. Technical Note CMU/SEI-2015-TN-005. Software Engineering Institute, Carnegie Mellon University. <https://insights.sei.cmu.edu/library/model-driven-engineering-automatic-code-generation-and-beyond/>
- [151] Luhunu, L. i Syriani, E. (2017). „Comparison of the Expressiveness and Performance of Template-Based Code Generation Tools”. U *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering (SLE 2017)*, str. 206-216. ACM. DOI: 10.1145/3136014.3136021.
- [152] Sousa, T., Syriani, E. i Paquin, N. (2017). „Feedback on How MDE Tools Are Used Prior to Academic Collaboration”. U *Proceedings of the 32nd ACM SIGAPP Symposium on Applied Computing (SAC 2017)*, str. 1295-1302. ACM. DOI: 10.1145/3019612.3019775.
- [153] Hu, K., Duan, Z., Wang, J., Gao, L. i Shang, L. (2019). „Template-Based AADL Automatic Code Generation”. *Frontiers of Computer Science*, 13(4), 698-714. DOI: 10.1007/s11704-017-6477-y.
- [154] Apache Software Foundation. (2024). *Apache FreeMarker Documentation*. <https://freemarker.apache.org/docs/>
- [155] Baeldung. „FreeMarker Common Operations”. <https://www.baeldung.com/freemarker-operations>
- [156] Sharman, R., Kishore, R. i Ramesh, R. (ur.). (2007). *Ontologies - A Handbook of Principles, Concepts and Applications in Information Systems*. Integrated Series in Information Systems, 14. Springer. ISBN 978-0-387-37019-4. DOI: 10.1007/978-0-387-37022-4.
- [157] Goguen, J. A. i Burstall, R. M. (1984). „Introducing Institutions”. U Clarke i Kozen (ur.), *Logics of Programs*, LNCS 164, str. 221-256. Springer. DOI: 10.1007/3-540-12896-4_366.
- [158] Foundation for Intelligent Physical Agents. (2002). *FIPA ACL Message Structure Specification*. Dokument SC00061G. <http://www.fipa.org/specs/fipa00061/SC00061G.html>
- [159] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S. i Kiela, D. (2020). „Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”. U *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, str. 9459-9474. <https://arxiv.org/abs/2005.11401>
- [160] Bogdanović, B., Marković, V., Segedinac, M. i Konjović, Z. (2026a). „BAB software framework and micro-service technology as the basis for cooperative process-aware information systems on the Java platform”. *Vojnotehnički glasnik / Military Technical Courier*, 74(3), 613-641. DOI: 10.5937/vojtehg74-59807.
- [161] Cousot, P. i Cousot, R. (1977). „Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints”. U *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages (POPL '77)*, str. 238-252. ACM. DOI: 10.1145/512950.512973.

Prilozi

Prilog A: Modularni BAB PIM 2.0 (RDF/OWL)

Elektronski prilog sadrži merodavne datoteke u formatu Turtle iz direktorijuma `resources/ontologies` proverene revizije generatora. Njihova podela prikazana je u Tabeli A.1.

Datoteka	Uloga	Veza uvoza
BABCore-2.0.ttl	Aplikacija, entitet, akter, atribut i neutralna binarna relacija	Ne uvozi dodatne PIM module
BABView-2.0.ttl	Prikazi, forme, projekcije i kolone	Uvozi BAB Core 2.0
BABBehavior-2.0.ttl	Procesi, stanja, zadaci i prelazi	Uvozi BAB Core 2.0
BABValidation-2.0.ttl	Deklarativna ograničenja vrednosti i kardinalnosti	Uvozi BAB Core 2.0
BABSecurity-2.0.ttl	Uloge i dozvole pristupa	Uvozi BAB Core 2.0
BABService-2.0.ttl	Servisi i njihove operacije	Uvozi BAB Core 2.0
BABClassification-2.0.ttl	Specifikacije klasifikacije	Uvozi BAB Core 2.0
BABCapability-2.0.ttl	Ponovljive sposobnosti i njihovi parametri	Uvozi BAB Core 2.0

Tabela A.1. Moduli BAB PIM-a 2.0 i njihove veze uvoza.

Smer uvoza čuva stabilnost jezgra: ekstenzije zavise od Core-a, dok Core ne zavisi od ekstenzija. Fizička podela na datoteke ne menja matematičku konstrukciju. U Poglavlju 4 kategorija C_{Core} gradi se nad unijom šematskih grafova svih učitanih PIM modula i BAB Mapping 2.0 rečnika, a ne samo nad sadržajem datoteke `BABCore-2.0.ttl`.

Prilog B: BAB Mapping 2.0 i BABj 2.0 (RDF/OWL)

`BABMapping-2.0.ttl` definiše apstraktnu hijerarhiju `mapping:ImplementationMapping`, pojam `mapping:RuntimeModule` i svojstva operativnog profila. `BABj-2.0.ttl` uvozi taj rečnik i sadrži konkretne individue za skup tehnologija Java, Jakarta EE i Vaadin, uključujući Maven koordinate BAB 2 modula. Aplikacioni PIM može svojstvom `mapping:hasImplementationMapping` izabrati kompatibilnu realizacionu individuu; kada eksplicitni izbor ne postoji, razrešavač bira odgovarajuće podrazumevano preslikavanje. Realizacije sposobnosti svojstvom `mapping:requiresRuntimeModule` određuju neobavezne zavisnosti, dok je `bab-core` obavezan za svaki generisani projekat. Ovo je opis aktuelnog ugovora, čija su podrazumevana preslikavanja sažeta u Tabeli B.1.

Podrazumevana BABj individua	Ciljni tip	Uloga artefakta	Šablon
<code>DefaultEntityMapping_J</code>	<code>AbstractEntity</code>	entity	<code>java/entity.ftl</code>
<code>DefaultPersistentEntityMapping_J</code>	<code>AbstractEntity</code> sa JPA anotacijama	entity	<code>java/entity.ftl</code>
<code>DefaultAttributeMapping_J</code>	Tip razrešen iz XSD/BABj rečnika	attribute	<code>java/entity.ftl</code>
<code>DefaultEntityHomeMapping_J</code>	<code>AbstractHome</code>	home	<code>java/home.ftl</code>

Podrazumevana BABj individua	Ciljni tip	Uloga artefakta	Šablon
DefaultFormMapping_J	GenericWindow	form	java/window.ftl
DefaultViewMapping_J	GenericView	view	java/view.ftl
DefaultGeneralServiceMapping_J	JAX-RS aplikacioni ugovor	service	java/rest-resource.ftl
DefaultProcessMapping_J	Jakarta EJB procesna komponenta	process	java/process-service.ftl
DefaultUserTaskMapping_J i DefaultServiceTaskMapping_J	Runnable	task	java/process-service.ftl

Tabela B.1. Podrazumevana preslikavanja BABj profila i ciljni artefakti.

BABj dodatno povezuje sposobnosti sa interfejsima i članovima za uloge `entity`, `home`, `view` i `window`. Potpuno kvalifikovana imena zapisana su svojstvom `woc:hasFullyQualifiedName`. Koristi se IRI tog svojstva, ali cela WoC ontologija nije deo zatvarača uvoza transformacionog profila.

Prilog C: Validacija, PSM razrešavanje i sertifikat transformacije

Postupak provere sastoji se od četiri odvojena koraka. HermiT najpre proverava logičku konzistentnost zatvarača uvoza PIM-a. Zatim namenski validatori na jeziku Java nad modelom Jena proveravaju zatvorene ugovore generatora: prisustvo i jednoznačnost obaveznih elemenata, tipove, relacije, projekcije putanje, procese, servise, uloge, sposobnosti i identifikatore na jeziku Java. PSM razrešavač filtrira realizacione individue prema ciljnom skupu tehnologija i ulozi artefakta, daje prednost eksplicitnom izboru pomoću svojstva `mapping:hasImplementationMapping` i uređuje kandidate prema prioritetu i IRI-ju. Tek se pripremljena specifikacija prosleđuje FreeMarkeru.

Sertifikat transformacije proverava sledeće obaveze:

1. kompletnost preslikavanja izabranih šematskih relacija;
2. injektivnost njihovih ciljnih oznaka, osim deklariranih normalizacija;
3. determinizam ponovljenog generisanja;
4. kompatibilnost sa deklariranim normalizacionim jednačinama;
5. odsustvo neželjenog sažimanja identiteta artefakata.

Za istraživačku verziju generatora iz potpoglavlja 6.6.1 sertifikat je imao status `VALID`: obuhvatio je 34 izabrane relacije, pet tipiziranih normalizacija i 27 šablona, a dva pokretanja proizvela su isti skup od 38 sažetaka artefakata. Svako pokretanje uključuje i strukturnu proveru stvarnog Java izlaza. Normalizacija obuhvata kanonsku orijentaciju alternativnih RDF zapisa i tipsku saglasnost krajeva; nasleđeni naziv provere `Quotient compatibility` ne znači da su proverene proizvoljne jednačine paralelnih staza. Rezultat važi za identifikovani šematski deo i konkretan BABj profil. Ne potvrđuje ABox očuvanje instanci, semantičku ekvivalenciju izvršavanja, sve moguće jednačine

prirodnosti niti povratnu rekonstrukciju PIM-a iz koda. SPARQL upiti u elektronskom prilogu služe pregledu modela i ne predstavljaju implementaciju transformacionog toka.

Prilog D: FreeMarker šabloni

Provereni skup sadrži 27 FreeMarker šablona. Projektni šabloni proizvode pom.xml, persistence.xml, beans.xml, aplikacione metapodatke i README. Šabloni za jezik Java proizvode entitete, DTO projekcije, klase tipa home, prikaze, prozore, procesne i REST komponente, bezbednosni početni nalog, sesijske pomoćne klase i zajedničke elemente korisničkog interfejsa. Posebni isecci za sposobnosti dodaju polja i metode aktera.

Za pregled transformacije dovoljno je priložiti reprezentativni skup java/entity.ftl, java/dto.ftl, java/home.ftl, java/view.ftl, java/window.ftl, java/process-service.ftl, project/pom.ftl i project/persistence.ftl. Merodavan dokaz čini ceo identifikovani direktorijum iz revizije generatora, jer pojedinačni isečak ne pokazuje međusobnu usklađenost svih generisanih slojeva. Šabloni dobijaju uređene podatke iz ApplicationSpec i PSM faze; ne izvršavaju samostalno ontološku transformaciju.

Prilog E: Generisani kod: PostOffice primer

Referentno generisanje istraživačkom verzijom iz potpoglavlja 6.6.1 proizvelo je 31 upravljaju PostOffice datoteku, od kojih su 24 izvorne datoteke na jeziku Java. Za štampani prilog izdvajaju se klase Customer, Parcel i PostalClerk, njihove DTO klase i klase tipa home, CustomerView, ParcelView, EditCustomerWindow, EditParcelWindow i ParcelDeliveryProcess. Za ponavljanje provere potreban je ceo upravljani izlaz, zajedno sa manifestom .owlbab-manifest, aplikacionim metapodacima application-metadata.json, normalizovanim modelom pim-schema.json i strukturnim profilom resolved-psm.json.

Priloženi kod predstavlja aplikacionu osnovu generisanu iz identifikovanog PIM-a. Njegov POM uvozi bab-bom i uključuje obavezni modul bab-core. Izgradnja je kompajlirala sve 24 Java izvorne datoteke i proizvela postoffice.war. Generisana aplikacija raspoređena je i pokrenuta na serveru WildFly 41.0.1, gde su ručno provereni prijavljivanje, CRUD operacije, navigacija, filtriranje i modelovano ponašanje. Projekat nije sadržao automatizovane aplikacione testove, pa ova provera ne obuhvata sve moguće poslovne scenarije, performanse ili dugotrajan rad. Nezavisna strukturna provera prihvatila je izlaz prema ugovoru navedenom u Prilogu G. Ponovljivost sadržaja zasebno je proverena na fiksiranom sertifikacionom korpusu iz Priloga C.

Prilog F: Matrica preslikavanja osobina ABPMS-a

Matrica razdvaja postojanje programskog mehanizma od potvrđenog ishoda na nivou celog sistema, kao što je prikazano u Tabeli F.1.

ABPMS osobina	BAB mehanizam	Potvrđeni nivo	Šta nije potvrđeno
Objašnjivost	Trag PIM-PSM-artefakt, manifest, metapodaci i brojači tokom izvršavanja	Strukturni pregled i pregled tokom izvršavanja	Razumljivost objašnjenja krajnjem korisniku
Adaptivnost	Razdvojeni PIM i realizacioni profil, ponovna generacija i kontrole agenata	Realizacioni izbori i upravljanje komponentama	Autonomno prilagođavanje poslovnog procesa prema izmerenim ishodima
Sposobnost delovanja kroz razgovor	UserHelpDeskAgent, klasifikacija namere i poruke	Postojanje pregledane komponente	Sistematska uspešnost razgovornog izvršavanja poslovnih zadataka
Autonomija u zadatim okvirima	Postupak predlaganja ispravke i kriterijum zabranjenih potpisa	Prototip predlaganja i ograničene provere	Stvarno automatsko izvršenje bezbedne ispravke
Samousavršavanje	Memorija razgovora i brojači tokom izvršavanja	Infrastrukturni preduslovi	Dokazano poboljšanje procesa na osnovu ranijih ishoda

Tabela F.1. Matrica preslikavanja osobina ABPMS-a na mehanizme BAB-a.

Matrica je sažetak nalaza iz potpoglavlja 6.5. Ona ne menja granice formalnog rezultata iz Poglavlja 4: sertifikat transformacije ne proširuje se na ponašanje jezičkog modela, bezbednost predložene radnje ili korisničku uspešnost ABPMS funkcija.

Prilog G: Tehnička evidencija i reproduktivnost provera

Evidencija razlikuje proveru istraživačke verzije generatora od ranijih provera modularne biblioteke i aplikacija. Datumi i revizije identifikuju uslove eksperimenta. Rezultati iz različitih skupova ne predstavljaju zajedničko izvršavanje testova. Identifikovane verzije i vremena provere navedeni su u Tabeli G.1.

Predmet	Identifikovana verzija ili skup	Vreme provere
Istraživačka verzija generatora	Radna nadogradnja nad osnovom bc95546a, određena manifestom izmena	10. septembar 2026.
Osnovna verzija generatora	bc95546a, sa prethodnom modularnom migracijom c7c5b2b	10. septembar 2026.
Generator posle modularne migracije	c7c5b2b	7. septembar 2026.
Biblioteka BAB 2	150c6ef5	Ranije zasebno zabeležena provera
SoilLab	cef60244	7. septembar 2026.
DentaScape 2	ce36110e	Ranije zasebno zabeležena provera

Tabela G.1. Identifikovane verzije i skupovi tehničkih provera.

Osnovna verzija generatora imala je 29 uspešnih testova u osam testnih klasa. Istraživačka nadogradnja ima 60 uspešnih testova u 12 klasa i uvodi nezavisnu proveru stvarnog Java izlaza. Zadržava osnovni bifunktorski model i BABj profil. Rečnici BAB Behavior, BAB Security, BAB Mapping i BABj u toj nadogradnji imaju vrednost owl:versionInfo jednaku 2.0.1, uz imenske prostore porodice 2.0. Ciljna konfiguracija koristi Java 26, Jakarta EE Web 11, Vaadin 25.2.6 i biblioteku BAB 2.0.0.

Izvršavanje i zapis provere. Regresioni testovi generatora pokreću se postupkom mvn clean test. Validacija ulaza, sertifikacija ugrađene transformacione specifikacije, generisanje, provera postojećeg izlaza i izgradnja generisanog projekta vode se kao odvojeni koraci. Sertifikat se izdaje komandom certificate, a postojeći izlaz ispituje komandom verify-output. Oznaka podržanog strukturnog ugovora je babj-entity-structure/1; proverivač razlikuje statusse PASS, FAIL i UNSUPPORTED. Poslednji status označava da konstrukcija nije obuhvaćena podržanom proverom.

JSON zapis provere izlaza čuva normalizovani PIM, razrešeni strukturni PSM, opažene Java deklaracije, izvorni i ciljni graf i njihove korespondencije. Deklaracije se čitaju iz aplikacionih izvora, odvojeno od testnih klasa. SHA-256 sažeci identifikuju ulazne datoteke i njihove lokalne uvoze, rečnike, izlazne datoteke i kompajliranu transformacionu implementaciju. Sažeci služe proveru identiteta i sadržaja artefakata; sami po sebi ne dokazuju strukturno očuvanje ili ispravnost programa. Trajanje provere beleži se odvojeno i ne ulazi u identifikator sertifikata.

Prateći zapisi. Putanje u nastavku relativne su u odnosu na istraživački radni prostor. Manifest izmena identifikuje radnu nadogradnju pored osnovne Git revizije; samo ime osnovne revizije nije dovoljno za njeno ponavljanje.

- verification/comsis-upgrade-20260910/implementation-changes.json i pripadajuća razlika izvora određuju sastav istraživačke nadogradnje.
- verification/comsis-upgrade-20260910/installed-tests.log sadrži rezultate njenih regresionih testova.
- verification/comsis-upgrade-20260910/installed/ sadrži validacije ulaza, sertifikat specifikacije, sertifikate pojedinačnih izlaza, generisane projekte i zapise izgradnje ispravnog i namerno izmenjenog Catalog primera.
- verification/implementation-prose-20260911/technical-evidence.md čuva potpune identifikatore i ranije operativne zapise izdvojene iz glavnog teksta, uključujući uslove provera BAB biblioteke i dodatnih aplikacija.

Raniji sertifikacioni korpus imao je 36 izlaznih artefakata. Istraživačka verzija čuva i normalizovani PIM i razrešeni strukturni PSM, pa odgovarajući korpus ima 38 artefakata. Istom dopunom broj PostOffice datoteka povećan je sa 29 na 31, a Catalog datoteka sa 36 na 38; broj Java izvora ostao je 24, odnosno 31. Ova razlika predstavlja proširenje evidencije transformacije, a ne povećanje aplikacione funkcionalnosti.

Bibliotečki podaci o doktorskoj disertaciji

Podatak	Vrednost
Autor	Borivoj Bogdanović
Vrsta dokumenta	Doktorska disertacija
Naslov rada	Metodologija za ubrzani razvoj procesno-svesnih informacionih sistema bazirana na ontološkom modelovanju i teoriji kategorija
Naslov rada na engleskom jeziku	<i>Methodology for Accelerated Development of Process-Aware Information Systems Based on Ontological Modeling and Category Theory</i>
Ustanova	Univerzitet Singidunum, Beograd
Studijski program	Intelligentno softversko inženjerstvo
Naučna oblast	Tehničko-tehnološke nauke
Uža naučna oblast	Elektrotehničko i računarsko inženjerstvo
Mentor	prof. dr Vidan Marković
Članovi komisije	prof. dr Vidan Marković; prof. dr Mladan Jovanović; prof. dr Milan Segedinac, redovni profesor Fakulteta tehničkih nauka Univerziteta u Novom Sadu
Jezik rada	srpski
Jezik sažetka	srpski i engleski
Pismo	latinica
Zemlja objavljivanja	Srbija
Mesto objavljivanja	Beograd
Godina	2026.
Ključne reči	procesno svesni informacioni sistemi; ontološko modelovanje; teorija kategorija; modelom vođeni razvoj; generisanje koda; BAB
Ključne reči na engleskom jeziku	process-aware information systems; ontological modeling; category theory; model-driven development; code generation; BAB
UDK	
Datum prihvatanja teme	26.03.2026.
Datum odbrane	
Broj stranica	x, 165 str.
Broj slika	8
Broj tabela	16
Broj listinga	19
Broj bibliografskih jedinica	161

Sažetak

U disertaciji se predlaže metodologija za ubrzani razvoj procesno svesnih informacionih sistema zasnovana na ontološkom modelovanju, modelom vođenom razvoju i teoriji kategorija. Modularni PIM opisuje domenske, procesne, korisničke i druge izabrane aspekte sistema, dok odvojeni sloj preslikavanja povezuje platformski nezavisne pojmove sa profilima ciljne platforme. Transformacioni lanac obuhvata izdvajanje specifikacije iz ontologije, njenu proveru, izbor preslikavanja, izgradnju unutrašnjeg modela i generisanje programskih artefakata. Kategorijalni aparat koristi se za precizno određivanje objekata, morfizama, funktora i uslova pod kojima se može govoriti o očuvanju šematske strukture. Metodologija je realizovana u okviru BAB i razmotrena na više aplikativnih primera. Evaluacija razdvaja formalne tvrdnje, inženjersku verifikaciju generatora i analizu konkretnih primena, uz izričito navođenje granica izvedenih zaključaka.

Library information about the doctoral dissertation

Data	Value
Author	Borivoj Bogdanović
Document type	Doctoral dissertation
Title	<i>Methodology for Accelerated Development of Process-Aware Information Systems Based on Ontological Modeling and Category Theory</i>
Title in Serbian	Metodologija za ubrzani razvoj procesno-svesnih informacionih sistema bazirana na ontološkom modelovanju i teoriji kategorija
Institution	Singidunum University, Belgrade
Study programme	Intelligent Software Engineering
Scientific field	Technical and Technological Sciences
Narrow scientific field	Electrical and Computer Engineering
Supervisor	Prof. Dr. Vidan Marković
Committee members	Prof. Dr. Vidan Marković; Prof. Dr. Mladen Jovanović; Prof. Dr. Milan Segedinac, Full Professor at the Faculty of Technical Sciences, University of Novi Sad
Language	Serbian
Abstract languages	Serbian and English
Script	Latin
Country of publication	Serbia
Place of publication	Belgrade
Year	2026
Keywords	process-aware information systems; ontological modeling; category theory; model-driven development; code generation; BAB
Keywords in Serbian	procesno svesni informacioni sistemi; ontološko modelovanje; teorija kategorija; modelom vođeni razvoj; generisanje koda; BAB
UDC	
Date of topic acceptance	26 March 2026
Date of defence	
Number of pages	x, 165 pages
Number of figures	8
Number of tables	16
Number of listings	19
Number of references	161

Abstract

The dissertation proposes a methodology for the accelerated development of process-aware information systems based on ontological modeling, model-driven development, and category theory. A modular platform-independent model describes selected domain, process, user-interface, and related aspects of a system, while a separate mapping layer connects platform-independent concepts with target-platform profiles. The transformation chain includes extracting a specification from an ontology, validating it, selecting mappings, constructing an internal model, and generating software artifacts. Category-theoretic concepts are used to define objects, morphisms, functors, and the conditions under which preservation of schema-level structure may be claimed. The methodology is implemented within the BAB framework and examined through several application examples. The evaluation distinguishes formal claims, engineering verification of the generator, and the analysis of concrete applications, while explicitly stating the limits of the conclusions.