UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTO IN INFORMATIKO

Irina Juhnov

Vključitev načrtovanja varnosti v načrtovanje informacijskih sistemov

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01633/2010
Datum: 15.02.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **IRINA JUHNOV**

Naslov: **VKLJUČITEV NAČRTOVANJA VARNOSTI V NAČRTOVANJE INFORMACIJSKIH SISTEMOV**

ENHACING SECURITY IN THE DESIGN OF INFORMATION SYSTEMS

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

V diplomski nalogi preučite področje načrtovanja varnosti pri načrtovanju informacijskih sistemov. Raziščite, katere od razširjenih tehnik in orodij za načrtovanje informacijskih sistemov imajo možnost primernih razširitev za načrtovanje varnostnih vidikov. Pri tem se osredotočite predvsem na UML. V praktičnem delu naloge preučite področje upravljanja z digitalnimi identitetami na primeru univerzitetnega okolja ter z eno od prej opisanih tehnik modelirajte življenjski cikel identitete študenta in pedagoga. Razložite varnostni vidik svojega dela ter kritično predstavite prednosti in slabosti.

Mentor:

M. Ciglarič

doc. dr. Mojca Ciglarič

Dekan:

prof. dr. Franc Solina

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTO IN INFORMATIKO

Irina Juhnov

Vključitev načrtovanja varnosti v načrtovanje informacijskih sistemov

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentorica: doc. dr. Mojca Ciglarič

Ljubljana, 2010

Namesto te strani vstavite original izdane teme dipomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne s študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisana Irina Juhnov,

z vpisno številko 63040062,

sem avtorica diplomskega dela z naslovom:

Vključitev načrtovanja varnosti v načrtovanje informacijskih sistemov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelala samostojno pod mentorstvom

  doc. dr. Mojce Ciglarič

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela;

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 1.3.2010 Podpis avtorice:

Zahvala

Najprej bi se zahvalila svoji mentorici, doc. dr. Mojci Ciglarič, za vse napotke, predloge in nasvete, s katerimi ste mi pomagali med pisanjem te diplomske naloge. Najlepša hvala za vašo pomoč med celotnim pisanjem diplome in predvsem na koncu, ko mi je bila zaradi časovne stiske najbolj potrebna.

Posebno zahvalo namenjam moji družini: očetu Petru, mami Veri, sestri Tamari in bratu Simonu, ki so mi bili v veliko podporo ne samo v času pisanja diplomske naloge, ampak tudi v času celotnega študija na Fakulteti za računalništvo in informatiko. Hvala vam za vse vzpodbudne besede v trenutnik, ko sem jih najbolj potrebovala, in za vso potrpežljivost, ki ste jo pokazali, predvsem v izpitnih obdobjih.

Nenazadnje gre zahvala tudi vsem sošolcem in prijateljem, ki so mi med študijem pomagali premagovati ovire na sami fakulteti in tudi zunaj nje. Hvala, da ste se z mano veselili vseh mojih uspehov in mi priskočili na pomoč, ko sem vas potrebovala.

Kazalo

Seznam uporabljenih kratic in simbolov

UML	Unified Modeling Language
MAC	Mandatory Access Control
DAC	Discretionary Access Control
RBAC	Role-Based Access Control
HAC	Hybrid Access Control
OCL	Object Constraint Language
MDA	Model Driven Architecture
XML	eXtensible Markup Language
LAN	Local Area Network
SSD	Static Separation of Duty
DSD	Dynamic Separation of Duty
EJB	Enterprise JavaBeans
SSI	Secure Subsystem Infrastructure
URA	User-Role Assignment
SS	Simple Security
SI	Simple Integrity
NP	Negative Permission
VPIS	Visokošolska prijavno-informacijska služba
KIU	Kadrovska informatika Univerze
PIU	Poslovna informatika Univerze

Povzetek

V diplomski nalogi je obravnavano področje načrtovanja računalniške varnosti v povezavi z upravljanjem z digitalnimi identitetami. Najprej so predstavljeni osnovni pojmi računalniške varnosti s poudarkom na kontroli dostopa. Nato je na kratko opisano področje upravljanja z identitetami in zgradba samih sistemov za upravljanje z digitalnimi identitetami.

V osrednjem delu sledi pregled tehnik, metod in priporočil za izgradnjo varnih informacijskih sistemov na podlagi večih strokovnih člankov. Pri tem je poudarek predvsem na samem načrtovanju varnosti, ki temelji na diagramski tehniki UML in njenih razširitvah. Za lažje razumevanje izbranih in predstavljenih metod je eno podpoglavje namenjeno predstaviti načrtovalskega jezika UML. Izmed predlaganih metod sem nato izbrala najboljše pristope in jih uporabila na konkretnem primeru.

Kot ogrodje za predstavitev konkretnega predloga izboljšave varnosti sem uporabila okolje študijske informatike Univerze v Ljubljani z vsemi pripadajočimi informacijskimi sistemi. Na osnovi obstoječega sistema je predstavljen moj predlog vpeljave dodatnih varnostnih mehanizmov, s poudarkom na upravljanju z digitalnimi identitetami, v sklopu katerega je predstavljen tudi življenjski cikel identitete študenta in pedagoga.

Na koncu sledi še kritična primerjava vseh predlaganih razširitev ter pregled prednosti in slabosti posameznih metod.

Ključne besede:

varnost informacijskih sistemov, načrtovanje, kontrola dostopa, RBAC, upravljanje z identitetami, UML, življenjski cikel identitete.

Abstract

This work covers security aspects in the design of information systems in connection with identity management. In the beginning we define terms of computer security based on access control. Later on we present the definition of identity management and describe the structure of indentity management systems.

The core of this work is an owerview of the proposed approach, methods and recommendations for building secure information systems based on articles of different authors. Common poit of all approaches is that they all use the designing standard UML which is also presented in one of the chapters. From among all proposals on security aspects improvement we take the best ones and use them on a concrete example.

As the framework for presenting best solutions we take the exsisting information system of University in Ljubljana with all the belonging applications. We demonstrate the best solutions for improving information system security with emphasis on management of digital identity and lifecycle of digital identity for a student and a teacher.

At the end we critically compare all proposed expansions and stress their main advantages and disadvantages.

Key words:

Information systems security, design of information systems, access control, RBAC, identity management, UML, identity lifecycle

1. Uvod

Včasih, ko še ni bilo interenta in smo računalnike ljudje uporabljali le za lastne potrebe, hobije in predvsem kot orodje, ki je nadomestilo pisanje na roko in pisalni stroj, se nam ni bilo treba preveč obremenjevati s tem, ali je naš sistem varen in ali je mogoče, da bi nam kdo vdrl v sistem ter zlorabil naše podatke na njem. Dovolj je bilo, da smo imeli osebni računalnik zavarovan z uporabniškim imenom in geslom ter s tem računalniku ob vpisu dokazali, da imamo pravico do uporabe le-tega.

Nato so se začela pojavljati lokalna omrežja, v katerih so več računalnikov povezali med seboj in omogočili lažji in hitrejši prenos podatkov med njimi. S tem pa smo naredili velik korak k povečanju ranljivosti naših računalnikov in tudi podatkov, ki jih hranimo. Naenkrat je postalo mogoče do računalnika dostopati brez neposrednega fizičnega kontakta. Ne samo da so dostopi neposredni, lahko so tudi skriti in se jih uporabnik sploh ne zaveda. Tako so se začeli pojavljati napadi in vdori v računalnike. Razlogi za njih so različni: kraja podatkov, zlonamernost, kraja identitete ali pa samo zabava in preizkušanje zmožnosti ter lastnih sposobnosti. Vse skupaj se je še stopnjevalo s pojavom interneta, ki med seboj povezuje na milijone računalnikov. Naenkrat so vdori del našega vsakdana in nemogoče se jim je izogniti. Na nas je, da poskušamo svoje računalnike in informacijske sisteme čim bolj zaščititi in s tem zmanjšati možnosti vdora ter okužb sistema.

Vendar pa izdelava varnih sistemov ni tako enostavna, kot se sprva sliši. Veliko problemov pri razvoju informacijskih sistemov nastaja zaradi preslabega predznanja razvijalcev na področju računalniške varnosti in tudi zato, ker je varnostne potrebe in zahteve težko analizirati ter načrtovati. To vodi v razvoj ranljivih in nestabilnih sistemov, ki pa si jih dandanes nihče ne more privoščiti. Če pogledamo na računalnik še z druge strani, kot na enega izmed členov ogromnega omrežja, kjer nam grozi »napad« iz tisoč in ene strani, bi morali na prvo mesto dati varnost pri razvoju novih informacijskih sistemov. Ne morejo namreč biti varnostni mehanizmi, kot je kontrola dostopa, zaščita podatkov, kriptirana povezava itd. kar na slepo vneseni v že postavljen system, ampak se morajo varnostni elementi razvijati od začetka, hkrati z vsemi ostalimi komponentami novega informacijskega sistema.

Glavni namen te diplomske naloge je pregled metod in priporočil za sistematično analiziranje varnostnih zahtev ter načrtovanje varnostnih lastnosti informacijskih sistemov. Najboljše in najbolj primerne predloge želimo podrobneje predstaviti, primerjati med seboj, ovrednotiti in prikazati uporabo na realnem primeru. V okviru tega smo si zadali naslednje cilje:

- povzeti osnovne pojme, povezane z varnostjo informacijskih sistemov;
- proučiti področje upravljanja z digitlanimi identitetami in ugotoviti povezanost tega področja s področjem računalniške varnosti;
- poiskati najprimernejše orodje in diagramske tehnike za analizo in načrtovanje varnostnih mehanizmov v informacijskih sistemih;

- poiskati, pregledati, ovrednotiti in primerjati strokovne članke, ki predlagajo izboljšave glede vpeljave varnosti z izbranimi diagramskimi tehnikami;
- najboljše ideje in predloge uporabiti in predstaviti na konkretnem primeru študijske informatike.

Kadar začnemo razmišljati o izdelavi novega, varnega infromacijskega sistema, je prvi korak zagotovo analiza morebitnega že obstoječega sistema in analiza okolja, v katerem bo sistem deloval. Nato sledi načrtovanje novega sistema oz. načrt prenovljenega sistema. Da pa bi ta dva koraka lahko karseda optimalno izvedli, je nujno potrebno dobro načrtovalsko orodje, s katerim lahko iz več zornih kotov prikažemo pogled na trenutno situacijo ter izdelamo načrt za nov sistem. Svetovno najbolj razširjen in najpogosteje uporabljan grafični načrtovalski jezik za vizualizacijo in specifikacijo komponent informacijskega sistema in aplikacij je UML (Unified Modeling Language). UML predstavlja zbirko najboljših praks, ki so se izkazale kot uspešne pri načrtovanju velikih in kompleksnih sistemov.

In ravno zato, ker moramo o varnosti začeti razmišljati že v prvih korakih razvoja novega sistema, v pomoč pri tem pa nam je ravno UML, smo se odločili raziskati, kakšna je povezava med tema dvema pojmoma. Se pravi, katere varnostne mehanizme UML že vsebuje in kako bi bilo mogoče UML nadgraditi, da bi še povečali stopnjo varnosti načrtovanega sistema.

Da pa ne bi o obravnavanju varnosti v fazi razvoja informacijskih sistemov govorili samo teoretično, smo naredili predlog vpeljave dodatnih varnostnih mehanizmov na konkretnem primeru. Izbrali smo študijski informacijski sistem e-Študent, ki ga uporabljajo študenti in zaposleni na Fakulteti za računalništvo in informatiko v Ljubljani. Poleg tega, da sistem poznamo z uporabniškega vidika, smo se z njim podrobneje seznanili tudi v okviru predmeta Teorija informacijskih sistemov v 4. letniku študija. Pri predmetu so nam predstavili zgradbo in delovanje obstoječega sistema, nato pa smo študentje informatike in programske smeri podoben sistem izdelali še sami.

V poglavju Osnovni pojmi, ki sledi, so opisane glavne značilnosti in pojmi, povezani z varnostjo informacijskih sistemov ter podrobnejši opis samega pojma in tudi sistema za upravljanje z identitetami. V tretjem poglavju je na kratko opisan koncept načrtovalskega jezika UML, nato pa sledijo predlagane varnostne razširitve različnih avtorjev, povezane z načrtovalskim jezikom UML. Za lažje razumevanje je med samimi opisi razširitve podanih tudi veliko konkretnih primerov, povezanih s prej omenjenim sistemom e-Študent. Praktični primer razširitve je opisan v četrtem poglavju, in sicer predlog integracije varnostnih razširitev v okolje študijske informatike, se pravi v sistem e-Študent in ostale infromacijske sisteme univerze. Nato sledijo še zadnje poglavje, zaključek in sklepne ugotovitve.

2. Osnovni pojmi

2.1. Varnost

Sčasoma se je pojem varnosti na področju računalništva in informacijskih sistemov spreminjal in še vedno spreminja svoj pomen. Varnost je bila sprva le zaščita pred fizičnimi napadi in napakami na posameznem računalniku. Nadaljevalo se je s potrebo po varovanju in zaščiti podatkov. S pojavom omrežij, ki povezujejo računalnike, se moramo spopadati z novimi nevarnostmi, kot so napadi prek omrežij. Zaradi le-teh je potrebno vpeljati nove varnostne mehanizme, ki ne le da preprečujejo vdore, ampak zagotavljajo pravilno in predvsem neprekinjeno delovanje sistemov.

Za začetek omenim tri pojme, ki se zelo pogosto pojavljajo v povezavi z varnostjo in katerih pomen je včasih narobe interpretiran. Prvi je ***identifikacija***. Vprašanje, ki ga postavi sistem uporabniku pri identifikaciji je: »Kdo, praviš, da si?«. Identifikacija je torej povezovanje neznanega uporabnika z znanim identifikatorjem v sistemu. Uporabnik se predstavi s svojim edinstvenim identifikatorjem – uporabniškim imenom, kar nato vodi do ***avtentikacije.*** Sedaj sistem že ve, za katerega uporabnika gre, ker pa ni prepričan, ali se je ta predstavil s pravo identiteto, ga vpraša: »Kako naj vem, da si to zares ti?« Najpogostejši način in tudi najbolj enostaven je z vnosom gesla, ki se ujema s prej vnesenim uporabniškim imenom. Pogosto se ta dva koraka izvedeta naenkrat, se pravi, da uporabnik hkrati vpiše uporabniško ime in geslo. Poznamo pa tudi mehanizme, ko je oboje združeno v eni funkciji, to je pri biometričnih varnostnih mehanizmih, kot sta identifikacija in avtentikacija s prstnim odtisom ali skeniranjem očesne mrežnice ali šarenice. Tretji korak oz. pojem je ***avtorizacija.*** Pri tem koraku sistem ugotavlja naslednje: »Zdaj, ko si tu - v sistemu, kakšne so tvoje pravice?«. Se pravi, še en mehanizem, ki ga potrebuje varen sistem, je, da vsakemu uporabniku dovoljuje izvajanje le tistih funkcij, za katere ima dovoljenje. Uporabnik je avtoriziran za uporabo točno določenih operacij v sistemu.

Računalniška varnost in varnost računalniških omrežij slonita na treh osnovnih stebrih.

- **Zaupnost**: pomen pojma zaupnosti podatkov je, da sistem onemogoči dostop do podatkov uporabnikom, ki nimajo pravice dostopati do njih. S tem podatke zaščitimo pred neavtoriziranim dostopom.
- **Popolnost:** sistem zahteva ohranitev pravilnosti podatkov. Sistem ne sme pokvariti podatkov ali dovoliti, da bi kdorkoli neavtorizirano, zlonamerno ali po nesreči podatke spremenil. Še posebej velik pomen dajejo popolnosti podatkov v finančnih sistemih, kjer je to najpomembnejši element varnosti.
- **Razpoložljivost**: osnovni pomen razpoložljivosti je, da strojna in programska oprema računalniškega sistema delujeta učinkovito. Sistem mora imeti podatke vedno na razpolago uporabniku, v primeru nesreče pa jih mora znati hitro in v celoti obnoviti.

Mednarodni konzorcij za certificiranje varnosti v informacijskih sistemih (The International Information Systems Security Certification Consortium) omenja naslednjih 10 področij v zvezi z računalniško varnostjo in varnostjo informacijskih sistemov:

- Informacijska varnost
- Metodologija kontrole dostopa
- Kriptografija
- Fizična varnost
- Poslovna varnostna infrastruktura
- Varnost aplikacij
- Telekomunikacijska, omrežna in internetna varnost
- Zakonodaja, preiskovanje in etika
- Načrtovanje poslovne kontinuitete
- Varnost operacij

Torej, računalniška varnost pokriva veliko področij, kot so: zaklepanje sob, v katerih so strežniki in telekomunikacijska oprema, zaklepanje računalnikov, zavarovanje uporabniških računov z močnimi gesli, uporaba zaščite za datoteke, kot je tudi izdelava varnostne kopije podatkov, kriptiranje omrežnih povezav in še mnoge druge varnostne preventive. Kljub temu pa imajo ljudje pogosto ob uporabi pojma računalniške varnosti v mislih le varovanje podatkov v računalniškem sistemu.

To pa zato, ker ravno varovanje podatkov predstavalja vedno večji problem na področju računalniške varnosti. Eden od razlogov je tudi ta, da se velikokrat začne o varovanju podatkov in informacij razmišljati prepozno. To je takrat, ko je sistem že postavljen in se začnejo pojavljati prve napake in vdori v sistem. Najenostavnejša rešitev tega problema bi bila, da o zaščiti in varnosti podatkov začnemo razmišljati takoj na začetku razvoja novega sistema, saj lahko le tako zgradimo varen in stabilen sistem, ki se bo uspešno postavil po robu nevarnostim, ki nam grozijo v digitalnem svetu.

V osnovi nam varen računalniški sistem zagotavlja, da naš računalnik počne tisto, za kar je namenjen, tudi če njegovi uporabniki ne počnejo tega, kar naj bi počeli in čemu je računalnik namenjen. Prav tako varen sistem varuje shranjene podatke pred izgubo, zlonamernimi ali nenamernimi spremembami ter pred neavtoriziranim dostopom.

Obstajajo štirje osnovni načini, kako zaščititi naš sistem oz. podatke, ki jih le-tahrani:

1. *Kontrola dostopa do sistema:* metode kontrole dostopa do sistema zagotavljajo, da neavtorizirani uporabniki ne morajo vstopiti v sistem. Prav tako skrbijo za to, da se avtorizirani uporabniki zavedajo pomena varnosti in redno spreminjajo svoja uporabniška gesla. Kontrola dostopa je odgovorna tudi za nadzor dogajanja v sistemu, predvsem nadzor operacij, povezanih s prijavo v sam sistem in pravicami uporabnikov.
2. *Kontrola dostopa do podatkov:* te metode določajo pravice za dostop do podatkov posameznim uporabnikom. Poleg tega tudi nadzorujejo, kdo dostopa do katerih podatkov in s kakšnim namenom. Poznamo več vrst kontrole dostopa, kot so DAC (Discretionary Access Control), MAC (Mandatory Access Control) in RBAC (Role-based Access Control), ki pa jih bomo podrobneje opisali v nadaljevanju.
3. *Administracija sistema:* določa naloge, ki jih opravlja administrator sistema. Te so: poučevanje uporabnikov o varnem načinu dela s sistemom in nato tudi sam nadzor

uporabnikov sistema pa tudi bolj globalno upravljanje z varnostjo, kot je ugotavljanje možnih nevarnosti v sistemu in kako se zaščititi pred njimi.

4. *Načrtovanje varnega sistema:* pri tem je mišljeno predvsem čim boljše izkoriščanje vseh že obstoječih varnostnih mehanizmov strojne in programske opreme. Na primer uporaba zmožnosti sistema za razdelitev spomina računalnika in s tem možnost ustvarjanja varnostnih kopij podatkov.

Kontrola dostopa

Kontrolo dostopa sestavlja množica pravil, pojmov in smernic za definiranje omejitev, kdo lahko dostopa do katerih podatkov sistema, kdaj in na kak način. Trije znani modeli za modeliranje kontrole dostopa so:

- obvezna kontrola dostopa (*mandatory access control - MAC),*
- neomejena kontrola dostopa *(discretionary access control - DAC) in*
- kontrola dostopa z vlogami *(role-based access control - RBAC).*

Vsi trije pristopi se uporabljajo za zaščito informacijskih objektov pred neavtoriziranim dostopom. Tako na primer študent ob prijavi v sistem e-Študent ne sme dobiti pravice za spreminjanje svojih ocen v elektronskem indeksu. Če želimo, da bo naš sistem preprečeval takšne neavtorizirane vdore, je potrebno vpeljati v sistem določena pravila in omejitve. In ravno pri tem so nam v veliko pomoč ogrodja RBAC, MAC in DAC.

MAC se uporablja v aplikacijah, kjer je varovanje informacij najpomembnejši element oz. bi izguba informacije ogrozila nacionalno varnost ali imela za posledice ogromne finančne izgube. V modelu MAC je vsak objekt označen s stopnjo tajnosti (javno, zaupno, tajno, strogo tajno), kar predstavlja pomembnost oz. občutljivost vsebovanih informacij. Vsak subjekt, ki dostopa do objekta, ima določeno stopnjo pravice dostopa do tajnih podatkov. Le- ta mora biti vedno večja ali enaka stopnji tajnosti objekta, do katerega subjekt dostopa.

DAC se uporablja za dinamične aplikacije. Pri DAC-u so dovoljenja določena direktno med subjekti in objekti. Subjekt lahko pridobi pravico za posredovanje svojih dovoljenj drugim subjektom.

RBAC grupira dovoljenja v neodvisne enote, imenovane vloge. Namesto posameznih dovoljenj je tako uporabniku ob prijavi v sistem dovoljena vloga. Z vlogo nato uporabnik avtomatsko pridobi vsa pripadajoča dovoljenja. Model RBAC je primeren za velike informacijskih sisteme z veliko uporabniki, ki jih lahko glede na njihove pravice v sistemu grupiramo v skupine. Tak model je zato najbolj primeren za informacijski sistem kot je e-Študent.

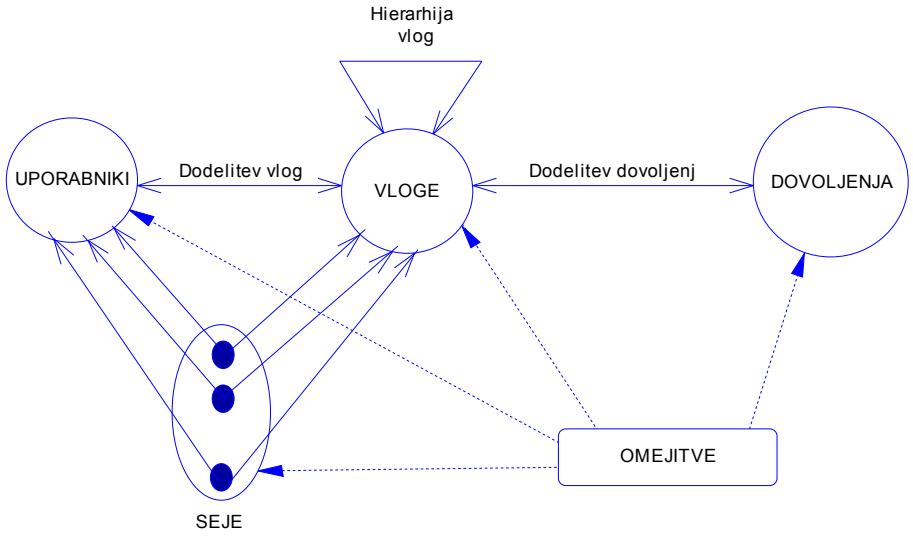
RBAC je sestavljen iz treh glavnih komponent: osnovnega modela, hierarhije vlog in omejitev.

Osnovni model vključuje bistvene poglede oz. lastnosti RBAC. Te so, da ima vsak uporabnik določeno eno ali več vlog v sistemu. Vsaka vloga je povezana z dovoljenji za izvajanje določene operacije na nekem objektu. Uporabnik pridobi potrebna dovoljenja z dodelitvijo

primerne vloge. Osnovni model vključuje še uporabniško sejo. Uporabnik ob prijavi v sistem aktivira sejo in takrat mu je dodeljena vloga, ki mu pripada. Vsak uporabnik lahko aktivira več sej, vsaka seja pa je lahko povezana le z enim uporabnikom.

Hierarhija vlog doda omejitve k osnovnemu modelu RBAC za podporo hierarhije vlog. Ta je pomembna in primerna predvsem za strukturiranje vlog neke organizacije. Hierarhija vlog predstavlja dedovalna razmerja med vlogami. Imamo torej starševske vloge in njim podrejene vloge (otroke). Podrejenim vlogam pripadajo poleg lastnih dovoljenj tudi vsa dovoljenja njihovih starševskih vlog. In tako je vsakemu uporabniku, ki mu je dodeljena neka vloga, hkrati dodeljena tudi vsaka tej vlogi nadrejena oz. starševska vloga, če seveda ta obstaja. Kot konkreten primer vzemimo vlogi študenta in adminstratorja v sistemu e-Študent. Starševska vloga je študent, dedovana vloga pa vloga administratorja. Administrator ima namreč poleg specifičnih dovoljenj glede administritanja sistema tudi vsa dovoljenja za uporabo sistema e-Študent, ki jih ima študent, in sicer pregled elektronskega indeksa, naročanje potrdil, prijava na izpit itd. Ne velja pa v obratni smeri. Študent torej nima vseh pravic in dovoljenj, kot jih ima administrator sistema. Zaradi prenosa dovoljenj pri hierarhiji vlog je pomembno, da se pred dodelitvijo katerekoli vloge uporabniku preverijo ne samo direktna dovoljenja in pravice, ki jih bo uporabnik s to vlogo pridobil, ampak tudi posredna dovoljenja, pridobljene zaradi hierarhije vlog.

Omejitve so eden od bistvenih delov modela RBAC. Nanašajo se lahko na vse elemente modela RBAC: na uporabnika, dovoljenja, sejo, vloge in tudi na samo hierarhijo vlog. Omejitve natančneje definirajo pravila v sistemu in situacije, do katerih v sistemu ne sme priti. Vrste omejitev bomo natančneje definirali v nadaljevanju, ob razlagi posameznih pristopov, ki so jih uporabili razni avtorji pri predstavitvi svoje ideje o vpeljavi modela RBAC v načrt razvijajočega informacijskega sistema. Omejitve lahko opišemo z uporabo jezika *Object Constraint Language (OCL)* v razrednih diagramih ali pa vizualno, z uporabo objektnih diagramov. Vizualni način je lažji za razumevanje, pristop z OCL pa je bolj formalen in natančen.



Slika 1: Celotni model RBAC.

2.2. Upravljanje z identitetami

Upravljanje z identitetami je nekaj, kar počnemo ljudje v vsakodnevnem življenju, s tem ko se odločamo, katere podatke o sebi bomo delili z drugimi in katerih ne. Vse je odvisno od situacije, v kateri se posameznik znajde, ter od vloge, ki jo v danem trenutku opravlja. Tako tudi samo okolje vpliva na to, na kakšen način se predstavimo drugim in katere podatke o sebi jim zaupamo. Ni vseeno namreč, ali smo doma, v pisarni, s sorodniki, na rekreaciji, v trgovini ali kje drugje. Zgodi se lahko, da človek uporablja drugačna imena oz. vzdevke glede na situacijo, v kateri se znajde. Ta pogost model obnašanja v realnem življenju se zrcali tudi v digitalnem svetu. Pri upravljanju z identitetami se lahko uporabnik sam odloči, komu bo zaupal svoje osebne podatke, vloge, ki jih opravlja, kdaj bo uporabil psevdonim in kdaj svoje pravo ime.

Na enostaven način povedano je identiteta množica informacij, ki jih vemo o določeni osebi oz. so informacije, ki to osebo določajo. V realnem svetu pod pojmom identiteta ponavadi razumemo ime človeka, njegov naslov, številko potnega lista in podobno. V poslovnem svetu identiteta dobi še dodatne attribute, kot so vloge, pravice, odgovornosti osebe. Za državo so kot identiteta človeka ponavadi uporabljeni podatki o človekovem imenu, njegov EMŠO in davčna številka ter drugi. Digitalna identiteta je podoben pojem kot identiteta, le s to razliko, da se uporablja v digitalnem svetu. Z njo se vsakodnevno srečujemo na internetu pa tudi v vseh ostalih informacijskih sistemih in aplikacijah, ki se uporabljajo v poslovnem in privatnem življenju.

Upravljanje z digitalnimi identitetami je množica procesov, orodij in dogovorov, povezanih z ustvarjanjem, vzdrževanjem ter prenehanjem digitalne identitete. Je integrirana rešitev, ki omogoča določanje in obvladovanje dostopov do različnih informacijskih virov v skladu s sprejetimi politikami. Sprva je bilo upravljanje z identitetami uporabljeno samo za upravljanje informacij o uporabniških računih ter za kontrolo dostopa do sistema in morda nekaj uporabniških aplikacij. Zadnje čase pa se je pomen upravljanja z identitetami zelo razširil glede na njegov prvoten namen ter postal ključni člen pri omogočanju elektronskega poslovanja. Je osnova za podporo poslovnih odnosov, za izboljševanje uporabniške izkušnje, za zaščito zasebnosti ter izvrševanje vseh potrebnih kontrol dostopa do najrazličnejših informacijskih sistemov.

Upravljanje z identitetami v digitalnem svetu srečamo na treh področjih: kot uporabniki interneta, kot zaposleni v nekem poslovnem okolju ter kot državljan oz. uporabnik državnih informacijskih sistemov. Sledi kratek opis posameznih področij.

- **Internet in e-trgovanje**

V bogastvu elektronskega življenja se zrcali širina našega fizičnega življenja. Aktivnosti, kot so nakupovanje, razprave, druženje, zabava in poslovno sodelovanje, se vedno bolj selijo iz fizičnega v kibernetski oz. virtualni svet. Ob povečanju možnosti in ponudb, ki so nam na voljo, ljudje od le-teh pričakujemo tudi vedno večjo koristnost ter udobnost. Ne želimo si ob vsaki uporabi določene aplikacije vedno znova vpisovati svojega uporabniškega imena in

gesla. Od spletne trgovine, kjer pogosto nakupujemo, pričakujemo, da si bo zapomnila naše osebne podatke, kot je na primer naš naslov za dostavo. Kadar iščemo novo glasbo in filme, želimo, da nam spletna stran sama predlaga izvajalce in naslove glede na naša predhodna iskanja ter nakupe.

Potrošniško spletno okolje je najbolj aktivno v povezavi z upravljanjem z identitetami. Obstaja kar nekaj rešitev za lajšanje interakcije med uporabniki oz. potrošniki in ponudniki storitev na internetu. Primer takih rešitev sta Microsoft MyServices in Liberty Alliance Project, ki olajšujeta e-trgovanje in transakcije, s tem da zmanjšata število večkratnih vpisov oz. avtorizacij na samo eno, začetno prijavo v sistem. Obstajajo tudi razna »združenja upravljanja z identitetami«, kamor spadajo razne tehnike in rešitve, ki jih nudijo *hranitelji oz. ponudniki identitet*. Njihova naloga je, da tvorijo zaupanja vreden člen med ponudniki storitev in internetnimi uporabniki. Preko njih se uporabnik samo enkrat prijavi v sistem, nato pa dostopa do večih strani, ponavadi spletnih trgovin, ki so povezane pod skupnim *oskrbovalcem identitet.* Na zunaj je tako uporabniku olajšano spletno trgovanje, na znotraj pa poteka izmenjava podatkov o identitetah in profilih uporabnikov med e-trgovinami. Pri tem pa se moramo zavedati tudi negativnih posledic, do katerih lahko pride ob morebitni zlorabi zaupnih podatkov.

- **Poslovno okolje**

Tudi v poslovnih informacijskih sistemih je upravljanje z identitetami zelo pomemben člen in s tem eden ključnih gradnikov varnostnih mehanizmov poslovnih sistemov. Dandanes imajo namreč uporabniki na svojih delovnih mestih vedno več gesel za dostop do različnih informacijskih virov. Administratorji porabijo veliko časa za vzpostavitev novega uporabniškega računa, za nastavitev vseh potrebnih pooblastil za novega uporabnika, za upravljanje z računi in pozabljenimi gesli ter tudi za ukinitev nepotrebnega računa. Vsako napredovanje uporabnika ali kakršnakoli druga sprememba v njegovi delovni vlogi zahteva spet dodatno delo administratorja. Ob večanju števila novih aplikacij in širitvi obstoječih sistemov se pojavljajo težave pri omejevanju in nadzoru dostopa do kritičnih podatkov ali aplikacij.

Na splošno postaja upravljanje z identitetami zaposlenih vedno bolj kompliciranо, ker so sami informacijski sistemi vedno bolj razsežni. V njih so informacije o posamezni identiteti pogosto razdrobljene ter razpršene in tudi upravljane s strani več administratorjev. Problem se še poslabša, ko začnejo podjetja in organizacije sodelovati med seboj. Tako dobimo še več privatnih in javnih e-trgovskih združenj, nabavnih verig ter internetnih poslovnih skupnosti.

Na tržišču obstaja precej programskih rešitev za celovito upravljanje z identitetami. Večina od njih je v pomoč administratorjem pri upravljanje zahtev za dostop do raznih informacijskih virov. Sama konfiguracija za nadzor dostopov lahko poteka na podlagi pooblastil in zgoraj opisanega modela RBAC (Role Based Access Control). Uspešna vpeljava tega modela v nek poslovni informacijski sistem administratorjem odvzame veliko dela in bremena. Ob spremembi statusa ali položaja nekega delavca administrator temu spremeni le vrsto vloge, sistem pa sam avtomatsko določi, kakšne so nove pravice in dolžnosti uporabnika. Tako administrator vsake spremembe ne opravlja ročno, ampak sistem sam poskrbi za to. Tudi v

primeru, da uporabnik potrebuje dodaten dostop, do katerega nima pravice, ga lahko preko samopostrežnega sistema zahteva kar sam. Ena glavnih nalog sistema za upravljanje z identitetami je tudi lažji nadzor nad celotnim dogajanjem v sistemu in sledenje dostopov do zaščitenih virov, ki jih sistem prav tako opravlja sam. Ob morebitnih vdorih in neobičajnih situacijah pa obvesti administratorja, ki nato ustrezno ukrepa.

Obstajajo tudi rešitve, ki temeljijo na infrastrukturi javnih ključev (PKI – Public Key Infrastructure). Po eni strani omogočajo varnejši način avtorizacije s pomočjo digitalnih certifikatov, po drugi strani pa otežijo in povečajo delo administratorjem, ki morajo konstantno nadzirati veljavnost certifikatov ter ob ugotovitvi neveljavnosti nekega certifikata le-tega preklicati.

- **Informacijski sistemi državne uprave**

Pojem identitete se pojavlja tudi v povezavi z državno upravo, kadar se nanaša na identifikacijo državljanov in njihovo interakcijo z javnimi storitvami ter vladnimi ustanovami. Ljudje se vsakodnevno srečujemo z opravki, povezanimi z našo identiteto, kot so urejanje osebnih dokumentov, vozniškega dovoljenja, zdravstvenega zavarovanja itd. Upravljanje z identitetami je za državne ustanove podobno tistemu za poslovne sisteme. Tudi tu ima veliko vlogo porazdeljenost in razpršenost osebnih podatkov, povezanih z državljani. Državljan je namreč s strani države identificiran na različne načine, glede na situacijo, v kateri se znajde. Veliko državnih ustanov in storitev se v zadnjem času seli na internet (e-uprava), kar zahteva spet nove načine identifikacije državljanov. Najpogosteje se v ta namen pojavljajo certifikati in pametne kartice. Glavni namen je racionalizirati državne storitve in jih narediti dostopnejše vsem državljanom. S prenosom določenih storitev na internet bi namreč zmanjšali čakalne vrste in izboljšali ter pospešili nekatere upravne postopke. Po drugi strani pa ne smemo pozabiti spet na skrb vzbujajoče posledice, do katerih lahko take spremembe vodijo. To so povečanje tveganja za izgubo zasebnosti ter možne zlorabe osebnih podatkov.

Kadar govorimo o upravljanju z digitalnimi identitetami, je potrebno razumeti, da se ne soočamo le s tehničnimi dejstvi in težavami, ampak so prisotni tudi socialni ter zakonodajni vidiki. Močno povečanje števila digitalnih identitet na uporabnika se sčasoma ne bo ustavilo, ampak se bo samo še povečevalo. Na žalost pa se bo s tem povečalo tudi število negativnih posledic, kot so kraja identitete in druge, z identitetami povezane prevare. Te lahko povzročijo resne probleme v našem zasebnem in poslovnem življenju. Zato je še kako pomembno imeti dober sistem za upravljanje z digitalnimi identitetami. Ne samo da lahko dober sistem zaščiti naše osebne podatke, ob morebitni zlorabi nam je lahko v veliko pomoč pri odkrivanju kriminalcev ter odskočna deska za forenzično analizo.

2.2.1. Elementi sistema za upravljanje z identitetami

Na upravljanje z identitetami moramo pogledati iz več perspektiv. Tri izmed pomembnejših so: tehnična, socialna in zakonodajna. Pri teh pogledih se srečamo z naslednjimi pojmi in zahtevami:

Avtorizacija: uporabnik se z avtorizacijo sistemu predstavi in dokaže, da ima pravico do uporabe le-tega. Z uspešno avtorizacijo dobi pravico dostopati do sistema oz. aplikacije. Vpliv in pomembnost avtorizacije sta premo sorazmerna s tveganjem pri izvedbi transakcije. Ni vseeno, ali gre za avtorizacijo pri manjšem nakupu preko spleta ali pomembno poslovno transakcijo.

Zaupanje: igra ključno vlogo v tem okolju. Pogost način avtorizacije je preko zaupanja vredne tretje strani, ki preveri, certificira ter po potrebi prekliče identiteto ter podatke o določenem profilu. Pojavljajo se tudi že razne storitve glede zaupanja, ki se ukvarjajo z več področji zaupanja. Ta so poleg certificiranj še kontrola identitete, priporočila, ocenitev kreditne sposobnosti, zaupen pregled računov itd.

Dinamičnost in minljivost: je velik problem, predvsem ko imamo opravka z informacijami na internetu, ki se lahko zelo hitro spreminjajo, kot naprimer limit na kreditni kartici, privilegiji, digitalne pravice itd. Na vrednosti teh podatkov lahko vpliva že ena sama transakcija oz. interakcija uporabnika s sistemom, zato je zelo pomembno, da te podatke ohranjamo ažurne in tako zagotovimo zanesljivost.

Življenjska doba identitet: nobena elektronska transakcija ni stoodstotno zanesljiva, zato se lahko pojavijo spori tudi več let po izvedbi transakcije. Takrat je ključni element, ali smo zmožni dokazati resničnost zabeleženih dogodkov. Seveda so se lahko od tiste situacije podatki glede neke identitete zelo spremenili, zato je pomembno, da so vse spremembe, ki so se zgodile, zabeležene. Potrebno je hranjenje vseh sprememb v poslovnem okolju, kjer se vloge ljudi v organizaciji spreminjajo, skupaj z ljudmi samimi. Poleg podatkov o posameznem zaposlenem je potrebno hraniti tudi zgodovino posameznih vlog. Zato je zelo pomembno, da sistem za upravljanje z identitetami vsebuje mehanizem za ohranjanje podatkov o vseh novih in tudi izteklih identitetah.

Tajnost: je pomemben del varnostnih zahtev, saj preprečuje zlorabo informacij o identitetah. Sistemi za upravljanje z identitetami namreč pogosto upravljajo s tajnimi podatki, ki morajo biti primerno varovani. Lastnikom identitet mora biti omogočeno odločanje, kateri njihovi podatki bodo vidni, kateri ne in seveda komu.

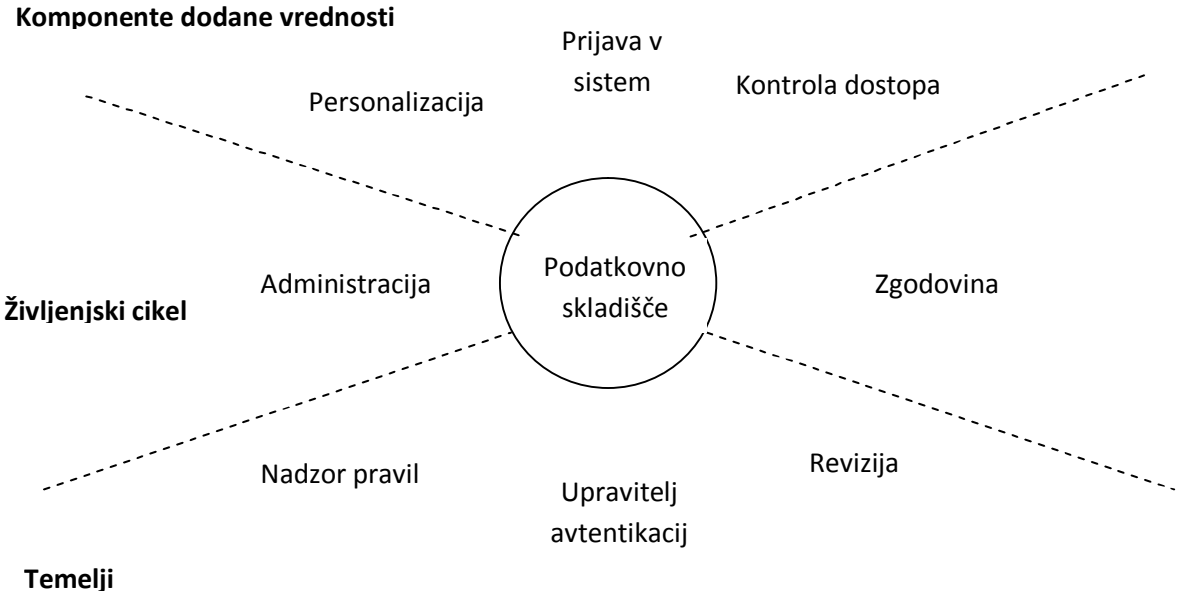
Odgovornost: Pomembno je, da so upravljavci identitet odgovorni pri svojem delu ter da so njihova opravila transparentna. To je pomembno za povečanje zaupanja ter verodostojnosti interneta.

Enostavnost in povezovanje: Sistem za upravljanje z identitetami naj bi bil enostaven za uporabo za vse, ki do sistema dostopajo. Še posebej je to pomembno s strani uporabnikov v svetu elektronskega nakupovanja, kjer avtorizacija ponavadi uporabniku predstavlja pregrado

oz. oviro pri dostopu do želene storitve. Prav tako je enostavnost sistema pomembna za same administratorje, ki se spopadajo s kompleksnimi nalogami in jim pregleden in jasen sistem lahko delo zelo olajša. Nenazadnje pa je enostavnost integracije ključni element za uspešnost sistema.

2.2.2. Model sistema za upravljanje z identitetami

Avtor Joseph Pato [9] opisuje zgradbo sistema takole: Zgradba sistema za upravljanje z identitetami je zelo kompleksna, sestavljena iz več komponent in storitev, saj le tako lahko zadosti vsem zgoraj opisanim zahtevam in potrebam. Primer predlagane rešitve je prikazan na spodnji sliki (Slika 2).



Slika 2: Zgradba sistema za upravljanje z digitalnimi identitetami.

Temelji:

- **Podatkovno skladišče:** v jedru sistema imamo logično podatkovno skladišče in podatkovni model identitet oz. model kontrole dostopa. Tu so shranjena tudi pravila za dostop do podatkov sistema.
- **Upravitelj avtentikacije:** ali drugače povedano upravitelj identitet je odgovoren za izvajanje primarne avtentikacije, ki ob prijavi v sistem posameznika z njegovo identiteto. Upravitelj avtentikacije ustvari poseben niz znakov, s pomočjo katerega ostale komponente prepoznajo, da je bila izvedena primarna avtentikacija. Pod primarno avtentikacijo razumemo mehanizme, kot so preverjanje uporabniškega gesla, preverjanje pametne kartice, biometrično skeniranje, preverjanje certifikata in podobno. Vsaka identiteta je lahko povezana z več kot enim overiteljem. Način posamezne avtentikacije ima lahko različno moč in zato lahko posamezna aplikacija zahteva avtentikacijo z določeno minimalno močjo.

- **Nadzor pravil:** dostop in uporaba identitete sta upravljana s pomočjo nadzora pravil. Tu gre za avtorizacijska pravila ter pravila za ohranjanje zasebnosti.
- **Revizija:** je mehanizem, ki sledi vsem podatkom, ki se v »skladišču« ustvarijo, spreminjajo ter uporabljajo. Je bistven člen forenzične analize, ki se uporablja za ugotavljanje, kdo in kako je zlorabil pravila v sistemu.

Komponente življenjskega cikla:

- **Administracija:** je avtomatizacija vseh postopkov in orodij za upravljanje z življenjskim ciklom identitete. Ti postopki so:
  - ustvarjanje *identifikatorja* za identiteto ter spojitev le-tega z upraviteljem avtorizacije,
  - določanje in spreminjanje atributov in pravic,
  - pridobivanje aktualnih informacij o identitetah od zaupanja vrednih virov ter preverjanje zanesljivosti in resničnosti podatkov,
  - odvzem pooblastil določeni identiteti.
- **Zgodovina:** Ta komponenta ustvarja in hrani zapise sprememb za vsako identiteto. Vsaka najmanjša sprememba identitete mora biti shranjena, najbolje če v razpršeni sistem, odporen na napake. S tem je omogočen naknadni pregled življenjskega cikla identitete.

Komponente dodane vrednosti (Consumable):

- **Prijava v sistem:** Uporabniku omogoča, da po uspešni primarni avtorizaciji dostopa do vseh aplikacij in sistemov, za katere ima dovoljen dostop v sklopu svoje identitete.
- **Personalizacija:** skupaj z upravljanjem prioritet dovoljuje, da je vsaka aplikacija prilagojena posamezni identiteti, ki jo uporablja. S pomočjo tega orodja aplikacija lahko prikroji nastavitve in delovanje glede na predhodne izkušnje posameznega uporabnika.
- **Kontrola dostopa:** Podobno kot nadzor pravil v temeljnih komponentah sistema, komponente kontrole dostopa dovoljujejo aplikacijam opravljanje avtorizacije in sprejemanje drugih odločitev na podlagi pravic in pravil, shranjenih v podatkovnem skladišču sistema.

Zmožnost prepoznavanja digitalne identitete ljudi in spletnih storitev, razumevanje, upravljanje ter potrjevanje njihovih profilov in pravic, je temelj za določanje odgovornosti v poslovnih procesih ter trgovskih transakcijah. Kot na vsako drugo stvar lahko tudi na upravljanje z digitalnimi identitetami pogledamo z dveh strani. Pozitivne in negativne. Po eni strani lahko poznavanje profilov uporabnikov in njihove preference pripomorejo k nudenju prilagojenih in boljših storitev, ki zadovoljijo uporabnika. Po drugi strani pa lahko že majhna zloraba vodi v kriminalno dejanje in v kršenje človekovih pravic. Pomembno je razumeti in poznati sisteme za upravljanje z digitalnimi identitetami, saj so stalnica v našem poslovnem in privatnem življenju.

3. Varnost pri načrtovanju informacijskih sistemov

Varnostni vidik je vsebovan v vsakem modernem informacijskem sistemu, ki se ne uporablja v nekem popolnoma zaupanja vrednem okolju. Kljub temu ne obstaja veliko pomoči za programerje, ki morajo te sisteme zgraditi. Poleg zagotavljanja neranljivosti sistemov morajo programerji znati zagotoviti, da bo progam oz. sistem pravilno nameščen, nastavljen, bo pravilno funkcioniral in ga bo administrator lahko tudi ustrezno upravljal. Poleg tega mora programer poskrbeti tudi za izdelavo ustrezne dokumentacije.

3.1. UML kot prevladujoče načrtovalsko orodje

Obsežni informacijski in aplikacijski sistemi, tisti, ki so jedro poslovnih aplikacij in omogočajo delovanje podjetja oz. organizacije, morajo biti več kot le množica programskih modulov. Zgrajeni morajo biti na način, da omogočajo razširljivost, varnost in stabilnost, predvsem pri delovanju v stresnih razmerah. Njihova struktura oz. arhitektura mora biti specificirana na jasen in enostaven način, ki omogoča vzdrževalcem in programerjem, da lahko hitro najdejo napako in jo tudi popravijo. Vse to pa je seveda zaželjeno tudi pri manjših sistemih in aplikacijah, ne samo obsežnih. Vendar je pri obvladovanju obsežnih kompleksnih sistemov še veliko bolj pomembno, kako dobra je struktura sistema. V veliko pomoč pri modeliranju kompleksne strukture so nam modeli pri načrtovanju. Ti nam omogočajo ponovno uporabo nekoč že definiranih elementov in kod, shranjenih v knjižnicah. Ko želimo v sistem dodati neko funkcionalnost, ki že obstaja, lahko to enostavno vključimo z vpeljavo primernega elementa oz. modela iz obstoječe knjižnice modelov.

Modeliranje je v bistvu izdelava oz. razvoj programskih aplikacij pred samim kodiranjem. Je zelo pomemben člen v razvoju informacijskih sistemov. V praksi se je izkazalo, da imajo obširni sistemi veliko verjetnost neuspeha. Pravzaprav je večja verjetnost, da neka večja aplikacija ne bo delovala, kot je bilo načrtovano v določenem razvojnem času ter v okviru načrtovanih stroškov, kot da bi to uspelo. Zato je nujno, da naredimo vse, kar je mogoče, da poskusimo v času razvoja čim bolj povečati verjetnost uspeha našega projekta. V tem primeru je modeliranje nepogrešljiv pripomoček, s katerim lahko vizualiziramo naš načrt in preverimo, ali je zadoščeno vsem zahtevam. Še ena prednost modela je ta, da nam omogoča delo na višjem nivoju abstrakcije, kar pomeni skrivanje nepotrebnih podrobnosti v času načrtovanja in s tem možnost prikaza večje, po potrebi celotne slike sistema. S tem imamo celovit pregled nad načrtovanim sistemom in njegovimi funkcionalnostmi, hkrati pa se lahko po želji osredotočimo le na določeni del, njegove lastnosti ter podrobnosti.

UML (Unified Modeling Language) je sodoben, široko razširjen grafični jezik za vizualizacijo, specificiranje, konstruiranje in dokumentiranje komponent informacijskega sistema, njihove zgradbe in oblike. Uporablja se v stopnji analize in načrtovanja aplikacij in informacijskih sistemov. Prav tako je primeren za poslovno načrtovanje informacijskih pa tudi neinformacijskih sistemov. Z njimi lahko naredimo model kakršnih koli vrst aplikacij, pri čemer ni pomembno, na kateri vrsti strojne opreme in v katerem omrežju se bodo izvajale, na katerem operacijskem sistemu ali s katerim programskim jezikom bodo realizirane. Ta fleksibilnost je glavni razlog, zakaj je UML tako razširjen in uporaben.

V osnovi je UML objektno orientiran, saj vsebuje razrede oz. objekte in operacije, zato je zelo dobra osnova za modeliranje z objektno usmerjenimi programskimi jeziki in okolji, kot so C++, Java in vedno bolj uporabljan C#. Seveda lahko UML uporabimo tudi za načrtovanje neobjektno usmerjenih aplikacij, napisanih v programskem jeziku Fortran, VB ali COBOL.

Obstajajo različna orodja – odprtokodna in plačljiva, za načrtovanje UML diagramov. Poleg samega načrtovanja imajo še razne dodatne uporabne funkcionalnosti. Danes znajo nekatera orodja analizirati že obstoječo izvorno kodo in jo nato spremeniti v ustrezne diagrame UML. Se pravi, da delujejo v nasprotni smeri od običajnega poteka, ko iz samih diagramov tvorimo kodo programa. Nekatera orodja na tržišču znajo preveriti delovanje samega modela UML in s tem preveriti, ali res model počne to, kar od njega pričakujemo, in ali dela to na pravilen način. Obstajajo tudi orodja, ki so ponavadi namenjena točno določenim področjem oz. domenam, kot so naprimer telekomunikacije ali finance, ki znajo direktno, iz diagramov UML, generirati programsko kodo. Takšna koda je zelo zanesljiva, saj temelji na modelih najboljše prakse, ki jih uporabi generator ob generiranju kode. Na tak način pridemo veliko hitreje do zanesljive in delujoče kode in s tem celotne aplikacije.

Modeli UML tvorijo ogrodje *modelno usmerjene arhitekture (MDA – Model Driven Architecture),* katere snovalci so skupina strokovnjakov OMG (Object Management Group), ki so hkrati tudi avtorji samega UML-ja. Model, narejen z jezikom UML, je lahko kreiran za točno določeno platformo, lahko pa je od platforme neodvisen. Če je neodvisen od platforme, predstavlja model poslovno funkcionalnost in zelo natančno opisuje obnašanje sistema, ne vključuje pa tehničnih aspektov sistema. Za pretvorbo od platforme neodvisnega modela v model za specifično platformo obstajajo razna orodja oz. generatorji. Ta postopek pretvorbe je avtomatiziran, vseeno pa ne moremo pričakovati neke čarovnije, ki bo preprost model spremenila v izvršljiv program. Načrt mora biti pred pretvorbo zelo natančno specificiran in narejen na tak način, da bo pomagal generatorju ob pretvorbi sprejemati pravilne odločitve in s tem ustvariti ustrezen produkt.

Preden se lotimo izdelave načrta, je potrebno zbrati vse potrebne informacije in analizirati zahteve novega sistema, kar je zelo kompleksna naloga in za katero obstaja veliko različnih metodologij, ki določajo procedure, kako analizo in načrt izpeljati. Pomembna lastnost UML-ja, ena od tistih, zaradi katere je tudi tako razširjen, je ta, da je *metodološko neodvisen.* Se pravi, ni pomembno, po kateri metodologiji se zgledujemo pri analizi in načrtovanju, rezultate lahko vedno predstavimo v obliki diagramov UML. Poleg tega lahko s pomočjo XMI (XML Metada Interchange) prenašamo modele UML iz enega orodja v drugega, glede na to, kako želimo model preoblikovati oz. dopolniti. Vse to so prednosti standardizacije, kot je UML.

Zaradi vseh teh razlogov in lastnosti je postal UML najbolj razširjen in dejansko uporabljan standard v industrijskem načrtovanju. Dandanes je veliko razvijalcev usposobljenih za uporabo UML-ja, kar zmanjšuje stroške potrebnega dodatnega izobraževanja. Poleg tega je UML v primerjavi s starejšimi načrtovalskimi jeziki zelo natančno definiran. Vedno znova, ob razvoju novih modelov in idej, se nadgrajuje tudi sam standard UML. Tako se sčasoma samo izboljšuje število in kakovost orodij, namenih načrtovanju diagramov UML.

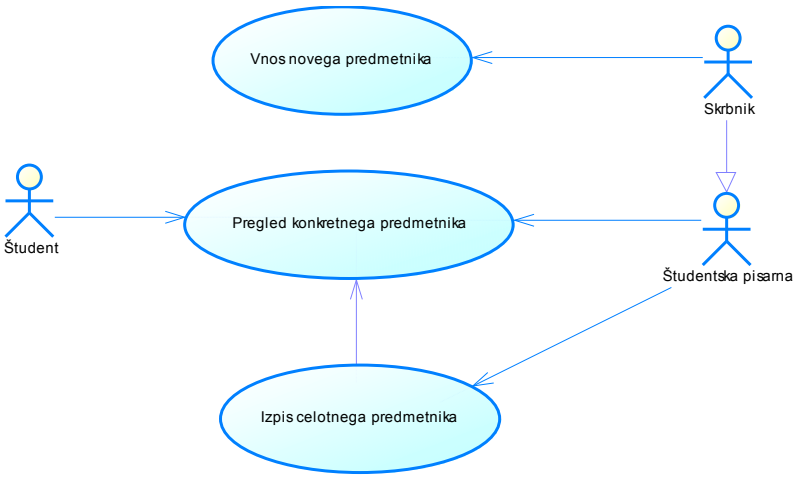
UML loči 13 diagramskih tipov, razdeljenih v 3 skupine. Šest diagramov predstavlja statično zgradbo aplikacij, trije predstavljajo osnovne tipe obnašanja sistema, štirje pa predstavljajo razne poglede na sistem in interakcije med elementi sistema.

- Strukturni diagrami:
  - razredni diagram *(class diagram),*
  - objektni diagram *(object diagram),*
  - diagram komponent *(component diagram),*
  - strukturni diagram *(composite structure diagram),*
  - diagram paketov *(package diagram),*
  - diagram postavitve (*deployment diagram*),
- Diagrami obnašanja:
  - diagram primerov uporabe *(use case diagram),*
  - diagram aktivnosti *(activity diagram),*
  - diagram stanj (*statechart diagram*).
- Diagrami interakcije:
  - diagram zaporedja *(sequence diagram),*
  - diagram sodelovanja oz. komunikacije *(collaboration/communication diagram).*
  - časovni diagram *(timing diagram),*
  - diagram pregleda interakcije *(interaction overview diagram).*

Sledi opis najbolj pogosto uporabljenih diagramskih tehnik, s katerimi se bomo srečali tudi v nadaljevanju. Pri opisu posamezne vrste diagrama je za lažjo predstavitev in boljše razumevanje dodan še preprost primerček iz področja študijske informatike.

❖ Diagram primerov uporabe

Vsak primer uporabe ponazarja določeno funkcionalnost sistema. Osnovni namen diagrama primerov uporabe je pomoč razvijalcem pri vizualizaciji zahtev sistema vključno s prikazom povezav med akterji oz. uporabniki sistema in osnovnimi procesi kot tudi povezav oz. razmerij med različnimi primeri uporabe. Diagram primerov uporabe prikazuje skupino primerov uporabe, lahko so to vsi primeri uporabe nekega sistema ali pa le podmnožica primerov uporabe, ki prikazujejo določeno funkcionalnost sistema. Grafično prikažemo primer uporabe kot elipso, znotraj katere je ime samega primera uporabe. Akter se ponazori z enostavno ikono v obliki človeka. Za ponazoritev razmerij med primeri uporabe in akterji se uporabijo črte, usmerjene ali ne.

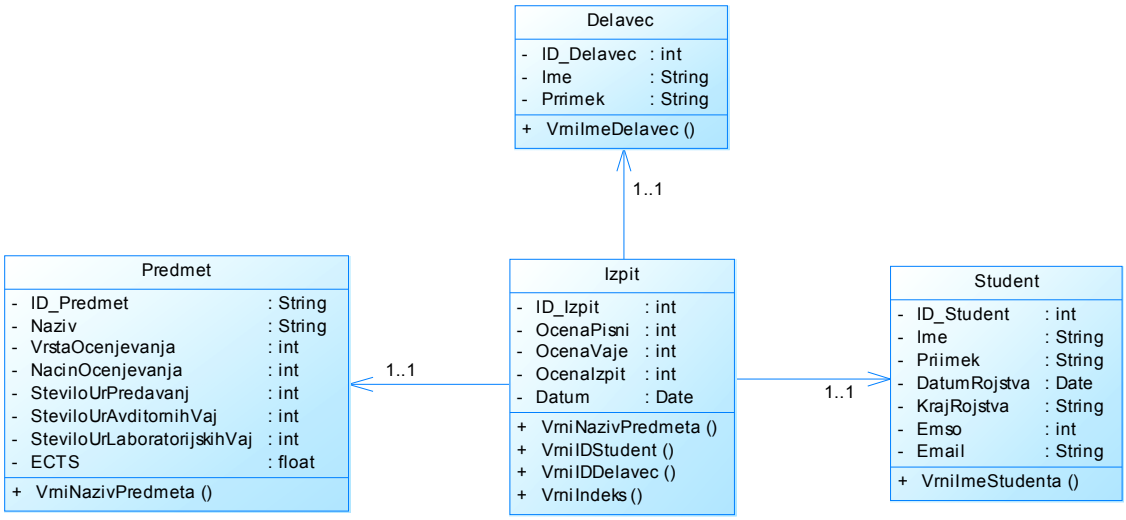


Slika 3: Primer diagrama primerov uporabe.

Tipično se diagram primerov uporabe uporablja za prikaz komunikacije v sistemu, na visokem nivoju. Iz slike (Slika 3) lahko hitro razberemo, katere funkcije nam omogoča naš sistem in kdo ima dostop do teh funkcij. V našem primeru imamo tri akterje: študent, skrbnik e-Študenta in študentska pisarna. Primer prikazuje vnos in pregled predmetnika. Sam vnos predmetnika lahko opravi le skrbnik informacijskega sistema. Študent ima pravice le do pregleda predmetnika za smer, ki jo študira. Študentska pisarna pa lahko pregleduje celoten predmetnik za študijsko leto. Ker skrbnik deduje od študentske pisarne, ima tudi on vse pravice, kot jih ima pisarna.

❖ *Razredni diagram*

Razred je množica objektov oz. entitet z enako zgradbo, obnašanjem in pomenom. Razredni diagram ponazarja, kako so različne entitete v sistemu (npr. ljudje, stvari, podatki) povezani med seboj. Z drugimi besedami, prikaže statičen vidik sistema, se pravi logične razrede, ki se pojavljajo v sistemu. Razredni diagrami se lahko uporabijo tudi za prikaz razredov, ki jih bodo razvijalci kasneje implementirali. Razred je na razrednem diagramu prikazan kot pravokotnik s tremi vodoravnimi predeli. Zgornji del vsebuje ime razreda, srednji del vsebuje atribute razreda, spodnji del pa seznam operacij oz. funkcij razreda. Relacije med razredi so predstavljene s povezavami, kot so asociacija in generalizacija oz. specializacija. Simbol na koncu povezave je odvisen od vrste povezave. Na primer generalizacija se ponazori z ravno črto in trikotnikom na enem koncu. Asociacija je lahko samo ravna črta ali pa usmerjena puščica, odvisno od tega, ali se oba razreda zavedata prisotnosti drugega ali ne.



Slika 4: Primer razrednega diagrama.

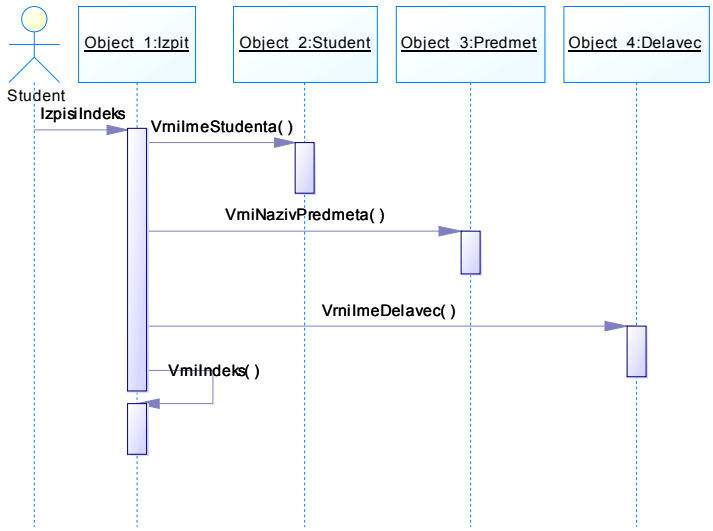
Na zgornji sliki (Slika 4) je prikazan enostavni razredni diagram, ki se nanaša na izpis elektronskega indeksa, torej ocen študenta. Imamo štiri razrede in med njimi usmerjeno asociacijo. To pomeni, da razred *Izpit* vidi ostale razrede (*Predmet, Delavec in Student*), zato lahko dostopa tudi do njihovih atributov in metod. Za omenjene tri razrede pa je razred *Izpit* neviden.

❖ *Objektni diagram*

Objektni diagram je statičen diagram in je zelo povezan z razrednim diagramom. Kot je objekt instanca nekega razreda, tako je tudi objektni diagram instanca razrednega diagrama. Objektni diagram opisuje statično zgradbo sistema v določenem trenutku in se zato uporablja predvsem za pojasnjevanje kompleksnih razredov ter preverjanje točnosti razrednega diagrama. Vsak objekt je predstavljen s pravokotnikom, ki vsebuje ime objekta in ime razreda, ki mu pripada. Tako kot imamo atribute v razredih, obstajajo v objektih prav tako atributi, katerim pa morajo biti dodeljene konkretne vrednosti. Z odebeljeno črto lahko označimo, kateri objekti so aktivni, to pomeni, da kontrolirajo tok podatkov. Množico več objektov lahko označimo tudi samo z enim simbolom, če niso pomembne vrednosti atributov v posameznem objektu. Sicer pa je sam objektni diagram po obliki enak razrednemu diagramu.

❖ *Diagram zaporedja*

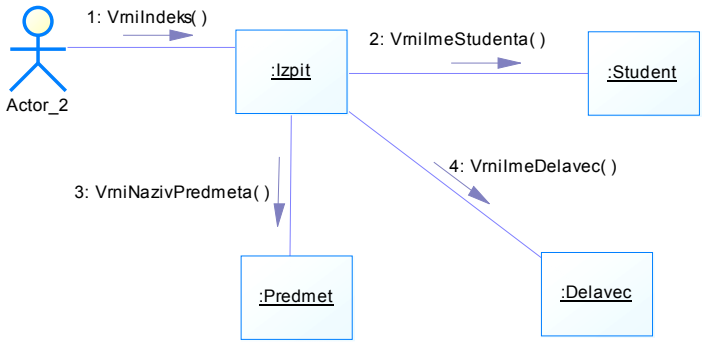
Diagram zaporedja prestavlja podroben tok podatkov za posamezen primer uporabe, lahko tudi samo za en del primera uporabe. Je torej realizacija primerov uporabe. Prikazuje klice med raznimi objekti, to je interakcija med komponentami sistema z vidika izmenjave sporočil. Diagram zaporedja ima dve dimenziji: navpična dimenzija prikazuje zaporedje sporočil oz. klicev, kot si sledijo v časovnem zaporedju; vodoravno dimenzijo pa sestavljajo instance objektov oz. razredov, ki si sporočila oz. podatke izmenjujejo. Na spodnjem primeru (Slika 5) je prikazano zaporedje dogodkov v sistemu ob izpisu elektronskega indeksa študenta. Po zahtevi za izpis indeksa s strani študenta sistem na polagi vpisne številke študenta poišče seznam vseh izpitov, ki jih je opravil študent s to vpisno številko. Nato za vsak izpit sistem poišče še podatke o nazivu predmeta in profesorja, pri katerem je študent izpit polagal.



Slika 5: Primer diagrama zaporedja za izpis elektronskega indeksa.

❖ *Diagram komunikacije (sodelovanja)*

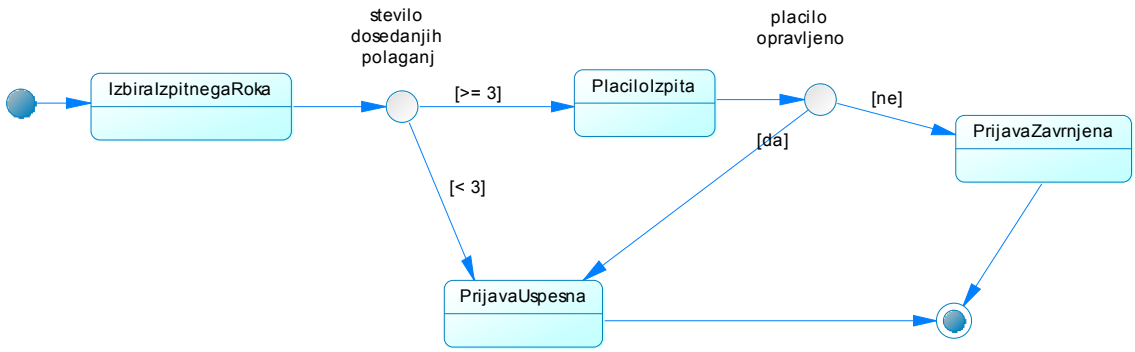
Diagram sodelovanja je zelo podoben diagramu zaporedja. Vsebuje iste elemente, le prikaz je malo drugačen. Dogajanje v diagramu sodelovanja (Slika 6) je torej isto kot v diagramu zaporedja. Bolje so vidni odnosi in organizacija objektov, težje pa se da razbrati časovni vidik.



Slika 6: Primer diagrama komunikacija.

❖ *Diagram stanj*

Diagram stanj prikazuje različna stanja, v katerih se lahko pojavi nek razred oz. objekt ter prehode med temi stanji, to je celotni življenjski cikel objekta. Spremembo stanja povzročajo dogodki. Ponavadi se diagram stanja naredi le za bolj »zanimive« razrede, ki lahko večkrat spremenijo svoje stanje v času delovanja sistema. Diagram stanj ima pet osnovnih elementov: začetna točka (poln krogec), prehajanje med stanji (črta s puščico), stanje (pravokotnik z zaobljenimi vogali), odločitvena točka (prazen krogec) ter končna točka (prazen krogec, ki vsebuje poln krogec).

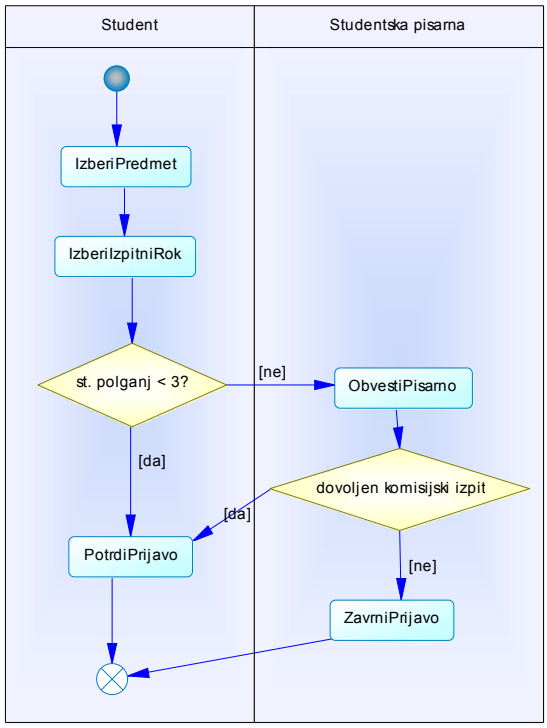


Slika 7: Primer diagrama stanj.

Zgornja slika (Slika 7) prikazuje enostaven primer diagrama stanja za prijavo na izpitni rok. Prijava se začne z izbiro izpitnega roka. Nato sistem preveri, kolikokrat je študent že polagal izpit pri izbranem predmetu. Če ga je polagal manj kot trikrat, je prijava uspešna. V primeru, da je izpit polagal že trikrat ali večkrat, pa mora najprej opraviti plačilo izpita. Če plačilo v določenem času izvede, je prijava uspešna, če pa izpita ne plača, prijavo sistem zavrne in študent na izpit ni prijavljen.

❖ *Diagram aktivnosti*

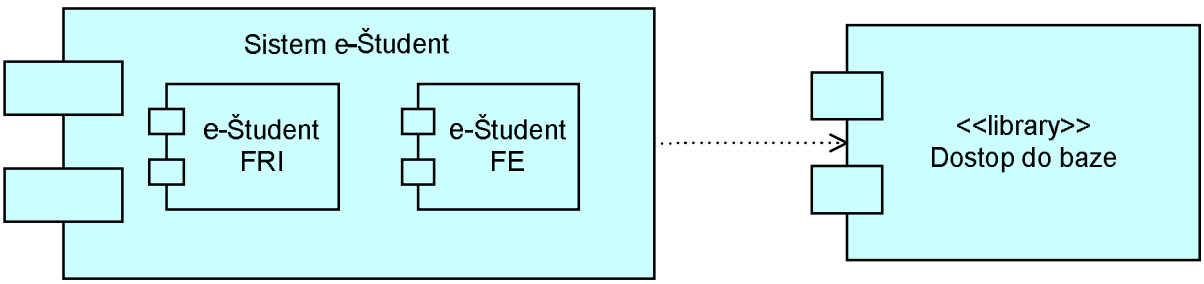
Diagram aktivnosti prikazuje kontrolni tok oz. tok podatkov med dvema ali več razredi med izvajanjem določene aktivnosti sistema. Diagrami aktivnosti se pogosto uporabljajo za opis poslovnih procesov, saj so manj tehnični po zgledu in lažji za razumevanje v primerjavi z diagrami zaporedja. Notacija je zelo podobna tisti pri diagramu sanj. Diagram aktivnosti (Slika 8) se prične s polnim krogcem in je povezan z začetno aktivnostjo. Aktivnost je ponazorjena kot pravokotnik z zaobljenimi vogali ter imenom aktivnosti v njem. Aktivnost je povezana z drugo aktivnostjo preko prehodnih poti ali z odločitveno točko, ki se povezuje z drugimi aktivnostmi, glede na pogoj odločitvene točke. Končne aktivnosti so povezane s končno točko, ki ima enak simbol kot v diagramu stanj (prazen krogec s polnim krogcem v sredini). Aktivnosti so lahko tudi grupirane v skupine in razporejene v proge, ki predstavljajo objekt oz. akterja, ki te aktivnosti izvaja.



Slika 8: Primer diagrama aktivnosti.

❖ *Diagram komponent*

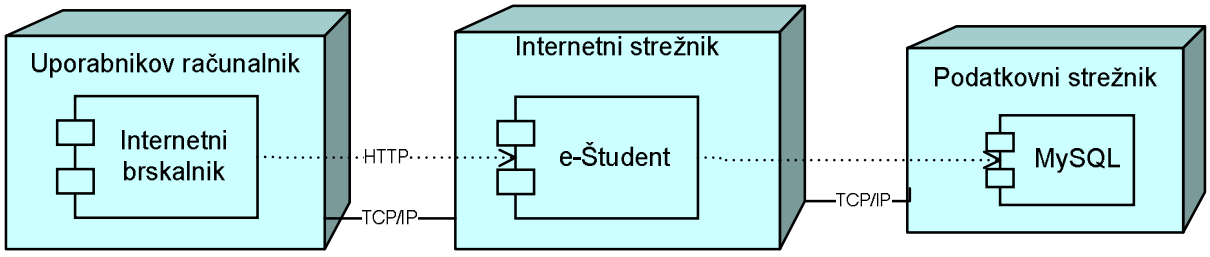
Diagram komponent prikazuje fizičen pogled na sistem. Njegov namen je prikaz odvisnosti med različnimi programskimi komponentami oz. aplikacijami, je torej prikaz med načrtovano aplikacijo in ostalimi aplikacijami in knjižnicami sistema. Diagram je lahko prikazan na zelo visokem nivoju abstrakcije, z velikimi okvirnimi komponentami ali pa na nivoju paketov posameznih komponent. Najboljši način opisa diagrama komponent je skozi primer. Na spodnjem primeru (Slika 9) vidimo, da je sistem e-Študent sestavljen iz večih podsistemov, kot sta e-Študent FRI in e-Študent FE. Sam e-Študent se med drugim povezuje s podatkovno bazo, ki je kot taka sam svoj sistem.



Slika 9: Primer diagrama komponent.

❖ *Diagram postavitve*

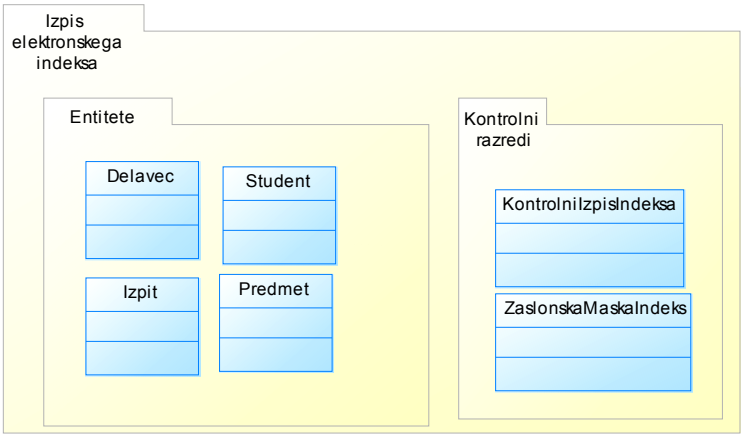
Diagram postavitve prikazuje, kako bo sistem fizično uveden/vpeljan na strojno opremo. Namen tega diagrama je prikazati, kje se bodo razne komponente sistema fizično nahajale in kako bodo med seboj komunicirale. Ker je tak diagram model fizičnega okolja, je najbolj koristen razvijalcem v produkciji. Notacija vsebuje iste elemente kot komponentni diagram, z nekaj dodatki (npr. koncept vozlišča). Vozlišče lahko predstavlja bodisi fizičen računalnik bodisi virtualno okolje. Simbol za označitev vozlišča je enostavna kocka z imenom vozlišča na vrhu kocke (Slika 10).



Slika 10: Primer diagrama postavitve.

❖ *Diagram paketov*

Z diagramom paketov (Slika 11) uredimo elemente sistema v sorodne skupine. Tako ponazorimo, kateri elementi so med sabo povezani, brez da bi dejansko risali vse povezave med njimi. S tem zmanjšamo število povezav. Paket označimo s simbolom za mapo. V zavihek paketa napišemo ime. Kakor razredi, imajo lahko tudi paketi lastne atribute. Pomembna lastnost paketov je vidnost, saj določa, kdo lahko dostopa do informacij znotraj paketa in kdo ne. Vidnost *privat(-)* pomeni, da atribut ali metoda nista dostopna zunaj paketa. Vidnost *javen (public(+))* dovoljuje dostop do atributa oz. metodo tudi zunaj paketa. Vidnost *zaščiten (protected(#))* pa naredi atribut oz. metodo dostopno paketom, ki dedujejo od tega osnovnega paketa. Odvisnost med paketi je določeno preko razmerij, ki ponazorijo, kako nek paket vpliva na ostale pakete.



Slika 11: Primer diagrama paketov.

3.2. UML varnostne razširitve

Kot smo omenili že v uvodu, je varnost eden izmed ključnih elementov današnjih informacijskih sistemov. Zaradi tega moramo o varnosti začeti razmišljati že takoj, v prvih fazah razvoja novega sistema, se pravi v fazi analize in načrtovanja informacijskega sistema. Tak pristop omogoča ugotavljanje in popravljanje napak, povezanih z varnostjo, že v zgodnji fazi razvoja novega sistema, kar predstavlja velik prihranek pri času in denarju.

Ker je ravno UML najbolj pogost načrtovalski standard, ki se uporablja v teh začetnih fazah, nas je zanimalo, kako podrobno je možno definirati varnostne mehanizme s samim jezikom UML. Za začetek smo pregledali kar nekaj literature in člankov na to temo, kjer smo našli razne ideje in predloge glede vpeljave varnostni v diagrame UML. Nekateri avtorji uporabljajo že obstoječe razširitvene mehanizme UML-ja, drugi pa vpeljejo nove modele, ki skrbijo za zaščito pred neavtoriziranim dostopom. Štiri najbolj zanimive ideje so opisane v nadaljevanju, v podpoglavjih z naslovi: UMLSec, Role-Based Access Control, Secure UML ter Enhancing UML with Security.

3.3. UMLsec

V člankih [1,2] avtor Jürjens opisuje predlog razširitve diagramske tehnike UML z imenom UMLsec, ki omogoča obravnavanje pomembnih varnostnih mehanizmov pri izdelavi načrta z diagrami UML. UMLsec je definiran tako, da uporablja standardne razširitvene mehanizme UML, torej ne uvaja nekih novih simbolov in oznak. Trije razširitveni mehanizmi, ki jih avtor uporablja, so: stereotipi (*stereotype)*, označene vrednosti (*tagged value*) in omejitve (*constraint*). **Stereotipi** definirajo nove elemente modela s pomočjo razširitve pomena obstoječih tipov oz. razredov. Uporabimo jih lahko pri vseh elementih modeliranja. Stereotipe lahko določimo podsistemom, razredom, povezavam ali katerim drugim elementom, ki jih želimo natančneje definirati. Označimo jih z imenom v dvojnih trikotnih oklepajih << >> ali z novim simbolom. **Označene vrednosti oz. oznake** uporabimo za eksplicitno določanje lastnosti elementov UML modela. Označena vrednost je par ime-vrednost, zapisana v zavitih oklepajih {*ime=vrednost*}. **Omejitve** so še ena možnost, kako dodati informacije oz. izpopolniti pravila, ki se nanašajo na elemente modela. Stereotipe lahko uporabimo tudi za določanje oz. dodajanje označenih vrednosti in omejitev elementov. Za vsak stereotip lahko točno določimo, katere lastnosti in omejitve mu pripadajo, in tako pri načrtovanju z določitvijo stereotipa avtomatsko določimo tudi pripadajoče omejitve in označene vrednosti.

Z uporabo teh razširitev se avtor osredotoči na rešitev štirih najpogostejših varnostnih zahtev, ki so nujne, če želimo, da bo naš sistem varno zgrajen. Te so:

- Poštena izmenjava (fair exchange): Pri izmenjavi elektronskega blaga oz. podatkov je pomembno, da izmenjava poteka tako, da prepreči goljufije na obeh straneh. Kot primer lahko vzamemo naročilo potrdil o šolanju preko interneta, ki zahteva, da po izvršenem naročilu študent dobi potrdilo o šolanju, dostavljeno po pošti ali pa mu sistem javi, da mora po potrdilo priti osebno.
- Tajnost/zaupnost (secrecy/confidentiality): Ena glavnih varnostnih zahtev je prav zaupnost. To pomeni, da lahko do nekega podatka pride le pooblaščena oseba. V

primeru e-študenta pomeni zaupnost to, da študent vidi le svoje ocene, ne pa tudi ocen vseh ostalih študentov.

- Varen pretok informacij (secure information flow): Med interakcijo dveh strani ne sme biti najmanjšega uhajanja podatkov k tretji, nepooblaščeni strani. Za to lahko poskrbimo z uvedbo kriptiranja in digitalnih certifikatov, ki jih uporabniki sistema uporabijo pri vpisu v sistem.
- Varen komunikacijski kanal (secure communication flow): Fizična komunikacijska pot med dvema vozliščema mora nuditi določeno varnostno garancijo. Na primer LAN je varen komunikacijski kanal zoper zunanje napadalce. Zato je v primerih, ko ne potrebujemo nujne povezave z internetom, dobra rešitev vzpostavitev lokalnega omrežja.

Poleg omenjenih varnostnih zahtev morajo biti zadoščene seveda še druge – osnovne varnostne zahteve, kot je razpoložljivost in popolnost oz. integriteta podatkov.

Za razširitev zgoraj omenjenih varnostnih zahtev avtor podaja nujne, na novo definirane stereotipe, pripadajoče označene vrednosti, omejitve ter nevarnosti (*threats)*. Seznam vseh teh je podan v spodnjih tabelah.

Stereotip	**Base Class**	**Označena vrednost**	**Opis**
internet	povezava		internetna povezava
kriptirano	povezava		kriptirana povezava
LAN	povezava		LAN povezava
varna povezava	podsistem		zahteva zaščitene komunikacijske povezave
tajno	odvisnost		zahteva tajnost
varna odvisnost	podsistem		<<kliči>>, <<pošlji>> spoštuje tajnost podatkov
kritično	objekt	tajno	kritični objekti
nepropustnost	podsistem		tok informacij; preprečevanje uhajanja podatkov
zaščita podatkov	podsistem		zagotavljanje tajnosti
poštena izmenjava	paket	začetek, konec	zahtevo pošteno izmenjavo

Tabela1: Stereotipi UMLsec (izvleček)

Označena vrednost	**Stereotip**	**Tip**	**Številnost**	**Opis**
tajno (secret)	<<kritično>>	niz	*	tajni podatki
začetek (start)	<<poštena izmenjava>>	niz	1	začetno stanje
konec (stop)	<<poštena izmenjava>>	niz	1	končno stanje

Tabela2: Označene vrednosti (izvleček)

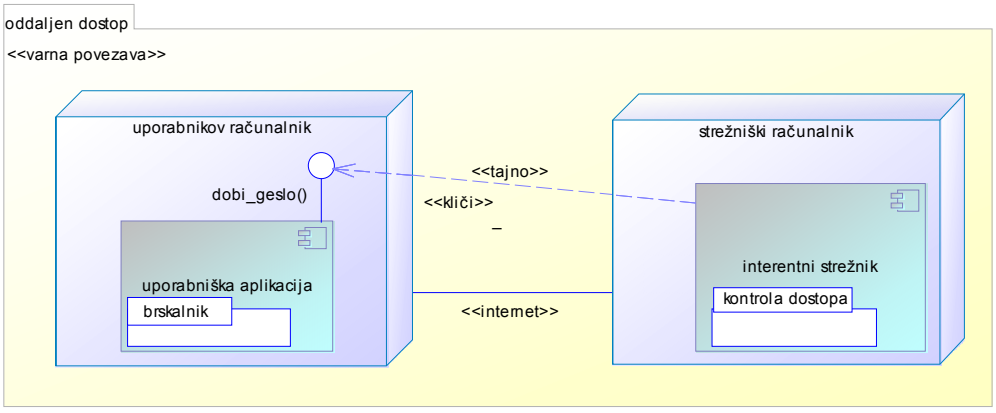
Stereotip	**Nevarnost**
internet	{briši, beri, spremeni}
kriptirana povezava	{briši}
LAN	/

Tabela3: Nevarnosti zunanjega napadalca

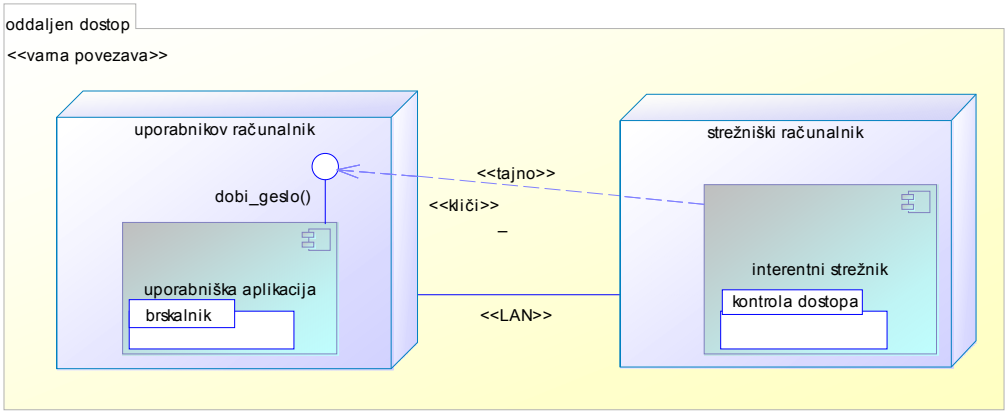
Pomen stereotipov in označenih vrednosti:

- *Internet, kriptirana povezava, LAN:* Ti stereotipi se uporabljajo na povezavah v diagramih implementacije in označujejo vrsto komunikacijske povezave. Vsaka povezava ima lahko določenega največ enega izmed teh treh stereotipov. Za vsakega izmed stereotipov {<<encrypted>>, <<LAN>>, <<Internet>>} moramo določiti nevarnosti, ki se lahko pojavijo pri uporabi posamezne povezave. Seznam teh nevarnosti je podan v Tabela 3. Primer uporabe stereotipov je podan v nadaljevanju.
- *Varna povezava (Secure links):* Ta stereotip se nanaša na nek podsistem in zagotavlja, da je med dvema vozliščema, ki komunicirata, varnost zagotovljena že na fizičnem nivoju. Ta omejitev zahteva, da mora biti med vozliščema povezava s takim stereotipom *s*, pri katerem ni možnih nevarnosti, ki bi naš sistem lahko ogrožale. Naprimer če želimo zaščititi sistem, moramo vzpostaviti povezavo s stereotipom, ki preprečuje neavtorizirano branje podatkov. Se pravi: branje ∉Nevarnost (povezave).
- *Tajno (Secrecy):* S tem stereotipom označujemo povezavi <<kliči>> ali <<pošlji>>, ki morata poskrbeti za tajnost prenesenih podatkov. Ta stereotip se uporablja kot omejitev znotraj podsistema označenega s stereotipom <<*varna povezava*>>.

Primer 1: Kršena je omejitev stereotipa <<*varna povezava*>>, ker sta komponenti podsistema povezani ne le s povezavo <<*kliči*>>, ki je varna, ampak tudi preko povezave <<*internet*>> (Slika12), povezava preko interneta pa ne zagotavlja potrebne varnostne zaščite, in sicer zaradi možnih nevarnosti – neavtoriziranega branja. Povezava, ki bi bila v tem primeru primerna, je bodisi kriptirana ali LAN povezava (Slika 13), saj sta ti dve povezavi varnejši in onemogočata vdor zunanjih napadalcev.



Slika 12: Prikaz napačnne uporabe stereotipa <<varna povezava>>.



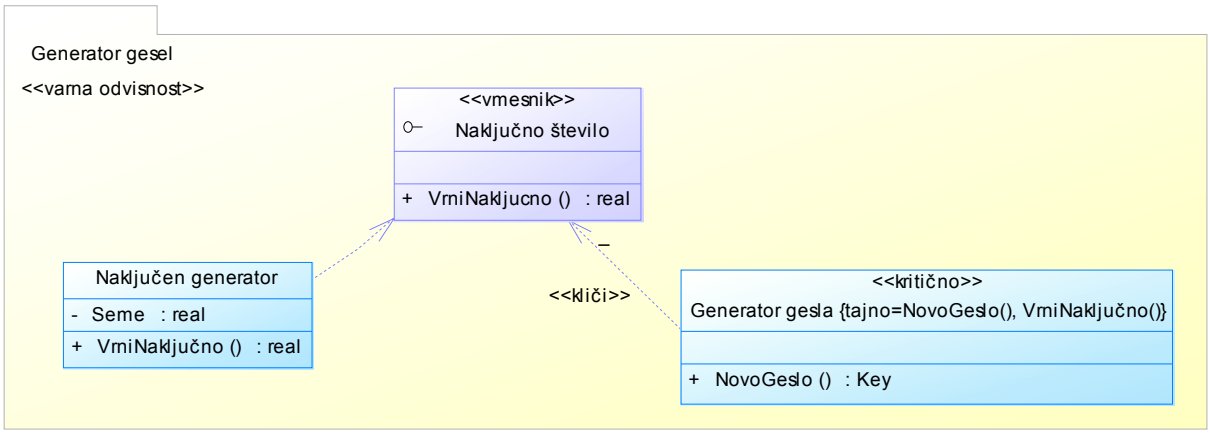
- *Varna odvisnost (Secure dependency):* uporablja se za

Slika 13: Prikaz pravilne uporabe stereotipa <<varna povezava>>.

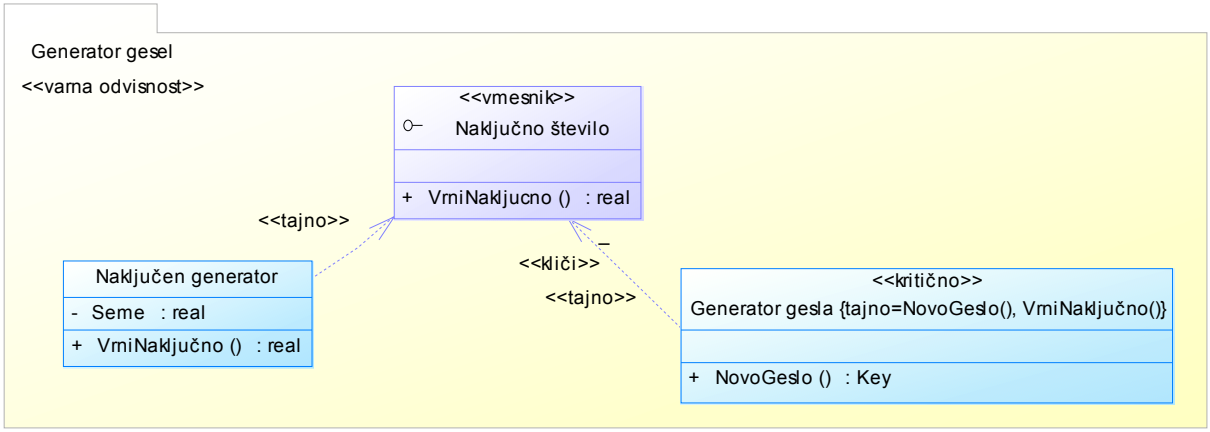
podsisteme, ki vsebujejo objektne in strukturne diagrame. Zagotavlja, da povezavi *<<kliči>>* in *<<pošlji>>* med objekti ali podsistemi (C in D) spoštujeta varnostne zahteve pri pretoku podatkov po njih. To pomeni:

- *Za vsako sporočilo n, ki ga pošlje D, se n pojavi v oznaki {tajno}, ko prispe do objekta ali podsistema C,* le če je tako označen tudi v D.
- *Če sporočilo, ki ga je poslal D, prispe v C v oznaki {tajno}, potem mora biti odvisnost oz. povezava med D in C označena s stereotipom <<tajnost>>.*

Primer 2: Na prvi sliki (Slika 14) ni upoštevana določitev stereotipa <<varna odvisnost>>, s katero je označen podsistem (Slika3). Razred *Generator gesla* namreč zahteva, da morata biti metodi *NovoGeslo() in VrniNaključno()* tajni, zato bi morali biti tudi povezavi med razredoma in vmesnikom označeni s stereotipom <<tajno>> (Slika 15).



Slika 14: Nepravilna uporaba stereotipa <<varna odvisnost>>.

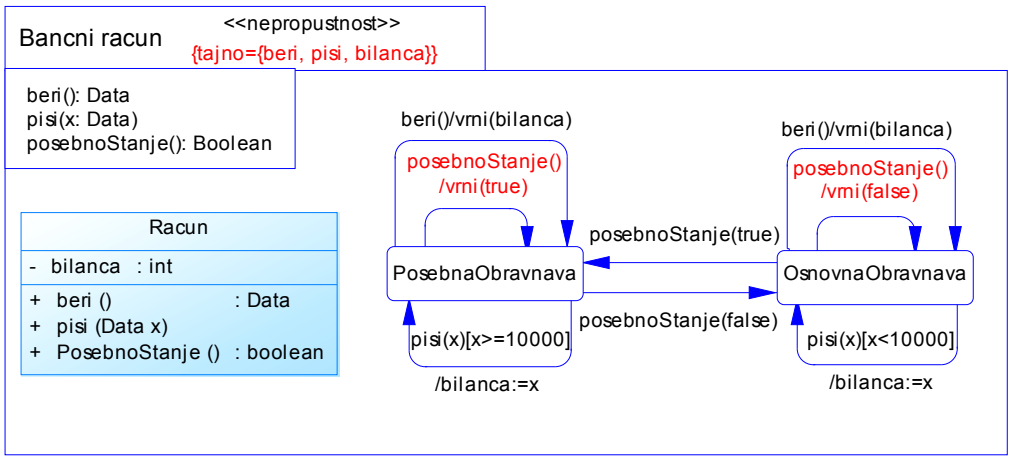


Slika 15: Pravilna uporaba stereotipa <<varna odvisnost>>.

- *Kritično (Critical):* Ta stereotip označuje objekte, katerih instance oz. primeri so kritični na kakršenkoli način. Se pravi, da vsebujejo atribute ali vrednosti, ki so tajne in morajo zato biti zaščitene z oznako {tajno}(secret). Stereotip <<kritično>> se uporablja znotraj podsistema, označenega s stereotipoma <<varnost podatkov>> ali <<neprepustnost>>, opisanima v nadaljevanju.

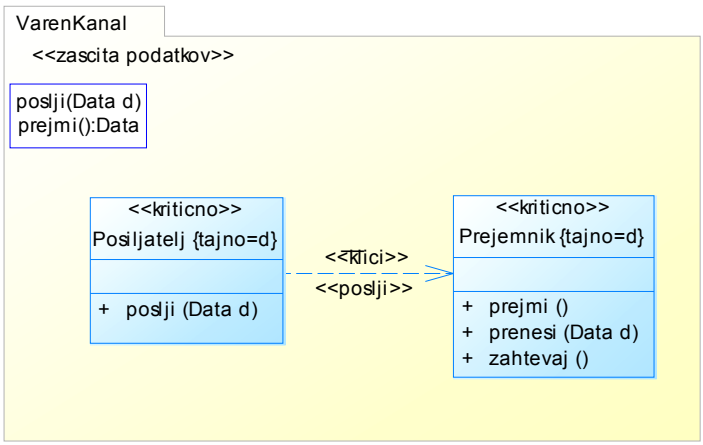
- *Neprepustnost:* je stereotip za označitev podsistema. Tako označen podsistem zagotavlja varen pretok informacij s pomočjo oznake {*tajno*}, s katero označimo vse funkcije, ki morajo biti tajne. S tem so tudi vrednosti, ki jih funkcije vračajo, tajne in jih napadalec ne more prebrati.

  Primer 3: Slika 16 prikazuje komponento bančne transakcije, ki omogoča branje tajne bilance z operacijo *beri()* in zapis tajne bilance s funkcijo *pisi()*. Vrednosti, ki jih funkcije vračajo, so prav tako tajne (imajo oznako {tajno}), saj to zahteva stereotip <<neprepustnost>>, s katerim je označen podsistem. Če je stanje na računu večje od 10000, je objekt v stanju *PosebnaObravnava,* sicer pa je v stanju *NormalnaObravnava.* Za ugotavljanje trenutnega stanja na računu uporabimo funkcijo *posebnoStanje(),* ki nam vrne true/false, odvisno, ali je bilanca večja od 10000 ali ne. Glede na zahteve stereotipa <<neprepustnost>> pride tu do kršitve zahteve po tajnosti podatkov, saj bi tudi funkcija *posebnoStanje()*morala biti označena kot tajna in bi tako tudi podatek o bilanci, ki ga *posebnoStanje()* vrača, bil tajen.



Slika 16: Primer uporabe stereotipa <<nepropustnost>>.

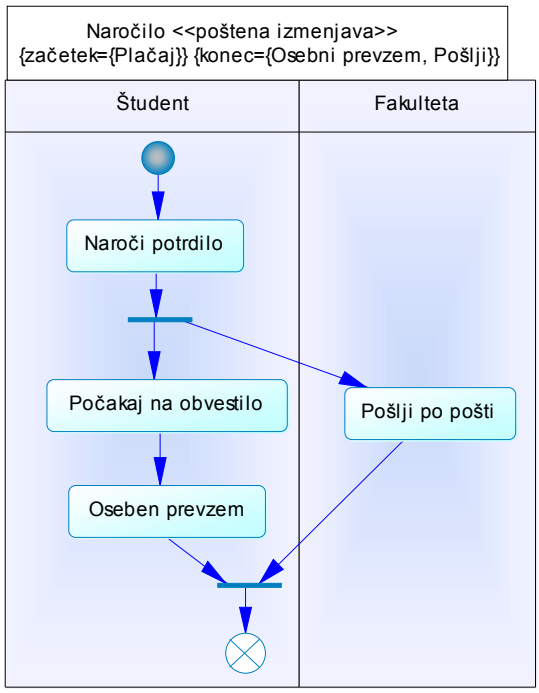
- *Zaščita podatkov:* Ta stereotip lahko označuje podsisteme in določa naslednjo omejitev. Podsistem se obnaša tako, da upošteva varnostne zahteve, povezane z zgoraj omenjenim stereotipom <<kritično>>, s katerim označimo kritične objekte v diagramu. Bolj natančno - ta stereotip obvaruje podatke, označene z oznako {*secret*}, znotraj kritičnih razredov pred nevarnostmi iz okolja. Primer je na spodnji sliki.



Slika 17: Primer varnega prenosa podatkov s stereotipoma <<zaščita podatkov>> in <<kritično>>.

- *Poštena izmenjava:* je stereotip podsistema in ima dve označeni vrednosti oz. oznaki: {začetek} in {konec}. Zahteva tega stereoptipa je, da če je doseženo stanje {začetek} v diagramu aktivnosti, bo zagotovo sčasoma nastopilo tudi stanje {konec}. Upoštevati pa je treba dejstvo, da tej zahtevi ne more biti zadoščeno v sistemih, ki jih napadalec lahko nasilno ustavi oz. ugasne.

Primer 5: Na sliki 18 je prikazana situacija elektronskega nakupa nekega izdelka oz. storitve in pravilna uporaba stereotipa <<poštena izmenjava>>. Glavna značilnost je, da morajo biti akcije, označene z {začetek} in {konec}, povezane med sabo na tak način, da če se zgodi ena izmed akcij z oznako začetek, se bo zagotovo sčasoma izvršila tudi ena od akcij z oznako konec. Na sliki je že zgoraj omenjeni primer naročila potrdil o šolanju preko interneta, ki zahteva, da po izvršenem naročilu študent dobi potrdilo o šolanju dostavljeno po pošti ali pa mu sistem javi, da mora po potrdilo priti osebno.



Slika 18: Primer pravilne uporabe stereotipa <<poštena izmenjava>>.

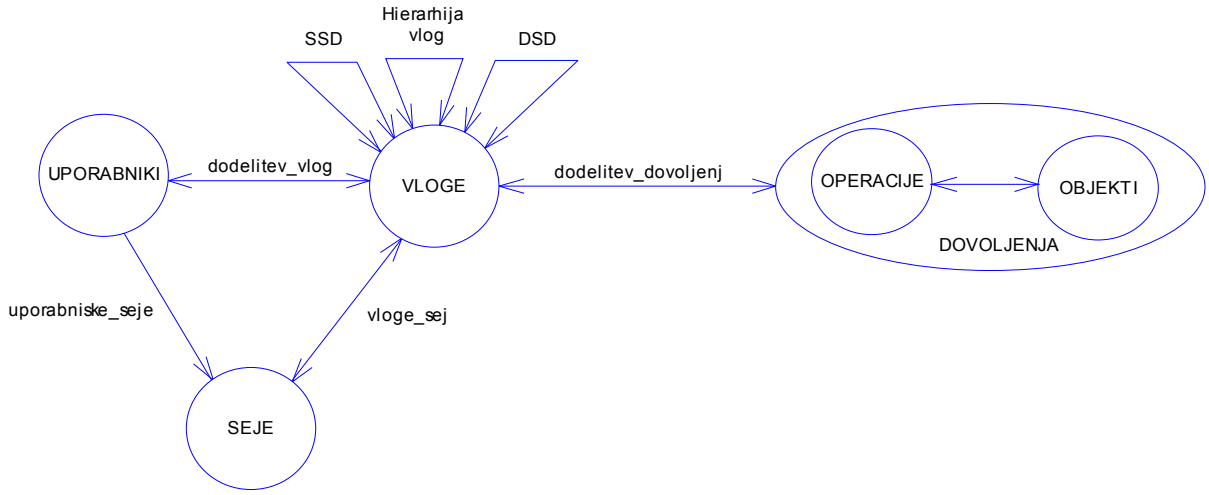
Omenjeni mehanizmi in predlogi pa nimajo samo teoretične podlage, ampak so bili uspešno uporabljeni tudi v praksi: pri projektih organizacije Visa International, pri projektu FairPay, financiranega s strani nemškega ministrstva za finance in pri projektih ene od večjih nemških bank.

3.4. Role-based access control (RBAC)

Avtorji v člankih [4] in [5] opisujejo, kako z uporabo jezika UML specificirati RBAC in MAC modela. Končni produkt je celovit model za kontrolo dostopa, razvit v obliki diagrama UML. Takega lahko dodamo v vsak načrtovani sistem, ki rabi poslovno logiko za varno kontrolo dostopa.

V članku [5] avtorji naredijo primerjavo med modeloma MAC in RBAC ter predlagajo nov, hibridni model (HAC – Hybrid access control) kot kombinacijo MAC in RBAC modela. Sam hibridni model se ne razlikuje kaj dosti od RBAC modela, pa tudi v kakšnih drugih člankih ga ne zasledimo, zato se bomo v nadaljnjem raziskovanju bolj osredotočili na najbolj razširjen RBAC model, ki je natančneje opisan v članku [4]. V tem članku je definiran model RBAC, ponazorjen s pomočjo jezika UML. Ta model lahko nato pri izdelavi informacijskega sistema kasneje vključimo v katerikoli načrt. Sam model ni predstavljen le vizualno, ampak s pomočjo objektnih diagramov predstavijo tudi pravila oz. omejitve modela RBAC.

Avtorji uporabijo tipičen RBAC model s tremi glavnimi komponentami, opisanimi v poglavju 2.1. Te so: osnovni model (*base model*), hierarhija vlog (*role hierarchies*) in omejitve (*constraints*). Splošna predstavitev modela RBAC je podana na spodnji sliki (Slika 19).



Slika 19: Splošna predstavitev modela RBAC.

Da bi model RBAC lažje vključili v nek načrtovan sistem, ga je potrebno definirati kot predlogo, narejeno z načrtovalskim jezikom UML. Na naslednji sliki (Slika 20) je razredni diagram, ki prikazuje hierarhični model RBAC z omejitvami SSD (Static separation of duty) in DSD (Dynamic separation of duty), opisanimi v nadaljevanju.

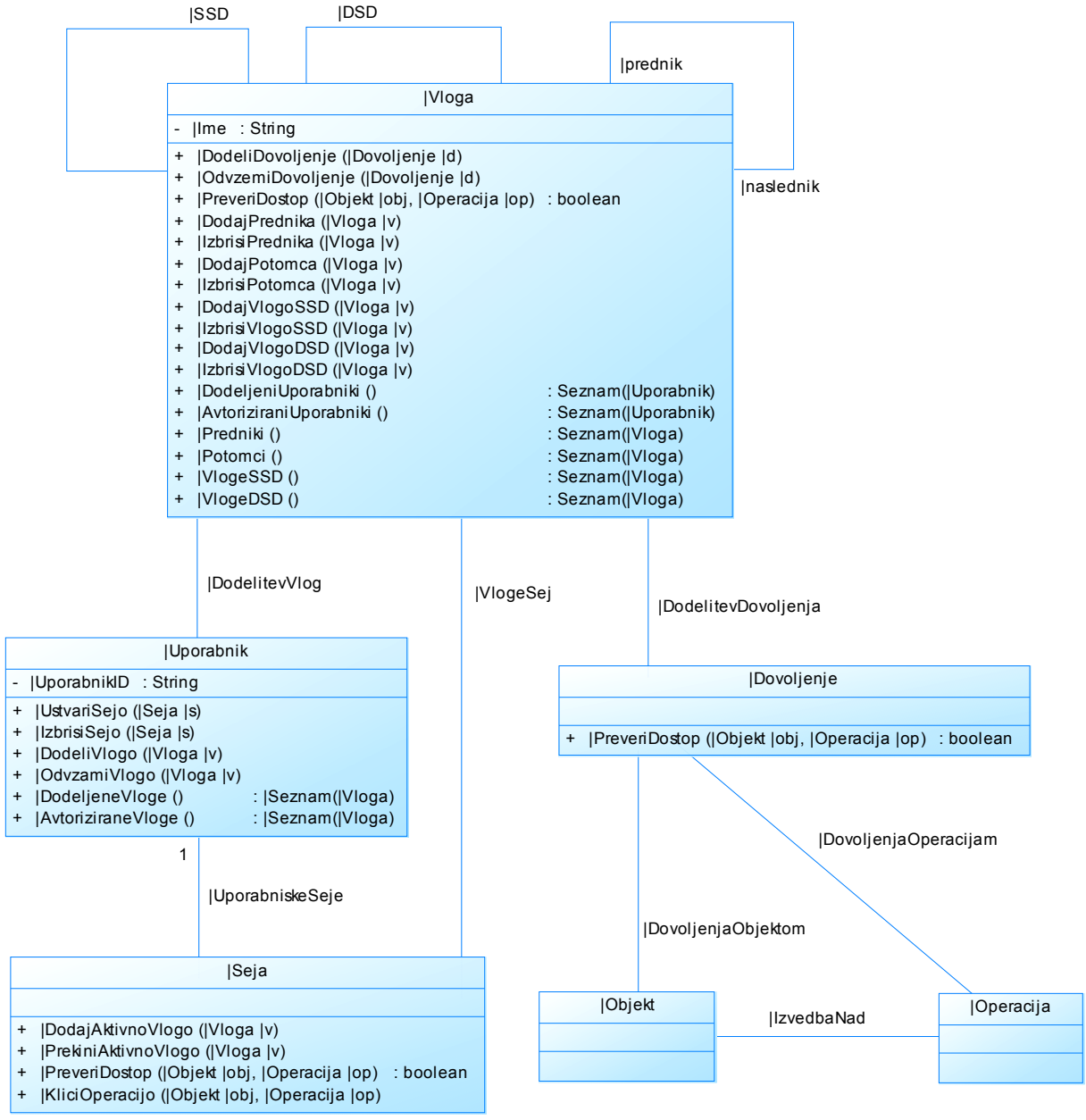
Predstavljen je s parametričnim razrednim diagramom. To pomeni, da je potrebno pri vsaki nadaljnji uporabi modela parametre nadomestiti s konkretnimi vrednostmi in tako dobimo specifičen model, prirejen za vsak posamezni načrtovani sistem. Parametri so označeni z znakom | pred samim imenom parametra. Parametriziran razred (npr. |*Vloga*) vsebuje predloge atributov (|Ime: *String*) in predloge operacij/funkcij (|*DodeliDovoljenje*). Parametrizirana relacija oz. povezava (npr. |*DodelitevVlog*) se sestoji iz parametra, ki določa ime povezave, in dveh parametrov, ki določata številnost.

Predloga razreda *Uporabnik* predstavlja uporabnika. Le-ta lahko ustvari novo sejo (*UstvariSejo*), izbriše/konča sejo (*IzbrisiSejo*), doda novo vlogo (*DodeliVlogo*) ter odstrani dodano vlogo (*OdvzamiVlogo*). Funkcija *DodeljeneVloge()* vrača seznam vlog, ki so uporabniku dodeljene direktno. Medtem ko funkcija *Avtorizirane Vloge()* vrne seznam vlog, direktno dodeljenih uporabniku, in tudi seznam tistih vlog, ki jih uporabnik podeduje od svojih direktnih vlog. Podobno kot ta razred so določeni oz. definirani tudi razredi *Vloga, Seja* in *Dovoljenje.*

Predloge povezav oz. relacij, kot sta *DodeliteVlog* in *VlogeSej,* ustvarjajo povezave med primerki oz. instancami razredov, ki jih povezujejo. Povezava *UporabniskeSeje* se ustvari ob klicu funkcije *UstvariSejo* in izbriše ob izvedbi funkcije *IzbrisiSejo.* Na podoben način se ustvarjajo in uničujejo tudi ostale povezave med razredi. Vsaka od funkcij oz. predloga funkcij je povezana s pripadajočim izrazom, definiranim z jezikom OCL. Vsebina tega izraza določa pred- in po- pogoje za izvršitev posamezne funkcije. Spodnji primer prikazuje izraz, ki

pripada predlogi funkcije *DodeliDovoljenje* in določa, da pred izvršitvijo te funkcije uporabnik nima določene pravice p, po izvršitvi pa to pravico p uporabnik pridobi.

context Role:: GrantPermission (p: Permission)
pre: self. |Permission -> excludes(|p)
post: self. |Permission -> includes(|p)



Slika 20: Predloga razrednega diagrama RBAC.

Kot je že omenjeno v poglavju 2.1., so omejitve eden glavnih elementov modela RBAC in jih lahko definiramo na dva različna načina. Prvi je vizualno, z objektnimi diagrami, drugi pa z uporabo jezika *Object Constraint Language (OCL)* v samem razrednem diagramu modela RBAC. Omejitev lahko določa, kakšna je lahko številnost na povezavi. Primer omejitve najdemo na povezavi |*UporabniskeSeje*, ki je povezana z razredom *User* in ima že vnaprej določeno številnost 1, kar pomeni, da je ena seja lahko povezana samo z enim uporabnikom.

Glede na vrsto omejitev poznamo naslednje tri: omejitve za ločitev dolžnosti (*separation of duty constrains*), predpogojne omejitve (*prerequisite constraints*) in omejitve številnosti (*cardinality constraints*). V nadaljevanju so te omejitve opisane, zraven pa so podani grafični primeri kršitev teh omejitev. Ti nam lahko služijo kot osnova pri ugotavljanju konfliktov v konkretnih primerih.

- **Omejitve za ločitev dolžnosti (Separation od Duty Constraints - SSD)**

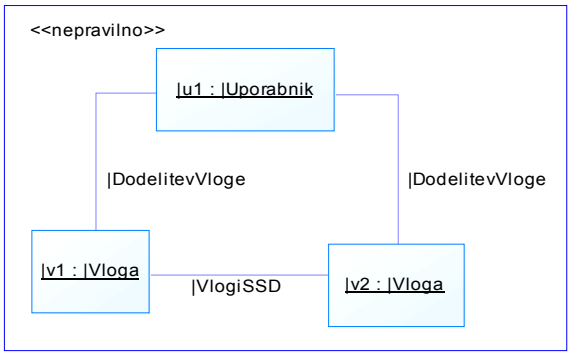
Omejitve SSD so namenjene preprečevanju konfliktov interesov, ki nastanejo, ko uporabnik pridobi dve vlogi, ki sta med seboj v konfliktu. SSD omejitve so določene za vsak par vlog, ki sta v konfliktu. Omejitev SSD postavi omejitev na metodo, ki dodeljuje vloge uporabnikom. Tako preprečimo, da bi uporabniku bila dodeljena vloga, ki je v konfliktu s katero od njegovih trenutnih vlog. Taka omejitev med pari vlog se imenuje omejitev vlog SSD (*SSD-Role constraint*) in lahko obstaja v primeru, ko ni hierarhije vlog, kakor tudi v primeru, ko hierarhija vlog obstaja. To potem seveda zakomplicira situacijo, saj je potrebno pred dodelitvijo vsake nove vloge uporabniku preveriti vse predhodno dodeljene vloge in njim podrejene oz. podedovane vloge. Nova vloga je lahko dodeljena uporabniku le, če ni v konfliktu z nobeno od teh, uporabniku že pripadajočih vlog.

Primer omejitve SSD opisane z OCL: Vlogi, ki sta v konfliktu, ne morati biti dodeljeni istemu uporabniku.

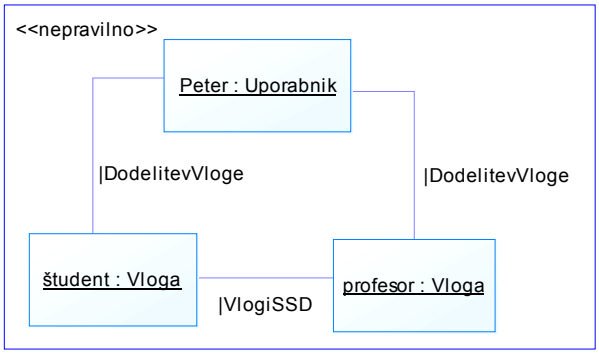
```
context |SSDRole inv:
        |r1. |User -> excludesAll(|r2. |User)
```

Trije primeri kršenja omejitev SSD:

Primer 1: Uporabniku sta dodeljeni 2 vlogi, ki sta v konfliktu. Naprimer uporabniku, ki ima vlogo študenta, ne more biti dodeljena še vloga profesorja, saj bi tako imel pravico do vpisovanja ocen samemu sebi, kar pa je proti pravilom.

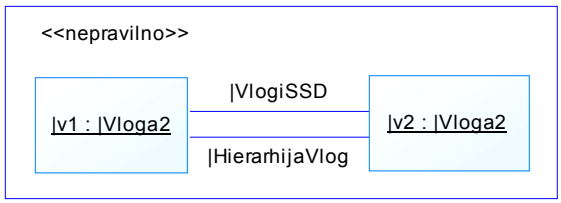


Slika 22: Primer napačne dodelitve dveh vlog, ki sta v konfliktu.



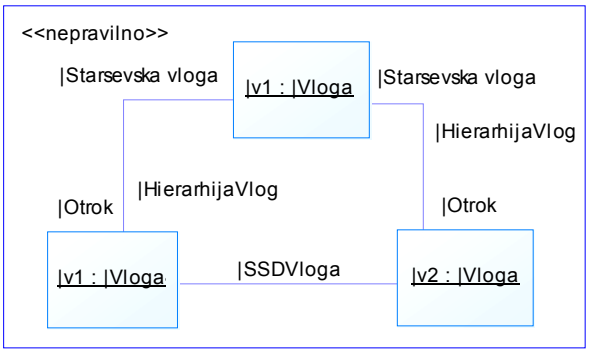
Slika 21: Konkreten primer kršitve omejitve SSD.

Primer 2: Povezava med dvema vlogama, ki sta hkrati povezani v hierarhijo in sta tudi v konfliktu. Primer takšne vloge sta vlogi profesorja in mladega raziskovalca. Po eni strani je vloga profesorja starševska vloga mladega raziskovalca. Se pravi, da profesor s prijavo v sistem lahko izvaja enake funkcije, kot jih lahko mladi raziskovalec. Hkrati pa sta ti vlogi v konfliktu v primeru, če je mladi raziskovalec tudi študent in tako bi imel profesor vpogled tudi do njegovih osebnih podatkov in ocen.



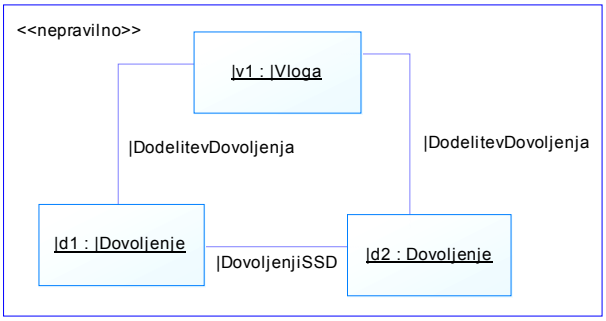
Slika 23: Primer nepravilne uporabe vlog, ki sta v konfliktu in hkrati povezani v hierarhiji.

Primer 3: Dve vlogi, ki sta v konfliktu, imata isto starševsko vlogo. Enostaven primer je povezava med vlogami referata, profesorja in študenta. Po eni strani je vloga študentskega referata starševska vloga profesorja in študenta. Se pravi, da referent s prijavo v sistem lahko izvaja enake funkcije, kot jih lahko študent in profesor . Hkrati pa sta si lahko vloga študenta in profesorja v konfliktu, saj študent ne sme imeti vseh enakih pravic v sistemu e-Študent kot profesor.

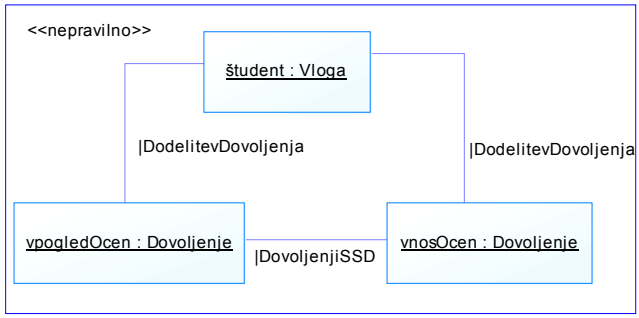


Slika 24: Primer nepravilne uporabe vlog, ki sta v konfliktu in imata isto starševsko vlogo.

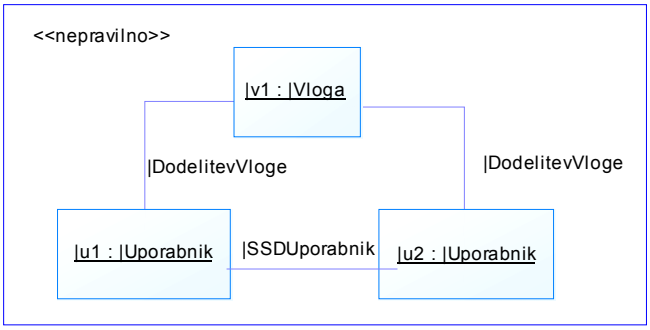
Poleg omejitev SSD (*SSD-Role constraint*) za vloge imamo tudi omejitve dovoljenj SSD (*SSD-Permission constraints*). Te omejitve se nanašajo na dovoljenja, ki so lahko dodeljena posameznim vlogam. Onemogočajo, da bi isti vlogi bili dodeljeni dve dovoljenji, ki sta med seboj v konfliktu. Študentu, ki ima pravico samo do vpogleda svojih ocen v e-Študentu, ne more biti dodeljena tudi pravica za vnos ocen v sistem. Prav tako velja isto tudi za omejitve uporabnikov SSD, ki določajo pare uporabnikov, ki so v konfliktu, tudi tem ne more biti dodeljena ista vloga. Primeri za vse tri opisane situacije so podani na naslednjih slikah.



Slika 25: Primer nepravilne dodelitve dveh dovoljenj, ki sta v konfliktu.



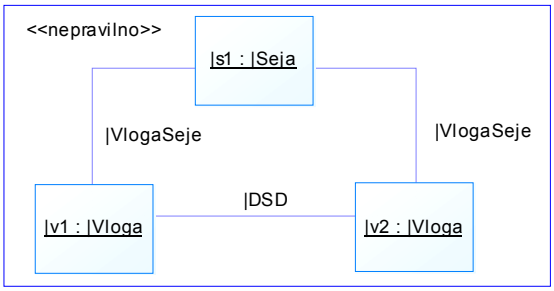
Slika 26: Konkreten primer dodelitve dovoljenj v konfliktu.



Slika 27: Primer nepravilne dodelitve iste vloge uporabnikoma, ki sta v konfliktu.

Namen omejitev **DSD** (Dynamic separation of duty) je prav tako preprečitev konfliktov interesov. Omejitve DSD se tako kot SSD nanašajo na pare vlog, vendar pa ne v povezavi z dodelitvijo vlog uporabniku, ampak glede na aktivacijo različnih vlog znotraj iste uporabniške seje. DSD se lahko nanašajo tudi konflikte med dovoljenji in uporabniki znotraj določene seje.

Primer omejitve DSD, opisane z OCL: Vlogi, ki sta v konfliktu, ne moreta biti aktivirani v isti seji.

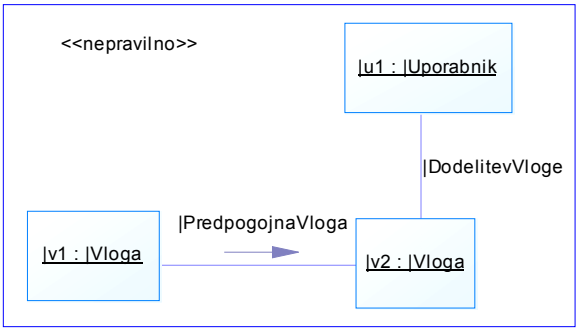


context |DSD **inv**:
|r1. |Session -> excludesAll(|r2. |Session)

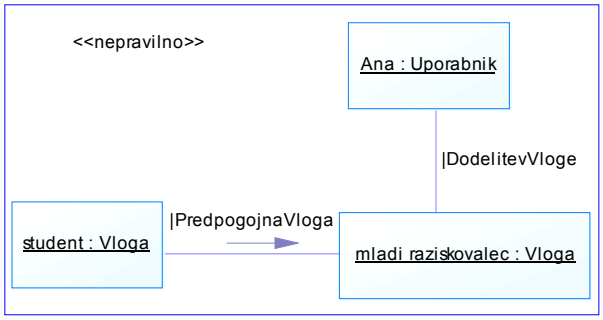
Slika 28: Prikaz kršitve omejitve DSD.

- **Predpogoji omejitev (Prerequisite Constraints)**

Zahteva predpogojev omejitev je, da je uporabniku lahko dodeljena neka vloga le, če mu je dodeljena tudi neka druga, predpogojna vloga. Predpogoj za dodelitev vloge mladega raziskovalca je, da ima uporabnik že dodeljeno vlogo študenta na tej fakulteti.

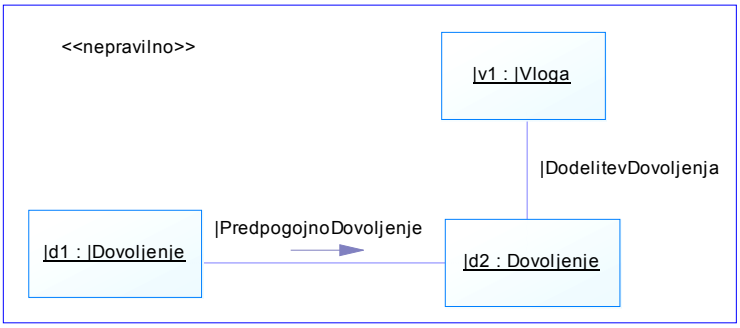


Slika 29: Primer kršitve predpogojnih omejitev.



Slika 30: Konkretni primer kršitve predpogojne omejitve.

Enako pravilo oz. omejitev se lahko nanaša tudi na dovoljenja. Profesorju je potrebno, preden mu dodelimo dovoljenje za vnos ocen v e-Študent, dodeliti še pravico do vpogleda na seznam študentov, ki poslušajo njegov predmet. Komaj takrat lahko profesor pridobi tudi pravico do vpisovanja in spreminjanja ocen izbranim študentom. Grafični prikaz je podan na spodnji sliki (Slika 31).

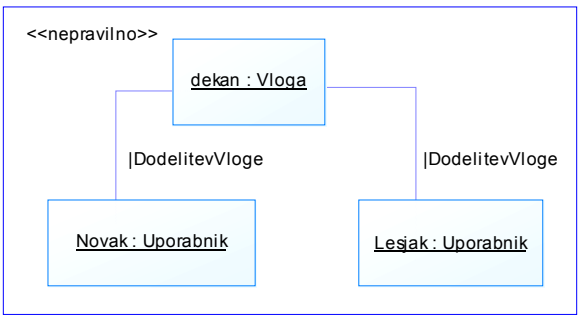


Slika 31: Primer kršitve predpogojnih omejitev za dovoljenja.

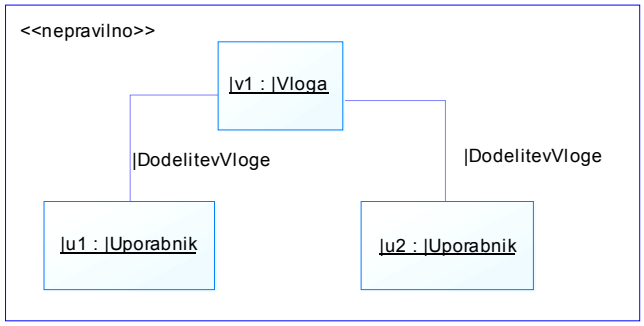
- **Omejitev številnosti (Cardinality Constraints)**

Še ena oblika omejitev so omejitve številnosti. Te omejitve lahko uporabimo za omejitev npr. števila uporabnikov, ki jim je lahko dodeljena neka vloga, ali števila vlog, ki jih lahko uporablja nek uporabnik, ali števila sej, ki jih uporabnik lahko aktivira hkrati. Te omejitve številnosti se dodajajo na povezave med elementi.

Primer: Napaka v shemi, ker je dvema uporabnikoma dodeljena ista vloga, ki pa glede na omejitve številnosti lahko pripada največ enemu uporabniku (Slika 32). Konkreten primer take vloge je vloga dekana na fakulteti (Slika 33). Ne moreta imeti dva uporabnika hkrati dodeljene te vloge, saj je na fakulteti lahko v določenem trenutku le en aktiven dekan.



Slika 32: Primer napačne uporabe omejitev številnosti.



Slika 33: Konkreten primer napačne uporabe omejitev številnosti.

Odkrivanje konfliktov

Če imamo nek objektni diagram, ki opisuje delovanje sistema, in v njem najdemo katerega izmed zgornjih diagramov omejitev, potem smo lahko prepričani, da v sistemu obstaja konfliktna situacija. Iskanje prisotnosti vzorca je v bistvu iskanje pod-grafa v objektnem diagramu.

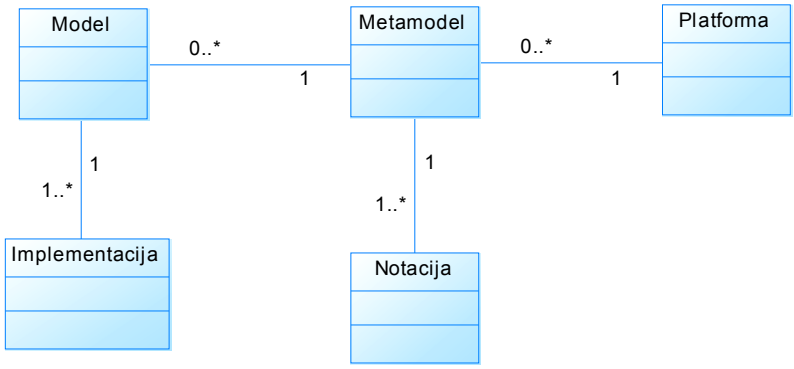
3.5. Secure UML

V članku [3] avtorji predstavijo metodologijo za izdelavo logike kontrole dostopa in njeno vpeljavo v proces modelno usmerjenega razvoja programske opreme (model-driven software development). Tehnika, ki jo uporabijo, temelji na načrtovalskem jeziku *SecureUML*. Namenjen je vključitvi informacij glede kontrole dostopa v aplikacijske načrte, narejene z grafičnim jezikom UML. Vpeljava varnosti v sam model sistema je predstavljena na visokem nivoju abstrakcije.

Modelno usmerjen razvoj programske opreme

Razlika med prejšnjim in tem pristopom je, da v tem primeru avtorji poudarjajo modelno usmerjen razvoj programske opreme. S pomočjo modelov namreč lahko dvignemo stopnjo abstrakcije našega dela. Tako se razvijalcu pri načrtovanju neke aplikacije ni potrebno že na začetku ukvarjati s tehnološkimi in implementacijskimi podrobnostmi, ampak se lahko osredotoči le na pomembnejši del, na samo arhitekturo aplikacije. Tako z modeli najprej specificiramo, kaj naj aplikacija dela, vprašanje, kako, pa prepustimo posameznim programerjem oz. avtomatiziranemu postopku generiranja kode. Modelno usmerjen razvoj programske opreme tako omogoča bistveno povečanje produktivnosti pri samem razvoju. Ker modeliranje poteka na visokem nivoju abstrakcije, omogoči skrivanje kompleksnosti današnjih programskih jezikov in platform. Ko imamo model narejen, s pomočjo avtomatiziranih preslikav oz. generatorjev kode, model preslikamo v želeno programsko kodo. Tako se izognemo ročnemu kodiranju in s tem povečamo kvaliteto programske kode, saj se izognemo napakam zaradi površnosti in nezbranosti programerjev. Poveča se produktivnost razvojnega procesa kot tudi kvaliteta končnega izdelka. Ta način je velik korak proti platformno neodvisnemu razvoju sistemov.

Skupina strokovnjakov *Object Management Group* se ukvarja z razvojem standarda za modelno usmerjen razvoj programske opreme, ki se imenuje *Model Driven Architecture* oz. modelno usmerjena arhitektura. Osnovni model tega standarda je predstavljen na spodnji sliki (Slika 34).



Slika 34: Referenčni model za modelno usmerjen razvoj programske opreme.

Model predstavlja abstrakten pogled na informacijski sistem. Iz modela lahko generiramo eno ali več instanc modela, kar predstavlja razred *Implementacija*. *Metamodel* definira zgradbo razredov modela in se nanaša na posamezno *platformo*. Platforma je izvedbeno okolje informacijskega sistema, kot na primer Java platforma. Vsak metamodel ima eno ali več *notacij*. Notacija predstavlja zgoščen format s katerim predstavimo posamezen model, kot instanco metamodela. Notacija je lahko tekstovna ali grafična (npr. UML Profile).

Potek procesa s pomočjo modelno usmerjene arhitekture poteka tako, da najprej izdelamo glede na platformo neodvisen model, ponavadi z notacijo UML. Tega nato s pomočjo avtomatiziranih transformacij pretvorimo v specifične modele, glede na platformo. Iz teh se nato v končni fazi generira izvorna koda za platforme kot so Java, MS .NET in podobno.

Vpeljava varnostnih mehanizmov v modelno usmerjen razvoj

Vpeljava varnosti v modelno usmerjen razvoj programske opreme ima več prednosti. Kot prvo je dejstvo, da so lahko varnostne zahteve izoblikovane in dodane oz. vpeljane v sistem na visokem nivoju abstrakcije. Na tak način lahko dosežemo razvoj res varnih aplikacij, ki so od osnove zgrajene z namenom preprečitve kršitev varnostnih pravil. Sama poizvedba v podatkovno bazo je lahko že v osnovi definirana tako, da dovoljuje uporabniku dostop le do tistih podatkov, do katerih ima pravico dostopati. Še ena prednost modelnega razvoja je lažji način detekcije in korekcije načrtovalskih napak, tudi tistih, povezanih z varnostjo.

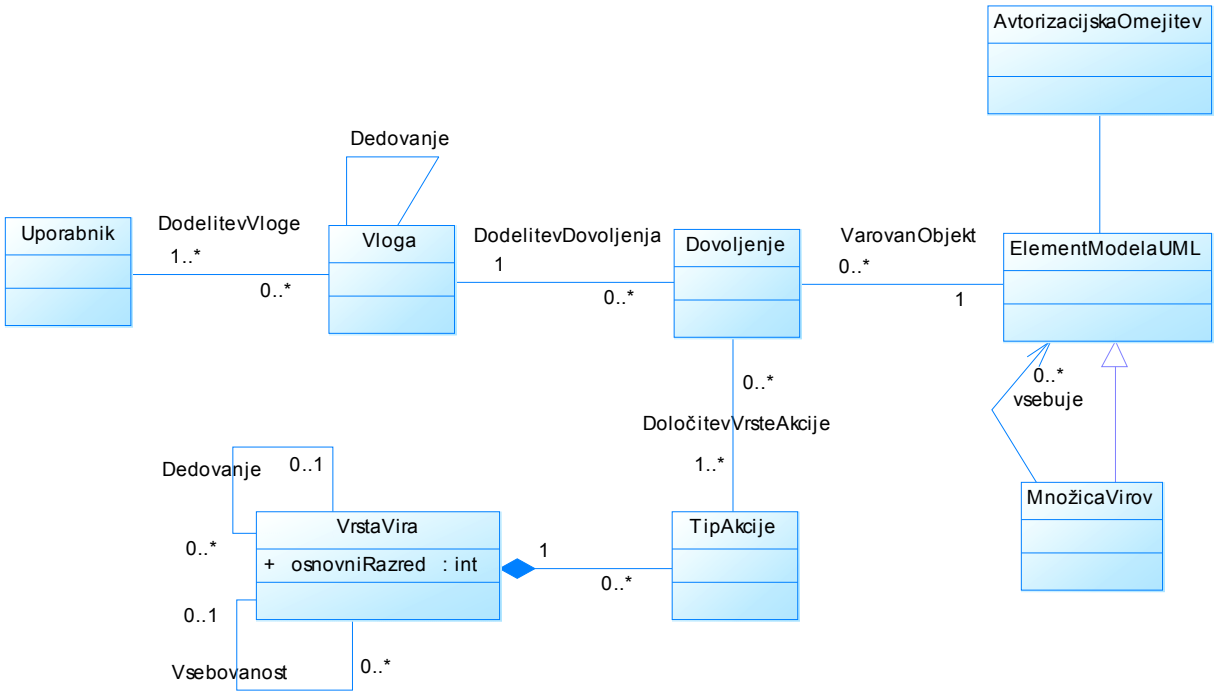
SecureUML je načrtovalski jezik, ki definira slovar za dopolnitev modelov UML s podatki glede kontrole dostopa, kot so vloge, dovoljenja in dodelitev vlog uporabnikom. SecureUML temelji na modelu RBAC. Model RBAC je dobro vpeljan model kontrole dostopa. Vseeno pa ima tudi določene pomanjkljivosti, kot na primer podpora za izražanje raznih pogojev kontrole dostopa, ki se nanašajo na trenutno stanje sistema (npr. vrednosti parametrov, datum ali čas …). Za še boljšo zaščito v takšnih primerih avtorji vpeljejo nov koncept *avtorizacijskih omejitev (authorization constraints)*. Avtorizacijska omejitev predstavlja nek predpogoj za pridobitev dostopa določeni operaciji oz. funkciji sistema. Za definiranje in opis omejitev

uporabimo že omenjeni logični jezik OCL (Object Constraint Language), ki je del načrtovalskega jezika UML. Na tak način lahko koristno uporabimo obstoječa orodja in koncepte za definiranje in analizo omejitev v zvezi s kontrolo dostopa. Ravno to pa je tudi glavni namen avtorjev SecureUML. SecureUML naj bi se namreč uporabljal kot dodatek drugim načrtovalskim jezikom, in sicer za področje kontrole dostopa. Tako bi različni modeli, ne glede na nivo abstrakcije, imeli enako zgradbo kontrole dostopa.

Če sedaj pogledamo na vse lastnosti načrtovalskega jezika SecureUML, lahko vidimo, da nam omogoča zelo fleksibilno načrtovanje varnosti, saj združuje enostavno grafično predstavitev kontrole dostopa z UML modelom, z dodatno močjo logičnih omejitev v modelu. Enostavnejša pravila tako lahko določimo samo z dovoljenji, vezanimi na posamezno vlogo. Bolj kompliciranе zahteve pa specificiramo z avtorizacijskimi omejitvami.

Secure UML metamodel, ki je prikazan na naslednji strani (Slika 35), je definiran kot razširitev metamodela UML.

Koncept kontrole dostopa RBAC je predstavljen z dodatnimi element metamodela, in sicer s podatkovnimi tipi ***Uporabnik***, ***Vloga*** in ***Dovoljenje*** ter relacijami med njimi. Elementi UML modela, ki jih je potrebno v sistemu zaščititi, so predstavljeni na poseben način. Namesto da bi definirali nov, poseben tip v metamodelu, vsakemu elementu UML modela, ki ga je potrebno zaščititi, dodelimo vlogo zaščitenega vira. ***MnožicaVirov*** vsebuje množico elementov, ki jih uporabimo pri definiranju pravil in avtorizacijskih omejitev.



Slika 35: Metamodel SecureUML.

Dovoljenje je povezovalni element, ki povezuje posamezno vlogo z elementom modela UML ali množico virov. Vsebina dovoljenja je določena z elementom ***Tip akcije***. S tipom akcije so

določeni razredi, ki vsebujejo varnostne operacije, povezane s posameznim dovoljenjem. Primer varnostne operacije je metoda *izvedi* ali atribut z metodami *spremeni, beri itd.* SecureUML ima za vsako tako operacijo definiran ustrezen *tip akcije*. Posamezni *tip akcije* leko predstavlja tudi bolj kompleksen razred večih operacij, predstavljenih na višjem abstraktnem nivoju.

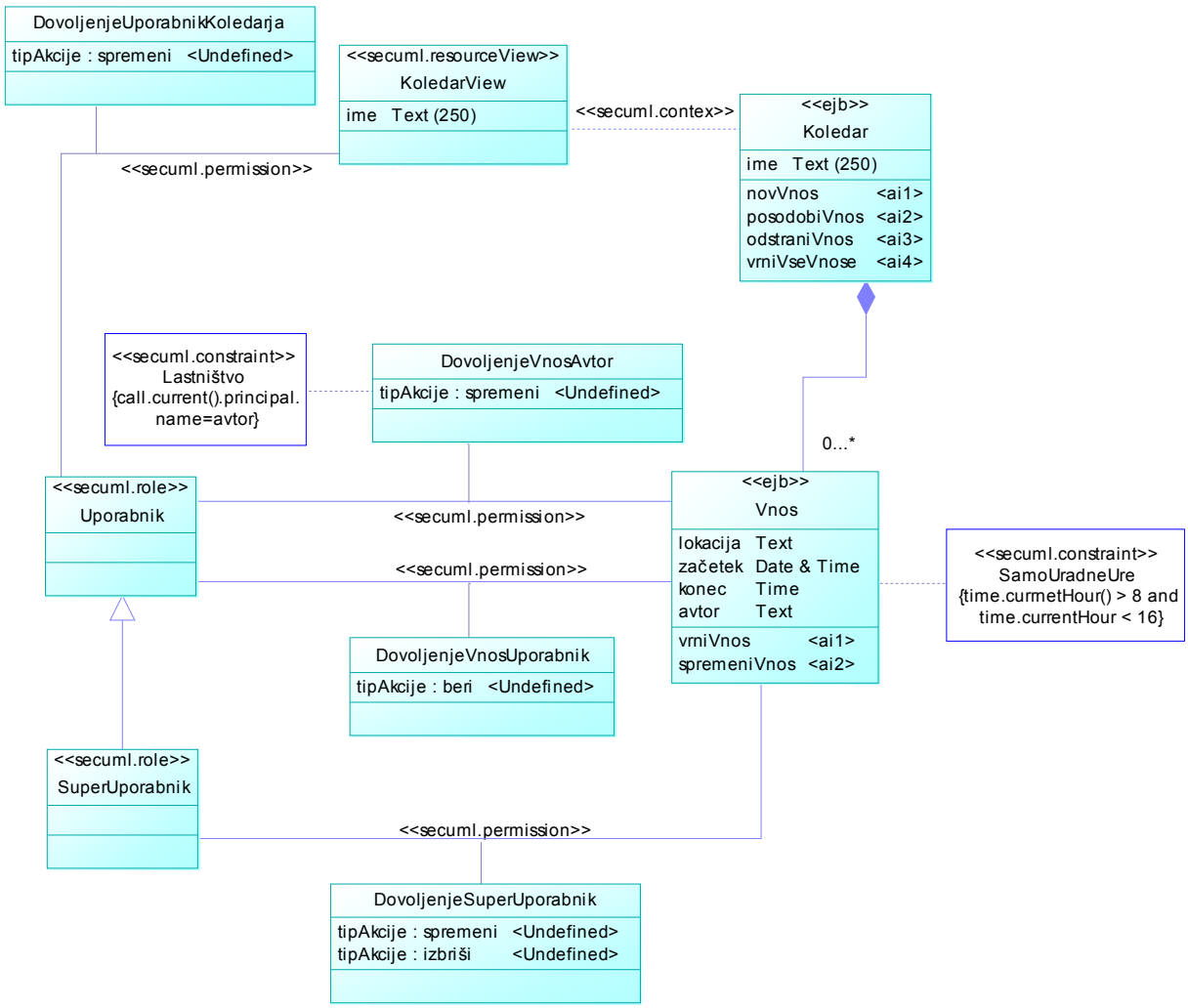
VrstaVira definira seznam oz. množico elementov *tip akcije*, ki so na razpolago v posameznem tipu metamodela. Povezava med vrsto vira in tipom metamodela, kateremu *tipi akcije* pripadajo, poteka preko atributa *osnovniRazred* v elementu *vrsta vira.*

Avtorizacijska omejitev je del kontrole dostopa do aplikacije. Je neke vrste predpogoj, ki mora biti izpolnjen pri vsakem klicu določenega elementa modela. Najpogosteje se nanaša na stanje elementa v trenutku klica in na okoliščine, v katerih pride do klica. Avtorizacijska omejitev je lahko na element modela povezana posredno – neposredno preko dovoljenja. Konkreten primer za uporabo take omejitve je v situaciji, če imamo na primer v e-Študentu omejitve, da lahko profesor vnaša nove izpitne roke samo v času delavnika, se pravi od 8h do 16h. Takrat uporabimo avtorizacijsko omejitev nad razredom oz. elementom tipa »VnosIzpitnegaRoka«. Bolj podrobno je konkretna omejitev opisana v nadaljevanju.

Primer

V opisu primera, ki sledi, uporabimo arhitekturo Enterprise JavaBeans (EJB), ki se v praksi pogosto uporablja, saj ima močno podporo za vpeljavo varnosti v razvoj programske opreme. Osnovni model kontrole dostopa v EJB je RBAC. Vendar pa če se varnosti lotimo šele pri implementaciji sistema, mora razvijalec ročno definirati vsa pravila in dovoljenja za posamezne metode. To pa je precej podrobno in dolgotrajno delo. Ravno zaradi tega je modeliranje varnostnih politik na višjem abstraktnem nivoju in avtomatično generiranje kode obetajoča in optimistična rešitev tega problema. Tako lahko vključimo kontrolo dostopa že v poslovni model in si s tem zelo olajšamo delo. V konkretnem primeru smo razrede, ki so komponente EJB, označili s stereotipom <<ejb>>. Razred s tem stereotipom vsebuje, kot vsi ostali razredi, svoje metode in atribute. Nekatere metode morajo biti definirane v osnovi (npr. iskanje), druge pa lahko dodamo po želji, glede na naš sistem. Vedno je potrebno definirati tudi tipe vira in pripadajoče tipe akcij.

Na spodnji sliki (Slika 36) je prikazan model za aplikacijo koledarja v e-Študentu skupaj z mehanizmom za kontrolo dostopa. Poleg osnovnih razredov aplikacije *koledar in vnos* so dodani še elementi SecureUML. Prepoznamo jih po imenu stereotipa (v dvojnih trikotnih oklepajih), ki vsebuje niz »secuml«. Koledar lahko vsebuje več vnosov, pri čemer vsak vnos predstavlja izpitni rok za nek predmet z določenim datumom, uro in prostorom izpita. Uporabnik, ki je vnesel izpitni rok, je v razredu *vnos* shranjen v atributu *avtor.* Ostali sestavni deli metamodela so namenjeni vzpostavitni mehanizma za kontrolo dostopa. Elementi SecureUML-ja so opisani v nadaljevanju.



Slika 36: Primer: varna aplikacija koledar.

Vloga – Predstavljena je z razredom s stereotipom <<secuml.role>>. Pri projekciji diagrama v komponente EJB-ja se vsaka vloga preslika glede na descriptor, določen v EJB. V zgornjem primeru obstajata 2 vlogi, in sicer Uporabnik in SuperUporabnik. Med sabo sta povezani z generalizacijo (dedovanje). Ta relacija se pojavi ob uporabi Role Hierarchy. SuperUporabnik je izpeljana vloga iz vloge Uporabnik.

Dovoljenje - Prikazano je kot povezovalni razred s stereotipom <<secuml.permission>>. To, da je povezovalni razred, pomeni, da se nanaša na povezavo med dvema razredoma modela. V najosnovnejšem primeru je dovoljenje povezano z enim od varovanih elementov. *Tip akcije,* ki določa vrsto dovoljenja, je določen kot tip atributa v razredu dovoljenja. Število akcij v enem dovoljenju ni omejeno. Mora pa tip akcije ustrezati elementu, na katerega se dovoljenje nanaša. V našem primeru imamo dovoljenje *DovoljenjeVnosUporabnik*, ki dovoljuje uporabniku dostop do elementa *Vnos* z metodo *beri* (in tudi z vsemi ostalimi metodami v EJB-ju, ki spadajo v skupino bralnih metod – ni nujno, da je to le ena metoda).

Ker je sintaktično nemogoče nasloviti samo posamezne atribute ali metode, uvedemo za klic posameznih atributov in metod nov stereotip <<secuml.resourceView>>. To je v bistvu realizacija tipa *množicaVirov* iz metamodela. Uporablja se kot neko ciljno dovoljenje in tudi

avtorizacijska omejitev. Povezava med resourceView-om in entiteto, na katero se nanaša, je označena s stereotipom **<<**secuml.context>>. Za vsak atribut oz. metodo, vsebovano v »resourceView«, mora obstajati atribut oz. metoda z enakim imenom v razredu, na katerega se »resourceView« nanaša. Za lažje razumevanje naj ponazorimo s primerom iz aplikacije študijski koledar. Ta vsebuje en »resourceView«, in sicer *KoledarView,* ki se sklicuje na en sam atribut *ime* entitete *Koledar*. Dovoljenje *DovoljenjeUporabnikKoledarja* dovoljuje uporabniku uporabo operacije *spremeni* nad atributom *ime*. To pa je mogoče ravno zato, ker je edino atribut name vsebovan v elementu *KoledarView*.

Avtorizacijska omejitev – Označimo jo s stereotipom <<secuml.constraint>>**.** Predvideva se, da se v večini primerov ena avtorizacijska omejitev nanaša na več ali celo vse metode nekega razreda v modelu. Omejitev je povezana z razredom preko UML povezave (črtkana črta). Z avtorizacijsko omejitvijo omejimo dostop do metod oz. jih zavarujemo. Za vsako metodo moramo definirati omejitev oz. predpogoj za njeno izvršitev in jo zapisati z jezikom OCL. Avtorizacijske omejitve imajo dostop ne samo do aplikacijskega modela, ampak tudi do same platforme sistema. S tem lahko pridemo do sistemskih informacij v času delovanja aplikacije, kot so naprimer trenutni čas in datum pa tudi vloga uporabnika in podobno. Ti atributi, ki so na voljo, so definirani v osnovnem modelu vsakega aplikacijskega modela.

Postavitev avtorizacijske omejitve je lahko direktna povezava z razredom ali »resourceView« preko UML povezave. Tako postavljena omejitev predstavlja predpogoj za izvedbo metod v tem omenjenem razredu.

Primer: Omejitev na razredu *Vnos*, do katerega lahko dostopamo samo v času uradnih ur. V sami omejitvi <<secuml.constraint>> SamoUradneUre zgleda pogoj tako:

```
time.currentHour() > 8 and time.currentHour() < 17
```

Nato pa je preveden v OCL ob klicu vsake metode, na katero se nanaša, na naslednji način:

```
context Entry::getEntryInfo():EntryInfo
pre: time.currentHour() > 8 and time.currentHour() < 17
```

Avtorizacijska omejitev se lahko nanaša tudi na kateri drugi element modela, ne samo na razred. Z njo lahko naprimer postavimo dodatne omejitve dovoljenju. V našem konkretnem primeru je taka omejitev dodana dovoljenju *DovoljenjeVnosAvtor*, ki omogoča posodobitev oz. spreminjanje vnosa v koledarju. Želimo, da bo pravico do spreminjanja vnosa izpitnega roka imel le profesor, ki je izpitni rok objavil. Omejitev se imenuje Lastništvo in poteka po principu, da pridobi sistem ime trenutnega uporabnika direktno s klicanjem metod platforme. Nato vrnjeno uporabniško ime primerja z atributom avtor razreda Vnos.

```
call.current().principal.name = owner
```

OCL stavek oz. izjava celotnega dovoljenja in avtorizacijske omejitve zgleda tako:

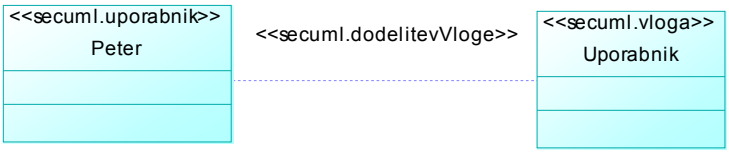
```
context Entry::setLocation(newValue:String):void
pre: call.current().principal.isInRole("User") and
call.current().principal.name = owner
```

V primeru, da imamo nad nekim elementov definiranih več dovoljenj in omejitev, dobimo predikat, ki je sestavljen iz vseh pogojev. Vrstni red upoštevanja omejitev in dovoljenj je naslednji. Najprej disjunktivno preverimo, če je izpolnjeno katero od dovoljenj (dovoljenje$_1$ ali dovoljenje$_2$ ali ... ali dovoljenje$_n$), nato pa še konjunktivno preverimo, ali so izpolnjene vse omejitve (omejitev$_1$ in omejitev$_2$ in ... in omejitev$_n$). Končni rezultat je predikat z vsemi omejitvami, ki se nanašajo na neko metodo. Kot primer lahko omenimo metodo elementa Vnos, *spremeniVnos*, na katero se nanašata dovoljenji *DovoljenjeVnosAvtor* in *DovoljenjeVnosUporabnik* ter avtorizacijska omejitev *SamoUradneUre*.

```
context Entry::setEntryInfo(EntryInfo):void
pre:                                    (call.current().principal.isInRole("SuperUser")
or(call.current().principal.isInRole("User") and

call.current().principal.name = owner) and
(time.currentHour() > 8 and time.currentHour() < 17)
```

Uporabnik in dodelitev vloge - Uporabnik je predstavljen z razredom s stereotipom <<secuml.user>>, vloga pa s stereotipom <<secuml.role>>. Dodelitev vloge uporabniku je prikazana s povezavo med vlogo in uporabnikom, s stereotipom <<secuml.roleAssignment>>. Primer je na spodnji sliki (Slika 37).



Slika 37: Primer: dodelitev vloge uporabniku s SecureUML.

Opisane koncepte načrtovalskega jezika SecureUML lahko preizkusimo tudi v praksi. Obstaja namreč prototip za modeliranje kontrole dostopa in tudi za samo generiranje kode. Avtor orodja je podjetje Interactive Objects, orodje pa se imenuje ArcStyler™ [12] in nudi okolje za modelno usmerjen razvoj programske opreme. Z njim je mogoče generirati aplikacije EJB s celotno logiko za kontrolo dostopa, ki temelji na modelu RBAC. Torej vsebuje definicije vlog, dovoljenja metod, določitev vlog uporabnikom ter avtorizacijske omejitve. S pomočjo tega orodja je bilo izdelanih že več aplikacij za banke, podobnih, kot je opisana v tem članku. Opravljeni eksperimenti so pokazali zmožnost natačnega in jedernatega oblikovanja kontrole dostopa z načrtovalskim jezikom UML in jezikom za definiranje omejitev OCL.

3.6. Stopnjevanje varnostni v UML modelih

Predlog, podan v članku [6], se nanaša na pristop modeliranja varnosti oz. kontrole dostopa s pomočjo razširitve osnovnega UMLmodela, z ločenimi, dodatnimi diagrami. Ti diagrami podajajo vizualen pogled na kontrolo dostopa. Vsak od njih vsebuje množico varnostnih značilnosti, ki povečajo varnost modelov, izdelanih z načrtovalskim jezikom UML.

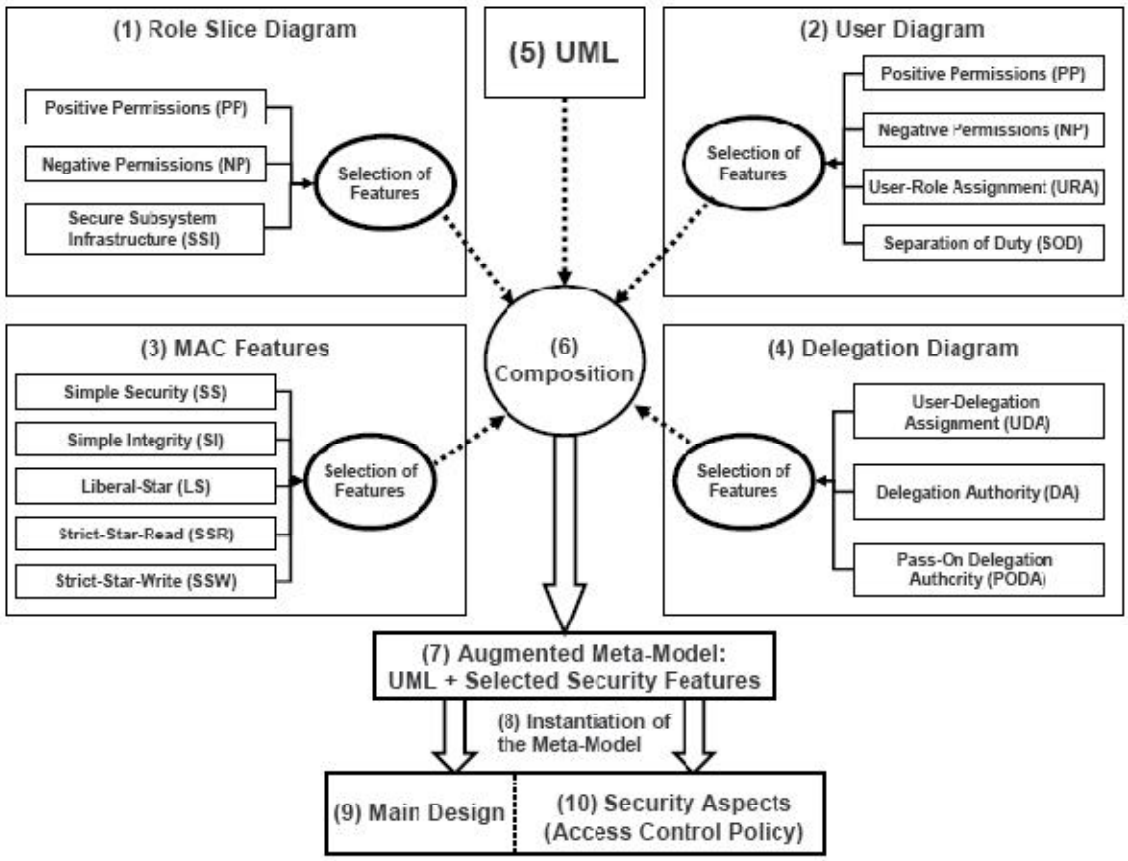
Če želimo analizirati varnost modelov v stopnji načrtovanja, moramo razumeti koncepte varnostnih shem kontrole dostopa. Trije najbolj pogosti modeli, opisani v poglavju 2.1., so:

- obvezna kontrola dostopa (*mandatory access control - MAC),*
- neomejena kontrola dostopa *(discretionary access control - DAC) in*

- kontrola dostopa z vlogami *(role-based access control - RBAC)*.

Te tri varnostne sheme, MAC, DAC in RBAC, specificirajo osnovno idejo za kontrolo dostopa. V nadaljevanju jih bomo s pomočjo vizualnega jezika UML združili v en model za obvladovanje varnosti. Vendar pa ni nujno, da se model uporabi vedno v celoti. Model je fleksibilen, zato lahko razvijalec sistema vedno vzame le tiste komponente, ki jih sistem potrebuje in bodo najbolj povečale varnost. Ker je tak model kontrole dostopa enoten, omogoča lažji pregled nad varnostnimi omejitvami.

Jedro pristopa je razširitev UML-ja z varnostnega vidika. Predlagana razširitev vsebuje dva elementa: ***varnostne komponente*** in ***varnostne diagrame***. Varnostne komponente ustrezajo specifičnim elementom kontrole dostopa, kot so dovoljenja, pooblastila itd. Varnostni diagrami pa določajo notacijo za upodobitev varnostnih komponent kot ločenega dela od celotnega modela aplikacije.



Slika 38: Pregled predlaganih varnostnih razširitev.

Na zgornji sliki (Slika 38) je predstavljen pregled predlaganih varnostnih razširitev UML-ja. *Role Slice Diagram(1)* je vizualna notacija za RBAC (vloge, pozitivna in negativna dovoljenja ter hierarhija vlog). *User diagram(2)* opisuje uporabnika, pozitivna in negativna dovoljenja (vlog uporabnikov), povezave z vlogami in omejitve glede dodelitve vlog. *Delegation Diagram(4)* je notacija za pooblastila, ki jih dodelimo uporabniku. *MAC Features(3)* predstavljajo ogrodje za 3 diagrame, ki opisujejo pravila modela MAC. Vsaka izmed opisanih štirih razširitev je povezana z množico varnostnih komponent, ki jo

sestavljajo. Razvijalec izbere podmnožico komponent, ki jih bo uporabil glede na problem, s katerim se sooča. Nato le-te združi v enoten model in poveže z osnovnim UML metamodelom aplikacije.

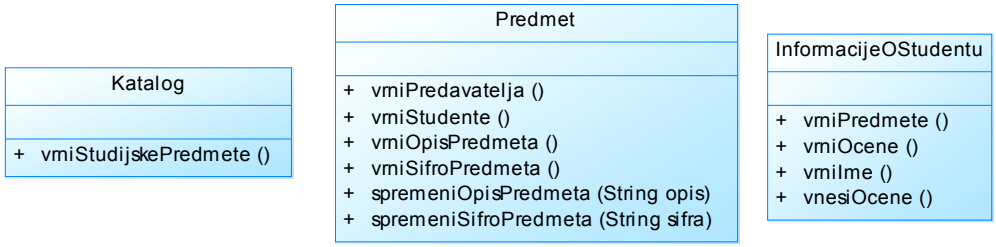
Da bi lahko definirali varnostne mehanizme v UML modelu in podrobno določili dovoljenja glede varnosti, uporabimo že omenjene *varnostne komponente*. Te predstavljajo zgradbo in vsebino varnostnih elementov. Vse možne varnostne komponente so naštete na zgornji sliki, in sicer so to:

- Secure Subsystem Infrastructure (SSI): predstavlja operacije v sistemu, ki zahtevajo kontrolo dostopa.
- User-Role Assignment (URA): predstavlja uporabnike, vloge in njihove povezave.
- Positive Permissions (PP) in Negative Permissions (NP): določajo, kaj uporabnik lahko počne v sistemu in česar ne sme.
- Separation of Duty (SOD): omejuje dodeljevanje vlog uporabnikom s tem, ko definira, katere vloge ne morejo biti hkrati dodeljene istemu uporabniku.
- User Delegation Assignment (UDA), Delegation Authority (DA) in Pass-On Delegation Authority: določajo razna pooblastila.

Ostale komponente skrbijo za osnovne varnostne elemente, kot so določanje stopnje tajnosti, pravica do branja in pisanja podatkov in druge. Razlog za razdelitev varnostnih politik na manjše enote je v lažjem razumevanju le-teh za razvijalce. Druga prednost teh vsebinsko in strukturno razdeljenih enot je lažje generiranje kode, saj so v kodo spremenjene le komponente, ki jih izbere razvijalec sistema. Nenazadnje vsaka varnostna komponenta realizira eno od varnostnih zahtev. Tako razvijalec z implementacijo vsake posamezne komponente stopnjuje varnost v sistemu, če pride do kakršnekoli spremebe, pa to ne vpliva na ostale varnostne komponente.

Primer: Aplikacija univerze

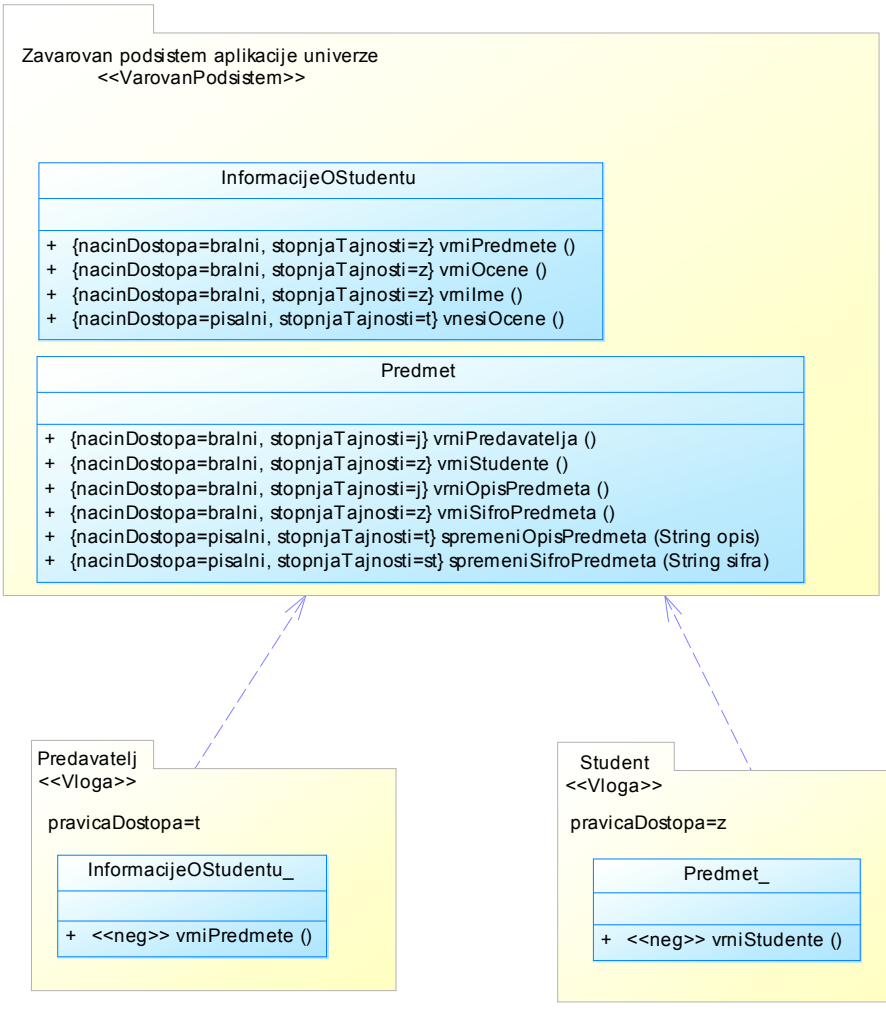
Aplikacija univerze upravlja podatke o predmetih, študentih, predavateljih in katalogu javnih informacij. Varnostne zahteve so naslednje: vsak predavatelj ima določeno število predmetov, za katere lahko piše in nato tudi pregleda učni načrt. Prav tako lahko predavatelj vidi kodo za vsak svoj predmet. Predavatelj lahko vidi seznam študentov, ki so prijavljeni na njegov predmet ter jim vpisuje ocene. Ne more pa videti seznama ostalih predmetov, ki jih obiskuje posamezni študent. Študent lahko vidi seznam predmetov, ki jih obiskuje ter predavatelje, ki mu predavajo. Poleg tega lahko pregleda tudi svoje ocene, učne načrte predmetov in kode premetov. Ne more pa videti seznama študentov, ki obiskujejo nek predmet. Dostop do kataloga javnih informacij imajo vsi.



Slika 39: Osnovni razredni diagram aplikacije univerze.

Za izdelavo varnostnih politik, mora razvijalec določiti tri ključne komponente: subjekte, objekte in dovoljenja. *Subjekti* so entitete, ki zahtevajo dostop do sistema. Sistem vsebuje množico objektov, to so entitete, ki potrebujejo zaščito pred subjekti. *Objekti* so lahko tudi metode razredov, ki morajo biti zaščitene. *Dovoljenja* določajo, do katerih objektov oz. operacij lahko dostopa posamezni subjekt v sistemu.

Če želimo zadostiti varnostnim zahtevam aplikacije univerze, moramo izbrati strukturo kontrole dostopa, ki predstavlja vse tri zgoraj omenjene komponente – subjekte, objekte in dovoljenja. Aplikacija univerze ima dve vrsti uporabnikov, predavatelje in študente, pri čemer ima vsaka skupina svoja dovoljenja. Za lažje grupiranje uporabnikov je najboljša alternativa uporaba kontrole dostopa na podlagi vlog. Določitev dovoljenj posamezni vlogi lahko poteka na dva načina: prvi je z dodelitvijo pozitivnih dovoljenj vlogi, drugi pa z obvezno kontrolo dostopa (MAC). V tem primeru uporabimo pravila obvezne kontrole dostopa. To pomeni, da vsakemu uporabniku dodelimo stopnjo pravice dostopa do tajnih podatkov, objektom oz. operacijam pa določimo stopnjo tajnosti. Tako omejimo, da lahko subjekt dostopa le do operacij, ki imajo stopnjo tajnosti manjšo ali enako njegovi stopnji za dostop do tajnih podatkov. V primeru, da obstajajo objekti, do katerih ima uporabnik glede na svojo vlogo pravico dostopati, a mu dostop ne bi smel biti dovoljen, lahko uporabimo negativna dovoljenja. Z negativnimi dovoljenji lahko uporabniku eksplicitno odvzamemo določene pravice.

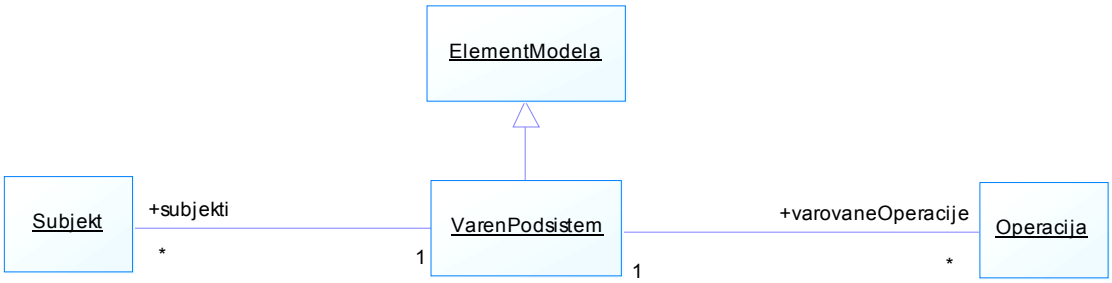


Slika 40: Diagram vlog v aplikaciji univerze.

Na zgornji sliki (Slika 40) je prikazan diagram vlog, ki mu je dodan MAC model. Paket, označen s stereotipom <<*SecureSubsystem*>>, označuje podsistem, ki vsebuje vse operacije v sistemu, ki zahtevajo kontrolo dostopa. Zavarovani podsistem definira tudi stopnjo tajnosti za vsako izmed vsebovanih operacij (javno (j), zaupno(z), tajno(t), strogo tajno(st)) ter način dostopa (bralni ali pisalni). Vlogi predavatelj in študent sta ponazorjeni s paketoma s stereotipom <<Vloga>>. Vlogi imata določeno stopnjo pravic dostopa do tajnih podatkov in negativna pravila. Povezavi med vlogama in varnim podsistemom pomenita, da lahko obe vlogi dostopata do teh operacij, seveda če stopnja tajnosti to omogoča.

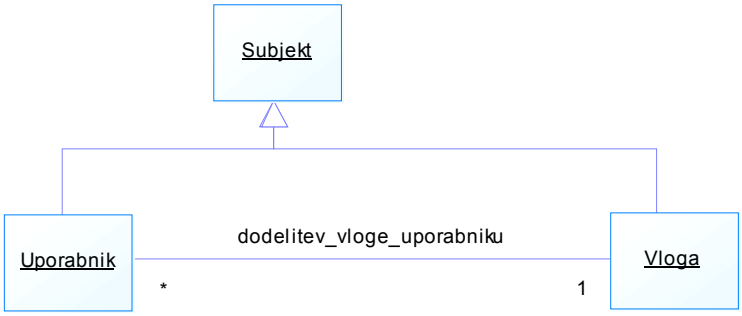
Za izdelavo varne aplikacije univerze so osnovnemu modelu dodane naslednje varnostne komponente:

- SSI (Secure Subsystem Infrastructure): Ta komponenta omogoča določitev množice operacij v sistemu, ki od uporabnikov oz. subjektov zahtevajo kontrolo dostopa. Na primer: za uporabo operacij, kot so vnos ocene v elektronski indeks, prijava na izpit, pregled ocen, se mora uporabnik najprej prijaviti v sistem.



Slika 41: Metamodel komponente SSI.

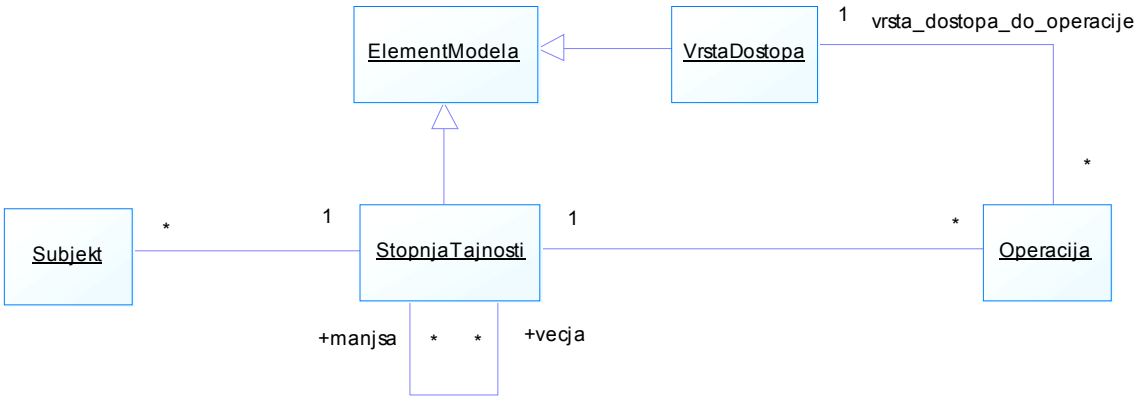
- URA (User-role assignment): S to komponento v sistem dodamo uporabnike, vloge (npr. študent, predavatelj) in povezave med njimi, s tem pa tudi dovoljenja, povezana s posameznimi vlogami.



Slika 42: Metamodel komponente URA.

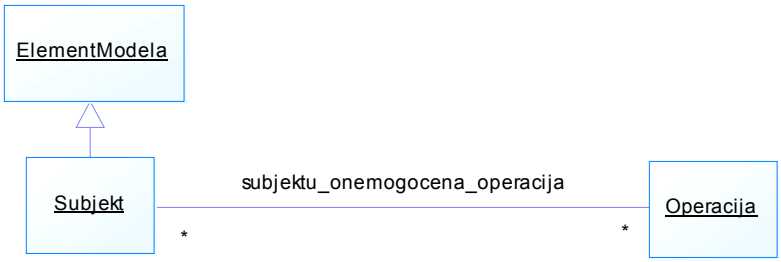
- SS (Simple Security) in SI (Simple Integrity): ti dve komponenti omogočata vpeljavo MAC modela v sistem. Obe komponenti imata isto zgradbo, ki subjektom določi stopnjo pravice dostopa do tajnih podatkov (clearance), objektom določi stopnjo tajnosti (classification), operacijam pa dodeli vrsto dostopa (bralni/pisalni). SS dovoli dostop subjektom do bralnih operacij, kadar je subjektova stopnja pravice dostopa

večja ali enaka stopnji tajnosti objekta. Komponenta SI pa omogoči isto za pisalne operacije.



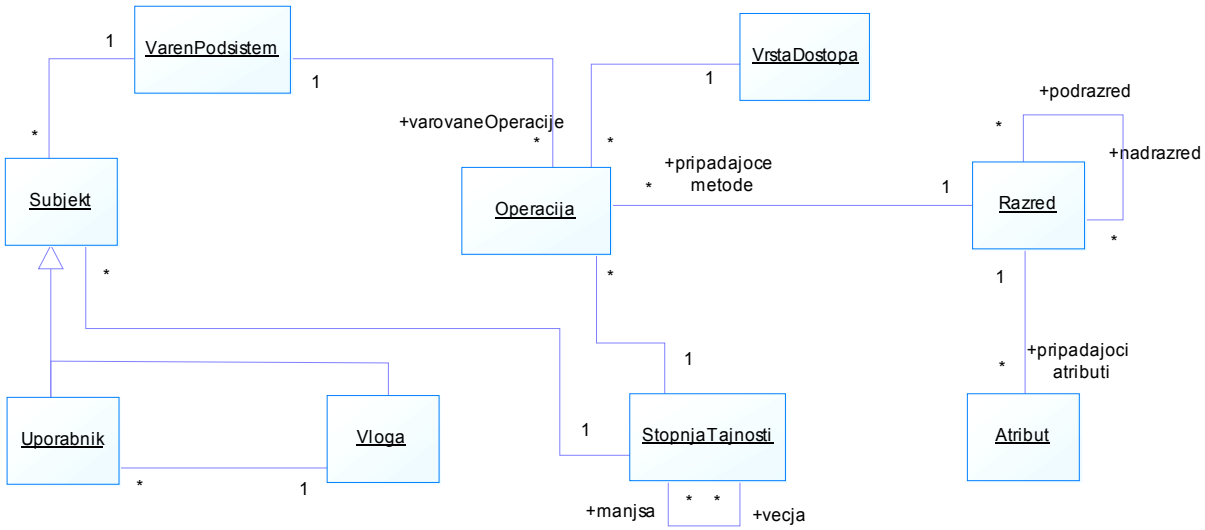
Slika 43: Metamodel komponent SS in SI.

- NP (Negative Permission): se uporabi za eksplicitno zavrnitev dostopa subjekta do določene operacije. Ta omejitev je prikazana kot povezava *subjektu-_onemogocena_operacija* med subjektom in objektom.



Slika 44: Metamodel komponente NP.

Kočni izdelek je metamodel, ki združuje vse opisane komponente: SSI+SS+SI+URA+NP (Slika 45). Če dva metamodela vsebujeta razred z istim imenom, bo končni metamodel vseboval samo en razred s tem imenom. V njem pa bodo združeni vsi atributi, relacije in metode njegovih originalnih razredov. Na koncu se dodajo še morebitna dovoljenja za operacije, ki nimajo posebnih varnostnih zahtev. To je v tem konkretnem primeru operacija *vrniStudijskePredmete* iz razreda *Katalog,* do katere ima pravico dostopati vsak uporabnik sistema.



Slika 45: Združen metamodel SSI+SS+SI+URA+NP.

Tak pristop pušča razvijalcu odprte roke pri razvoju varnostnih mehanizmov glede na vsak konkreten primer, s katerim se sooča. Nobeden izmed prej omenjenih pristopov ne združuje treh varnostnih shem v eni, kot to omogoča ta pristop. Glede na neodvisnost med varnostno in osnovno shemo so možne tudi naknadne spremembe in izboljšave varnosti, ki pa ne vplivajo na ostale, nevarnostne komponente sistema.

4. Predlog integracije varnostnih razširitev v okolju študijske informatike

Predlog vpeljave varnostnih mehanizmov, ki ga bomo opisali v nadaljevanju, se nanaša na obstoječe informacijske sisteme, ki jih uporablja Univerza v Ljubljani in njene fakultete, med njimi tudi Fakulteta za računalništvo in informatiko. Obstajata dva razloga, zakaj smo se odočili ravno za okolje študijske informatike. Prvi je ta, da smo se z enim od informacijskih sistemov (e-Študent), ki je del celotenega sistema, srečevali zadnjih 5 let, vse od vpisa na fakulteto. Sistem je znan z uporabniškega vidika, hkrati pa tudi iz načrtovalskega, saj smo se z arhitekuro in celotno funkcionalnostjo sistema spoznali pri predmetu Teorija informacijskih sistemov v 4. letniku študija. Drugi razlog je kompleksnost celotnega sistema, pri katerem niso pomembni le funkcionalni vidki sistema, ampak tudi varnostni. Pri tako obsežnem in razčlenjenem sistemu, ki ga uporablja preko 50.000 uporabnikov, je zelo pomembno zagotavljanje zaupnosti, popolnosti in razpoložljivosti podatkov v sistemu. Varnost, zaupnost in tajnost so pojmi, mimo katerih v sklopu načrtovanja informacijskih sistemov ne moremo. Hkrati so ti pojmi močno povezani z mehanizmom upravljanja z identitetami.

Celoten sistem uporablja veliko različnih skupin uporabnikov, od študentov do predavateljev, kadrovskih delavcev, administratrojev, vodstva univerze ter mnogih drugih. Vsi ti uporabniki so v sistemu predstavljeni s svojimi digitalnimi identitetami. Upravljanje the identitet ureja, kdo ima kakšne pravice v sistemu, do katerih informacijskih virov lahko dostopa in kako lahko s tem vpliva na podatke v sistemu. V nadaljevanju sledi predlog, kako z vpeljavo modela RBAC izboljšati nadzor dostopa do posameznega sistema ter do podatkov v celotnem sistemu. Na osnovi modela RBAC sta nato predstavljena še življenjska cikla digitalne identitete študenta in zaposlenega (pedagoškega delavca) na fakulteti.

4.1. Teoretični pogled

Sistem tako imenovane študijske informatike Univerze v Ljubljani je sestavljen iz petih večjih informacijskih sistemov. Ti so:

1. **Centralni imenik:** v centralnem imeniku so shranjeni podatki, ki so potrebni za avtentikacijo za vse štiri spodaj naštete sisteme. Vsebuje osnovne podatke o uporabnikih, uporabniških imenih in geslih.
2. **Visokošolska prijavno-informacijska služba (VPIS):** to je sistem za obdelavo in hranjenje podatkov o dijakih in ostalih kandidatih, ki se v določnem razpisanem vpisnem roku vpisujejo na kateregakoli od visokošolskih zavodov v Sloveniji. Zbrani so torej osebni podatki kandidatov, podatki o dosedanjem izobraževanju, opravljeni maturi in nenazadnje tudi o njihovi prijavi na študij.
3. **Kadrovska informatika Univerze:** podatki, ki jih najdemo v tem sistemu, se nanašajo na zaposlene v sklopu celotne Univerze, tu so njihovi osebni podatki, izobrazba, izkušnje. Poleg tega so tu še vsi podatki, povezani s sklenjenim zaposlitvenim razmerjem z Univerzo oz. s posamezno fakulteto. To so zaposlitvene

pogodbe, plače, dopust, dodatna izobraževanja, napredovanja, podatki o prenehanju delovnega razmerja in podobno.

4. **Poslovna infromatika Univerze:** sistem poslovne informatike upravlja s podatki, povezanimi s samim poslovanjem Univerze. Torej s financami, investicijami, nabavo, skratka z vsemi funkcijami, ki so poleg kadrovske in študijske potrebne, da lahko Univerza in pripradajoče fakultete uspešno opravljajo svoje poslanstvo.
5. **E-študent:** je informacijski sistem, ki hrani podatke o študentih, predmetih, predavateljih, izpitnih rokih, ocenah ter ostale podatke, povezane s študijem. Glavni uporabniki tega sistema so študentje, študentski referat, učitelji, tudi asistenti in seveda administratorji, ki skrbijo za sam sistem.

Vseh pet sistemov je med seboj povezanih: nekatere entitete nastopajo v več sistemih in nekateri procesi tečejo čez meje več sistemov. Prav tako lahko uporabnik dostopa do večih, nekateri celo do vseh petih sistemov, odvisno od uporabnikove pozicije in delovnega mesta, zato je pomembno, da ne prihaja do podvajanja podatkov. Sistem mora delovati kot enotna, usklajena celota.

Glede na veliko število uporabnikov bi bilo nesmisleno vsakega posebej obravnavati kot svojo identiteto. Optimalen način je združiti uporabnike, ki imajo določene skupne lastnosti, v skupine in s tem zelo zmanjšati število različnih vrst uporabnikov. Vsako tako skupino nato predstavimo kot svojo vrsto identitete, za katero veljajo določena pravila. Taka vrsta identitete je v bistvu neka vloga, ki jo lahko dodelimo uporabniku pri uporabi sistema. Glede na prebrane članke, opisane v 3. poglavju, se zdi v takem sistemu najbolj primerna vpeljava RBAC modela. Naj ponovimo, to je model, ki temelji na vpeljavi vlog v sistem. Z dodelitvijo vloge uporabniku ta s pomočjo sistema za upravljanje z identitetami pridobi vsa dovoljenja za upravljanje s sistemom, ki tej vlogi pripadajo.

Prednost tega pristopa je gotovo lažji nadzor dostopa, saj lahko že na podlagi vloge, ki je dodeljena uporabniku, sistem ugotovi, do katerih informacijskih virov in podatkov ima uporabnik dostop in do katerih ne. Na primer: študent lahko dostopa samo do sistema e-Študent svoje fakultete, do ostalih sistemov pa nima pravice dostopati. Če bi se na primer želel študent prijaviti v kadrovski informacijski sistem, bi sistem že ob prijavi iz vloge, ki je dodeljena študentu, ugotovil, da uporabnik nima pravice dostopati do tega sistema. Poleg lažjega nadzora dostopa obstajajo še druge prednosti uporabe RBAC modela. Že sama vpeljava vlog v sistem olajša nadzor dostopa do informacijskih virov sistema in posameznih funkcionalnosti.

Seveda pa dejstvo, da je več uporabnikom dodeljena ista vloga, še ne pomeni, da veljajo za vse uporabnike povsem ista pravila v sistemu. Najboljši način za določanje natančnejših pravil glede pravic in dolžnosti posameznega uporabnika je z uporabo omejitev v sklopu RBAC modela (Slika 1).

Vzemimo za primer vlogo *študent*, s katero lahko uporabnik dostopa do sistema e-Študent in opravlja osnovne naloge, kot so pregled izpitnih rokov, pregled ocen, naročanje potrdil itd. Vloga *študent* je ob prijavi dodeljena vsakemu študentu, ki je na študij vpisan redno, lahko je ponavaljec, pavzer ali absolvent. Vendar pa vsi ti uporabniki nimajo enakih dovoljenj v

sistemu e-Študent. Torej ni dovolj, da je uporabniku dodeljena vloga študenta, ampak je potrebno vsakokrat določiti še dodatno vlogo, ki natančneje opredeljuje uporabnikove pravice v sistemu. V tem primeru je koristno vpeljati hierarhijo vlog, ki je prav tako del modela RBAC. Torej, če imamo dva uporabnika in imata oba dodeljeno vlogo *študent* in je eden redno vpisani študent 1. letnika, drugi pa že absolvent, mora sistem ob prijavi to dejstvo upoštevati ter študentu glede na njegovo stanje vloge dodeliti še dodatna dovoljenja oz. dodatno vlogo, ki jo deduje od vloge *študent*. Vloga *študent* je torej starševska vloga, od nje pa lahko deduje več različnih vlog, kot so *redno vpisan, ponavaljlec* itd.

En uporabnik ima tako lahko hkrati dodeljenih več vlog. Glede na vsako vlogo so nato uporabniku dodeljena ustrezna dovoljenja. Seveda se skozi študij lahko vloge uporabnika spreminjajo. Pri študentu, ki se prvič vpiše na študij, imamo v začetku vlogo *redno vpisan*. V primeru neopravljenih pogojev za vpis v naslednji letnik študent izgubi vlogo rednega študenta in mu je dodeljena vloga *ponavljalec*. Ti prehodi med različnimi vlogami so del življenjskega cikla identitete. V podpoglavju, ki sledi, bomo za lažje razumevanje podrobneje opisali življenjski cikel identitete za študenta in pedagoškega delavca na fakulteti.

Naj omenim še eno situacijo oz. funkcionalnost, ki je potrebna v sistemu v primeru spreminjanja vloge uporabnika. Za lažjo predstavo vzemimo situacijo, ko študent konča študij, nato pa se zaposli kot asistent na isti ali kateri drugi fakulteti. Ena od možnosti bi bila izbristati (arhivirati) stari profil uporabnika in ustvariti novega. Ker pa so nekateri podatki v sistemu še vedno potrebni, bi bilo nesmisleno ustvariti povsem nov profil uporabnika. Zato je lahko zelo koristno, če imamo v sistemu funkcionalnost, ki nam omogoča prehajanje med različnimi vlogami tako, da sistem sam poskrbi za to, katere podatke ohraniti, katere shraniti, morda nekatere izbrisati in kako spremeniti dovoljenja uporabniku glede na novo dodeljeno vlogo. Do podobne situacije lahko pride že v sklopu življenjskega cikla identitete študenta in tudi med spreminjanjem administrativnih delovnih mest. Npr. zamenjava delovnega mesta med kadrovsko, poslovno, administrativno funkcijo itd.

Naj se vrnemo nazaj na model RBAC, ki je osnovno ogrodje za postavitev kontrole dostopa v sistemu. Ker se avtentikacija za vse sisteme izvaja s pomočjo centralega imenika, bi model RBAC lahko vključili v centralni imenik. Tako bi imeli na enem mestu shranjene osnovne podatke o uporabnikih in hkrati podatke o vlogah, stanjih vlog, ki so dodeljene posameznemu uporabniku, ter tudi o sejah, aktivnih v določenem trenutku. Pri prijavi v kateregakoli izmed sistemov bi se glede na vlogo uporabnika nato konkretnemu uporabniku določile pravice in dovoljenja za uporabo sistema. Če se pedagoški delavec prijavi v sistem kadrovske informatike, dobi dovoljenje za vpogled podatkov o svoji plači, dopustu, možnostih izobraževanja in podobno. Če se v isti sistem prijavi nekdo iz računovodstva, so mu dodeljena povsem drugačna dovoljenja, naprimer možnost vnosa podatkov za obračun plač. Vloge morajo biti torej oblikovane glede na potrebe uporabnikov. V sistemu celotne študijske informatike so potrebne različne vloge in podvloge. Ena od možnih razdelitev oz. opredelitev vlog je naslednja:

- Pedagoške vloge: redni profesor, izredni profesor, docent, asistent, mladi raziskovalec.
- Kadrovske vloge: zaposlovanje, obračun, zaposleni.

- Administrator: splošen, e-študent, vpis, KIU, PIU, centralni.
- Poslovne vloge: referat, računovodstvo, finance, uprava, dekanat.
- Študent: kandidat, študent, redno vpisan, ponavljalec, pavzer, zamrznjen status, absolvent, alumni, podiplomski.

4.2. Življenjski cikel identitete za študenta in zaposlenega

❖ Identiteta: ŠTUDENT

Možne vloge študenta so natančeje opisane v spodnji tabeli. Poleg možnih vlog so našteta še okvirna dovoljenja v sklopu sistema e-študent.

Vloga	Dovoljenja v sistemu e-Študent
kandidat	- pregled obvestil - vpis v naslednji letnik
študent	- pregled izpitnih rokov - pregled ocen - naročilo potrdila o opravljenih izpitih - spreminjanje nastavitev - pregled obvestil
redno vpisan (deduje od vloge *študent)*	- prijava/odjava na izpit - naročilo potrdil o vpisu - vpis v naslenji letnik
ponavljalec (deduje od vloge *študent)*	- naročilo potrdil o vpisu - vpis v naslednji letnik
pavzer (deduje od vloge *študent)*	- prijava na izpit s plačilom - odjava z izpita - vpis v naslednji letnik
izpisan	- pregled ocen (2 leti)
zamrznjen status (če ni opravil izpita več kot 2 leti zaporedoma)	- pregled obvestil - pregled ocen
absolvent (deduje od vloge *študent)*	- prijava/odjava na izpit - naročilo potrdila o vpisu - pregled tem za diplomsko nalogo - naročilo potrdila o dvignjeni temi diplomske naloge - vloga za podaljšanje statusa
alumni	- naročilo potrdila o opravljenih izpitih - pregled obestil

Tabela 4.1: Seznam možnih vlog identitete študenta ter dovoljenj, dodeljenih posamezni vlogi.

Študent se kot digitalna identiteta v sistemu študijske informatike prvič pojavi z oddajo prijavnice za vpis na kateregakoli od visokošolskih zavodov v Sloveniji. Takrat se njegovi podatki shranijo v sistemu VPIS, in sicer njegovi osebni podatki, podatki o dosedanjem šolanju ter podatki o njegovi prijavi na študij. Po opravljeni maturi oz. zaključnem izpitu se v sistem vnesejo doseženi rezultati. Takrat prijavna služba podatke uredi in kandidate obvesti o tem, na kateri študijski program so sprejeti. Hkrati obvestijo o tem tudi posamezne fakultete

in jim posredujejo podatke o prijavljenjih kandidatih. Vsaka fakulteta mora namreč že pred samim vpisom v sistem e-Študent vnesti podatke o morebitnih novih študentih, sprejetih na študij - kandidatih, in se tako pripraviti na dejanski vpis. Vsak kandidat dobi obvestilo o datumu prvega vpisa na izbrano fakulteto. Pred vpisom kandidat sam aktivira svoj uporabniški račun, dodeli se mu ustrezna vloga (*kandidat*). Po vpisu se vloga spremeni v *redno vpisan*. Ker ima vloga *redno vpisan* tudi starševsko vlogo (*študent*), dobi redno vpisan študent tudi vsa dovoljenja te starševske vloge. Enako se zgodi ob vsaki dodelitvi določene vloge, ki ima v hierarhični shemi nad seboj neko starševsko vlogo. Vloga, ki je dodeljena študentu, se med šolskim letom načeloma ne spreminja, razen če se študent odloči izpisati s fakultete. Takrat se študentu odvzame vloga *redno vpisan* (in posredno vloga *študent*) ter se mu dodeli vloga *izpisan.* S tem izgubi skoraj vsa dovoljenja v sistemu e-Študent.

Pred začetkom vsakega novega študijeskega leta (ob vpisu v naslednji letnik) mora sistem za vsakega študenta preveriti, ali izpolnjuje pogoje za vpis v naslednji letnik. Če jih, se študent lahko redno vpiše in dodeli se mu vloga *redno vpisan*. Kadar gre za študenta 2. ali 3.letnika, obstajata dve možnosti v primeru, če pogojev za vpis v nasednji letnik ne izpolnjuje:

1. če študent še ni ponavljal nobenega letnika, potem se lahko ponovno redno vpiše isti letnik,
2. če je študent že ponavljal letnik v času študija, se v naledjem študijskem letu ne more redno vpisati in mora pavzirati.

Pri študentu 1. letnika, ki ne izpolnjuje pogojev za vpis v 2. letnik, je pomembno preveriti, ali izpolnjuje pogoje za ponavljanje. Če jih ne izpolnjuje, se ne more ponovno vpisati v 1. letnik, kljub temu da pred tem še ni ponavljal letnika. Tako mu ostane le opcija pavziranja ali izpisa iz fakultete. Glede na izpolnjene oz. neizpolnjene pogoje se ob začetku vsakega šolskega leta študentu dodel ustrezna vloga, in sicer ena izmed vlog *redno vpisan, ponavljalec* ali *pavzer* in posredno tudi vloga *študent.*

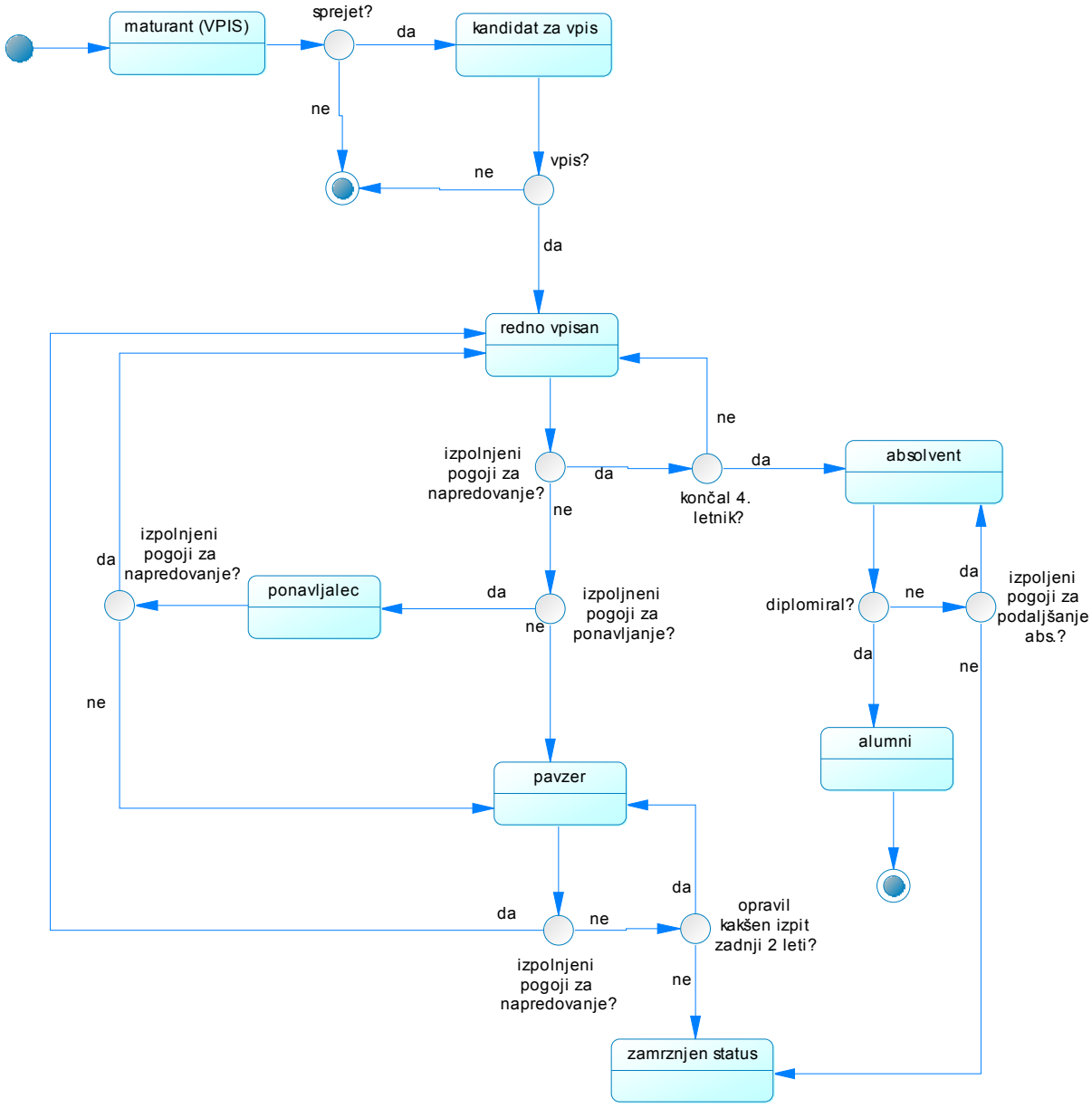
Po končanem 4. Letniku postane študent absolvent. Temu primerno se mu vloga spremeni. Po uspešno opravljenih vseh izpitnih obveznostih dobi dodatna dovoljenja, in sicer za pregled tem diplomskih nalog. Ko se odloči za mentorja in temo diplomske naloge ter jo prijavi v študentskem referatu, se mu v e-Študentu pojavi še opcija naročitve potrdila o dvignjeni temi diplomske naloge. Po zagovoru diplome se študij za študenta konča in takrat mu preneha status študenta. V e-Študentu se to označi tako, da uporabnik izgubi vlogo *študent* in se mu dodeli le vloga *alumni*. S tem bivši študent fakultete izgubi skoraj vsa dovoljenja, ki jih je imel kot študent v sistemu e-Študent. Edini funkciji, ki sta mu še omogočeni za naslednji 2 leti, sta naročilo potrdila o opravljenjih izpitih in pregled obvestil.

Izjemna situacija, do katere še lahko pride, je v primeru, ko študent pavzira in več kot 2 leti zapored ne opravi nobenega izpita. Takrat se mu vloga v sistemu spremeni na *zamrznjen status*, v sistemu pa so mu onemogočene vse funkcije razen pregleda ocen in obvestil. Kadar želi tak študent svoj račun znova aktivirati (se torej odloči, da bo s študijem nadaljeval), se mora osebno oglasiti v pisarni študentskega referata, kjer se dogovori, kako bo potekalo nadaljevanje študija. Če se odloči študij nadaljevati, mu študentski referat dodeli vlogo

pavzer. Isto se zgodi tudi v primeru, ko študentu mine absolventski status preden diplomira. Takrat se mu namesto vloge *absolvent* dodeli vloga *zamrznjen status*.

Po zaključku študija se podatki v sistemu ne izbrišejo, ampak se vsi podatki v sistemu e-Študent arhivirajo. Do njih lahko dostopajo zaposleni v študentskem referatu in seveda administrator sistema.

Za lažjo predstavo je zgoraj opisan življenjski cikel identitete študenta predstavljen na spodnji sliki (Slika 46) z diagramom stanj diagramske tehnike UML.



Slika 46: Življenjski cikel identitete študenta.

❖ **Identiteta: PEDAGOG**

V spodnji tabeli so opisane vloge, ki so lahko dodeljene pedagoškemu delavcu na fakulteti.

Vloga	**Dovoljenja**	
	e-Študent	**kadrovska informatika univerze**
pedagog (deduje od vloge *asistent)*	- vnos podatkov o predmetih - vnos ocen - pregled podatkov o svojih študentih - vnos tem za diplome - pregled svojih diplomantov - vnos pooblastil asistentu	- pregled osebnih podatkov - pregled plač - pregled obvestil
redni professor	(deduje od vloge *pedagog*)	(deduje od vloge *pedagog*) - ocenjevanje dela asistentov in raziskovalcev
izredni professor	(deduje od vloge *pedagog*)	(deduje od vloge *pedagog*) - prijava na habilitacijo
docent	(deduje od vloge *pedagog*)	(deduje od vloge *pedagog*) - prijava na habilitacijo
asistent (deduje od vloge *raziskovalec*)	- vnos izpitnih rokov in kolokvijev	(deduje od vloge *pedagog*) - prijava na dodatno izobraževanje
mladi raziskovalec	- pregled podatkov o predmetih - pregled prijavljenih na izpit - pošiljanje obvestil študentom	(deduje od vloge *pedagog*)

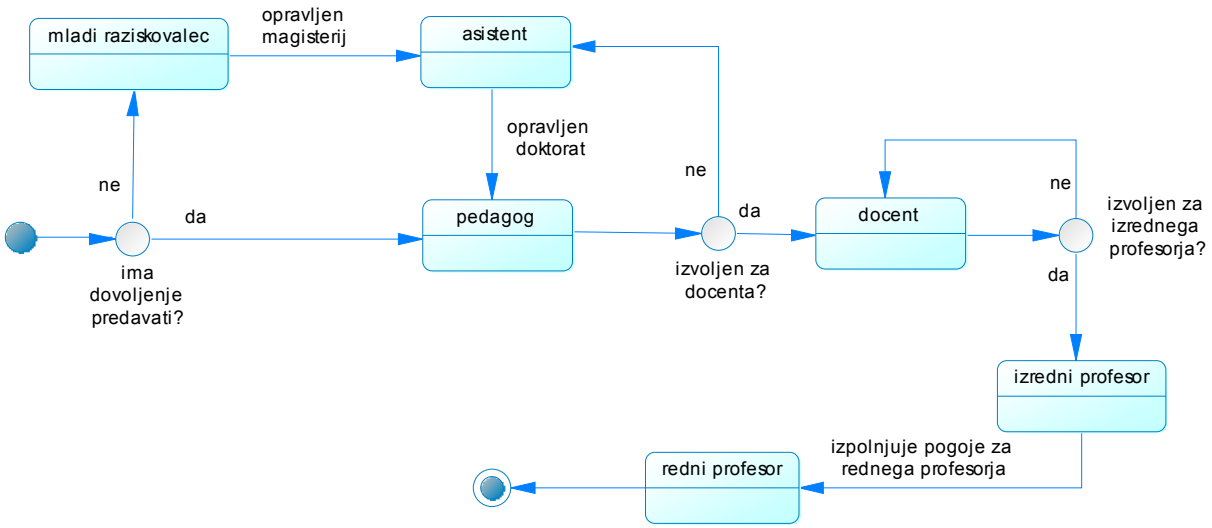
Tabela 4.2: Seznam možnih vlog identitete pedagoga ter dovoljenj, dodeljenih posamezni vlogi.

Življenjski cikel digitalne identitete pedagoga ni tako razvejan kot življenjski cikel identitete študenta. Že samo prehajanje med vlogami ni tako pogosto kot pri študentu. Kljub temu ima pedagoški delavec lahko dodeljenih več različnih vlog, ki se skozi leta po potrebi spreminjajo.

Za razliko od življenjskega cikla identitete študenta se lahko življenjski cikel pedagoškega delavca začne na različne načine. Najbolj osnovni je ta, da se študent po diplomi zaposli na fakulteti kot mladi raziskovalec. S tem so mu v sistemu e-Študent omogočene osnovne funkcije, ki jih potrebuje pri svojem delu. Naštete so v zgornji tabeli (Tabela 4.2). Mladi raziskovalec lahko poleg dela na raziskovalnem področju pomaga pri izvajanju vaj v sklopu svojega laboratorija. Kadar opravi magisterij, lahko napreduje na položaj asistenta. Kot asistent ima zaposleni nekaj več pravic v e-Študentu, vendar še vedno ne more izvajati predavanj in biti nosilec nekega predmeta. Asistent je torej hkrati tudi mladi raziskovalec, zaradi tega vloga *asistent* deduje dovoljenja od vloge *mladi raziskovalec.* Kadar asistent doktorira in nadaljuje s pedagoško in raziskovalno dejavnostjo, lahko zaprosi za naziv docenta. Če je izvoljen, dobi naziv (in s tem tudi vlogo) *docent* za naslednjih 5 let. Podoben postopek se ponovi v primeru izvolitve za izrednega profesorja. Tudi ta naziv je časovno omejen, zato vsakih 5 let poteka habilitacija oz. ponovna izvolitev v pedagoški naziv. Poleg tega mora pedagog ves čas opravljati raziskovalno delo, sodelovati na seminarjih, objavljati strokovne članke in podobno. Najvišji pedagoški naziv, ki ni časovno omejena, je *redni profesor*. To je tudi zadnja stopnja, v življenjskem ciklu pedagoga. Ni pa nujno, da poteka življenjski cikel identitete pedagoga po vseh the stopnjah zaporedoma. V primeru, da se na

fakulteti zaposli nekdo, ki že ima status docenta, izrednega ali rednega profesorja, mu je dodeljena ustrezna vloga in lahko takoj začne z izvajanjem predavanj in vseh ostalih pripadajočih funkcij. S tem je v življenjskem ciklu idenitete preskočil nekaj korakov, kar pa ne vpliva na nadaljnje upravljanje z njegovo identiteto v sistemu.

Na spodnji sliki (Slika 47) je s pomočjo diagrama stanja diagramske tehnike UML prikazan opisan življenjski cikel pedagoškega delavca.



Slika 47: Življenjski cikel identitete pedagoga.

Za lažjo predstavo, kako lahko poteka samo dodeljevanje vlog in dovoljenj v sistemu, je podan kratek primer z dejanskimi vrednostmi podatkovnih tipov modela RBAC. Torej seznam nekaj vlog uporabnikov in dovoljenj ter sej, v katerih nastopajo različne kombinacije omenjenih podatkovnih tipov.

VLOGE
mladi raziskovalec
asistent
pedagog
docent
redni profesor
izredni profesor

DOVOLJENJA
vnos izpitnih rokov
vnos ocen
vnos pooblastil asistentu
pregled plač
prijava na dodatno izobraževanje
pošiljanje obvestil študentom

UPORABNIKI
Janez Novak
Matej Hren
Tina Koren

SEJA	UPORABNIK	DODELJENE VLOGE	UPORABLJENA DOVOLJENJA
seja 1	Janez Novak	redni profesor	- vnos pooblastil asistentu (e-Študent) - vnos ocen (e-Študent) - pregled plač (KIU)
seja 2	Janez Novak	asistent	- vnos izpitnih rokov (e-Študent)
seja 3	Matej Hren	mladi raziskovalec	- pošiljanje obvestil študentom (e-Študent)
Seja 4	Tina Koren	docent	- prijava na dodatno izobraževanje (KIU)

4.3. Kritični pogled

Osnovi koncept predloga za izboljšanje varnosti v celotnem informacijskem sistemu Univerze je torej vpeljava modela RBAC in s tem olajšano upravljanje z identitetami v sistemu. Glede na vse štiri predloge sistematičnega obravnavanja načrtovanja varnosti, obravnavane v 3. poglavju, bi poleg omenjenega predloga predlagali še naslednje razširitve:

- Uvesti omejitve nad elementi modela RBAC, opisane v poglavju 3.4 (*omejitve za ločitev dolžnosti, predpogojne omejitve in omejitve številnosti*). Te določajo natančna pravila glede dodeljevanja vlog uporabnikom, o pravilnih in nepravilnih kombinacijah dodeljenih dovoljenj določeni vlogi in ostalih situacijah, ki lahko povzročijo ranljivost sistema. Omejitve je najboljše vključiti že v sam načrt sistema in jih definirati s pomočjo jezika OCL. Tako lahko v fazi implementacije sistema načrtovane omejitve lažje in pravilno realiziramo.
- Vpeljava avtorizacijskih omejitev, opisanih v poglavju 3.5. Te se ne nanašajo na dodelitev vlog v sistemu, ampak na konkretne entitete oz. razrede sistema, v katerih hranimo podatke, ki jih je potebno zaščititi (npr. osebne podatke, ocene …) ne glede na to, ali gre za sistem e-Študent ali kateregakoli izmed ostalih sistemov. S pomočjo jezika SecureUML in OCL lahko dodatno omejimo dostop do kritičnih entitet. To so entitete, pri katerih mora biti izpolnjen določen predpogoj, da lahko uporabnik dostopa do njih.
- Eden glavnih ciljev vpeljave sistema za upraljanjne z identitetami je poenostavitev oz. poenotenje vstopa v sistem in s tem do vseh njegovih aplikacij.V našem primeru bi celoten sistem študijske informatike potreboval enotno vstopno točko, ki bi bila skupna za vse uporabnike in sicer ne glede na to, ali gre za študenta, predavatelja, računovodjo ali kateregakoli drugega delavca na univerzi. Po uspešni avtentikaciji, ki poteka preko centralnega imenika, bi uporabnik brez dodatnih prijav v posamezni sistem lahko dostopal do vseh sistemov študijske informatike, do katerih ima pravico dostopati. Za študenta bi bil to samo sistem e-Študent, medtem ko bi predavatelj lahko izbiral med sistemoma e-Študent in sistemom kadrovske informatike Univerze. Hkrati bi tako lahko dostopal do vseh funkcij obeh sistemov, do katerih ima na podlagi svoje vloge in stanja te vloge pravico dostopati. S takim pristopom dosežemo visoko stopnjo varnosti v sistemu, hkrati pa izpolnimo enega od poglavitnih namenov upravljanja z digitalnimi identitetami, in sicer enkratni vpis v sistem (single sign-on).
- Nenazadnje je potrebno na informacijski sistem pogledati še z druge perspektive oz. z bolj oddaljenega zornega kota. Ni dovolj, da poskrbimo le za informacijsko varnost oz. varnost aplikacij, pomembni sta tudi fizična in omrežna varnost, ki ju je potrebno prav tako skrbno načrtovati. Poskrbeti je treba za varne povezave med posameznimi računalniki in strežniki, med katerimi poteka prenos občutljivih podatkov. Ena izmed možnih rešitev je uvedba lokalnega, po potrebi kriptiranega omrežja, ki bo povezovalo vse računalnike znotraj Univerze, kjer se obdelujejo in hranijo občutljivi podatki. Načrt varne infrastrukture lahko naredimo s pomočjo diagrama komponent in diagrama postavitve, kot je predlagano v poglavju 3.3.

5. Zaključek

Potreba po integraciji različnih aplikacij in s tem tudi kontrole dostopa narašča z večanjem kompleksnosti informacijskih sistemov v današnji informacijski družbi. Vedno večim uporabnikom prestavlja digitalni svet njihovo vsakdanje delovno okolje, zato je potrebno tako kot na vseh ostalih področjih v življenju najprej poskrbeti za varnost - za varnost svojega delovnega okolja, torej varnost informacijskih sistemov.

V diplomski nalogi so predstavljeni osnovni pogledi na računalniško varnost, predvsem z vidika kontrole dostopa in s tem tudi mehanizmi, ki naredijo kontrolo dostopa enotno in varnejšo. Najbolj razširjeno in tudi najbolj primereno ogrodje kontrole dostopa je RBAC (Role-Based Access Control). Njegova glavna značilnost je vpeljava vlog v sistem in dodelitev ustreznih dovoljenj, ki omejijo dostop do informacijskih virov in s tem povečajo stopnjo varnosti v sistemu. Kontrola dostopa do večih različnih aplikacij in sistemov se v zadnjem času večkrat združuje v enotni točki. Sistemom, ki skrbijo za enotno avtentikacijo in ostale podatke o uporabnikih, pravimo sistemi za upravljanje z digitalnimi identitetami. V sklopu 2. poglavja sta opisanani njihova osnovna zgradba in funkcionalnosti.

Ne smemo pozabiti na dejstvo, da je o vpeljavi kontrole dostopa in ostalih varnostnih mehanizmov potrebno začeti razmišljati že v fazi načrtovanja novega informacijskega sistema. Le tako lahko namreč zagotovimo izgradnjo stabilnega, robustnega in varnega sistema. Kako samo načrtovanje izvesti, katere mehanizme vpeljati in s katero digramsko tehniko to najbolj nazorno prikazati, so glavna vprašanja, ki so postavljena v sklopu te diplomska naloge. Po pregledu razne literature je ugotovljeno, da se tako v praksi kot teoriji dandanes najbolj pogosto uporablja diagramski načrtovalski jezik UML. Nato smo med strokovnimi članki izbrali štiri, ki predstavljajo najbolj zanimive ideje in predlog vpeljave varnostnih mehanizmov pri načrtovanju informacijskih sistemov ter pri tem uporabljajo načrtovalski jezik UML.

Prvi članek predlaga razširitev same diagramske tehnike UML s pomočjo že obstoječih elementov UML, kot so stereotipi, označene vrednosti in omejitve. Na tak način lahko izdelamo podrobnejši načrt aplikacije oz. informacijskega sistema s poudarkom na varnosti. Avtor se osredotoča predvsem na štiri vidike varnosti v sistemih, in sicer kako zagotoviti pošteno izmenjavo podatkov, ohraniti zaupnost podatkov, zagotoviti varen pretok informacij in varen komunikacijski kanal. Predlaga nove stereotipe, ki jih uporabi pri načrtovanju. Vendar pa nam sam načrt še ne zagotavlja, da bo novi sistem res vseboval vse načrtovane mehanizme. Izdelovalec sistema mora namreč sam poskrbeti za to, da bodo vsi načrtovani varnostni mehanizmi kasneje tudi implementirani.

Drugi članek se osredotoča predvsem na kontrolo dostopa. Omenjena sta dva modela, in sicer MAC ter RBAC. Glavna razlika med njima je ta, da v modelu RBAC uporabniku dodelimo vloge, ki določajo, do katerih funkcij in podatkov v sistemu lahko dostopa, medtem ko model MAC vlog ne pozna, ampak omejuje dostop do objektov v sistemu s pomočjo določanja stopnje tajnosti podatkov. Tak pristop je lahko strožji, zato se MAC uporablja predvsem v

sistemih, ki hranijo tajne podatke. V praksi se pogosteje uporablja model RBAC, saj ga lažje prilagodimo različnim sistemom. Za razliko od prvega članka, kjer varnostne mehanizme vplejemo v načrt le vizualno, omogoča vpeljava modela RBAC tudi natančnejše definiranje varnostnih zahtev oz. omejitev z jezikom OCL (Object Constraint Language). Definicije teh omejitev so osnova programerju pri implementaciji končne aplikacije. Kljub temu, da je s tem delo programerju olajšano, se ne izognemo možnim napakam pri dejanskem kodiranju nove aplikacije oz. sistema.

Podoben pristop z RBAC in definiranjem omejitev z OCL opisujejo tudi avtorji tretjega članka, kjer svjo različico načrtovalskega jezika poimenujejo SecureUML. Ideja je podobna kot v prejšnjem članku, razlika je le v tem, da avtorji tega članka poudarjajo modelno usmerjen razvoj programske opeme. Z uvedbo modelov pri načrtovanju se dvigne nivo abstrakcij in omogoči ponovna uporaba že izdelanih modelov. Poleg tega avtorji predlagajo vključitev dodatnih funkcionalnosti v model RBAC, in sicer avtorizacijske omejitve, ki predstavljajo neke vrste predpogoje za dostop do posameznih elementov sistema. Osnovne omejitve SecureUML lahko tako kot omejitve RBAC definiramo ne le grafično ampak tudi z jezikom OCL. Avtorji poudarjajo dejstvo, da že obstajajo generatorji, s katerimi lahko iz načrtov direktno generiramo kodo aplikacije. To je velika prednost, ki pospeši izdelavo varnih aplikacij in zmanjša število napak med samim kodiranjem.

V četrtem članku avtorji združujejo tri modele kotrole dostopa (MAC, DAC in RBAC) ter hkrati puščajo vsakemu načrtovalcu proste roke pri izbiri potrebnih varnostnih mehanizmov, glede na primer s katerim se sooča. Celoten varnostni model je razdeljen na štiri skupine komponent, kjer vsaka vsebuje več manjših komponent za obvladovanje varnosti. Ta pristop je torej zelo flekibilen in primeren predvsem za sisteme, kjer želimo varnost načrtovati po meri. Slabost tega pristopa je ta, da lahko v primeru nenatančnega načrtovanja spregledamo kakšno od varnostnih lukenj v sistemu in izdelamo varnostno shemo, ki bo pomajkljiva.

Iz vseh obravnavanih člankov so povzete najboljše ideje in uporabljene v praktičnem delu diplomske naloge, kjer je proučeno področje upravljanja z digitalnimi identitetami na primeru univerzitetnega okolja. Opisan je predlog vpeljave RBAC modela kontrole dostopa v sistem za upravljanje z digitalnimi indentitetami. Prednosti predlaganega pristopa so lažji in učinkovitejši nadzor dostopa do univerzitetnega sistema in pripadajočih informacijskih virov. Poleg tega je sistem prilagojen vsaki skupini uporabnikov glede na njihove potrebe, pravice in dolžnosti. Nato je predstavjen še predlog življenjskega cikla identitete za študenta in pedagoga, ki podrobneje prikazuje dogajanje v sistemu skozi čas. Oba življenjska cikla sta predstavljena opisno in z diagramom stanj diagramske tehnike UML.

Predlagana rešitev temelji na obstoječem informacijskem sistemu Univerze. Ker vsebuje nove predloge, kako izboljšati celoten sistem Univerze, bi jo lahko uporabili v novem informacijskem sistemu Univerze, ki je trenutno v izdelavi. Ena od možnih razširitev, o kateri bi bilo v primeru uporabe predloga dobro razmisliti, je povezava večih življnjskih ciklov različnih identitet. Na primer prehod iz identitete študenta v identiteto pedagoga, kadar se diplomant zaposli na fakulteti. S povezavo večih življenjskih ciklov bi se kompleksnost

sistema povečala. Omogočena bi bila avtomatizacija nekaterih postopkov v sistemu, s tem bi se olajšalo delo administratorju in povečala zanesljivost sisitema.

Idej in predlogov za izdelavo varnih informacijskih sistemov je ogromno. Pomembno je predvsem, da se zavedamo resničnega pomena varnosti pri načrtovanju in izdelavi informacijskih sistemov. Na nas pa je, da se glede na potrebe in zahteve v konkretnem primeru odločimo, katere mehanizme bomo pri načrtovanju uporabili in kako jih nato v zaključni fazi tudi implementirali.

6. Kazalo slik

7. Viri

[1] Jürjens, Jan, *UMLsec: Extending UML for secure systems development*, v: The Unified Modeling Language, LNCS 2460 (2002), strani 412–425, dostopno na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.5211&rep=rep1&type=pdf

[2] Jürjens, Jan, *Deveoping safety-critical systems with UML*, v: UML 2003 – The Unified Modeling Language, Model Languages and Applications, LNCS 2863 (2003), strani 360-372

[3] T. Lodderstedt, D. Basin, in J. Doser, *SecureUML: A UML-Based Modeling Language for Moel-Driven Security,* v: UML 2002, LNCS 2460, strani 426-441, dostopno na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.57.7885&rep=rep1&type=pdf

[4] Indrakshi Ray, Na Li, Robert France, *Using UML to Visualize Role-Based Access Control Constraints*, v: Proceedings of the Symposium on Access Control Models and Technologies (SACMAT), (2004) strain 31-40, dostopno na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.9736&rep=rep1&type=pdf

[5] Indrakshi Ray, Na Li, Dae-Kyoo, Robert France: *Using Parameterized UML To Specify And Compose Access Control Models,* v: Proc. Of the 6th IFIP Working Conference on Integrity & Internal Control in Info. Systems, 2003, dostopno na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.7689&rep=rep1&type=pdf

[6] J. Pavlich-Mariscal, L. Michel, S. Demurijan: *Enhancing UML to Model Custom Security Aspect*, v: Proceedings of the 11th International Workshop on Aspect-Oriented Modeling, 2007, dostopno na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.98.102&rep=rep1&type=pdf

[7] Gail-Joon Ahn, Michael. E. Shin: *Role-Based Authorization Constraints Scecification Using Object Constraint Language*, v; 10th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE Computer Society, 2001, strani 157-162, dostopno na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.1111&rep=rep1&type=pdf

[8] Tomaž Lukman, *Modelno usmerjen razvoj programske opreme – domensko specifičen pristop,* v zborniku 10. mednarodne multikonference Informacijska družba – IS 1007, strani 287- 291.

[9] Joseph Pato, *Identity Management: Setting Context,* v: Technical report, Trusted Systems Laboratory, HP Laboratories Cambridge, HPL-2003-72, 2003, dostopno na: http://artemis.cs.yale.edu/classes/cs155/spr03/idmgmt-tr.pdf

[10] Marco Cassa Mont, Pete Bramhall, *Identity Managemen: a Key e-Business Enabler,* v: Technical report, HP – Trust, Security and Privacy, 2002, dostopno na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.18.5717&rep=rep1&type=pdf

[11] Rick Lehtinen, *Computer Security Basics*, O'Reilly Media, 2nd Edition, 2006

[12] Interactive objects, dostopno na: http://www.interactive-objects.com/en/model-driven-engineering/46-model-zentrische-software-entwicklung-mdamdd.html