Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jan Meznarič

Dinamično generiranje validacijske logike z uporabo sistemov za upravljanje s poslovnimi pravili

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: prof. dr. Matjaž Branko Jurič

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Opišite pristope k validaciji vnosov v spletnih aplikacijah. Proučite in analizirajte sisteme za upravljanje s poslovnimi pravili. Opišite koncept poslovnega pravila in načine zapisa. Podrobno preglejte orodje Drools. Izdelajte zasnovo sistema za validacijo vhodnih podatkov z uporabo poslovnih pravil oz. sistemov za upravljanje s poslovnimi pravili. Implementirajte sistem, ga predstavite in izdelajte spletno aplikacijo za testiranje zasnovanega sistema.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Jan Meznarič, z vpisno številko **63110268**, sem avtor diplomskega dela z naslovom:

Dinamično generiranje validacijske logike z uporabo sistemov za upravljanje s poslovnimi pravili

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Matjaža Branka Juriča,
- 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 na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 15. septembra 2014 Podpis avtorja:

Hvala mentorju prof. dr. Matjažu B. Juriču za pomoč pri izbiri teme in izdelavi naloge.

Hvala staršem, ki so mi omogočili, da sem tu, kjer sem.

Hvala prijateljem, ki mi lepšajo življenje.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
BRMS	Business Rule Management System	Sistem za upravljanje s poslovnimi pravili.
DRL	Drools Rule Language	Drools jezik za zapis pravil.
KRR	Knowledge Representation and Reasoning	Prikaz znanja in sklepanje.
SQL	Structured Query Language	Strukturirani povpraševalni jezik za delo s podatkovnimi bazami.
XSS	Cross-Site Scripting	Napad skriptnega izvajanja za več mest.

Povzetek

Validacija vnosov v spletnih aplikacijah predstavlja pomemben del njihovega delovanja. Vneseni podatki morajo biti skladni s tehničnimi omejitvami vnosa, ki jih določi razvijalec, ali pa z omejitvami iz poslovnega okolja. V programskih ogrodjih za izdelavo spletnih aplikacij je validacijska logika sklopljena s programsko kodo. Zaradi tega je ob spremembi posameznega pravila potrebno ponovno prevajanje in postavitev spletne aplikacije, kar oteži njihovo upravljanje in vzdrževanje. V tem delu smo razvili sistem za validacijo vnosov s pomočjo sistema za upravljanje s poslovnimi pravili. Validacijska pravila so shranjena v centralnem repozitorju, ločeno od implementacije spletne aplikacije. S tem smo dosegli enostaven in pregleden zapis validacijske logike v obliki deklarativnih poslovnih pravil ter poenostavili vzdrževanje aplikacij ob spremembah validacijske logike.

Ključne besede: validacija vnosov, spletne aplikacije, poslovna pravila.

Abstract

Input validation in web applications represents an important part of their functionality. With proper validation we ensure that provided input data is in accordance with technical constraints, defined by the developer and with business-related constraints. In web development frameworks, validation logic is coupled with program code. If one validation rule is changed, application needs to be recompiled and redeployed. In this thesis we developed a system for input validation based on business rules management system. Validation rules are stored in central repository, separated from implementation of web applications. Thus, we have achieved a simple and transparent way of declaring validation logic in the form of declarative business rules as well as simplifed applications maintenance in case of changes in validation logic.

Keywords: input validation, web applications, business rules.

Poglavje 1

Uvod

Spletne aplikacije nas danes spremljajo na vsakem koraku. Milijoni uporabnikov jih vsakodnevno uporabljajo za številne aktivnosti, npr. opravljanje finančnih transakcij, urejanje dokumentacije, komuniciranje, pridobivanje informacij, druženje, sprostitev ipd.

Zagotoviti moramo, da podatki, vneseni preko spletnega vmesnika, ustrezajo bodisi tehničnim omejitvam bodisi omejitvam v poslovnem okolju, kar dosežemo z ustrezno validacijo vnosov. Validacija izboljša varnost aplikacije ter skrbi, da aplikacija izloči vnose, ki bi lahko povzročili napačno delovanje.

V spletnih aplikacijah, izdelanih z namenskimi ogrodji, je validacijska logika sklopljena s programsko kodo. Zaradi tega je ob spremembi posameznega pravila potrebno ponovno prevajanje in postavitev spletne aplikacije, kar oteži njihovo upravljanje in vzdrževanje.

V diplomski nalogi smo zasnovali sistem za validacijo vnosov, ki se opisanega problema loteva s pomočjo sistema za upravljanje s poslovnimi pravili. Ta nam omogoča centralizacijo validacijskih pravil v osrednjem repozitoriju, ločeno od implementacije aplikacije, kar poenostavi njeno vzdrževanje, saj ni potrebe po ponovnem prevajanju in namestitvi aplikacije v primeru spremembe validacijskih pravil. Z zapisom validacijske logike s pomočjo poslovnih pravil se izognemo proceduralnim programskim jezikom in validacijsko logiko zapišemo deklarativno kot množico poslovnih pravil. Zasnovan sis-

tem omogoča vzdrževanje validacijske logike preko spletnega vmesnika, samodejno generiranje kode JavaScript za validacijo na odjemalcu in možnost implementacije validacije na strežniku.

Diplomska naloga je sestavljena iz 6 poglavij. V poglavju 2 opišemo pomen in način implementacije validacije vnosov v spletnih aplikacijah. Poglavje 3 predstavi sisteme za upravljanje s poslovnimi pravili. V poglavju 4 opišemo zasnovo in implementacijo razvitega sistema za validacijo vnosov, v poglavju 5 implementiran sistem preizkusimo s pomočjo zato izdelane spletne aplikacije, zaključke pa podamo v poglavju 6.

Poglavje 2

Validacija vnosov v spletnih aplikacijah

Validacija vnosov je zelo pomemben del razvoja spletnih aplikacij. Z validacijo zagotovimo, da podatki, ki jih vnese uporabnik preko spletnega vmesnika, ustrezajo omejitvam pri vnosu, ki so bile določene med razvojem aplikacije, ali pa so v okviru omejitev iz poslovnega okolja. Z validacijo povečamo varnost in stabilnost aplikacije ter zagotovimo pravilno delovanje [7].

2.1 Varnost spletnih aplikacij

Validacija vnosov preprečuje, da bi napadalec v aplikacijo vnesel potencialno škodljive podatke. Z ustreznim oblikovanjem vnosov lahko napadalec v aplikacijo podtakne ukaze, ki lahko vplivajo na potek izvajanja aplikacije. Tako lahko napadalec pridobi dostop do zaupnih podatkov, ponaredi rezultate, ki jih vrača aplikacija, ali celo povzroči prenehanje njenega delovanja.

Uporabniški ukazi so podatki, ki jih uporabnik vnese in imajo neposredni vpliv na izvajanje aplikacije. Glede na način vnosa ukaze ločimo na tekstovne ukaze, miškine klike, dotike zaslona, zvokovne ukaze, datoteke ipd. Veliko programskih defektov je posledica napačne obdelave ukazov. Aplikacija mora biti sposobna pravilno obdelati in se ustrezno odzvati na vse

možne uporabnikove ukaze, saj le tako zagotovimo pravilno delovanje v vseh možnih scenarijih. V zadnjem času se hitro širi uporaba glasovnih ukazov, ki jih je zaradi narave govora težje pravilno obdelati, zato je pomen pravilne validacije vnosov pri glasovnih ukazih še večji [13].

Aplikacija mora vse zunanje podatke obravnavati kot potencialno škodljive. Vsak podatek in ukaz, ki vstopi v aplikacijo, obravnavamo kot nevarnega, dokler nismo prepričani v njegovo neškodljivost. Na enak način moramo obravnavati tudi zunanje podatke, kamor spadajo uporabniški vnosi, podatki prejeti preko mreže ter podatki, ki jih aplikacija prejme od drugih procesov. Vhodni podatki ne smejo imeti nobenih nepričakovanih in nezaželenih vplivov na aplikacijo. Aplikacija mora tako vse prejete podatke obdelati, jih preučiti in ugotoviti, ali ustrezajo vnaprej postavljenim validacijskim zahtevam. Takemu načinu validacije pravimo validacija z belim seznamom (*whitelisting*). Pri tej vrsti validacije aplikacija sprejme le tiste vnose, ki ustrezajo pogojem na belem seznamu. Poznamo tudi validacijo s črnim seznamom (*blacklisting*), pri kateri aplikacija sprejme vse vnose razen tistih, ki ustrezajo pogojem na črnem seznamu. Pristop s črnim seznamom je manj primeren, saj moramo črni seznam redno vzdrževati, pri tem pa nanj ne moremo vključiti še neodkritih ranljivosti, prav tako pa lahko pri vzdrževanju pride do napak in tako kak škodljiv vnos pozabimo uvrstiti na seznam [20].

Najpogostejša principa napadov na spletne aplikacije sta vrivanje SQL (*SQL Injection*) [8] in napad skriptnega izvajanja za več mest (*Cross-Site Scripting - XSS*) [11]. V napadih z vrivanjem SQL napadalec kot del vnosa v aplikacijo poda veljaven SQL stavek, s katerim lahko pride do zaupnih podatkov ali povzroči drugo poslovno škodo. Pri XSS napadu napadalec podtakne zlonamerno programsko kodo na spletno stran. Podtaknjena koda se nato izvaja v brskalnikih uporabnikov, ki obiščejo spletno stran. V [19] so avtorji pokazali, da je z ustrezno validacijo vnosov mogoče preprečiti večino napadov XSS in napadov vrivanja SQL na obsežnem naboru znanih ranljivih spletnih aplikacij. Ustrezno validacijo zagotovimo z dvema konceptoma validacije, ki jo bomo predstavili v nadaljevanju: validacije vnosov na osnovi

preverjanja podatkovnih tipov in poslovne validacije vnosov.

2.2 Validacija vnosov na osnovi preverjanja podatkovnih tipov

V spletni aplikaciji je potrebno zagotoviti, da so vsi vneseni podatki pravilnega podatkovnega tipa. Pri vnosih, ki pričakujejo numerične podatke, s to vrsto validacije na primer omogočimo nemoteno izvajanje matematičnih operacij nad vnesenimi podatki. Pri nizih lahko preverjamo njihovo dolžino in s tem izločimo nize, ki bi bili predolgi za zapis v bazo ali nadaljnjo obdelavo.

Nekaj tipičnih primerov podatkovnih tipov in pripadajoče validacijske logike v poslovnih okoljih:

- **Število** - Preverimo, ali vnesen podatek predstavlja veljavno število. Upoštevamo lahko tudi, ali je podatek celo število ali ne, ali je lahko negativno, kakšna je največja in najmanjša dovoljena vrednost ipd.

- **Niz** - Preverimo, ali niz vsebuje samo dovoljene znake. V nizih praviloma ne dovolimo posebnih znakov, kot so narekovaji, dvopičja, podpičja, podčrtaji ipd. S tem se obvarujemo pred napadi s podtikanjem zlonamerne kode.

- **Datum** - Preverimo, ali vnesen podatek predstavlja veljaven datum. Potrebno je biti pozoren na število dni v določenem mesecu in na prestopna leta. Pri datumu, ki predstavlja leto rojstva, lahko omejimo najmanjšo možno letnico.

- **IBAN številka** - Potrebno je preveriti dolžino številke, ustreznost znakov, ki jo sestavljajo, in veljavnost kontrolne številke.

2.3 Poslovna validacija vnosov

Z uporabo poslovne validacije vnose validiramo na osnovi vsebine. Validacijska pravila za določeno vnosno polje so lahko odvisna od vrednosti ostalih polj v obrazcu. Vrednost nekega polja je tako lahko odvisna od vrednosti nekega drugega polja. Tak primer je polje, ki hrani dan v mesecu. Njegova vrednost je odvisna od vrednosti polja, ki hrani mesec. Podobna odvisnost velja pri obveznosti vnosov polj. Pri tem velja tudi, da je obveznost vnosa nekega polja lahko odvisna od vrednosti nekega drugega polja.

2.4 Pristopi k validaciji vnosov

Danes ne obstaja standardizirana rešitev za preverjanje vnosov v spletnih aplikacijah. Validacija na strani odjemalca je implementirana s tehnologijami, ki jih podpirajo spletni brskalniki. Trenutno najpopularnejša rešitev je implementacija validacije s programskim jezikom JavaScript, saj je jezik zelo razširjen in dobro podprt v večini modernih brskalnikov [9]. V novejših aplikacijah, izdelanih s tehnologijo HTML5, lahko del validacije vnosnih polj implementiramo z namenskimi atributi vnosnih polj, ki jih za ta namen uvaja standard HTML5 [23]. Na strani strežnika se validacija implementira z jezikom, v katerem je razvita spletna aplikacija [12] [15].

Primer deklarativnega sistema za validacijo vnosov so predstavili avtorji v [5]. To so dosegli z razširitvijo jezika HTML z namenskimi značkami, ki omogočajo deklarativen zapis validacijske logike. Njihova rešitev rešuje samo problem validacije na strani odjemalca, ne naslavlja pa problema validacije vnosov na strani strežnika. V ogrodju JavaServer Faces (JSF) lahko validacijsko logiko zapišemo deklarativno, z anotacijami na nivoju programske kode, definirana pravila pa se uporabijo za izvajanje validacije tako na strani odjemalca, kot na strani strežnika [2]. V našem primeru pa smo takšen način še dodatno razširili in omejitve izvzeli iz aplikacije in jih definirali v obliki poslovnih pravil.

V nadaljevanju bomo podrobneje predstavili načine implementacije va-

lidacijske logike na odjemalcu in strežniku ter izpostavili probleme, ki se pojavijo ob vzdrževanju aplikacij.

2.4.1 Validacija na odjemalcu in na strežniku

Ustaljena praksa je, da se validacija vnosov implementira na strani odjemalca in na strani strežnika. Z validacijo na strani odjemalca se izognemo nepotrebni komunikaciji s strežnikom v primeru, da vneseni podatki ne ustrezajo zahtevam. S tem izboljšamo odzivnost aplikacije in zmanjšamo omrežni promet. Validacijo na strani odjemalca tipično implementiramo s pomočjo JavaScripta, zato ga vzamemo kot primer. Validacijska funkcija se kliče ob poljubnem dogodku vnosnega polja in uporabnika sproti opozarja na nepravilne vnose. JavaScript lahko prav tako skrbi, da oddaja obrazca ni mogoča, dokler niso vsi vnosi veljavni. Programska koda JavaScript se izvaja v uporabnikovem spletnem brskalniku, ločeno od preostalega dela spletne aplikacije. Pri razvoju moramo upoštevati dejstvo, da lahko uporabnik izklopi izvajanje kode JavaScript v brskalniku in s tem zaobide validacijo vnosov na strani odjemalca, zato je potrebno validacijo vnosov implementirati tudi na strani strežnika. Pri tem se pojavijo težave s podvajanjem programske kode, ki pa jih napredna ogrodja za razvoj spletnih aplikacij že rešujejo.

Način implementacije validacije na strani strežnika je odvisen od platforme, ki jo uporabljamo. Naštejmo nekaj tipičnih primerov. V okolju JavaServer Faces (JSF) validacijo na strežniku izvajamo z uporabo vnaprej definiranih validatorjev, z ročno spisanimi razredi za pretvorbo in validacijo, ali pa jih deklarativno navedemo. Okolje ASP.NET uporablja vnaprej definirane validatorje, ki jih dodamo posameznim vnosnim poljem, in možnost vključitve validatorjev v model aplikacije. V Ruby on Rails validatorje vključimo v model. Pri uporabi jezika PHP validacijsko logiko definiramo sami znotraj same aplikacije.

2.4.2 Validacija vnosov pri vzdrževanju aplikacij

Pomemben del življenjskega cikla vsake aplikacije je njeno vzdrževanje. Prav tako kot pri razvoju, je potrebno tudi pri vzdrževanju aplikacije zagotavljati ustrezno validacijo vnosov. Ker je validacijska logika v večini sistemov implementirana na dveh mestih, je vzdrževanje takšnih aplikacij zelo zahtevno. Prav tako so zaradi velike kompleksnosti validacijske logike verjetnejše napake pri implementaciji, ki zmanjšajo varnost aplikacije, ali celo vodijo v napačno delovanje aplikacije. V [18] avtorji navajajo, da se 40% ranljivosti pojavi v prvi izdaji aplikacije, preostalih 60% pa se pojavi šele ob dodajanju novih funkcionalnosti.

Idealen sistem za validacijo vnosov v spletnih aplikacijah bi bil torej tak, pri katerem bi lahko validacijsko logiko definirali na enem mestu, ločeno od programske kode, s pomočjo deklarativnega programskega jezika. Sistem bi potem sam na podlagi teh definicij generiral programsko kodo za validacijo na odjemalcu in kodo za validacijo na strežniku. Tak sistem bi olajšal tudi vzdrževanje aplikacije, saj bi bila vsa validacijska logika zbrana na enem mestu. Skrbnik aplikacije bi validacijska pravila spreminjal v namenski aplikaciji, sistem bi pa poskrbel za dinamično generiranje programske kode validatorjev na odjemalcu in strežniku. Tak koncept smo upoštevali pri razvoju predlaganega sistema za validacijo vnosov, predstavljenega v tej diplomski nalogi.

Poglavje 3

Sistemi za upravljanje s poslovnimi pravili

Sistemi za upravljanje s poslovnimi pravili (SUPP) omogočajo deklarativen zapis validacijske logike v obliki poslovnih pravil. Z njimi lahko definiramo pogoje, pri katerih je vnos veljaven, in se izognemo klasičnemu pristopu zapisa validacijske logike s proceduralnimi programskimi jeziki. Prav tako lahko validacijsko logiko ločimo od programske kode in se s tem izognemo ponovnemu prevajanju aplikacije ob spremembi posameznega pravila. Zaradi teh lastnosti sistem za upravljanje s poslovnimi pravili predstavlja osnovo našega sistema za validacijo vnosov.

3.1 Poslovna pravila

S poslovnimi pravili se največkrat srečujemo v poslovnih okoljih. Poslovno pravilo definira določen aspekt poslovanja podjetja ali korporacije. Primer poslovnega pravila bi lahko bil naslednji: "Če je stranka mlajša od 30 let in ima status študenta, ji pripada študentski popust." Razne institucije pri poslovanju redno uporabljajo poslovna pravila. Banke jih na primer med drugim uporabljajo pri obravnavi vlog za kredit, države imajo kompleksna poslovna pravila za dodeljevanja viz priseljencem, v večjih trgovinah pa jih

npr. uporabljajo za uravnavanje zalog [22].

Vsako poslovno pravilo je sestavljeno iz dveh delov, in sicer levega in desnega. Levi del pravila sestavljajo pogoji in omejitve, desni del pa predstavljajo akcije, ki se izvedejo, ko so pogoji v levem delu zadoščeni.

Jezik, v katerem zapisujemo poslovna pravila, je deklarativen programski jezik. Z njim ne definiramo, kako naj se program izvaja, temveč povemo, kaj naj se zgodi, ko so določeni pogoji izpolnjeni.

Poslovno pravilo (3.1) prikazuje primer tipičnega programa "Pozdravljen svet", zapisanega v jeziku za zapis pravil DRL, ki je del okolja Drools, katerega podrobneje opišemo v podpoglavju 3.2.1.

Pravilo 3.1 *Pozdravljen svet*

```
rule "Pozdravljen svet"
when
    eval(true)
then
    System.out.println("Pozdravljen svet");
end
```

Poslovna pravila izvajamo nad množico dejstev. Dejstva so instance objektov, s katerimi predstavimo entitete iz realnega sveta in njihove atribute. V nadaljevanju opis poslovnih pravil prikažemo na primeru depozita v okolju Drools.

Depozitu obrestno mero določimo glede na njegovo ročnost in vrednost. Obrestna mera za depozit vreden 250€ in vezan za 31 dni je npr. enaka 1.15%. To lahko zapišemo kot poslovno pravilo (3.2). *Depozit* je javanski razred, predstavljen z izvorno kodo (3.1). Pravilo (3.2) v delovnem spominu poišče vse instance objekta *Depozit*, ki ustrezajo podanim kriterijem, in jim določi ustrezno obrestno mero.

Izvorna koda 3.1 *Razred Depozit*

```
class Depozit(){

    private int vrednost;
    private int rocnost;
    private double obrestnaMera;

    public Depozit(){
    }

    //ustrezne getter in setter metode

}
```

Pravilo 3.2 *Obrestna mera depozita*

```
rule "Depozit"
when
    $d : Depozit(vrednost >= 250,
                 vrednost < 5000,
                 rocnost >= 31,
                 rocnost < 60)
then
    $d.setObrestnaMera(1.15);
end
```

3.2 Opis sistemov za upravljanje s poslovnimi pravili

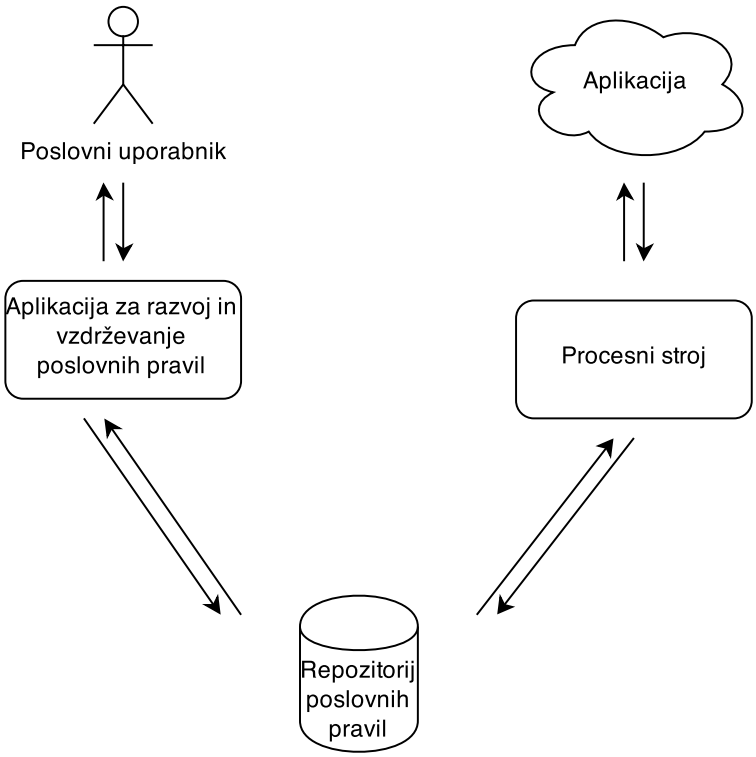
Poslovna pravila so v večini primerov zapisana v naravnem jeziku. Ko jih želimo uporabiti v aplikaciji, jih je potrebno pretvoriti iz naravnega jezika v splošnonamenski programski jezik. Ta pretvorba je velikokrat zahtevna

in zahteva veliko sodelovanja med poslovno in programersko sfero. Sistemi za upravljanje s poslovnimi pravili (*Business Rule Management System - BRMS*) omogočajo zapis poslovnih pravil v jeziku, ki je zelo podoben naravnemu, a ga razume tudi računalnik. Sistemi za upravljanje s poslovnimi pravili nam omogočajo, da poslovno znanje zapišemo v obliki "kaj želimo" in ne, "kako želimo".

Velikokrat v podjetjih težavo predstavlja tudi razdrobljenost znanja. Znanje namreč ni zapisano na enem mestu, temveč je porazdeljeno v dokumentaciji, glavah zaposlenih in raznih aplikacijah. Sistemi za upravljanje s poslovnimi pravili omogočajo hrambo vsega znanja podjetja v centralnem repozitoriju pravil. Ta pravila lahko urejajo in dopolnjujejo vsi zaposleni z ustreznimi pravicami [16].

Z vpeljavo sistema za upravljanje s poslovnimi pravili v poslovno aplikacijo poenostavimo zapis poslovne logike v aplikaciji in ga strnemo na eno mesto. Tipično aplikacijo sestavlja predstavitveni nivo, poslovni nivo in podatkovni nivo. Predstavitveni nivo skrbi za interakcijo z uporabnikom, podatkovni pa za shranjevanje in posredovanje podatkov. Poslovni nivo vsebuje poslovno logiko podjetja, ki je skupek poslovnih pravil. Če je poslovna logika zapisana kot množica programske kode v različnih delih aplikacije, je vzdrževanje aplikacije zaradi razdrobljenosti zelo zahtevno. Prav tako je za spremembo poslovnih pravil v aplikaciji potreben poseg programerja. Z uporabo sistemov za upravljanje s poslovnimi pravili poslovna pravila niso več del programske kode, temveč so ločena in shranjena v repozitoriju pravil. Aplikacija nato na poslovnem nivoju dostopa do pravil in jih izvaja nad dejstvi.

Tipičen sistem za upravljanje s poslovnimi pravili sestavljajo aplikacija za razvoj in vzdrževanje poslovnih pravil (*Authoring and Rules Management Application*), repozitorij poslovnih pravil (*Rules Repository*) in procesni stroj (*Business Rules Engine*). Aplikacija za razvoj in vzdrževanje poslovnih pravil poslovnim analitikom omogoča enostavno spreminjanje obstoječih pravil, tako da odražajo poslovna pravila podjetja. Prav tako omogoča dodajanje



Slika 3.1: Zgradba sistema za upravljanje s poslovnimi pravili

novih pravil. Aplikacija je namenjena poslovni sferi, ki lahko poslovna pravila ureja brez pomoči programerske sfere. Poslovna pravila hrani repozitorij poslovnih pravil, procesni stroj pa jih izvršuje. Zgradbo tipičnega sistema prikazuje slika 3.1.

3.2.1 Drools

V diplomski nalogi smo sistem za upravljanje s poslovnimi pravili uporabili za upravljanje z validacijskimi pravili. Izbrali smo odprtokoden sistem za upravljanje s poslovnimi pravili Drools [1] [6] [3]. Za njegovim razvojem stoji skupnost JBoss.org. Izbrali smo ga zaradi njegove odprtokodne narave in dobre integracije z okoljem Java EE. Alternativna izbira bi bila komercialna produkta IBM BRMS ali Oracle Business Rules.

Drools sestavljajo pogon za izvajanje poslovnih pravil, spletna aplikacija za razvoj in vzdrževanje poslovnih pravil Drools Workbench in dodatek za razvojno okolje Eclipse namenjen kompleksnejšemu razvoju.

Sistem Drools uporabljamo v navezi s programskim jezikom Java, kar nam omogoča predstavitev dejstev z javanskimi razredi, ki jih nato uporabimo pri zapisu pravil. Procesni stroj sistema Drools lahko vključimo v poljubno aplikacijo, razvito s programskim jezikom Java, ki se lahko izvaja lokalno ali pa na oddaljenem strežniku.

Poslovna pravila v okolju Drools zapisujemo z jezikom za zapis pravil (*Drools Rule Language - DRL*). Shranimo jih v datoteko s končnico *.drl*. Možen je tudi zapis s pomočjo odločitvenih tabel, ki jih lahko urejamo s poljubnim programom za urejanje preglednic, kot je npr. Microsoft Office Excel. Vsaka vrstica v tabeli predstavlja eno poslovno pravilo. Odločitvene tabele je pametno uporabiti v okoljih, kjer veliko pravil sledi enakemu vzorcu.

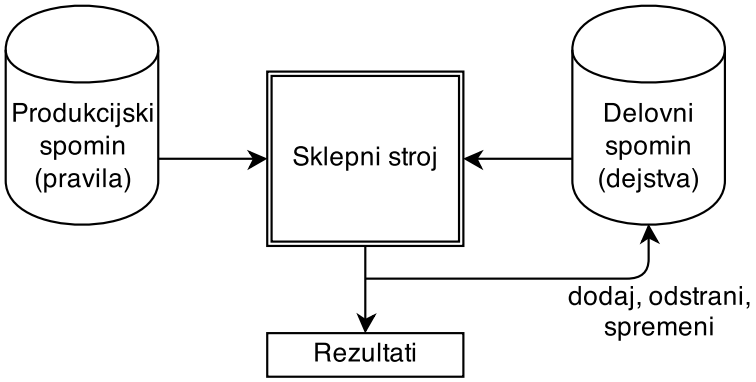
V naslednjih podpoglavjih bomo predstavili zgradbo in delovanje procesnega stroja ter algoritem Rete, kateraga izpeljanke uporablja procesni stroj sistema Drools.

3.3 Procesni stroj

Procesni stroj je osrednji del sistema za upravljanje s poslovnimi pravili. Stroj prejme zahtevo za obdelavo, nato poišče pravila, na katera se zahteva nanaša, ter jih izvrši. Rezultate izvajanja nato v obliki sklepa vrne aplikaciji.

Procesni stroj poleg rezultatov vrne tudi sledljivost sklepa. To pomeni, da lahko za vsak rezultat, ki ga stroj vrne, korak po korak pogledamo, kako je procesni stroj prišel do končnega sklepa.

Prikaz znanja in sklepanje (*Knowledge Representation and Reasoning - KRR*) je področje umetne inteligence, ki se ukvarja s tem, kako znanje o svetu predstaviti v obliki, ki bo razumljiva tudi računalniku. Nad tako zapisanimi podatki lahko nato računalniški sistemi rešujejo kompleksne probleme, kot je na primer medicinska diagnostika. Tudi procesni stroj deluje kot sistem



Slika 3.2: Zgradba procesnega stroja

KRR.

Procesni stroj v osnovi sestavljajo trije deli: produkcijski spomin, delovni spomin in sklepni stroj. Produkcijski spomin hrani poslovna pravila, v delovnem spominu so shranjena dejstva, sklepni stroj pa izvršuje poslovna pravila iz produkcijskega spomina nad dejstvi, ki se nahajajo v delovnem spominu. Akcije v poslovnih pravilih lahko dodajajo dejstva v delovni spomin, jih odstranjujejo ali spreminjajo. Zgradbo procesnega stroja prikazuje slika 3.2.

Procesni stroj sistema Drools, ki smo ga uporabili pri implementaciji sistema za validacijo vnosov, za izvrševanje poslovnih pravil do verzije 5 uporablja razširjen algoritem Rete, imenovan ReteOO. Od Drools verzije 6 naprej uporablja algoritem PHREAK, ki predstavlja logično nadaljevanje algoritma ReteOO, vendar je že v tolikšni meri dopolnjen, da ga več ne uvrščamo med Rete implementacije.

Za osnovno razumevanje delovanja procesnega stroja je dovolj, da smo seznanjeni z algoritmom Rete, ki ga predstavimo v nadaljevanju.

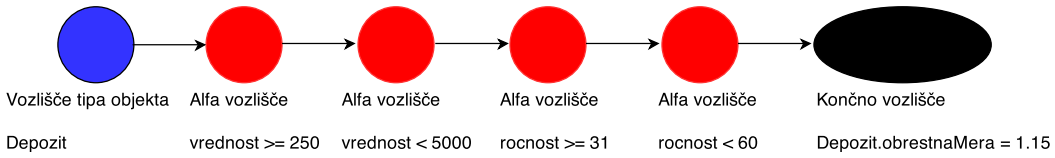
3.4 Algoritem "Rete"

Algoritem Rete je algoritem za primerjanje vzorcev. Njegovo izpeljanko, algoritem PHREAK, pri izvrševanju poslovnih pravil uporablja procesni stroj sistema Drools. Algoritem je razvil Dr. Charles Forgy. Latinska beseda "Rete" pomeni "mrežo" ali "omrežje".

Algoritem Rete lahko razdelimo na dva zaporedna koraka izvajanja delovanja: prevajanje pravil in izvrševanje pravil. V postopku prevajanja pravil algoritem iz pravil shranjenih v produkcijskem spominu gradi razločevalno omrežje. To omrežje se uporablja za filtriranje dejstev, ki vstopajo v sistem. Dejstvo se ujema z vozliščem, če zadostuje omejitvam, ki jih predstavlja vozlišče. Vozlišča na vrhu omrežja imajo veliko ujemanj. Bolj kot se bližamo dnu omrežja, manj je ujemanj. Na dnu omrežja so tako imenovana končna vozlišča, ki predstavljajo akcije.

Algoritem pozna pet tipov vozlišč:

- **Korensko vozlišče** (*Root Node*) - Vozlišče, preko katerega vsi objekti vstopajo v omrežje.
- **Vozlišče tipa objekta** (*Object Type Node*) - Zagotavlja, da pogon ne opravlja več dela, kot je potrebno. Takoj, ko objekt zapusti korensko vozlišče, nadaljuje pot po ujemajočem vozlišču tipa objekta.
- **Alfa vozlišče** (*Alpha Node*) - Alfa vozlišča se uporabljajo za vrednotenje omejitev pravila. Za vsako omejitev na objektu obstaja v omrežju pripadajoče alfa vozlišče.
- **Beta vozlišče** (*Beta Node*) - Beta vozlišča se uporabljajo za primerjanje enakosti dveh objektov.
- **Končno vozlišče** (*Terminal node*) - Končna vozlišča se nahajajo na dnu omrežja. Dosežemo jih, če so bile zadoščene vse omejitve pravila. Če pravilo vsebuje pogoj *ali*, ima lahko omrežje več končnih vozlišč. Končna vozlišča predstavljajo akcije v pravilu.



Slika 3.3: Primer omrežja za pravilo (3.2)

Procesni stroj poslovna pravila izvršuje tako, da najprej zgradi razločevalno omrežje, nato pa dejstva vstavi v korensko vozlišče zgrajenega omrežja. Dejstva se potem skladno s pravili širijo po omrežju. Če dejstvo zadosti vsem omejitvam, doseže končno vozlišče, ki predstavlja desno stran pravila, torej hrani akcije.

Primer omrežja za poslovno pravilo (3.2) prikazuje slika 3.3.

Poglavje 4

Zasnova in implementacija sistema za validacijo vnosov z uporabo sistemov za upravljanje s poslovnimi pravili

Omenili smo že, da ima validacija vnosov, kot se je lotevamo danes, naslednji pomanjkljivosti:

- pomanjkanje namenskih, standardnih sistemov namenjenih validaciji vnosov, ki so neodvisni od platforme, v kateri je razvita spletna aplikacija;
- validacijska logika je zapisana kot del aplikacije, kar otežuje njeno posodabljanje in vzdrževanje, zato bi bilo bolje, če bi bi bila zapisana posebej.

Cilj tega diplomskega dela je zasnova in gradnja sistema za validacijo vnosov, ki v celoti odpravlja ti pomanjkljivosti.

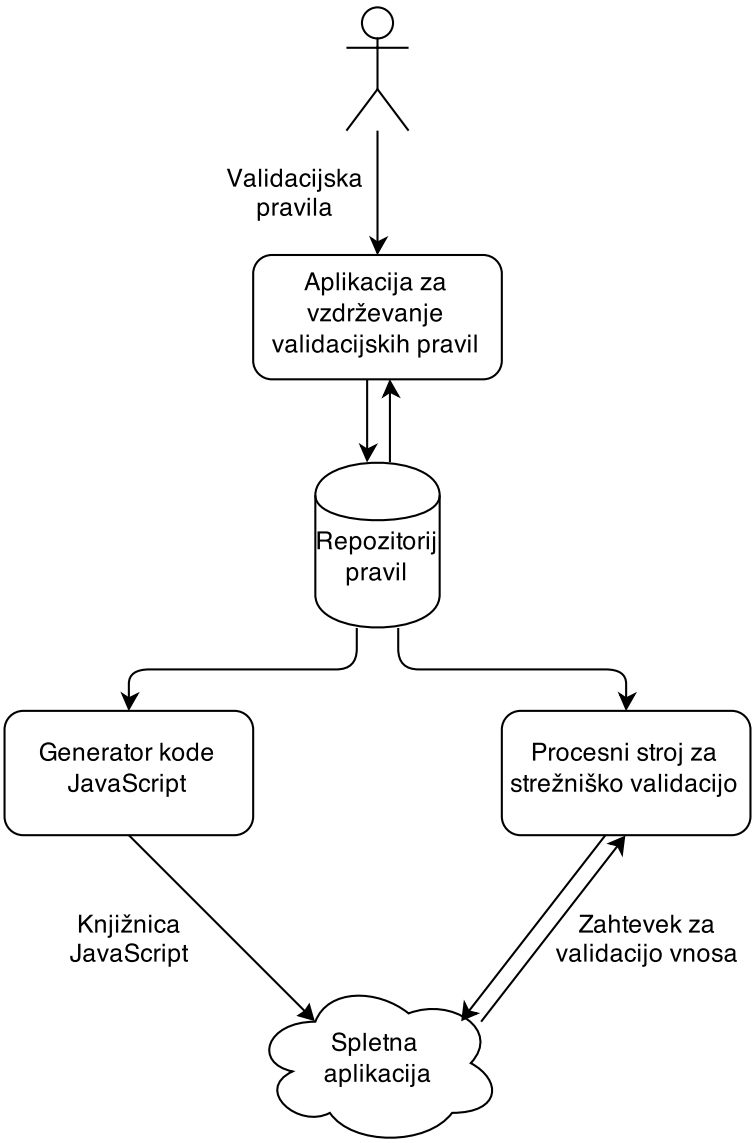
Zasnovan sistem za validacijo vnosov temelji na sistemu za upravljanje s poslovnimi pravili. Validacijska logika je v sistemu predstavljena kot množica poslovnih pravil, ki definirajo veljavnost vnosov.

Validacijska pravila v zasnovan sistem dodajamo in vzdržujemo s pomočjo namenske spletne aplikacije, ki smo jo razvili kot del sistema za validacijo vnosov. Uporaba aplikacije je enostavna in predvideva uporabo s strani poslovnih analitikov brez večjih posegov programerjev. Vsa validacijska pravila so shranjena na enem mestu, v centralnem repozitoriju.

Razviti sistem validacijska pravila iz centralnega repozitorija uporabi za dinamično generiranje validacijske logike za stran odjemalca. Generator validacije za odjemalca je v sistemu implementirana kot preprost prevajalnik. Ta prevede validacijsko logiko iz jezika poslovnih pravil v knjižnico JavaScript, ki jo nato vključimo v spletno stran. Knjižnica se avtomatsko posodablja ob vsaki spremembi poslovnih pravil in odraža validacijsko logiko shranjeno v centralnem repozitoriju. Implementacija validacije na strani strežnika je odvisna od uporabljene tehnologije. Ob izvajanju validacije na strani strežnika, le-ta komunicira s sistemom preko spletnih storitev. Sistemu pošljemo tip vnosa in njegovo vrednost. Sistem nato odgovori, ali je vnos pravilen ali ne.

Shemo razvitega sistema za validacijo vnosov s pomočjo sistemov za upravljanje s poslovnimi pravili prikazuje slika 4.1. Sistem sestavljajo javanski razred *Input*, ki ga uporabimo pri zapisu poslovnih pravil v okolju Drools, spletna aplikacija za zapisovanje in urejanje validacijskih pravil, repozitorij pravil, za implementacijo validacije pa skrbita generator kode JavaScript za stran odjemalca ter procesni stroj za stran strežnika. Posamezne gradnike sistema podrobneje opišemo v nadaljevanju.

Sistem smo implementirali s platformo Java Enterprise Edition [21] [10]. Java EE je platforma, namenjena razvoju kompleksnejših aplikacij za uporabo v poslovnih okoljih. Omogoča izdelavo skalabilnih, distribuiranih, večnivojskih strežniških aplikacij. Predstavlja univerzalen standard v poslovnih informacijskih sistemih. Platformo smo izbrali zaradi njene razširjenosti v poslovnih okoljih ter dobre podpore skupnosti.



Slika 4.1: Shema sistema za validacijo vnosov

4.1 Razred Input

V sistemih za upravljanje s poslovnimi pravili so dejstva predstavljena kot instance objektov. Za zapis validacijske logike s poslovnimi pravili smo razvili javanski razred *Input*, ki predstavlja uporabnikov vnos. Razred vsebuje atribut *value* tipa niz, ki hrani vrednost vnosa. Atribut *type* predstavlja tip vnosa. Tip je lahko poljuben niz. V nadaljevanju tipu vnosa priredimo ustrezno validacijsko logiko. Atribut *valid* predstavlja veljavnost vnosa. Za vsak nov vnos predpostavimo, da ni veljaven.

Razredu smo definirali dodatne atribute in metode, ki po našem mnenju predstavljajo najpomembnejše lastnosti vnosa, potrebne pri implementaciji validacije:

- **length** - Atribut vrača dolžino vnosa.
- **matches(String s)** - Funkcija preveri, ali se vrednost vnosa ujema z regularnim izrazom, podanim s spremenljivko *s*.
- **isEmail()** - Funkcija preveri, ali je vrednost vnosa veljaven email naslov.
- **isInteger()** - Funkcija preveri, ali je vrednost vnosa veljavno celo število.
- **isNumber()** - Funkcija preveri, ali je vrednost vnosa veljavno število.
- **numericValue** - Vrne numerično vrednost vnosa.
- **nAlphanumeric** - Vrne število alfanumeričnih znakov v nizu.
- **nUpperCase** - Vrne število velikih črk v nizu.
- **nLowerCase** - Vrne število malih črk v nizu.
- **nDigits** - Vrne število števk v nizu.
- **containsWhitespace()** - Funkcija preveri, ali niz vsebuje bele znake.
- **isDate()** - Funkcija preveri, ali je vnos veljaven datum.

4.2 Validacijska pravila

Validacijska pravila v sistemu zapisujemo v obliki poslovnih pravil. Za opis lastnosti vnosa uporabimo razred *Input*. Validacijska pravila zapisujemo v jeziku za zapis poslovnih pravil sistema Drools, ki smo ga predstavili že v podpoglavju 3.2.1.

V nadaljevanju podajamo nekaj tipičnih primerov za zapis validacijskih pravil v našem okolju. Primer poslovnega pravila za vnos, ki predstavlja starost osebe, prikazuje primer 4.1. Njegov formalen zapis prikazuje pravilo (4.1).

Primer 4.1 *Vnos tipa starost je pravilen, če vrednost vnosa predstavlja naravno število, če je numerična vrednost vnosa pozitivna in manjša od 130.*

Pravilo 4.1 *Zapis pravila za primer 4.1*

```
rule "starost"
when
        $i:Input(type == "starost",
                 isInteger(),
                 numericValue >= 0,
                 numericValue < 130
                 )

then
        $i.setValid(true);
end
```

Poglejmo si primere validacijskih pravil za vnose tipa *ime* (4.2), *email* (4.3) in *pozitivno število* (4.4). Vnos tipa *ime* je pravilen, ko je njegova dolžina večja od ena, sestavljajo pa ga velika črka, ki ji sledi poljubno mnogo malih črk, kar zapišemo z regularnim izrazom. Vnos tipa *email* je pravilen, ko funkcija *isEmail()* razreda *Input* vrne vrednost *true*. Vnos tipa *pozitivno število* je pravilen, ko je vrednost vnosa veljavno število in je hkrati večje od nič.

Pravilo 4.2 *Primer pravila za vnos tipa ime*

```
rule "ime"
when
        $i:Input( type == "ime",
                  length > 1,
                  matches("^[A-Z][a-z]+$")
                  )

then
        $i.setValid(true);
end
```

Pravilo 4.3 *Primer pravila za vnos tipa email*

```
rule "email"
when
        $i:Input( type == "email",
                  isEmail()
                  )

then
        $i.setValid(true);
end
```

Pravilo 4.4 *Primer pravila za vnos tipa pozitivno število*

```
rule "pozitivno stevilo"
when
        $i:Input( type == "number",
                  isNumber(),
                  numericValue > 0
                  )

then
        $i.setValid(true);
end
```

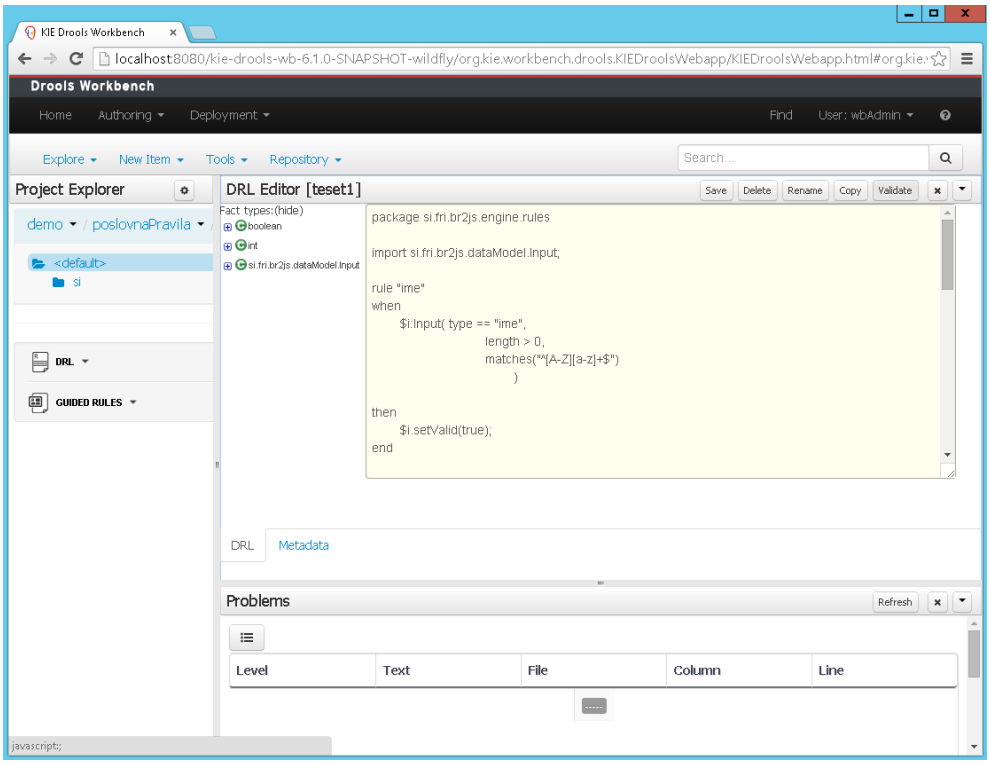
4.3 Aplikacija za vzdrževanje validacijskih pravil

Sistem vsebuje aplikacijo, namenjeno vzdrževanju validacijske logike, ki je v obliki poslovnih pravil shranjena v centralnem repozitoriju. Najprimerneje je, da je aplikacija implementirana kot spletna aplikacija. To zagotavlja neodvisnost od platforme in operacijskega sistema ter omogoča uporabo na poljubnem računalniku brez predhodne namestitve.

Aplikacija je namenjena pregledovanju, urejanju in dodajanju validacijskih pravil. Uporaba aplikacije mora biti enostavna, tako da lahko validacijska pravila urejajo zaposleni brez pomoči programerjev.

V naši rešitvi smo za ta namen uporabili obstoječo aplikacijo Drools Workbench. Aplikacija je del sistema za upravljanje s poslovnimi pravili Drools. Aplikacija je odprtokodna in prosto dostopna. S spleta prenesemo arhivsko datoteko tipa *war*, ki jo namestimo na aplikacijski strežnik.

Z aplikacijo lahko ustvarimo datoteke s poslovnimi pravili, ki jih nato urejamo s pomočjo vgrajenega urejevalnika. Urejevalnik pravil ima vgrajen validator kode, ki preveri, ali so pravila sintaktično pravilna. Zaslonsko masko urejevalnika pravil v aplikaciji Drools Workbench prikazuje slika 4.2.



Slika 4.2: Zaslonska maska orodja Drools Workbench

Zraven tekstovnega urejevalnika Workbench vsebuje tudi voden urejevalnik pravil, ki uporabnika z izbirnimi meniji vodi skozi postopek oblikovanja poslovnega pravila.

V aplikacijo lahko uvozimo tudi artefakte iz repozitorija Maven. V našem primeru smo uvozili razred Input, na katerega se lahko sklicujemo v poslovnih pravilih.

4.4 Repozitorij pravil

V repozitoriju pravil je shranjena vsa validacijska logika v obliki poslovnih pravil. Repozitorij zagotavlja upravljanje z verzijami validacijskih pravil.

Do repozitorija dostopajo vsi bistveni gradniki sistema. Aplikacija za urejanje pravil vanj dodaja nova pravila ter spreminja že obstoječa. Kompo-

nenta za generiranje kode JavaScript pravila shranjena v repozitoriju prevede v kodo JavaScript. Procesni stroj pravila iz repozitorija izvaja ter rezultate vrača aplikaciji, ki je poslala zahtevo.

Pri implementaciji smo uporabili repozitorij pravil, ki je del aplikacije Drools Workbench, ki smo jo podrobneje opisali v podpoglavju 4.3. Pravila, ki jih zapišemo s pomočjo aplikacije, shranimo v lokalen repozitorij. Do teh pravil nato dostopata generator kode JavaScript in procesni stroj.

4.5 Generator kode JavaScript

Generator kode JavaScript iz validacijske logike, shranjene v repozitoriju pravil, dinamično generira funkcijo JavaScript, namenjeno validaciji vnosov na strani odjemalca. Generirano kodo nato vključimo v spletno stran.

Generator deluje kot preprost prevajalnik. Validacijska pravila razčleni na posamične omejitve ter jih nato zapiše s kodo JavaScript. Izvorna koda (4.1) prikazuje psevdokod algoritma, ki ga uporablja generator. Pravilo predstavlja validacijsko pravilo, zapisano v okolju Drools. Omejitev predstavlja posamezno omejitev, ki jo definiramo objektu *Input* v validacijskem pravilu. Funkcija *prevediOmejitev* (4.2) posamezno omejitev prevede v kodo JavaScript, ki implementira ustrezno validacijsko logiko.

Izvorna koda 4.1 *Algoritem generatorja kode JavaScript*

```
function prevedi(pravilo){
    String validiraj;
    for omejitev in pravilo {
        validiraj.append(prevediOmejitev(omejitev);
    }
    return validiraj;
}
```

Izvorna koda 4.2 *Pseudo koda funkcije prevediOmejitev*

```
function prevediOmejitev(omejitev){
...
// prevedi omejitev tipa isEmail
if omejitev == isEmail{
        String js = "";
        js = js.concat("/^[A-Z0-9._%+-]+@[A-Z0-9.-]
        +\\.[A-Z]{2,4}$/" +
        ".test(value.toUpperCase())");

        return js;
}
...
}
```

Generator vrača funkcijo *validiraj*, ki jo dodelimo elementom *input* na spletni strani. Parameter *name* elementa določa tip vnosa. Izvorna koda (4.3) prikazuje primer dodajanja validacije elementu *input*. Elementu tipa *text* smo dodelili funkcijo *validiraj*, ki se kliče ob dogodku *onBlur*. Validacija na strani odjemalca se torej izvede, ko uporabnik premakne kurzor iz vnosnega polja, in uporabnika opozori ob morebitnem neveljavnem vnosu.

Izvorna koda 4.3 *Dodajanje validatorja elementu input*

```
<input type="text" name="ime" onblur="validiraj(this)">
```

Generator kode preslika validacijsko pravilo (4.5) v funkcijo (4.4).

Pravilo 4.5 *Primer validacijskega pravila za vnos tipa starost*

```
rule "starost"
when
        $i:Input(type == "starost",
                 isInteger(),
                 numericValue > 0
                 )

then
        $i.setValid(true);
end
```

Izvorna koda 4.4 *Primer preslikave pravila* (4.5)

```
function validiraj(sender) {
        var value = sender.value;
        var name = sender.name;

        if(name=="starost"){
          if( /^-?\d+$/.test(value) &&
              parseFloat(value)>0) {
                sender.style.backgroundColor="white";
                return true;
              }else{
                sender.select();
                sender.style.backgroundColor="red";
                return false;
              }
        }

}
```

Generator smo implementirali z uporabo javanskih strežniških zrn, ki

so pomemben sestavni del platforme Java EE. Omogočajo modularno gradnjo poslovnih strežniških aplikacij. Poznamo dva tipa zrn: sejna zrna (*Session Bean*), v katerih se izvaja poslovna logika aplikacije in sporočilna zrna (*Message-Driven Bean*), ki omogočajo asinhrono komunikacijo med gradniki aplikacije. Javanska zrna omogočajo opisno določevanje vedenja. Vedenje zrn tako določamo z deklarativnimi metapodatki in tako prilagodimo vedenje zrn brez implementiranja logike. Zrna se izvajajo na aplikacijskem strežniku, ki zagotavlja okolje za izvajanje vsebnikov Java EE. Aplikacijski strežnik zraven izvajanja aplikacij implementira tudi storitve kot so gručenje, uravnavanje obremenitve in obravnavanje izpadov. Za izdelavo praktičnega primera smo uporabili odprtokodni aplikacijski strežnik WildFly, ki ga razvija skupnost JBoss.org.

4.6 Procesni stroj

Procesni stroj je namenjen preverjanju pravilnosti vnosov na strani strežnika. Je del sistema za upravljanje s poslovnimi pravili, ki izvršuje poslovna pravila. V našem primeru ta poslovna pravila predstavljajo validacijsko logiko.

Do procesnega stroja lahko dostopamo iz vseh delov aplikacije. Prav tako je možen klic preko spletne storitve. Validacijo na strani strežnika smo implementirali tako, da validatorji kličejo procesni stroj, ki preveri ustreznost vnosov in validatorjem vrne odgovor.

Pri implementaciji našega sistema smo uporabili procesni stroj, ki je del sistema za upravljanje s poslovnimi pravili Drools. Procesni stroj se podobno kot javanska zrna izvaja na aplikacijskem strežniku. Implementacijo javanskega zrna, v katerem se izvaja procesni stroj, prikazuje izvorna koda (4.5).

Izvorna koda 4.5 *Zrno, v katerem se izvaja procesni stroj Drools*

```
@Stateless
public class FireRulesZrno implements
        FireRulesZrnoRemote, FireRulesZrnoLocal {
...
public Input fireRules(Input i){
    String url; // inicializacija URL repozitorija

    // preberi pravila iz repozitorija
    ReleaseIdImpl releaseId = new ReleaseIdImpl
        ("si.fri.validaija", "Pravila", "LATEST");
    KieServices ks = KieServices.Factory.get();
    ks.getResources().newUrlResource(url);
    KieSession kSession; // inicializacija procesnega
                         stroja


    // dejstvo i vstavimo v delovni spomin
    kSession.insert(i);

    // izvedemo pravila
    kSession.fireAllRules();

    return i;
}
}
```

Poglavje 5

Preizkus zasnovanega sistema

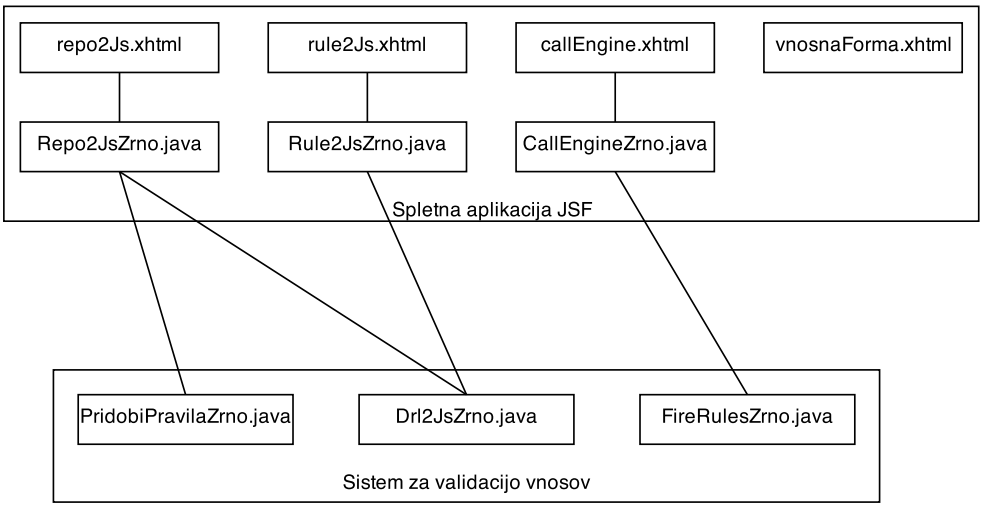
V tem poglavju predstavimo aplikacijo, ki smo jo razvili za testiranje zasnovanega sistema, nato pa opišemo preizkus sistema za validacijo na praktičnem primeru.

5.1 Spletna aplikacija za testiranje zasnovanega sistema

Končna verzija sistema za validacijo vnosov s pomočjo sistemov za upravljanje s poslovnimi pravili predvideva popolno integracijo sistema za validacijo s poljubno spletno aplikacijo.

Za potrebe testiranja in demonstracije smo razvili preprosto spletno aplikacijo, ki omogoča testiranje delovanja komponente za generiranje validacijske kode JavaScript in procesnega stroja za izvajanje validacijskih pravil. Aplikacija omogoča naslednje aktivnosti:

- **Generiranje kode JavaScript** - validacijsko kodo lahko generiramo iz pravil, ki so zapisana v repozitoriju, ali iz poljubnega validacijskega pravila, ki ga vnesemo v spletno aplikacijo.
- **Testiranje generirane kode** - aplikacija vsebuje spletni obrazec, s



Slika 5.1: Zgradba aplikacije za testiranje

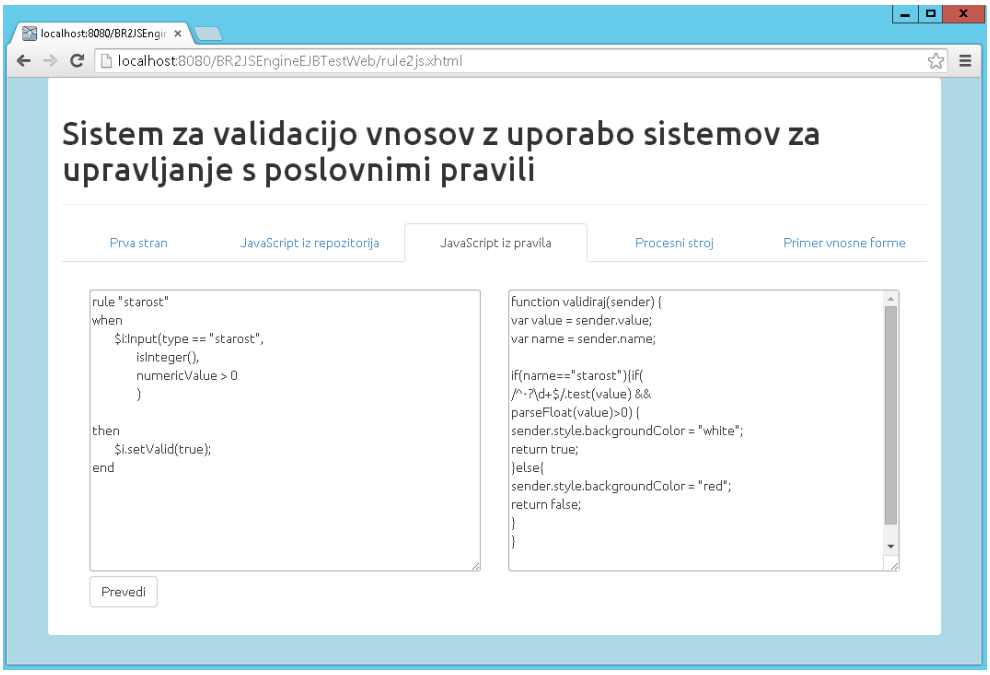
katerim lahko testiramo delovanje generirane validacijske kode JavaScript.

- **Validacija v procesnem stroju** - za dostop do procesnega stroja je na voljo obrazec, v katerega vnesemo tip vnosa in njegovo vrednost, stroj pa nam glede na validacijska pravila iz repozitorija odgovori, ali je vnos veljaven ali ne.

Spletno aplikacijo smo implementirali z ogrodjem JavaServer Faces (JSF), ki je del platforme Java EE. Ogrodje je namenjeno izdelavi uporabniških vmesnikov spletnih aplikacij. JSF omogoča gradnjo uporabniških vmesnikov s pomočjo pripravljenih gradnikov, ki so del knjižnic ogrodja JSF. Omogoča preprosto izmenjavo podatkov s preostalim delom aplikacije.

Aplikacijo sestavljajo spletne strani s pripadajočimi upraviteljskimi zrni (*Managed Beans*). V upraviteljskih zrnih se vršijo klici sejnih zrn, v katerih se izvaja sistem za validacijo vnosov. Zgradbo sistema prikazuje slika 5.1.

Slika 5.2 prikazuje zaslonsko masko aplikacije, ki omogoča generiranje validacijske kode JavaScript na podlagi vnesenega validacijskega pravila. Validacijsko pravilo zapišemo v tekstovno polje na levi. Ob kliku na gumb se v



Slika 5.2: Zaslonska maska spletne aplikacije za generiranje validacijske kode JavaScript

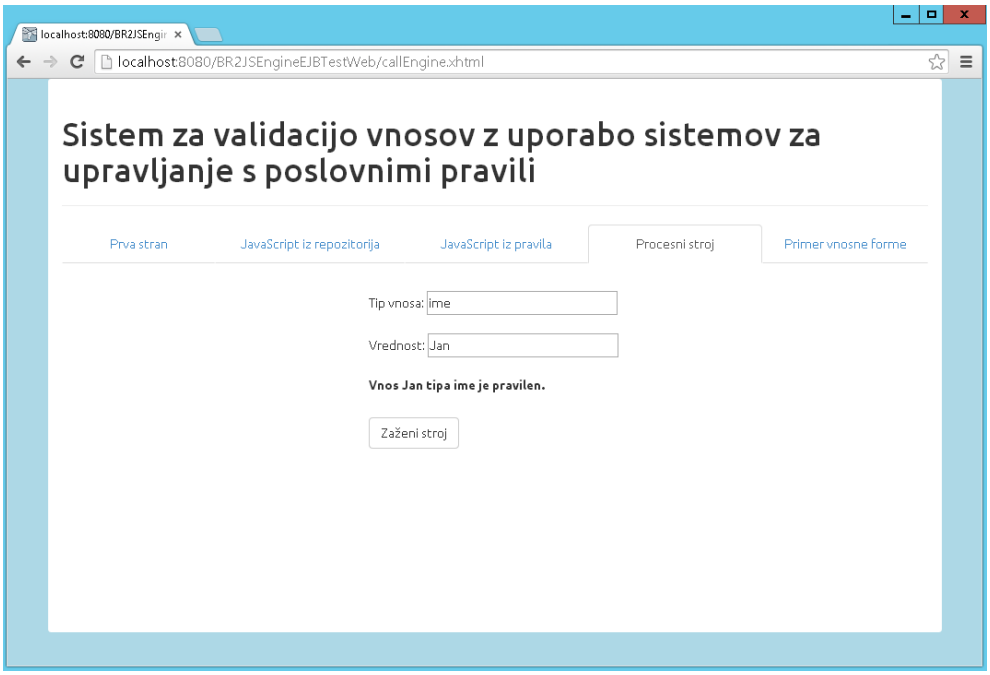
desnem tekstovnem polju izpiše ustrezna validacijska funkcija.

Slika 5.3 prikazuje zaslonsko masko aplikacije, ki omogoča validacijo vnosov v procesnem stroju. V aplikacijo vnesemo tip vnosa in njegovo vrednost, aplikacija pa nam odgovori z veljavnostjo vnosa.

5.2 Preizkus sistema na praktičnem primeru

Praktični primer smo implementirali na primeru aplikacije za spletno bančništvo, ki omogoča operacije kot so pregled stanja, plačevanje položnic, prenos sredstev med računi, oddaja vloge za depozit in podobno.

Aplikacija vsebuje veliko vnosnih polj različnih tipov, in sicer: ime, priimek, email naslov, davčna številka, številka EMŠO, številka bančnega računa, številka kartice ipd. Vsako vnosno polje potrebuje ustrezno validacijsko pravilo.



Slika 5.3: Zaslonska maska spletne aplikacije za validiranje vnosov s procesnim strojem

Pred vpeljavo našega sistema za validacijo vnosov je validacija v aplikaciji implementirana z validacijskimi orodji ogrodja JSF. Validacijska pravila so sklopljena s programsko kodo. Če se validacijska pravila za določen tip vnosnega polja spremenijo, je potrebno implementacijo validacijske logike posodobiti v programski kodi, aplikacijo pa ponovno prevesti in namestiti.

Z vpeljavo našega sistema se zapis validacijskih pravil za vnosna polja izolira od implementacije aplikacije in prestavi v centralni repozitorij pravil. Validacijska logika je sedaj zapisana v obliki deklarativnih poslovnih pravil, ki jih vzdržujemo na enem mestu, preko namenske spletne aplikacije. Ob spremembi posameznega pravila ni potrebno ponovno prevajanje aplikacije. Generiranje knjižnice JavaScript bi bilo mogoče tudi avtomatizirati in sicer tako, da se samodejno proži ob spremembi kateregakoli izmed validacijskih poslovnih pravil v repozitoriju.

Ocenjujemo, da razvit sistem za validacijo vnosov uspešno odpravlja ključne slabosti in pomanjkljivosti validacije vnosov, ki smo jih izpostavili v tej nalogi.

Poglavje 6

Zaključek

V diplomski nalogi smo se seznanili s pomenom validacije vnosov v spletnih aplikacijah. Spoznali smo vpliv validacije na delovanje spletne aplikacije, opisali najpogostejše varnostne ranljivosti ter se seznanili s pomenom validacije podatkovnih tipov.

Danes se validacije vnosov lotevamo brez namenskih rešitev, s pomočjo splošnonamenskih proceduralnih programskih jezikov. V diplomski nalogi smo razvili sistem za validacijo vnosov, ki omogoča izvajanje validacije z deklarativnim programskim jezikom, kjer validacijska pravila definiramo in vzdržujemo izven programske kode.

Zasnovali smo sistem za validacijo vnosov s pomočjo sistemov za upravljanje s poslovnimi pravili. Sistem omogoča zapis validacijske logike z validacijskimi pravili, zapisanimi v jeziku poslovnih pravil. Pomembna lastnost razvitega sistema je centralizacija validacijske logike in hramba validacijskih pravil v centralnem repozitoriju, ločeno od implementacije aplikacije. Sistem smo implementirali in ga preizkusili.

Razvit sistem je možno uporabiti v vseh spletnih aplikacijah, kjer potrebujemo validacijo vnosov. Z njim poenostavimo zapis validacijske logike, ki jo zapišemo deklarativno v obliki poslovnih pravil. Validacijska pravila shranimo v centralnem repozitoriju, sistem pa dinamično generira kodo za validacijo na odjemalcu in izvaja validacijo na strani strežnika.

Za produkcijske namene bi bilo potrebno sistem razširiti tako, da bi povečali množico možnih lastnosti vnosa, ki jih lahko uporabimo pri zapisu validacijskih pravil. Prav tako bi bilo potrebno dodati verzioniranje validacijskih pravil in generirane validacijske knjižnice JavaScript, predvsem zaradi zagotavljanja konsistentnosti validacijske logike odjemalca in strežnika.

Literatura

[1] Drools documentation. `http://docs.jboss.org/drools/release/6.1.0.Final/drools-docs/html_single/index.html`. Dostop: 10.08.2014.

[2] Javaserver faces documentation. `http://www.oracle.com/technetwork/java/javaee/documentation/index-137726.html`. Dostop: 14.07.2014.

[3] Lucas Amador. *Drools developer's cookbook*. Packt Publishing Ltd, 2012.

[4] Michal Bali. *Drools JBoss Rules 5.0 Developer's Guide*. Packt Publishing Ltd, 2009.

[5] Claus Brabrand, Anders Møller, Mikkel Ricky, and Michael I Schwartzbach. Powerforms: Declarative client-side form field validation. *World Wide Web*, 3(4):205–214, 2000.

[6] Paul Browne. *JBoss Drools business rules*. Packt Publishing Ltd, 2009.

[7] Kalyan Chatterjee and William Samuelson. *Game theory and business applications*. Springer, 2001.

[8] Justin Clarke. *SQL injection attacks and defense*. Elsevier, 2012.

[9] David Flanagan. *JavaScript: the definitive guide*. "O'Reilly Media, Inc.", 2011.

[10] Antonio Goncalves. *Beginning Java EE 6 Platform with GlassFish 3*, volume 2. Springer, 2010.

[11] Jeremiah Grossman. *XSS Attacks: Cross-site scripting exploits and defense*. Syngress, 2007.

[12] R. Grove. *Web Based Application Development*. Jones & Bartlett Learning, 2009.

[13] Jane Huffman Hayes and A Jefferson Offutt. Increased software reliability through input validation analysis and testing. In *Software Reliability Engineering, 1999. Proceedings. 10th International Symposium on*, pages 199–209. IEEE, 1999.

[14] Mark Proctor. Relational declarative programming with jboss drools. In *Symbolic and Numeric Algorithms for Scientific Computing, 2007. SYNASC. International Symposium on*, pages 5–5. IEEE, 2007.

[15] Semmy Purewal. *Learning Web App Development*. "O'Reilly Media, Inc.", 2014.

[16] K. Roebuck. *Brms - Business Rule Management Systems: High-impact Strategies - What You Need to Know*. Emereo Pty Limited, 2011.

[17] Theodoor Scholte, Davide Balzarotti, and Engin Kirda. Have things changed now? an empirical study on input validation vulnerabilities in web applications. *Computers & Security*, 31(3):344–356, 2012.

[18] Theodoor Scholte, Davide Balzarotti, and Engin Kirda. Quo vadis? a study of the evolution of input validation vulnerabilities in web applications. In *Financial Cryptography and Data Security*, pages 284–298. Springer, 2012.

[19] Theodoor Scholte, William Robertson, Davide Balzarotti, and Engin Kirda. Preventing input validation vulnerabilities in web applications

through automated type analysis. In *Computer Software and Applications Conference (COMPSAC), 2012 IEEE 36th Annual*, pages 233–243. IEEE, 2012.

[20] John Viega and Matt Messier. *Secure Programming Cookbook for C and C++: Recipes for Cryptography, Authentication, Input Validation & More*. "O'Reilly Media, Inc.", 2003.

[21] Deepak Vohra. *Java EE Development with Eclipse*. Packt Publishing Ltd, 2012.

[22] Graham Witt. *Writing Effective Business Rules*. Elsevier, 2012.

[23] Denise Woods. *HTML5 and CSS: Complete*. Cengage Learning, 2012.

Slike