UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Marjan Pušnik

RAZVOJ STANDARDNIH JAVANSKIH SPLETNIH PORTALOV Z UPORABO SISTEMA LIFERAY

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor:

doc.dr. Matjaž Kukar

Ljubljana, 2009

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: 01497/2008
Datum: 01.09.2008

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

Kandidat: **MARJAN PUŠNIK**

Naslov: **RAZVOJ STANDARDIZIRANIH JAVANSKIH SPLETNIH PORTALOV Z UPORABO SISTEMA LIFERAY**

DEVELOPING STANDARDIZED WEB PORTALS IN JAVA WITH THE LIFERAY SYSTEM

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

V zadnjih letih se je v okviru standardnih javanskih razširitev pojavilo kar nekaj zahtev za standardizacijo dela s spletnimi portali. Med najpomembnejše specifikacije sodijo JSR-168 (javanski portleti), JSR-170 (dostop do vsebin v repozitoriju) in njegova razširitev JSR-283, JSR-127 (JavaServer Faces, razvoj JSF portletov). Naštete specifikacije omogočajo standardizirano in od implementacije neodvisno gradnjo spletnih portalov. Eden izmed najpomembnejših sistemov, ki uporabljajo naštete specifikacije, je odprtokodni sistem Liferay. Kandidat naj v okviru diplomskega dela prouči in natančno oriše javanske standarde za delo s spletnimi portali in sistemi za upravljanje z vsebinami (CMS), ter prednosti in omejitve standardiziranega razvoja spletnih portalov. Opiše naj zmogljivosti in namembnost celotnega odprtokodnega sistem Liferay. S pomočjo pridobljenega znanja naj s sistemom Liferay ilustrira uporabo standardizirane izgradnje na primeru manjšega spletnega portala.

Mentor:
doc. dr. Matjaž Kukar

Dekan:
prof. dr. Franc Solina

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a Marjan Pušnik

z vpisno številko 63000254,

sem avtor/-ica diplomskega dela z naslovom:

RAZVOJ STANDARDNIH JAVANSKIH SPLETNIH PORTALOV Z UPORABO SISTEMA LIFERAY

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  doc.dr. Matjaž Kukar

  in somentorstvom (naziv, ime in priimek)

  _____

- 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 16.03.2009 Podpis avtorja/-ice: _____

Zahvala

Najprej bi se zahvalil mentorju doc. dr. Matjažu Kukarju za nasvete in pomoč pri izdelavi diplomske naloge.

Zahvalil bi se tudi svojim staršem in sestri za vzpodbudo in pomoč pri pisanju diplomske naloge. Prav tako se zahvaljujem sodelavcem za njihov porabljen čas, predloge in nasvete, ki so mi jih namenili glede pisanja diplomske naloge.

Kazalo vsebine

Seznam kratic in simbolov

Classloader	Program, ki skrbi za iskanje in nalaganje javanskih razredov.
Application context	Eden izmed kontekstov spletnih aplikacij v katerega je mogoče shranjevati instance objektov.
servlet	Javanski razred, ki omogoča razširitev spletnega strežnika z dodatnimi funkcionalnostmi(obdelavo HTTP zahtevkov in generiranje HMTL-ja).
portlet	Javanski razred, ki predstavlja razširitev funkcionalnosti servletov z funkcionalnostmi za generiranje fragmentov in portleta na portalu.
JSP	JavaServer Pages je tehnologija, ki omogoča enostavno in hitro dinamično generiranje spletnih vsebin.
JSF	JavaServer Faces je tehnologija, ki omogoča standarden razvoj uporabniških vmesnikov na strani strežnika (server-side).
Request dispatcher	Javanski razred, ki skrbi za usmerjanje oz. preusmerjanje zahtevkov med Servlet-i ali JSP-ji.
JSP tag library	Javanska knjižnica, ki skrbi za obdelavo posebnih označitvenih elementov uporabljenih v JSP in JSF datotekah.
ResourceBundle	Javanski objekt, ki običajno predstavlja podatke zapisane v formatu ključ=vrednost .
Deployment descriptor	Namestitveni opisnik. Običajno je to xml datoteka v kateri je zapisana konfiguracija potrebna za zagon javanskega razreda ali komponente.
Runtime environment	Okolje v katere se izvaja javanska komponenta.
Instanca	Instanca javanksega razreda, ki predstavlja prevedeni javanski razred, ki se izvaja v virtualnem stroju.
Render	Komponenta, ki skrbi za izris uporabniških vmesnikov.
Garbage collection	Način za uporavljanje z delavnim pomnilnikom virtualnega stroja, ki sam skrbi za odstranjevanje razredov in resursov, ki niso več v uporabi.
URL	Enolični krajevnik vira.
Action URL	Enolični krajevnik vira, ki sproži pri portletih obdelavo action zahtevka.

Render URL	Enolični krajevnik vira, ki sproži pri portletih obdelavo render zahtevka.
HTTP	Je eden izmed glavnih protokolov za prenos podatkov po spletu.
HTML	Označitveni jezik, ki omogoča gradnjo spletih strani.
Redirection	Preusmeritev oz. sprememba iz trenutnega URL-ja na nek drug URL.
Content type	Tip vsebine.
Namespace	Imenski prostor.
Avtentikacija	Postopek preverjanje istovetnosti uporabnika glede na podatke, ki so dostopni aplikaciji.
User role	Uporabniška vloga, ki določa, katere akcije lahko uporabnik izvaja.
Security constraints	Varnostne omejitve.
Content integrity	Integriteta vsebine, ki zagotavlja, da se vsebina od njenega nastanka naprej ni več spreminjala.
Confidentiality	Zaupnost podatkov, ki se običajno zagotavlja z kriptiranjem .
SSL	Secure Sockets Layer je množica kriptografskih protokolov, ki omogoča varen prenos pri povezavah preko TCP/IP omrežij.
Data source	Opisnik izvora podatkov
JNDI	Javanski vmesnik, ki omogoča enoten dostop do množice poimenskih(naming) in direktorijskih servisov
UUID	Universally Unique Identifiers je univerzalni enolični identifikator, ki enoločno določa vozlišče v repozitoriju vsebin.
Repozitorij vsebine	Hierarhičen pomnilnik vsebine, ki omogoča shranjevanje strukturirane in nestrukturirane vsebine.
Transient	Začasen
XML	Extensible Markup Language je razširljiv označitveni jezik.
SAX	Simple API for XML je skupina vmesnikov, ki omogoča zaporedno branje podatkov shranjenih v xml formatu. Je alternativa DOM-u.
DOM	Document Object Model je objektni model za prestavitev podatkov, ki so običjano shranjeni v xml formatu.

URI	Uniform Resource Identifier je enolični identifikator vira.
Referenčne integritete	Povezava med elementi repozitorija vsebin, ki uporabniku onemogoča izvedbo določenih akcij.
Version	Verzija
Shared set	Množica, katere elementi (vozilšča) so razredljeni med več delavnih prostorov repozitorija vsebin.
Lifecycle	Življenski cikel.
SQL	(Structured Query Language) je strukturirani povpraševalni jezik za delo s podatkovnimi bazami.
AOP	(Aspect-oriented programming) je programska paradigma s katero je mogoče povečati modularnost programske opreme preko uporabe t.i. ločitve presečnih zadev (separation of cross-cutting concerns)
DDD	(Domain-driven design) je pristop k načrtovanju programske opreme na osnovi problemske domene in domenske logike. Za razliko od običajnega razvoja pri katerem je pomembna izbira tehnologije za implemetacijo sistema.
AQM	(Abstract Query Model) je abstraktni povpraševalni model
JQOM	(Java Query Object Model) je javanski objektni povpraševalni model
Xpath	Je jezik, ki omogča izbiro vozlišč XML dokumenta.
AJAX	(Asynchronous JavaScript and XML) je tehnologija asinhroni javascript in xml
RSS	Je družina XML datotečnih oblik za spletno zlaganje
Wiki	Je strežniški program, ki uporabnikom omogoča prosto ustvarjanje in urejanje spletnih strani s spletnim brskalnikom
Blog	Spletni dnevnik je spletna stran, ki periodično prikazuje besedila, slike in druge elemente, ki jih njihovi avtorji sproti dodajajo.

Povzetek

V zadnjih leti se pojavlja vedno več spletnih portalov, ki omogočajo tudi upravljanje s spletnimi vsebinami in dokumenti t.i. sistemi za upravljanje s spletnimi vsebinami(WCMS). WCMS sistemi so dandanes razviti za večini programskih platform kot so npr. .NET, PHP, Python in Java. .NET je programsko ogrodje, ki je dostopno na večini operacijskih sistemov Microsoft Windows in se uporablja za izdelavo aplikacija za platformo Microsoft Windows. PHP je skriptni jezik narejen za generiranje dinamičnih spletnih strani. Python in Java sta programska jezik za splošne namene, ki ju je mogoče uporabiti na večini današnjih operacijskih sistemov. Potrebe po standardizaciji razvoja programske opreme izhajajo predvsem iz zamnjševanja stroškov, medsebojne zamenljivosti komponent oz. modulov, povečanja interoperabilnosti, ter povečanja varnosti. Primer javanskega WCMS sistema je portal Liferay. Portal vsebuje implementacije številnih javanskih specifikacij. Primeri takšnih specifikacij so specifikacije: javanskih portletov JSR-168, JavaServer Faces JSR-127, javanskega repozitorija vsebin JSR-170. Omenjene specifikacije predstavljajo osnovo za gradnjo standardnih javanskih portalov, portletov in sistemov za upravljanje z spletnimi vsebinami. Portal Liferay omogoča izdelavo naprednih visoko zmogljivih spletnih strani preko katerih je mogoče upravljati spletne skupnosti, dinamično generirati spletne strani, upravljati z vsebino in njenim verzioniranjem, itd. Visoko zmogljivost portala omogoča ta napredna arhitekutra in uporaba predpomnenja generiranih vsebin, ki se vsekakor odraža tudi v večji porabi delovnega pomnilnika v primerjavi s primerljivimi WCMS sistemi napisanimi v programskem jeziku PHP. Predstavnik takšnega WCMS sistem je portal Joomla, ki za shranjevanje podatkov uporablja podatkovno bazo MySQL in je dokaj razširjen med uporabniki WCMS sistemov predvsem zaradi njegove enostavne uporabe in namestitve. Za večjo porabo delovnega pomnilnika pri portalu Liferay gre vzroke iskati predvsem v predpomnenju in uporabi javanske tehnologije, ki hkrati omogoča izvedbo napredne arhitekture in s tem obdelavo večjega števila zahtevkov na časovno enoto glede na primerljiv WCMS sisteme Joomla. Specifikacija javanskih portletov JSR-168 oz. njena implementacija omogoča izdelav spletnih strani, ki vsebujejo večje število med seboj neodvisnih komponent oz. portletov. Vsak portlet generira del vsebine spletne strani. Omenjeni del vsebine se imenuje fragment. Celotna stran portala je tako sestavljen iz posameznih fragmentov, ki se združijo v celoto in tako generirajo ena stran spletnega portala. Specifikacija JavaServer Faces JSR-127 oz. njena implementacija omogoča razvijalcem spletnih aplikacij, da se s pomočjo abstrakntega programskega modela skoraj popolnoma izognejo posebnostim oz. učenju označitvenga jezika HTML. Javanska specifikacija repozitorija vsebin JSR-170 in njena nadgradnja JSR-283 oz. njuni implementaciji omogočata razvijalcem gradnjo repozitorijev vsebin. Repozitoriji vsebin so hierarhične drevesne struktur s pomočjo katerih je mogoče shranjevati, urejati, verzionirati in izvajati številne druge akcije nad vsebino shranjeno v repozitoriju vsebin.V pričujočem diplomskem delu bom natančno opisal omenjene javanske specifikacije, zmogljivosti portala Liferay in prikazal njihovo uporabo pri izvedbi portleta znotraj portala Liferay.

Ključne besede

liferay, java server faces, portlet, javanski repozitorij vsebine

Abstract

In recent years, there is a growing number of web portals that enable the management of online content and documents. These are so called web content management systems(WCMS). WCMS systems are today developed for the most software platforms such as. . NET, PHP, Python and Java. . NET is a software framework that is accessible on most Microsoft Windows operating systems and is used to make applications for the Microsoft Windows platform. PHP Scripting Language is designed for generating dynamic web pages. Python and Java are for general purposes programming languages, which can be used on most of today's operating systems. The need for standardization of software development are derived primarily from reducing costs, interchangeability of components, increased interoperability and increased safety. An example of such WCMS system is a portal Liferay. Portal contains the implementation of many Java specifications. Examples of such specifications are specifications: Java portlet JSR-168, JavaServer Faces JSR-127, Java content repository JSR-170. These specifications form the basis for the construction of the standard Java portals, portlets and systems for content management. Liferay Portal allows users to generate advanced high-performance web sites through which it is possible to manage online communities, dynamically generated web pages, manage and version content, etc. Advanced architecture and use of generated content caching enables Liferay portal to be one of the high performance portals. Content caching is on the other hand, reflected in higher consumption of random access memory compared to WCMS systems written in the programming language PHP. A representative of such a system is WCMS portal Joomla, which uses MySQL database to store data and is fairly widespread among the users WCMS systems primarily because of its easy use and easy installation. The increased use of working memory in the Liferay portal is mainly becouse of use of caching and the use of Java technology, which on the other hand allows the advanced architecture and the processing of a significantly more number of requests per unit time according to comparable systems WCMS Joomla. Java portlet specification JSR-168 implementation allows generation of web pages that contain a number of mutually independent components called portlets. Each portlet generates part of the contents of the website. That part of the contents is called a fragment. The entire portal page is composed of individual fragments, which are aggregated and as such represent one web page of portal. JavaServer Faces specification JSR-127 implementation allows developers to implement web applications with components of the abstract programming model, which almost completely avoids specificity of Hyper Text Markup Language. Java content repository specification JSR-170 and its upgrade JSR-283 allow developers to build content repositorys. Content repository is a hierarchical tree structure through which you can be store, edit, version and perform many other

actions over the content stored in the content repository. In this graduate work I will accurately describe these Java specifications, the capacity of Liferay Portal and show their use in the implementation of Portlet within Liferay Portal.

Keywords

liferay, java server faces, portlet, java content repository

1 Uvod

Z vse večjo uporabo spletnih tehnologij se je hkrati povečal tudi obseg dela potrebnega pri vzdrževanju vsebin in objavi sprememb na spletnih straneh. Zaradi lažjega vzdrževanja in manjših stroškov se je pri uporabnikih pojavila potreba po samostojnem spreminjanju vsebine spletnih portalov brez posega programerjev in vzdrževalcev spletnih aplikacij. Nastali so t.i. WCMS sistemi oz. sistemi za upravljanje s spletnimi vsebinami. Pri tem je potrebno pripomniti, da CMS sistemi niso vezani samo na spletne vsebine in se običajno uporabljajo tudi za urejanje drugih tipov vsebin npr. različnih tipov dokumentov.

Kaj pravzaprav je CMS sistem in za kakšne namene ga je mogoče uporabiti? Za odgovor na to vprašanje je potrebno najprej odgovoriti na naslednji vprašanji:

- Kaj je vsebina?
- Kaj je upravljane z vsebino?

Vsebina je kakršenkoli tip ali enota digitalne informacije kot npr. slike, grafike, video, zvok, dokumente, itd. Vsebina je skratka vse kar je mogoče upravljati v elektronski obliki. Upravljanje z vsebinami je upravljanje z uporabo pravil, procesov, delavnih procesov, ki omogočajo shranjevanje, kontroliranje, verzioniranje in objavo dokumentov oz. vsebin. Torej kaj je sistem za upravljanje z vsebinami? Je orodje oz. skupek orodij, ki omogočajo učinkovito izvedbo željenega učinka z uporabo vsebine, ki jih upravlja uporabnik [5].

Leta 1998 je bil ustanovljen t.i. »Java Community Process (JCP)«. JCP je odprt proces razvoja in revizje specifikacij Java™ tehnologije, referenčnih implementacij in testov [7]. JCP proces od njegovega nastanka naprej skrbi za napredek in razvoj Java platforme v sodelovanju z mednarodno skupnostjo Java razvijalcev. Vse specifikacije omenjene v tej diplomski nalogi so prav tako del JCP procesa.

Prednosti standardiziranega razvoja aplikacij [8]:

- Zmanjševanje stroškov namestitve;
- Omogočanje medsebojne zamenljivosti sestavnih delov oz. komponent;
- Zmanjševanje stroškov vzdrževanja;
- Povečanje interoperabilnost;
- Izboljšava načrta oz. arhitekture z manj dodatnega napora načrtovalcev;
- Povečanje varnosti;
- Standardi predstavljajo leta izkušenj ekspertov in odpravljajo potrebo gradnje komponent oz. projekta popolnoma od začetka (ground up).
- Prihranki pri stroških oz. denarju potrebnih pri lastnem razvoju komponent;

Slabosti standardiziranega razvoja aplikacij:

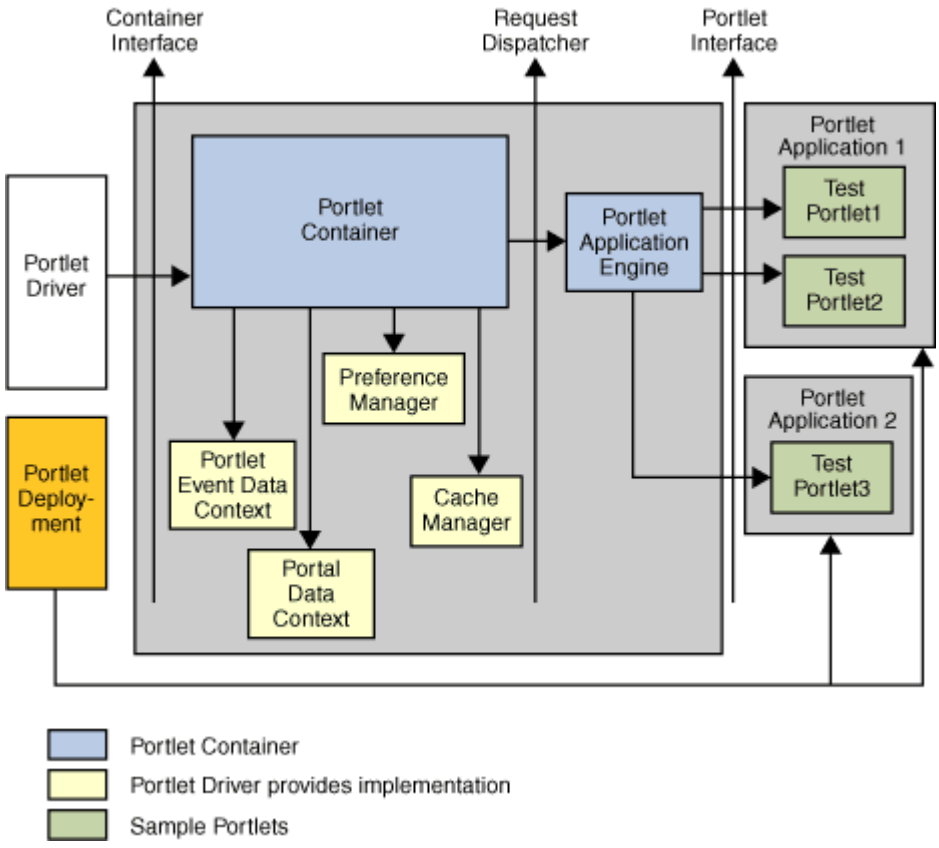
- Potreben čas za začetno učenje implementacije specifikacije;
- Težje uvajanje posebnih sprememb oz. zahtev, ki jih zahteva naročnik in so specifične za posamezen projekt;
- Zagotavljanje spoštovanja oz. kompatibilnosti implementiranih komponent s specifikacijo. (Ta težava je pri javanskih specifikacijah deloma odpravljena z uporabo kompleta kompatibilnosti tehnologije [29] (TCK - Technology Compatibility Kit), ki omogoča preverjanje, če posamezna implemetacija zadošča zahtevam, ki so zapisane v specifikaciji)
- Počasnejši razvoj implementacije specifikcije oz. funkcionalnosti, ki jih specificira specifikacija. Zaradi potreb po usklajevanju članov skupine oz. skupnosti, ki sodelujejo pri pripravi specifikacije, je običajno razvoj implementacije funkcionalnosti v manjših skupinah brez zunanjih sodelavcev oz. skupnosti hitrejši.

Prednosti standardiziranega razvoja spletnih aplikacij so vsekakor večje od njihovih slabosti. Slednje je razvidno tudi iz primera [9], ki predstavlja implementacijo javanskega repozitorija vsebin (JSR-283) v jeziku PHP oz. WCMS sistemu TYPO3. TYPO3 je podobno kot Joomla odprtokodni WCMS sistem napisan v skriptnem programskem jeziku PHP. Repozitorij vsebin omogoča robustno shrambo vsebin in vsebuje enoten, dobro definiran API, ki prinaša vrsto prednosti pred običajnim shranjevanjem podatkov v običajne podatkoven baze npr. MySQL. Ker trenutno ne obstajata nobena popolna implementacija specifikacije JSR-283 v skriptnem jeziku PHP in so bili poiskusi uporabe implementacije repozitorija vsebin Jackrabbit preko t.i. javanskega mostu z skriptnim jezikom PHP neuspešni (predvsem zaradi težav z uporabo delovnega spomina in t.i. wrapped klicev v javo) se je skupina razvijalcev WCMS sistema odločila za izvedbo lastne PHP implementacije specifikacije JSR-283. Implementacija je razvita z uporabo FLOW3 [32] ogrodja za razvoj aplikaciji z uporabo AOP programske paradigme (varnost, logiranje) in DDD (Domain-driven design) pristopa k načrtovanju programske opreme. Trenutna implementacija vsebuje naslednjo podmnožice lastnosti, ki so zapisane v specifikaciji JSR-283 : osnovno branje in pisanje, registracijo imenskih prostorov, odkrivanje tipov vozlišč in njihovo registracijo. Primer dokazuje tudi, da javanske specifikacije niso omejene na implementacijo samo v jeziku Java.

V pričujoči diplomski nalogi bom opisal naslednje javanske specifikacije :

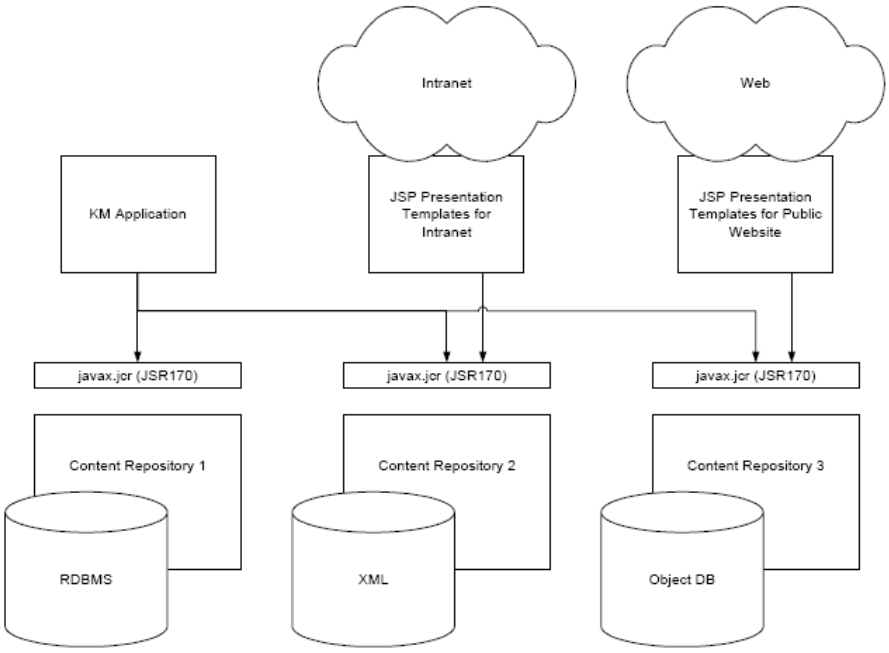
- Specifikacija portletov JSR-168;
- Repozitorij vsebin verzija 1.0 za tehnologijo Java JSR-170;
- Repozitorij vsebin verzija 2.0 za tehnologijo Java JSR-283;
- JavaServer Faces JSR-127;

Specifikacija JSR-168 specificira javanske portlete in okolje v katerem se izvajajo. Portleti so spletne komponente, ki temeljijo na tehnologiji Java in procesirajo zahtevke, ter generirajo dinamično vsebino. S portleti upravlja t.i. portlet zabojnik. Ta predstavlja okolje v katerem se izvajajo. Portleti se uporabljajo kot uporabniški vmesnki, ki jih je mogoče vključiti v portale in predstavljajo del predstavitvene plasti informacijskih sistemov. Portal je spletna aplikacija, ki običajno omogoča personalizacijo, enotni sistem za prijavo, agregacijo vsebin iz različnih virov in vsebuje prezentacijsko plast informacijskega sistema. Agregacija je integracija vsebin iz raličnih virov znotraj spletne aplikacije. Portlete je mogoče izvajati na vsakem portalu, ki vsebuje implemetacijo portlet zabojnika. Vendar pa je pri tem potrebno poudariti, da je običajno potrebno dodatan konfiguracija za vključitev portleta v portal, ki je specifična za portal v katerega bo portlet vključen. Ena izmed internetnih stran preko katere je mogoče pridobiti portlete za javanske portale je tudi stran [33]. Portleti, grobo rečeno, predstavljajo nadgradnjo servletov z dodatnimi funkcionalnostmi in posameznimi specifičnimi lastnosti, kot je npr. generiranje fragmentvo spletne strani in ne celotne spletne strani. Podobnosi in razlike s specifikacijo servletov so natančno opisane v podpoglavju opisa specifikacije JSR-168 *2.1.3 Povezava s specifikcija servletov v2.3*.



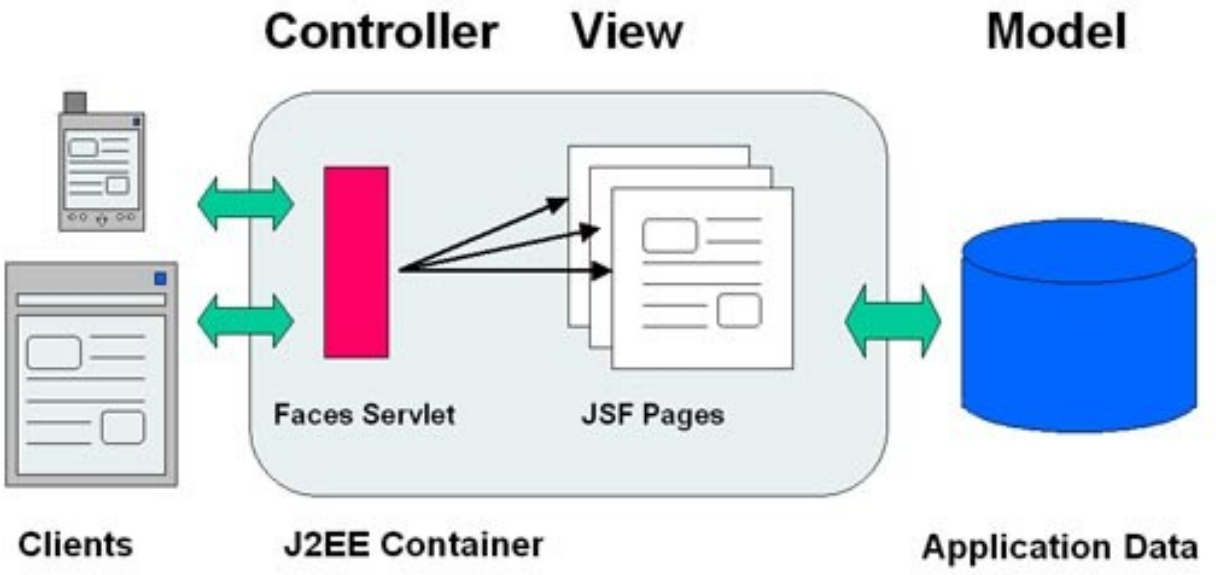
Slika 1. Shema arhitekture implemetacije portlet specifikacije JSR-168(slika je dostopna na [34]).

Specifikacija JSR-170 in njena naslednica JSR-283 definirata standarden vmesnik API za dostop do javanskega repozitorija vsebin. Repozitorij vsebine omogoča shranjevanje in upravljanje vsebine.



Slika 2. Shema prikazuje integracijo repozitorijem vsebin v aplikaciji oz. spletnem portalu (slika je dostopna na [1]).

Specifikacija JavaServer Faces JSR-127 je javanska specifikacija, ki omogoča gradnjo spletnih aplikacij s pomočjo javanskih objektov skoraj brez uporabe znanj o označevalni jezikih kot je npr. HTML. Ogrodje JSF omogoča razvijalcu, da definira uporabniški vmesnik s pomočjo javanskega vmesnika API oz. z uporabo t.i. tag knjižnic (tag libraries), ki jih nato samo ogrodje JSF uporabi za generiranje dejanskega uporabniškega vmesnika npr. HTML stran. Ogrodje JSF prav tako skrbi za pošiljanje in sprejemanje zahtevkov, ter dekodiranje in enkodiranje vrednosti, ki se npr. pri uporabi protokola HTTP pošiljajo iz strežnika do uporabnika in nazaj. Če povzamem ogrodje JSF razvijalcu ponuja abstrakten programski model preko katerega je poenostavljena izgradnja uporabniških vmesnikov za spletne aplikacij. Razvijalci različne ravni usposobljenosti lahko hitro gradijo spletne aplikacije z uporabo ponovno uporabljivih UI komponent znotraj ene spletne strani. Ogrodje JSF omogoča tudi sprejemanje dogodkov, ki se prožijo na klientih npr. v brskalnikih, s pomočjo razredov za obdelovanje dogodkov (event handler), ki se izvajajo na strežniku.



Slika 3. Shema poenostavljene arhitekture JavaServer Faces (slika je dostopna na [35]).

Liferay je odprtokodni WCMS sistem namenjen uporabi v poslovnih informacijskih sistemih in intranetih. Vsebuje implementacije vseh pravkar omenjenih specifikacija javanske tehnologije. Poleg teh pa vsebuje tudi implentacijo naslednjih specifikacij oz. tehnologij: JSR – 220 (Hibernate oz. javanska tehnologija za persistiranje podatkov), AJAX in JSON, WRSP (specifikacija spletnega servisa, ki omogoča objavo portletov na oddaljenih portalih), itd. Implementacijo specifikacije JSR-170 predstavlja ogrodje Jackrabbit, ki ga je mogoče uporabiti preko CMS portelov, ki so že priloženi WCMS sistemu Liferay. Lifery prav tako omogoča integracijo z javanskimi CMS repozitoriji kot sta Alfresco in Magnolia. Vsebuje vgrajenih že preko 60 konfigurabilnih portletov s pomočjo katerih je mogoče ustvarjati spletne skupnosti, upravljati dokumente in spletne vsebine, upravljati z uporabniki portala in generirati hierarhično urejene spletne strani z poljubnim naborom lastnih portletov ali portletov, ki so že priloženi WCMS sistemu.

Poleg opisa omenjenih specifikacij bom v pričujoči diplomski nalogi uporabil pridobljeno znanje o specifikacijah za implementacijo portleta brskalnika po repozitoriju vsebin. Omenjeni portlet bom vključil v seznam portletov WCMS sistema Liferay, kar bo omogočalo, da se portlet umesti na katero koli generirano stran WCMS sistema Liferay. Razvoj portleta oz. potek dela sem razdelil v štiri faze. V prvi fazi bom vzpostavil razvojno okolje, ter namestil aplikacijski strežnik, v katerega je že vključen WCMS sistem Liferay na računalnik, kjer po potekal razvoj. Liferay omogoča nadgradnjo sistema z uporabo dveh načinov programiranja. Prvi način je t.i. »extension« ali razširljiv način pri katerem razvijalec dodaja funkcionalnosti direktno v kodo, ki

predstavlja jedro »kernel« WCMS sistema Liferay in s tem razširja funkcionalnosti, ki jih ponuja sistem. Drugi način programiranja pa je t.i. »plugin« ali vtičniški način, pri katerem razvijalec preko posebne arhitekture vtičnikov dodaja funkcionalnosti v WCMS sistem Liferay. Pri vtičniškem razširanju sistema Liferay je dodatna funkcionalnost pakirana v obliki war (web archive) datotek, ki jih je mogoče dodajati in odvzemati sistemu Liferay medtem, ko le ta obratuje (t.i. hotdeploy). V grobem povedana je razlika med obema načinoma programiranja v paketu, ki je končni rezultat razvoja funkcionalnosti. V primeru razširljivega načina programiranja je rezultata war datoteka, ki vsebuje prevedeno kodo celotnega WCMS sistema Liferay in prevedeno kodo dodatnih funkcionalnosti, ki jih je dodal razvijalec(ta način ne omogoča dodajanja novih fukcionalnosti medtem, ko sistem obratuje). Pri vtičniškem načinu, pa je rezultat war datoteka, ki vsebuje samo prevedeno kodo nove funkcionalnost(ta način omogoča dodajanja novih fukcionalnosti medtem, ko sistem obratuje). Predvsem zaradi predhodnega poznavanja razvoja z razširljivim načino sem se odločil, da ga bom uporabil pri razvoju portleta za WCMS sistem Liferay. V drugi fazi bom ustrezno konfiguriral repozitorij vsebin Jackrabbit, ki predstavlja implementacijo specifikacije JSR-170. Knjižnica oz. implemenacija je že vključena v WCMS sistem Lifery, zato sem se odločil, da bo potrebno v tej fazi samo napisati novo kofiguracijo za repozitorij vsebin (predvsem zato, ker tudi sistem Lifery uporablja repozitorij vsebin za shranjevanje in upravljanje vsebin vnešene preko CMS portletov). Za ustrezno konfiguriranje repozitorija vsebin se bo potrebno seznaniti s konfiguracijo, preko katere je mogoče generirati in upravljati repozitorij vsebin(Jackrabbit vsebuje samo eno konfiguracijsko xml datoteko, preko katere je mogoče upravljati z repozitorijem vsebin). V tretji fazi bom razvil/implementiral portlet, ki bo omogočal brskanje po repozitoriju vsebin. Zato bo potrebno ustrezno skonfigurirati implementacijo specifikacije JSF (JSR-127), ter konfiguracijo potrebno za vključitev JSF portleta v WCMS sistem Liferay. Polege omenjenega bo potrebno napisati tudi ustrezno kodo za prikaz portleta na straneh WCMS sistem Liferay, ter kodo za shranjevanje vrednosti vnosnih polj, ki jih bo uporabnik vnesel preko spletne forme znotraj portleta. V tej fazi bo prav tako potrebno napisati javanske metode, ki bodo s pomočjo podatkov vpisanih s strani uporabnika, omogočale brskanje oz. komuniciranje z repozitorijem vsebin. Za sam prikaz vnosnih polj in portleta za brskanje po repozitoriju vsebin bo uporabljena tehnologija JSF, ki bo omogočila predvsem lažji in hitrejši razvoj portleta na prezentacijski plasti pa deloma tudi komponent portleta na strežniški strani. Prednost uporabe tehnologije JSF je, da njen abstraktni model genriranja uporabniških vmesnikov skoraj v popolnosti skrije posebnosti tehnologije uporabljenje za prikaz uporabniškega vmesnika uporabnika(v primeru spletnih aplikacij je to HTML) oz. hkrati tudi odpravi posebnosti, ki so pri spletnih aplikacijah povezane z različnimi implementacijami spletnih brskalnikov. V zadnji fazi razvoja portleta bo potrebno napisati še ustreno konfiguracijo oz. konfiguracijske datoteke, ki bodo portlet vključile v WCMS sistem Liferay in ga dodale v ustrezno kategorijo portletov omenjenega sistema. Po uspešno zaključenih vseh fazah razvoj portleta bo potrebno portlet namestiti še na aplikacijski strežnik Apache Tomcat 5.5, ki sem ga izbral za razvoj portleta oz. na katerem je že prednameščen WCMS sistem Liferay. Stežnik s prednameščenim WCMS sistemo Lifery je mogoče prenesi z uradne strani

WCMS sistema Liferay. Prav tako je mogoče z omenjene strani prenesti tudi paket datotek, potrebni za razvoj dodatnih funkcionalnosti v razširljivem načinu, za WCMS sistem Liferay.

Poleg razvoja portleta brskalnika po repozitoriju vsebin bom v pričujoči diplomski nalogi naredil tudi test zmogljivosti WCMS sistema Liferay in WCMS sistema Joomla napisanega v jeziku PHP. Primerjava bo narejena na podlagi števila postreženih zahtevkov na časovno enoto. Za WCMS sistem Joomla sem se odločil predvsem zaradi njegove enostavnosti za uporabi in njegove razširjenosti med uporabniki. Ker aplikacijski strežnik Apache Tomcet predstavlja implementacijo zabojnika za javankse servlete, kot tak ne podpira spletnih strani napisanih v skriptnem programskem jeziku PHP. Zato bo za prikaz omenjenih spletnih strani potrebno v aplikacijski strežnik vključiti javanski servlet, ki bo opravljal prevajanje PHP spletnih strani v HTML spletne strani, ki se prikazujejo v spletnih brskalnikih. Orodje s katerim bom izvedel testiranje je Apache JMeter. Apache JMeter je javanska aplikacija, ki omogoča beleženje podatkov in izdelavo naprednih testov predvsem za protokola HTTP, HTTPS(omogoča pa tudi testiranje drugih protokolov). Ker WCMS sistema ne vsebujeta prednameščene strani, ki bi bile medsebojno podobne sem se odločil, da bom na vstopno stran WCMS sistema Liferay za potrebe testiranja dodal ustrezno število portletov in vsebine tako, da bo slednja vsaj približno vsebovala enako število vsebine kot to vsebuje vstopna stran WCMS sistema Joomla.

2 Javanski standardi za gradnjo spletnih portalov in WCMS sistemov

2.1 Specifikacija portletov JSR-168

Podatki in slike opisani v tem poglavju so povzeti glede na vir [1].

Specifikacija portletov opisuje delovanje portletov in okolje v katerem se izvajajo.
Namenjena je :

- Proizvajalcem portalov, ki želijo zagotoviti, da njihovi izvajalniki portletov zadostijo zahtevam opisanim v tej specifikacij;
- Razvijalcem avtorskih orodij , ki želijo vključiti podporo za spletne aplikacije skladne s to specifikacijo;
- Izkušenim avtorjem portletov, ki želijo razumeti mehanizme tehnologije portletov;

Specifikacija se navezuje tudi na naslednje javanske specifikacije :

- Java 2 Platform, Enterprise Edition, v1.3 (J2EE™);
- Java Servlet™, v2.3;
- JavaServer Pages™, v1.2 (JSP™);

Kot je bilo že povedano so portleti spletne komponente, ki temeljijo na javanski tehnologiji in jih upravlja portlet zabojnik. Obdelujejo zahtevek in genirirajo dinamično vsebino. Uporabljajo se v portalih kot komponente v uporabniških vmesniki, ki skrbijo za del predstavitvene plasti informacijskega sistema.

Vsebina, ki jo generirajo portleti je imenovana tudi fragment. Fragment je del kode napisane v označevalnem jeziku npr. HTML, XHTML, WML, ki spoštuje določena pravila in ga je mogoče združiti z ostalimi fragmenti, ter tako geneirati celotne dokument oz. stran portala. Stran portala je običajno sestavljena iz vsebine, ki jo generira posamezen portlet in je agregirana oz. združena z vsebino drugih portletov na tej strani. Življenski cikel portleta opravlja t.i. portlet zabojnik.

Interakcija med spletnim klientom in portleti se izvaja preko t.i. zahtevek/odziv(request/response) paradigme, ki jo implementira portal. Vsebina, ki jo generirajo portleti ni enaka za vse uporabnike, ampak se razlikuje glede na konfiguracijo posameznega portleta.

2.1.1 Portlet zabojnik

Portlet zabojnik izvaja portlete in predstavlja okolje v katerem se izvajajo t.i. »runtime« okolje. Prav tako upravlja z življenskim ciklom portletov in omogoča obstojno shrambo njihovih

nastavitev. Portlet zabojnik sprejema zahtevke od portala in jih usmerja oz. izvaja na portletih, ki jih vsebuje.

Portlet zabojnik ne skrbi za agregacijo vsebin, ki jih proizvjajo portleti. To je namreč naloga portala.

Portal in portlet zabojnik je mogoče narediti skupaj kot eno komponento aplikacije ali pa kot ločeni komponenti portalske aplikacije. Slika 1 prikazuje arhitekturo portlet zabojnika. Slika 6 pa prikazuje komunikacijo portlet zabojnikom s portalom in portleti. Portlet zabojnik sprejema zahtevke za izvedbo portletov s strani portala in jih izvaja. Generirano vsebino portletvo pa nato posreduje nazaj portalu.

2.1.2 Primer dostopa do strani portala

Prikazan je primer zaporedja dogodkov, ki jih sproži uporabnik ob prihodu na stran portal, ki vsebuje portlete:

1. Klient (npr. spletni brskalnik), po uspešni avtentikaciji pošlje HTTP zahtevek portalu;
2. Portal sprejeme HTTP zahtevek;
3. Portal ugotovi, če zahtevek vsebuje akcijo namenjeno, kateremu izmed portletov povezanih s stranjo portal;
4. Če zahtevek vsebuje akcijo namenjeno, kateremu izmed portletov, zahteva od portlet zabojnik, da izvede portlet in izvede zahtevano akcijo;
5. Portal izvede portlet preko portlet zabojnika in generira stran portala, ki vključuje fragment generiran preko portleta.
6. Portal agregira vse fragmente portletov in sestavi stran portala, ki jo pošlje kot odgovor na zahtevek nazaj klientu.

2.1.3 Povezava s specifikcija servletov v2.3

Servleti so spletne komponente java tehnologije, ki jih upravlja servlet zabojnik in generirajo dinamične vsebine. Servlet zabojniki so razširiteve spletnega strežnika, ki omogočajo nadgradnjo spletnih strežnikov z dodatnimi funkcionalnostmi.

Podobnosti med servleti in portleti:

- So spletne komponente, ki temeljijo na javanski tehnologiji;
- Upravljajo jih specializirani zabojniki;
- Generirajo dinamične vsebine;
- Zabojnik upravlja z njihovim življenskim ciklom;

- Interakcija s klientom poteka preko request/response paradigme;

Razlike med servleti in portleti:

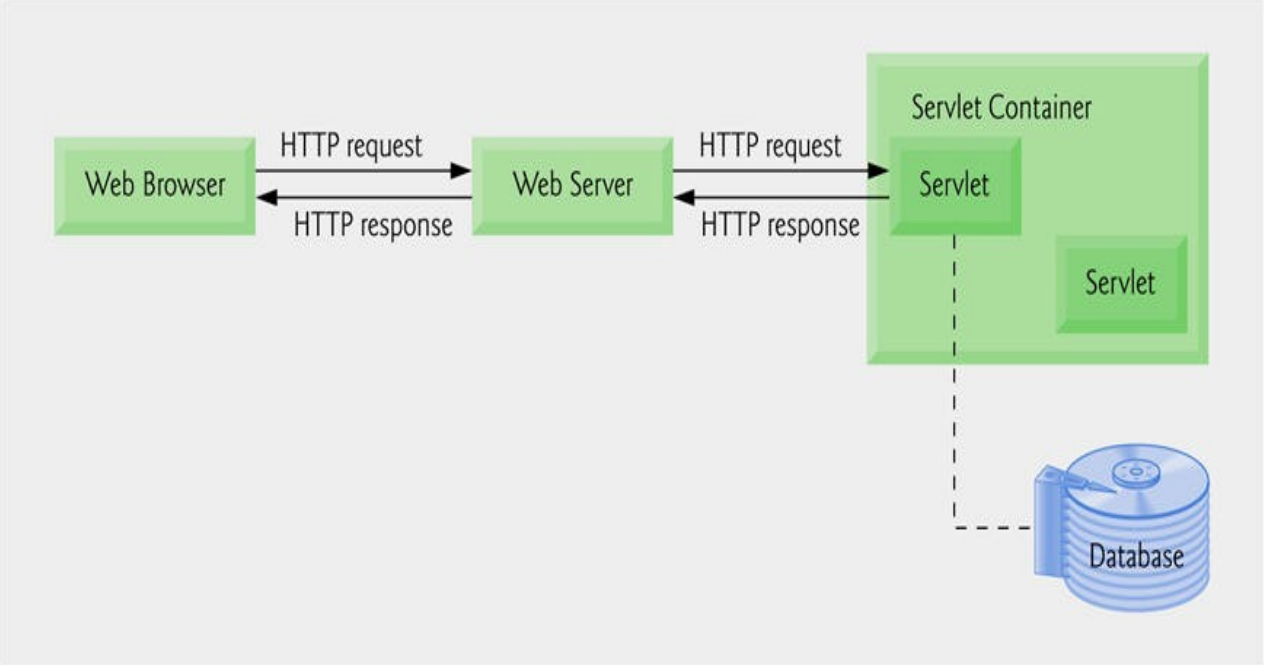
- Portleti ne generirajo celotnih dokumentov ampak samo fragmente. Portal združuje oz. agregira fragmente v celotno stran portala;
- Portleti niso direktno vezani na URL;
- Interakcija med spletnimi klienti in portleti poteka preko portalskega sistema oz. portala;
- Portleti imajo bolj podrobno obravnavo zahtevkov(action zahtevkov, render zahtevkov);
- Portleti imajo predefinirane načine prikaza in stanja okna;
- Portleti so lahko prisotni večkrat na isti portalski strani;
- Portleti imajo možnost shranjevanja in dostopa do konfiguracijskih in prilagoditvenih podatkov;
- Portleti imajo dostop do podatkov uporabnikovega profila;
- Portleti imajo možnost URL prepisovanja za kreiranje hiperpovezav znotraj vsebine, ki jo generirajo;
- Portleti lahko shranjujejo svoje prehodne podatke v dva različna področja znotraj portlet seje(aplikacijsko področje in privatno področje portleta);

Portleti za razliko od servletov nimajo dostopa do naslednjih funkcijonalnost :

- Nastavljanje znakovnega kodiranja odgovora (responsa);
- Nastavljanje HTTP glave(header) odgovora (respons);
- Dostop do URL-ja iz klientovega zahtevka poslanega na portal;

Zaradi teh razlik se je ekspertna skupina odločila, da morajo postati portleti nove komponente, kar omogoča definiranje jasnejših vmesnikov in obnašanja teh komponent. Zaradi ponovne uporabe, čim večjega deleža infrastrukture servletov, uporablja specifikacija portletov funkcionalnosti, ki jih vsebuje tudi specifikacija servletov, povsod kjer je to mogoče.

Portleti, servleti in JSP-ji so združeni v razširjeni spletni aplikaciji imenovani portlet aplikacija. Hkrati si portleti, servleti in JSP-ji znotraj iste spletne aplikacije delijo tudi nalagalnik razredov (classloader), aplikacijski kontekst (application context) in strežniško sejo (session).



Slika 4. Osnovna shema komunikacije med servleti in spletnim brskalnikom (slika je dostopna na [36]).

2.1.4 Most med portleti, servleti in JSP-ji

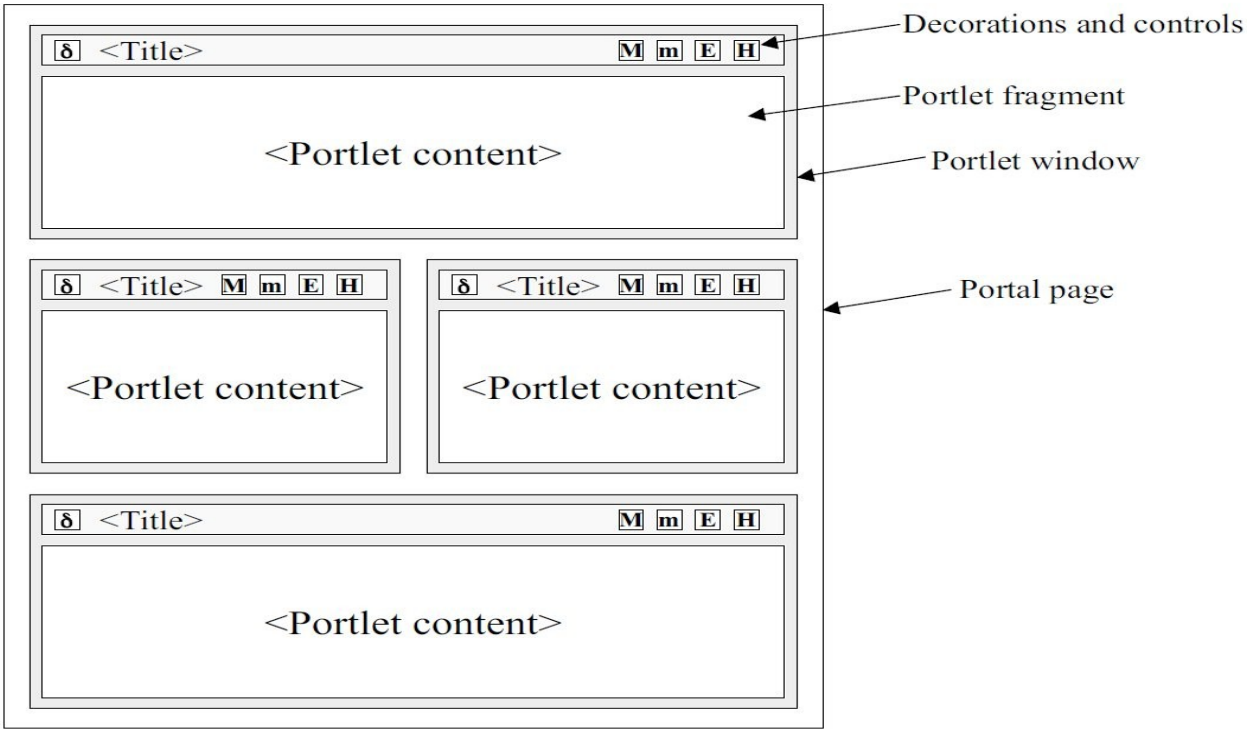
Portleti lahko generirajo vsebino s pomočjo servletov, JSP-jev in JSP tag knjižnic. Lahko kličejo servlete in JSP-je tako kot lahko servleti in JSP-ji kličejo druge servlete in JSP-je s pomočjo usmerjevalnika zahtevkov(request dispatcher). Ko je znotraj portleta poklican servlet oz. JSP je servlet zahtevek(request) osnovan na portlet zahtevku(request) predan servletu oz. JSP-ju. Enako velja za servlet odziv (response) osnovan na portlet odzivu.

2.1.5 Povezava med servlet zabojnikom in portlet zabojnikom

Portlet zabojnik je razširitev servlet zabojnika. Iz tega sledi, da je portlet zabojnik lahko narejen na osnovi obstoječega servlet zabojnika ali pa mora implementirati vse funkcionalnost, ki jih omogoča servlet zabojnik. Ne glede na izbiro načina za implementacijo portlet zabojnika, mora slednji podpirati servlet specifikacijo 2.3.

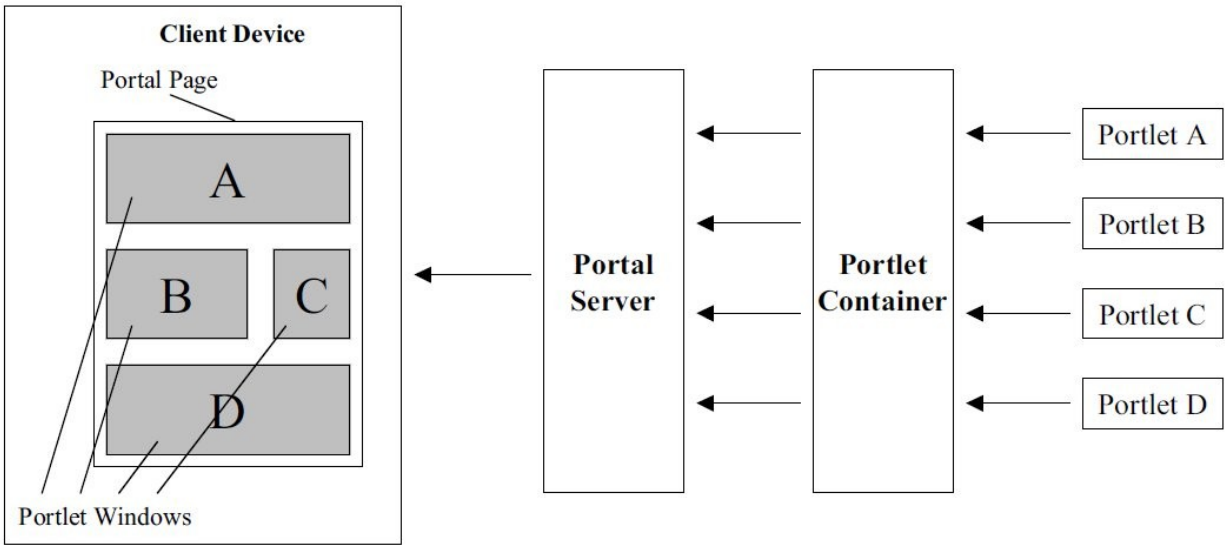
2.1.6 Osnovni koncepti

Portlet generira fragment, ki ga portal nato okrasi z naslovom, kontrolnimi gumbi in ostalimi dekoracijami. Ta nov fragment se imenuje portlet okno. Portal nato agregira oz. združi vsa portlet okna in tako generira celotno stran portala.



Slika 5. Stran portala s štirimi portlet okni. Portlet okno je sestavljeno iz dekoracij in kontrol, ter fragmetna vsebine, ki jo generira portlet.

Portlet zabojnik posreduje vsebino oz. fragment, ki ga generira portalskemu strežniku. Portalski strežnik to vsebino združi v stran portala in jo posreduje klientu (npr. brskalniku) kot odgovor na njegov zahtevek.



Slika 6. Interakcije med klientom in spletno stranjo portala, ki vsebuje štiri portlete. Portal preko portlet zabojnika izvaja posamezne portlete, ter agregira generirano vsebino.

Portal ob sprejemu zahtevka za določeno stran portala preveri seznam portletov, ki jih je potrebno izvesti za uspešno strežbo zahtevka. Portal nato preko portlet zabojnika izvede te portlete. Fragmente portletov združi v stran, ki jo vrne klientu.

2.1.7 Vmesniki specifikacije portletov

2.1.7.1 Vmesnik Portlet

Vmesnik Portlet predstavlja glavno abstrakcijo portleta API-ja. Vsi portleti morajo ali implementirati ta vmesnik oz. bolj pogosto razširjajo razred, ki implementira ta vmesnik npr. GenericPortlet, ki je vključen v portlet API.

Sekcija definicije portletov v namestitvenem deskriptorju portlet aplikacije določa, kako portlet zabojnik kreira instance portletov. Portlet zabojnik lahko za posamezno definicijo portleta v nedistribuiranem okolju ustvari samo eno instanco portleta. V distribuiranem okolju pa lahko ustvari po eno instanco za posamezno definicijo portleta v namestitvenem deskriptorju (deployment deskriptor) na posamezen virtualni stroj.

Življenski cikel portleta se upravlja preko metdo: init, processAction, render in destory, ki so definirane v vmesniku Portlet.

Inicializacija objekta Portlet se izvede s klicem metode init. Klic metode init se izvede z enoličnim (za vsako definicijo portleta) parametrom tipa vmesnika PortletConfig. Slednji omogoča dostop do inicializacijskih parametrov in skupine resursov (ResourceBundl), ki so

definirani v definiciji portleta v namestitvenem deskriptorju. Hkrati konfiguracijski objekt omogoča tudi dostop do kontekstnega objekta, ki predstavlja izvajajoče okolje (runtime okolje).

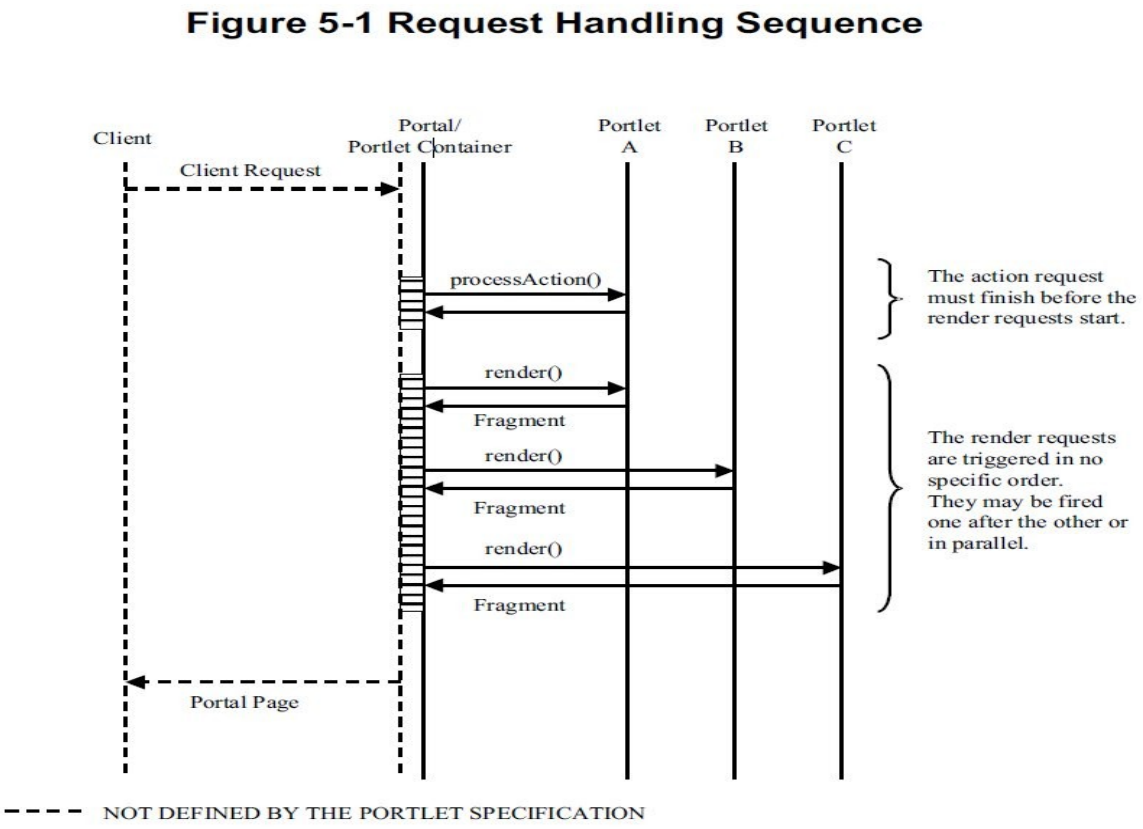
2.1.7.2 Portlet okno

Portletova definicija lahko vsebuje množico izbirnih nastavitev. Uporabljajo se za ustvarjanje objekta z izbirnimi nastavitvami (preferences).

Ob postavitvi portleta na stran portala se Portlet objekt poveže z objektom izbirnih nastavitev, ki jih lahko bere, spreminja ali dodaja. Povezavo upravlja portal/portlet zabojnik. Glede na te nastavitve lahko portlet prilagaja svoje obnašanje. Portal/portlet zabojnik lahko omogoča administracijska sredstva za kreiranje novega objekta izbirnih nastavitev. Pojavitev objekta Portlet in objekta izbirnih nastavitev na strani portala se imenuje portlet okno. Stran portala lahko vsebuje več portlet oken, ki se navezujejo na ista objekta Portlet-a in izbirnih nastavitev.

2.1.7.3 Obdelava zahtevkov

Vmesnik Portlet ima definirani metodi processAction in render za obdelavo zahtevkov, ki ju je mogoče izvajati šele po uspešni inicializaciji instance objekta Portlet. Metodi je mogoče izvesti preko t.i. portlet URL-ja, ki je lahko naslednjih dveh tipov : action URL in render URL. Običajno klientov zahtevek tipa action URL povzroči izvedbe ene metode processAction in večih metod tipa render (enkrat za vsak portlet na portalski strani). Izjema so le portleti, katerih generirana vsebina se pomni v predpomnilniku. V primeru, da klient pošlje zahtevek tipa render URL pa portlet zabojnik izvede samo render metode vseh portletov na portalski strani, razen na portletih, katerih generirana vsebina je shranjena v predpomnilniku. Odločitev o uporabi vsebine v predopomnilniku oz. klicu metode render je prepuščena Portler zabojniku.



Slika 7. Obdelave klientovega zahtevka s strani portal/portlet zabojnika

2.1.7.3.1 Action zahtevek

Običajno kot odziv na Action zahtevek portlet spremeni svoje stanje glede na podatke, ki jih dobi v obliki parametrov. Metoda processAction dobi pri klicu dva parametra tipa ActionRequest in ActionResponse. Objekt ActionRequest vsebuje parametre action zahtevka, stanje okna portleta, portlet način (mode), portlet kontekst, portlet sejo (session) in podatke o izbirnih nastavitvah portleta (preferences). Portlet lahko s pomočjo objekta ActionResponse med obdelavo Action zahtevka spremeni stanje svojega okna in portlet načina.

2.1.7.3.2 Render zahtevek

Kot odziv na Render zahtevek portlet generira vsebino glede na njegovo trenutno stanje. Metoda render dobi pri klicu dva parametra tipa RenderRequest in RenderRespons. Objekt RenderRequest vsebuje parametre render zahtevka, stanje okna portleta, portlet način (mode), portlet kontekst, portlet sejo (session) in podatke o izbirnih nastavitvah portleta.

2.1.7.4 GenericPortlet

Abstrakten razred GenericPortlet vsebuje priročne metode za obravnavo render zahtevkov. Metoda render razreda GenericPortlet nastavi naslov(title) zapisan v namestitvenem deskriptorju in pokliče metodo doDispatch. Ta metoda vsebuje logiko za pomoč pri procesiranju zahtevkov glede na portlet način v katerem je trenutno portlet (metoda doDispatch pokliče eno izmed doView,doEdit,doHelp). Če je stanje okna portleta »MINIMIZED« metoda render ne izvede nobene izmed metod za obdelavo portleta gleda na način v katerem se nahaja.

Tipčno portleti razširjajo razred GenericPortlet in preobolžijo(override) metode doView, doEdit, doHelp in getTitle (namesto metod render in doDispatch).

2.1.7.5 Večnitnost med obdelavo zahtevkov

Portlet zabojnik obravnava sočasne zahtevek na istem portletu s sočasnim klicom metod za obravnavo zahtevkov na različnih nitih. Zato morata biti metodi processAction in render narejeni tako, da omogočata sočasno izvedbo zahtevkov.

Implementacija objektov request in response ni nujno varna za uporabo v večnitnem okolju.

2.1.7.6 Napake pri obravnavi zahtevkov

Med obravnano zahtevkov lahko portlet dvigne naslednje napake:

- PortletException (Med izvedbo zahtevka prišlo do napake. Portlet zabojnik mora počistiti zahtevek);
- PortletSecurityException (uporabnik nima potrebni pravic za izvedbo te akcije);
- UnavailableException (portlet ne more obdelati zahtevkov bodisi začasno ali trajno);

Napaka RuntimeException dvignjena med obdelava zahtevka se mora obravnavati kot napaka PortletException.

2.1.7.7 Vmesnik PortletConfig

Vmesnik PortletConfig definira metodi **getInitParameterNames** in **getInitParameter**, ki omogočata dostop do inicializacijskih parametrov definiranih v namestitvenem deskriptorju.

Objekt tipa vmesnik PortletConfig omogoča dostop do informacij potrebnih pri inicializaciji, portlet konteksta in skupkov resursov (resource bundle), ki se uporabljajo npr. za prikaz naslova (title-bar), skrajšanega naslova (short-title), kategorizacijo portleta znotraj portala, ključnih besede. Te informacije se lahko vključijo direktno v definicijo portleta znotraj namestitvenega

deskriptorja ali pa se shranijo v obliki t.i. skupka resursov (resource bundles) pri katerih je v namestitiveni deskriptor vpisan samo razred preko katerega je mogoče priti do omenjenih resursov.

Primer direktne definicije in definicije z uporabo skupka resursov v namestitvenega deskriptorja:

```
<portlet>
 ...
  <portlet-info>
   <title>Stock Quote Portlet</title>
   <short-title>Stock</short-title>
   <keywords>finance,stock market</keywords>
  </portlet-info>
 ...
</portlet>

<portlet>
 ...
  <resource-bundle>com.foo.myApp.QuotePortlet</resource-bundle>
 ...
</portlet>
```

2.1.7.8 Vmesnik PortletURL

Občasno mora portlet kot del svoje vsebine generirati URL-je, ki kažjo na sam portlet. To so t.i. portlet URL-ji. Te URL-je mora portlet generirati s pomočjo vmesnika PortletURL. Mogoče jih je ustvariti s klicom metod createActionURL (kreira action URL) in createRenderURL (kreira render URL) na objektu tipa RenderResponse.

Razlika med action URL-ji in render URL-ji je v tem, da slednji ne povzorčijo klica metode porcessAction. Vmesnik PortletURL ima definirano tudi metodo addParameter, s pomočjo katere lahko portleti dodajo svoje parametre za potrebe spletne aplikacije. Primer uporabe razreda PortletURL:

```
PortletURL url = response.createRenderURL();
url.addParameter("customer","foo.com");
url.addParameter("show","summary");
writer.print("<A HREF=\""+url.toString()+"\">Summary</A>");
```

Preko vmesnika PortletURL je mogoče z uporabo metod setPortletMode in setWindowState nastavljati način prikaza portleta in stanje okna v katerem je prikazan portlet. Portlet ne more nastaviti portlet načina, ki ga portlet ne podpira oz. uporabnik nima ustreznih pravic za njegovo uporabo. V tem primeru mora metoda setPortletMode dviginit napako PortletModeException. Prav tako portlet ne more spremeniti stanje okna v stanje, ki ga ne podpira portlet zabojnik. V tem primeru mora metoda setWindowState dvigini napako WindowStateException.

Vmesnik PortletURL vsebuje tudi metodo setSecure, ki omogoča generiranje t.i. varnih portlet URL-jev (HTTPS oz. HTTP).

2.1.7.9 Portlet načini (modes) delovanja

Portlet način določa funkcijo, ki jo trenutno opravlja portlet. Običajno portlet izvaja različne naloge in generiran različne vsebine gleda na to v katerem načinu se nahaja. Portlet zabojnik ob klicu portleta slednjemu posreduje tudi trenuten portlet način, ki ga lahko portlet v primeru action zahtevka tudi spremeni.

Portlet specifikacija definira tri portlet načine VIEW, EDIT in HELP. Razred PortletMode definira konstanete za te načine.

Dostopnost posameznega portlet načina je odvisan od uporabnikovih vlog(role) oz. njegovih pravic. Anonimen uporabnik lahko npr. uporablja portlet samo v portlet načinih VIEW in HELP. EDIT portlet način pa je vidne samo avtenticiranim uporabnikom.

VIEW portlet način morajo podpirati vsi portleti. Njegova funkcija je generiranje vsebine, ki odraža trenutno stanje portleta. Metoda v razredu GenericPortlet, ki obravnava zahtevke, ko je portlet v VIEW način se imenuje doView.

EDIT portlet način ni potrebno podpreti vsem portletom. Omogoča uporabniku, da prilagodi obnašanje portleta. Običajno portleti v EDIT načinu spreminjajo oz. posodabljajo svoje izbirne nastavitve. Metoda v razredu GenericPortlet, ki obravnava zahtevke, ko je portlet v EDIT način se imenuje doEdit.

HELP portlet način ni potrebno podpreti vsem portletom. V tem načinu portlet prikaže informacije o pomoči glede uporabe portleta. Metoda v razredu GenericPortlet, ki obravnava zahtevke, ko je portlet v HELP način se imenuje doHelp.

Izdelovalci portalov lahko definirajo tudi svoje portlet način.

2.1.7.10 Stanja portlet okna (window states)

Stanje okna je pokazatelj koliko prostora na strani portala bo dodeljenega vsebini generirani s strani portleta. Portleti lahko spremenjajo stanje svojega okna med obdelavo action zahtevka.

V specifikacija so definirana tri stanja okna:

- NORMAL (portlet, si deli spletno stran z drugimi portleti oz. ima naprava, ki prikazuje stran omejen možnosti prikaza);
- MAXIMIZED (portlet je edini portlet oz. ima več prostora v primerjavi z drugimi portleti na spletni strani);
- MINIMIZED (v tem stanju portlet generira minimalno oz. nič vsebine);

Izdelovalci portalov lahko definirajo tudi svoje stanja portlet okna.

2.1.7.11 Vmesnik PortletContext

Vmesnik predstavlja pogled portleta na aplikacijo v kateri se izvaja. S pomočjo objekta tipa PortletContext lahko portlet dostopa do inicializacijskih parametrov konteksta, pridobi oz shrani kontekstne atribute, pridobi statične resurse in usmerjevalnik zahtevkov (request dispatcher) za vključevanje servletov in JSP-jev. Za vsako portlet aplikacijo, ki se namesti v portlet zabojnik se ustvari natanko ena instance objekta tipa PortletContext na virtualni stroja (virtual machine).

2.1.7.12 Vmesnik PortletRequest

Objekt tipa vmesnika PortletRequest vsebuje vse informacije o klientovem zahtevku in še nekatere druge podatke. Vmesnik je oče vmesnikoma :

- ActionRequest (objekti tega tipa se uporabljajo za klice metode processAction);
- RenderRequest (objekti tega tipa se uporabljajo za klice metode render);

Vmesnik PortletRequest definira metode za dostop do parametrov, atributov, lastnosti (properties npr. HTTP glave) klientovega zahtevka. Definira tudi metode za dostopo do kontekstne poti zahtevka (request context path), varnostnih atributov (vsebujejo podatke o uporabniku in povezavi med portalom in uporabnikom), tipov vsebine odgovora (response content types), trenutnega portlet načina , trenutnega stanja portlet okna, internacionalizacije.

Vmesnik ActionRequest dodatno definira metodo za pridobivanje naloženih (uploaded) podatkov.

Vmesnik RenderRequest ne definira nobene dodatne metode.

Vsak objekt tipa PortletRequest je veljavne samo znotraj klica metode processAction ali render. Zato naj se reference na ta objekt ne bi uporabljale zunaj teh dveh metod.

2.1.7.13 Vmesnik PortletResponse

Objekt tipa PortletResponse vsebuje vse podatke, ki jih portlet posreduje portlet zabojniku med oz. po obdelavi zahtevka. Portal/portlet zabojnik uporabi te informacije za generiranje odgovora, običajno portalske strani, ki se nato kot odgovor posreduje klientu. Vmesnik je oče vmesnikoma:

- ActionResponse (objekti tega tipa se uporabljajo za klice metode processAction);
- RenderResponse (objekti tega tipa se uporabljajo za klice metode render);

Vmesnik PortletResponse definira metode za dodajanje in spreminjanje lastnosti (properties), enkodiranje URL-jev.

Vmesnik ActionResponse dodatno definira metode za pošiljanje preusmeritev (redirections), spreminjanje portlet načina, spreminjanje stanja portlet okna, nastavljanje »render« parametrov.

Vmesnik RenderResponse dodatno definira metode za nastavljanje tipa vsebine odgovora (conten type), nastavljanje naslova(title) portleta, enkodiranje imenskega prostora (namespace encoding). Prav tako vmesnik definira metode za generiranje vsebine s pisanjem v objekta tipov OutputStream ali Writer, ter metode za shranjevanje generiranih vsebin v medpomnilnik, zaradi boljše učinkovitosti pri obravani zahtevkov.

Vsak objekt tipa PortletResponse je veljavne samo znotraj klica metode processAction ali render. Zato naj se reference na ta objekt ne bi uporabljale zunaj teh dveh metod.

2.1.7.14 Vmesnik PortalContext

Vmesnik definira metode za dostop do informacij o portalu, ki izvaja portlet. Metode omogočajo pridobivanje informacij o proizvajalcu portala, lastnostih(properties) portala, podprtih portlet načinih in portlet stanjih oken, ki jih podpira portal.

2.1.7.15 Vmesnik PortletPreferences

Portleti običajno omogočajo prirejen izgled oz. obnašanje namenjene različnim uporabnikom. To omogočajo t.i. portlet nastavitve, ki so predstavljene kot pari ime/vrednost. Za dostop in spreminjanje teh nastavitev skrbi portlet zabojnik. Portlet nastavitve so portletu dostopne preko objekta, ki implemenitra vmesnik PortletPreferences in ga je mogoče pridobiti iz zahtevka (request).

Portleti imajo dostop do objekta tipa PortletPreferences samo med obdelavo zahtevka. Hkrati lahko spreminjajo portlet nastavitve samo znotraj metode processAction.

Metoda store vmesnika PortletPreferences omogoča ob njenem uspešnem klicu trajno shranjevanje sprememb portlet nastavitve v trajni pomnilnik. Mogoče jo je poklicati samo znotraj metode processAction.

Primer uporabe portlet nastavitev :

```
String url = prefs.getValue(”quotesFeedURL”,null);
int refreshInterval =
 Integer.parseInt(prefs.getValue(”refresh”,”10”));
```

Specifikacija predvideva, da so portlet nastavitve vezane na uporabnika oz. ima vsak uporabnik portleta svoje portlet nastavitve, ne določa pa načinov za delitev portlet nastavtev med različe uporabnike.

Primer dela portlet nastavitev iz namestitvenega deskriptorja:

```
<portlet>
```

```
...
<!-- Portlet Preferences -->
<portlet-preferences>
      <preference>
            <name>PreferredStockSymbols</name>
            <value>FOO</value>
            <value>XYZ</value>
            <modifiable>0</modifiable>
      </preference>
      <preference>
            <name>quotesFeedURL</name>
            <value>http://www.foomarket.com/quotes</value>
      </preference>
      <validator>com.foo.portlets.XYZValidator</validator>
</portlet-preferences>
</portlet>
```

Preko razredov, ki implementirajo vmesnik PreferencesValidator, je pri uporabi metode store vmesnika PortletPreferences možno izvesti validacijo dodanih oz. spremenjnenih portlet nastavitev in ob napaki ustaviti trajno shranjevanje teh nastavitev.

2.1.7.16 Vmesnik PortletSession

Strežniška sejaomogoča spremljanje oz. povezovanje zahtevkov, ki prihajajo od istega klienta oz. uporabnika. Portlet specifikacija, zato vsebuje definicijo vmesnika PortletSession. Objekt tega tipa je vezan na uporabnika in mora biti enak za vse portlete portlet aplikacije znotraj ene uporabniške seje.

Vmesnik definira naslednja dva področji (scopes), kamor lahko portleti shranjujejo attribute:

- APPLICATION_SCOPE (atributi v tem področju so dostopni vsem portletom znotraj portlet aplikacije);
- PORTLET_SCOPE (atributi v tem področju morajo biti dostopni samo portletu, ki jih je shranil);

2.1.7.17 Vmesnik PortletRequestDispatcher

Vmesnik definira metode, ki omogočajo portletom, da deligirajo generiranje vsebine servletom in JSP-jem. Objekt tipa PortletRequestDispatcher, lahko portlet uporabi samo znotraj metode render. Portlet ta objekt lahko pridobi s klicem ene izmed metod getRequestDispatcher ali getNamedDispatcher na objektu tipa PortletContext.

Primer uporabe objekta tipa PortletRequestDispatcher :

```
String path = "/raisons.jsp?orderno=5";
PortletRequestDispatcher rd = context.getRequestDispatcher(path);
 rd.include(renderRequest, renderResponse);
```

2.1.7.18 Podatki o uporabnikih

Običajno portleti generirajo prilagojeno vsebino glede na uporabnika, ki je poslal zahtevek. Če želijo to početi učinkoviti morajo imeti dostop do uporabniških atributov kot so npr. ime, elektronski naslov, telefon, naslov. Te podatke je mogoče zapisati v namestitveni deskriptor portlet aplikacije.

Primer namestitvenege deskripotrja s podatki o uporabniku:

```
<portlet-app>
…
<user-attribute>
      <description>User Given Name</description>
       <name>user.name.given</name>
</user-attribute>
<user-attribute>
      <description>User Last Name</description>
      <name>user.name.family</name>
</user-attribute>
…
 <portlet-app>
```

Portleti lahko med obdelavo zahtevka pridobijo objekt tipa vmesnik Map, ki ga ne morejo spreminjati in vsebuje podatke o uporabniku. Če je uporabnik avtenticiran vsebuje ta objekt podatke o uporabiku, ki so vpisani v namestitvenem deskriptorju.

Primer pridobivanja uporabniških atributov:

```
Map userInfo = (Map) request.getAttribute(PortletRequest.USER_INFO);
      String givenName = (userInfo!=null)?
      (String) userInfo.get("user.name.given") : "";
```

2.1.7.19 Predpomnjenje

Predpomnjenje (caching) generiranih vsebin omogoča hitrejši odziv portletov na zahteveke uporabnikov. Pripomore tudi k zmanjšanju obremenitve strežnikov. Portlet specifikacija definira t.i. »expiration« predpomnjenje. Predpomnjenje vsebin je vezano na posamezene portlet in uporabnika. Glede na portlet specifikacijo, portlet zabojnikom ni potrebno obvezno implementirati podpore za predpomnjenje.

2.1.7.20 Portlet aplikacija

Portlet aplikacija je spletna aplikacija kot je definirana v specifikaciji Servlet Specification 2.3, v poglavju SRV.9. Poleg JSP-jev, servletov, HTML strani, razredov, itd. Vsebuje portlete in portlet namestitveni deskriptor, ki se nahaja v datoteki /WEB-INF/portlet.xml. Portlet zabojnik naj bi omogočal zamenjavo portlet aplikacij z novo brez ponovnega zagona zabojnika in z ohranjanjem podatkov shranjenih v uporabniških sejah.

2.1.7.21 Varnost

Varnost pri portletih temljeni na varnostnih mehanizmih, ki jih uporabljajo servleti. Portlet zabojnik je zadolžen za sporočanje podatkov o uporabniku oz. uporabniških vlogah (role) portletu do katerege uporabnik dostopa. Za samo avtentikacijo uporabnikov skrbi servlet zabojnik. Portlet specifikcija uporablja iste definicije vlog (roles) kot so definirane v sekciji SRV.12.4 specifikacije Servlet Specification 2.3. V namestitvenem deskriptorju portleta je mogoče z uporabo elementa security-role-ref definirati povezavo na vlogo (role), ki je zapisana v namestitvenem deskriptorju aplikacije oz. v datoteki web.xml.

Portlet specifikacija omogoča uporabo tudi t.i. varnostnih omejitve (security constraints) s pomočjo katerih je mogoče na množici portletov definirati varnostno omejitev, ki zahteva od transportne omrežne plasti, da pri prenosu podatko zagotovi integriteto(content integrity), zaupnost (confidentiality) podatkov. Zabojnik za zagotavljanje ustreznega nivoja varnosti, glede na varnostne omejtive zapisane v namestitvenem deskriptorju, v večini primerov uporabi SSL. Primer uporabe varnostne omejitve v namestitvenem deskriptorju portleta:

```
<security-constraint>
      <portlet-collection>
            <portlet-name>accountSummary</portlet-name>
      </portlet-collection>
      <user-data-constraint>
            <transport-guarantee>CONFIDENTIAL</transportguarantee>
      </user-data-constraint>
</security-constraint>
```

2.2 Specifikacija repozitorija vsebin JSR-170 (JCR 1.0)

Podatki in slike opisani v tem poglavju so povzeti glede na vir [2].

S povečanjem števila ponudnikov repozitorijev vsebin se je pojavila potreba po skupnem vmesniku API-ju, ki bo omogočal standarden način dostopa do repozitorija vsebin. Razvijalcem bo API omogočil zmanjšanje stroškov povezanih z učenjem novega vmesnika API posameznega ponudnika repozitorija vsebin. Uporabnikom pa bodo prihranjeni stroški ob zamenjavi ponudnika, ker standarden vemesnik API omogoča zamenjavo ponudnika brez sprememb same aplikacije. Osnova načela pri načrtovanju vmesnika API so bila :

- Vmesnik API naj bo neodvisen od arhitekture, protokola in izvora podatkov (data source);
- Vmesnik API naj bo enostaven za uporabo;
- Vmesnik API naj omogoča enostavno implementacijo na obstoječih repozitorijih vsebin;
- Hkrati mora vmesnik API standardizirati tudi bolj kompleksne funkcionalnosti za potrebe naprednih vsebinsko usmerjenih (content related) aplikacij;

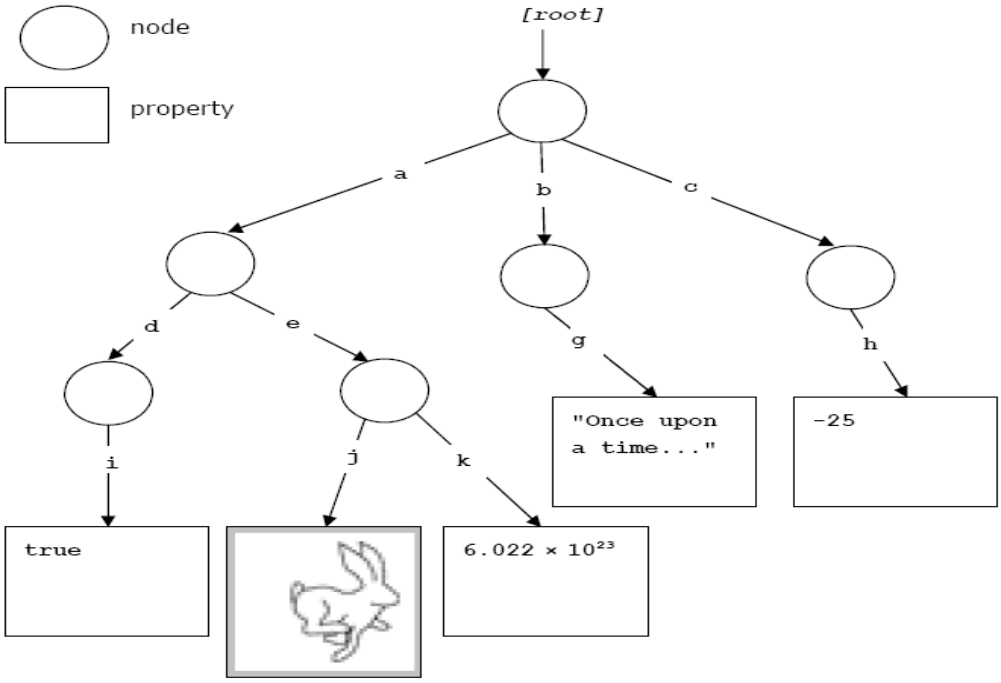
2.2.1.1 Osnovna zgradba repozitorija

Repozitorij vsebin vsebuje enega ali več delavnih prostorov (workspaces). Vsak delavni prostor vsebuje drevo elementov. Element je lahko vozlišče ali lastnost (property). Vsako vozlišče ima lahko nič ali več otrok tipa vozlišče ali tipa lastnost. V posameznem delavnem prostoru obstaja natanko eno t.i. korensko vozlišče (root node), ki nima očeta. Vsa ostala vozlišča imajo vsaj enega očeta. Element tipa lastnost imajo natanko enega očeta oz. vozlišče, ki ni tipa lastnost. Hkrati ne morejo imeti otrok in predstavljajo t.i. liste drevesa. Vsa vsebina drevesa je shranjena v vrednostih elementov tipa lastnost.

Elemente repozitorija vsebin je mogoče naslavljati preko absolutnih in relativnih poti. Absolutne poti se začnejo z znakom / npr. pot /a/d/i v sliki 8 predstavlja lastnost z vrednostjo true. Relativne poti predstavlja pot do elementa, ki je podana relativno glede nek drugi element znotraj drevesa. Na sliki 8 je relativno glede na vozlišče /a je pot do lastnosti true podana za izrazom d/i. Poti je mogoče podajati v t.i. »unix« stilu preko izrazov ».« (this) in »..«(parent oz. oče). Na sliki 8 je, zato mogoče lastnost z vrednostjo -25, relativno glede na vozlišče /a, zapisati kot izraz ../c/h .

Drevesna hierarhija repozitorija vsebin omogoča gradnjo hierarhičnih struktur, ki opisujejo vsebino v repozitoriju. Vozlišča in lastnosti predstavljajo opis strukturo vsebine kot tudi samo vsebino shranjeno v repozitoriju vsebin. Razlika med vozlišči in lastnostmi je v tem, da slednji vsebujejo podatke in meta podatke vsebine, vozlišča pa omogočajo dostop do te vsebine preko

absolutnih in relatvinih poti oz. preko UUID identifikatorja. Vozlišča prav tako omogočajo verzioniranje lastnosti oz. vsebine shranjene v njihovih lastnostih in podrejenih vozliščih. Hkrati drevesna struktura repozitorija vsebin odraža tudi povezavo z strukturo XML dokumentov po kateri so se zgledovali pisci specifikacije pri njenem nastajanju. Od tod izhaja tudi lastnost, da je po repozitoriju vsebin mogoče iskati preko t.i. Xpath izrazov.



Slika 8. Primer drevesa vsebin znotraj delavnega prostora repozitorija vsebin.

2.2.1.2 Osnove vmesnika API

Repozitorij vsebin je predstavljen z razredom Repository. Klient se poveže z repozitorijem preko klica metode Repository.login pri katerem lahko doda kot parametra ime delavnega prostora (workspace name) in objekt Credentials. Klient kot odgovor pridobi objekt Session, ki je vezan na delavni prostor naveden ob klicu metode login. Kako aplikacija pridobi objekt Repository je odvisno od implementacije. Ena izmed možnosti je, da aplikacija pridobi objekt iz JNDI konteksta kot je to narjeno na sliki 9 Vsi razredi omenjeni v specifikaciji so del paketa (package) javax.jcr, razen v primerih, ko je paket eksplicitno zapisan poleg razreda.

```
// Get the Repository object
InitialContext ctx = ...
Repository repository = (Repository)ctx.lookup("myrepo");

// Get a Credentials object
Credentials credentials =
   new SimpleCredentials("MyName",
                         "MyPassword".toCharArray());
// Get a Session
Session mySession =
  repository.login(credentials, "MyWorkspace");
```

Slika 9. Dostopa do repozitoraj vsebin.

2.2.1.3 Dostop do elemetnov repozitorija

2.2.1.3.1 Posredni dostop

Do posameznega elementa repozitorija je mogoče posredno oz. preko drugi elemetnov drevesa dostopati z uporabo metod :

- Node Session.getRootNode();
- Node Node.getNode(String relPath) ;
- Property Node.getProperty(String relPath);
- String Property.getString() ;
- Value Property.getValue() ;
- String Value.getString();

```
// Get the root node
Node root = mySession.getRootNode();

// Traverse to the node you want
Node myNode = root.getNode("a/e");

// Retrieve a property of myNode
Property myProperty = myNode.getProperty("k");

// Get the value of the property
Value myValue = myProperty.getValue();

// Convert the value to the desired type
double myDouble = myValue.getDouble();

// The variable myDouble will contain the
// value 6.022 x 10^23
```

Slika 10. Primer posrednega dostopa do elementa drevesa, glede na sliko 8.

2.2.1.3.2 Direkten dostop

Z uporabo absolutnih poti je mogoče dostopati direktno do posameznega elementa v drevesu delavnega prostora. Za te potrebe specifikacija definira metodo Item Session.getItem(String abspath)

```
Property myProperty =
         (Property)mySession.getItem("/a/e/k");

// Directly convert to a double
double myDouble = myProperty.getDouble();
```

Slika 11. Primer direktnega dostopa do elementa drevesa, glede na sliko 8 z uporabo absolutne poti.

Do nekaterih vozlišč je mogoče dostopati tudi preko t.i. UUID identifikatorja. To omogoča metoda Node Session.getNodeByUUID(String uuid). Identifikator UUID enolično določa vozlišče znotraj delavnega prostora repozitorija vsebin in ga repozitorij vsebin generira avtomatsko za vsako vozlišče, ki je referencabilno oz. ima lastnost mix:referenceable.

```
Node myNode =
     mySession.getNodeByUUID("1111 2222 3333 4444");

// and then get the property and convert it to a double
double myDouble = myNode.getProperty("k").getDouble();
```

Slika 12. Primer direktnega dostopa do vozlišča drevesa, glede na sliko 8 z uporabo identifikatorja UUID.

2.2.1.4 Pisanje v repozitorij

V primeru, da repozitorij implementira 2.nivo združljivosti specfikacije JSR-170 je mogoče preko objekta session tudi pisati vanj. Klient lahko dodaj ali odstranjuje vozlišča in lastnosti, ter spreminja vrednosti lastnosti.

```
// Retrieve a node
Node myNode = (Node) mySession.getItem("/a/e");

// Add a child node
Node newNode = myNode.addNode("n");

// Add a property
newNode.setProperty("x", "Hello");

// Persist the changes
mySession.save();
```

Slika 13. Primer dodajanje novega vozlišča in lastnosti v repozitorij vsebin.

2.2.1.5 Odstranjevanje elementov iz repozitorija

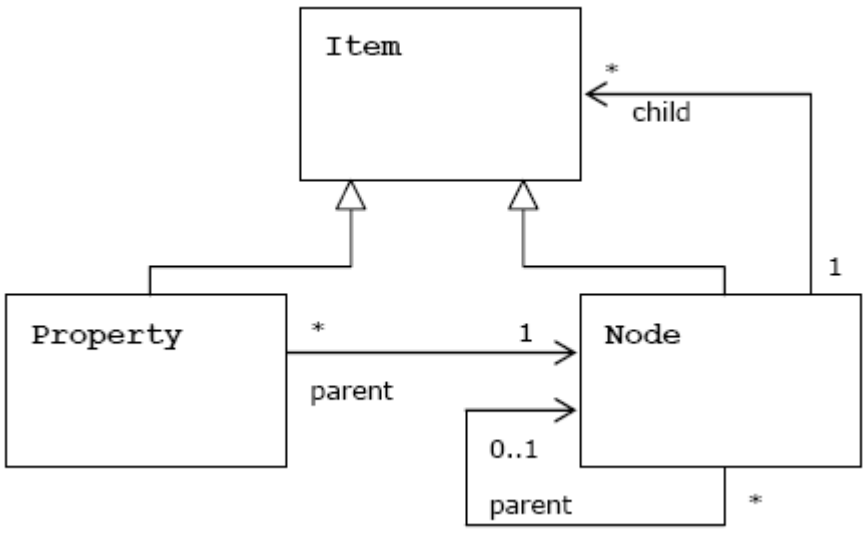
Element je mogoče odstraniti iz repozitorija s klicem metode Item.remove(). V primeru lastnosti lahko kot alternative metode remove() uporabimo nastavljanje vrednosti lastnosti na null. Slednje je mogoče doseči na dva različna načina in sicer s klicoma metod setValue((Value)null) na objktu tipa Property in s klicem metode setProperty("q", (Value)null) na objektu tipa Node.

2.2.1.6 Prehodne(transient) spremembe znotraj seje (session)

Vse spremembe narejene na objektih Session, Node, Property so prehodne (transient) in se ne persisitrajo avtomatsko v delavni prostor. Spremembe se hranijo v prehodnem pomnilniku, ki je povezan z objektom seja vse dokler ni poklicana ena izmed metod Session.save() ali Item.save(). Spremembe na elementih repozitorija je mogoče tudi razveljaviti s klicem ene izmed metod Session.refresh(false) ali Item.refresh(false). Razlika med klicem ene izmed metod save ali refresh na objektu Session in objektu Item je v tem, da slednji omogoča bolj drobnozrnato kontrolo nad spremembami, ki se persistirajo ali zavržejo. Medtem, ko klic metode save ali refresh na objektu Session persistira ali zavrže vse spremembe narejene znotraj seje, ista klica metod na objektu Item persistira ali zavržeta spremembe narejene samo na vozlišču in njegovih otrocih ali lastnosti odvisn od tega, kaj predstavlja objekt Item.

2.2.1.7 Osnovni vmesniki vozlišč in lastnosti

Vmesnika Property in Node sta podvmesnika vmesnika Item, ki vsebuje skupne metode za funkcionalnosti, ki so potrebne za delo z objekti tipa vozlišče oz. tipa lastnost. Vsaka lastnost ima natanko enega očete tipa vozlišče. Vozlišče ima lahko nič (korensko vozlišče) ali enega očeta. Vsako vozlišče ima lahko enega ali več otrok tipa Item.



Slika14. UML diagram osnovnih vmesnikov vozlišč in lastnosti repozitorija vsebin.

2.2.1.8 Nivo združljivosti

Specifikacija JSR-170 defnira dva nivoja združljivosti in množico dodatnih funkcionalnosti, ki jih lahko podpirajo repozitoriji na obeh nivojih.

1. Nivo 1
    - Pridobivanje in sprehajanje po vozliščih in lastnostih;
    - Branje vrednosti lastnosti;
    - Prepisovanje prehodnega imenskega prostora;
    - Izvoz v XML/SAX;
    - Izvajanje poizvedb s sintakso XPath;
    - Odkrivanje možnih tipov vozlišč;
    - Odkrivanje kontrole pravic dostopa;
2. Nivo 2
    - Dodajanje in odstranjevanje vozlišč in lastnosti;
    - Zapisovanje vrednosti lastnosti;
    - Spreminjanje obstoječih imenskih prostorov;
    - Uvoz iz XML/SAX;
    - Dodeljevanje tipov vozlišč vozliščem;
3. Dodatne funkcionalnosti
    - Transakcije;
    - Verzioniranje;
    - Opazovanje, dogodki (Observation, Events);
    - Zaklepanje (Locking);
    - Izvajanje poizvedb s sintakso SQL;

2.2.1.9 Imenski prostori (namespaces)

Vozlišče ali lastnost imata lahko kot del imena predpono (prefix), ki je zaključena z znakom ":" in predstavlja imenski prostor elementa.

Imenski prostor v repozitoriju vsebin je povzet po imenskem prostoru, ki se uporablja v XML-ju. Kot v XML-ju je predpona okrajšava za celotno poimenovanje imenskega prostora, ki ga predstavlja t.i. URI. URI-ji se uporabljajo kot imenski prostori z namenom zmanjšanja konfliktov pri poimenovanju. Vsak repozitorij združljiv po 1. ali 2. nivoju specifikacije JSR-170 ima register imenskih prostorov, ki vsebuje vsaj naslednje imenske prostore :

- jcr (rezerviran za elemente, ki uporabljajo že vgrajene tipe vozlišč);
- nt (rezerviran za imena vgrajenih tipov primarnih vozlišč);
- mix (rezerviran za imena vgrajenih tipov t.i. "mixin" vozlišč);
- xml (rezerviran zaradi kompatibilnosti z XML-jem) ;
- "" (prazna predpona predstavlja privzet imenski prostor) ;

V repozitoriju združljivem po nivoju 1 je se lahko začasno predpona vsakemu registriranemu imenskemu prostoru spremeni za čas trajanja seje. Združljivost repozitorija po nivoju 2 omogoča dodatne možnosti dodajanja, odstranjevanja in spreminjanja imenskih prostorov shranjenih v registru imenskih prostorv (ne velja za zgoraj omenjene imenske prostore).

2.2.1.10 Tipi lastnosti

Vsaka lastnost v repozitoriju je enega izmed naslednjih tipov:

- PropertyType.STRING
- PropertyType.BINARY
- PropertyType.DATE
- PropertyType.LONG
- PropertyType.DOUBLE
- PropertyType.BOOLEAN
- PropertyType.NAME
- PropertyType.PATH
- PropertyType.REFERENCE

Lastnosti tipa NAME se uporabljajo za shranjevanje imen v obliki nizov (strings), ki so umeščena v določen imenski prostor. To so npr. imena tipov vozlišč in imena elementov repozitorija. Lastnost tipa NAME si lahko predstavljamo kot niz, ki se zaveda svojega imenskega prostora. Lastnost se nastavlja s klicem metode setProperty("aNodeType", "nt:file"), vendar pa se pri shranjevanju predpona nt zamenja s celotnim URI-jem imenskega prostora. Pri branju vrednosti te lastnosti se postopek ponovi v obratno smer oz. celotnim URI imenskega prostora se zamenja s predpono.

Lastnosti tipa PATH predstavlja pot v delavnem prostoru (relativni ali absolutno). Lastnost ne uveljavlja referenčne integritete oz. drugače povedan lahko kaže na pot, ki več ne obstaja v repozitoriju. Tudi ta lastnost se zaveda imenskega prostora, kar se odraža v predponi lastnosti pri njenem branju, ki je vedno enaka trenutni vrednosti predpone za določen URI.

Lastnosti tipa REFERENCE predstavlja referenco na neko drugo vozlišče znotraj delavnega prostora. Vrednost lastnosti je t.i. UUID vozlišča na katerega se nanaša. Lastnost se lahko nanaša samo na vozlišča, ki so referenciabilna. Ta lastnost onemogoča izbris vozlišča, iz repozitorija oz. povedano drugače ohranja referenčno integriteto repozitorija. Metodo Node.getReferences() je mogoče uporabiti za pridobivanje vseh lastnosti tipa REFERENCE, ki kažejo na dotično vozlišče. Z uporabo metode Node.setProperty(String name, Node value) je mogoče nastaviti vrednost lastnosti tipa REFERENCE na specifičen UUID določenega vozlišča.

Pri lasnostih je potrebno omeniti, da ne smejo zavzeti vrednosti null. Če želimo lastnosti dodeliti vrednost null je to ekvivalentno odstranitvi omenjene lastnosti iz repozitorija, kar moramo v tem primeru tudi storiti.

2.2.1.11 Tipi vozlišč

Vsako vozlišče mora imeti natanko en primaren tip vozlišča. Primarni tip vozliša določa imena, tipe in ostale karakteristike lastnosti in vozlišč otrok, ki jih lahko ima oz. jih mora imeti obezno vozlišče s tem primarnim tipom. Primarni tip vozlišča se določi z lastnostjo jcr:primaryType.

Poleg primarnega tipa imajo lahko vozlišča tudi enega ali več t.i. mešanih (mixin) tipov. Ti določajo dodatne karakteristike za vozlišča otroke, lastnosti, ter njihovih imen in tipov. Mešani tipi vozlišč se določajo preko večvrednostne lastnosti jcr:mixinTypes.

Nivo 1 specifikacije definira metode za odkrivanje tipov obstoječih vozlišč in odkrivanje, ter branje definicij tipov vozlišč, ki so dostopna v repozitoriju.

Nivo 2 specifkacije definira metode za prirejanje primarnih in mešanih tipov vozliščem.

2.2.1.12 Referencabilna vozlišča

Referenciablina vozlišča predstavljajo osnovo za mnoge funkcionalnosti repozitorija kot so verzioniranje in več delavnih prostorov.

Če repozitorij podpira referenciabilna vozlišča mora podpirati tudi tip mix:referenceable. Ta tip zahteva od vozlišča, da ima prirejeno tudi lastnost tipa jcr:uuid. Lastnost jcr:uuid je obvezna, zaščitena in avtomatsko kreirana. Kar pomeni, da njeno generiranje in urejanje nadzoruje sistem, klienti pa jo lahko samo berejo. Vrednost lastnosti jcr:uuid predstavlja t.i. UUID ali enolični identifikator vozlišča. V vsakem delavnem prostoru ne sme biti več kot eno vozlišče z določenim UUID-jem.

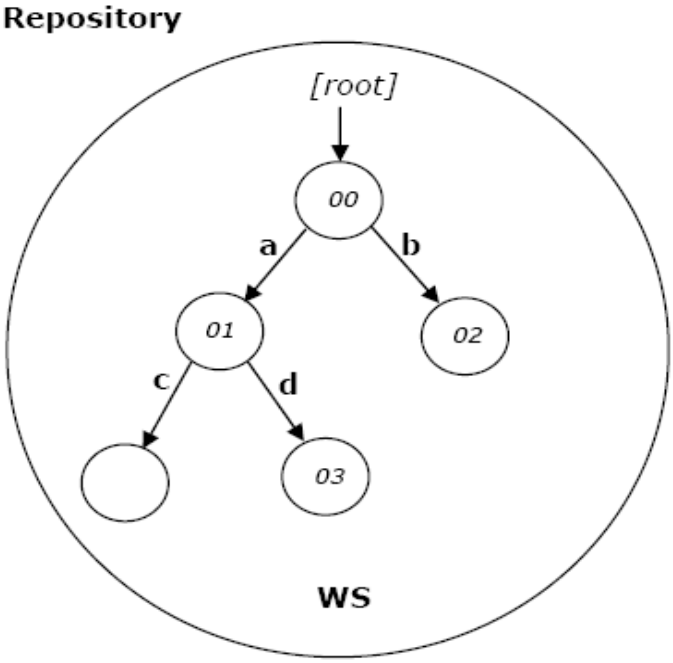
UUID referenciablinega vozlišča lahko uporabimo v lastnostih tipa PropertyType.REFERENCE. Lastnost tipa REFERENCE shrani UUID vozlišča znotraj istega delavnega prostora in uveljavi referenčno integriteto. Izjema uveljavljana referenčne integritete lastnosti tipa REFERENCE je dovoljena samo v primeru, ko je lastnost shranjena kot verzionirano stanje v pomnilnik verzij.

2.2.1.13 Delavni prostori (workspaces)

Repozitorij vsebin je sestavljen iz številnih delavnih prostorov. Vsak izmed njih vsebuje eno drevo elementov s korenskim vozliščem. V najenostavnješem primeru repozitorij vsebuje samo en delavni prostor, v bolj kompleksnih primerih pa jih vsebuje več.

2.2.1.13.1 Repozitorij z enim delavnim prostorom

Repozitorij z enim delavnim prostorom vsebuje natanko eno drevo z vozlišči in lastnostmi. To hkrati tudi pomeni, da je največ eno vozlišče z dodeljenim UUID-jem znotraj celotnega repozitorija.



Slika 15. Repozitorij z drevesom elementov. Puščice usmerjene od očeta k otroku so poimenoven z imenom otroka. Ime korenskega vozliča predstavlja prazen niz, vendar je na sliki zaradi lažjega razumevanja prikazan kot niz [root]. Števila v vozliščih predstavljajo enolične identifikatorje t.i. UUID-je. Vozlišče /a/c nima UUID-ja, ker ni referenciabilno.

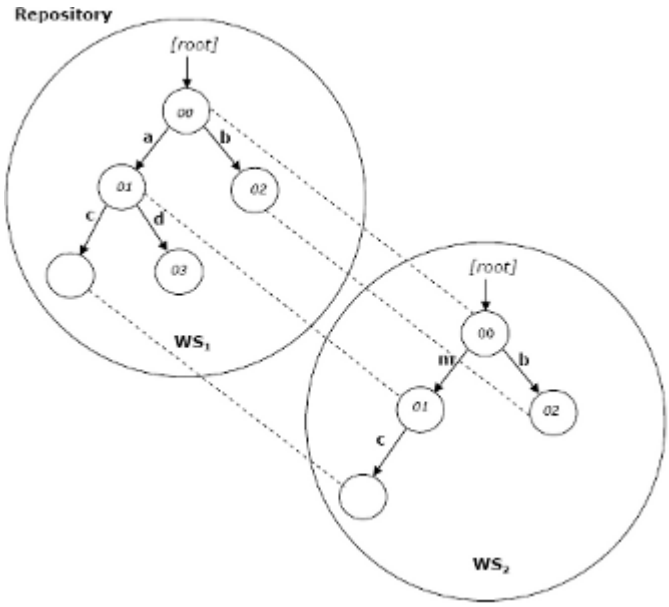
2.2.1.13.2 Repozitorij z večimi delavnimi prostorom

V repozitoriju z večimi delavnimi prostori ima lahko vozlišče v enem delavnem prostoru korespondenčno vozlišča tudi v drugih delavnih prostorih. Korespondenčno vozlišče je definirano kot :

- Vozlišču korespondenčno vozlišče je takšno, ki ima korespondenčni identfikator;
- Korespondenčni identifikator referenciabilnega vozlišča je njegov UUID;
- Korespondenčni identifikator vozlišča, ki ni referenciabilno z vsaj enim referenciabilnim naslednikom je sestavljeno iz UUID-ja najbližjega referenciabilega predhodnika in relativne poti od predhodnika;
- Korespondenčni identifikator vozlišča, ki ni referenciabilno brez referenciabilnih predhodnikov je njegova absolutna pot;

Korespondenčnim vozliščem je skupen samo korespondenčni identifikator. Lahko imajo popolnoma različne lastnosti in vozlišča otroke. Pri tem je potrebno poudariti, da je še vedno samo eno vozlišče z določeni UUID-jem znotraj posameznega delavnega prostora.

Metoda Node.update(String srcWorkspace) omogoča zamenjavo vozlišča in njenega poddrevesa z vozlišče in poddrevesom iz delavnega prostora srcWorkspace.



Slika 16. Povezava med dvema repozitorijema oz. drevesoma elementov. Črtkane črte prikazujejo korespondenčna vozlišča z enakimi UUID-ji. Vozlišči /a/c in /m/c, sta korespondenčni, ker imata obe enako relativno pot od svojega najbližjega referenciabilnega vozlišča c.

2.2.2 Verzioniranje

Podpora repozitorija vsebin verzioniranju je opcijska. Sistem verzioniranja je narejen z uporabo delavnih prostorov in referenciabilnih vozilišč.

Če repozitorij podpira verzioniranje, lahko delavni prostori vsebujejo vozlišča, ki jih je mogoče, ter vozlišča, ki jih ni mogoče verzionirati. Vozlišče je mogoče verzionirati, če mu je dodeljena lastnost z mešanim tipom mix:versionable. Tip mix:versionable je podtip tipa mix:referenceable, kar pomeni, da se vozlišču avtomatsko dodeli UUID v primeru, da mu je dodeljen tip mix:versionable.

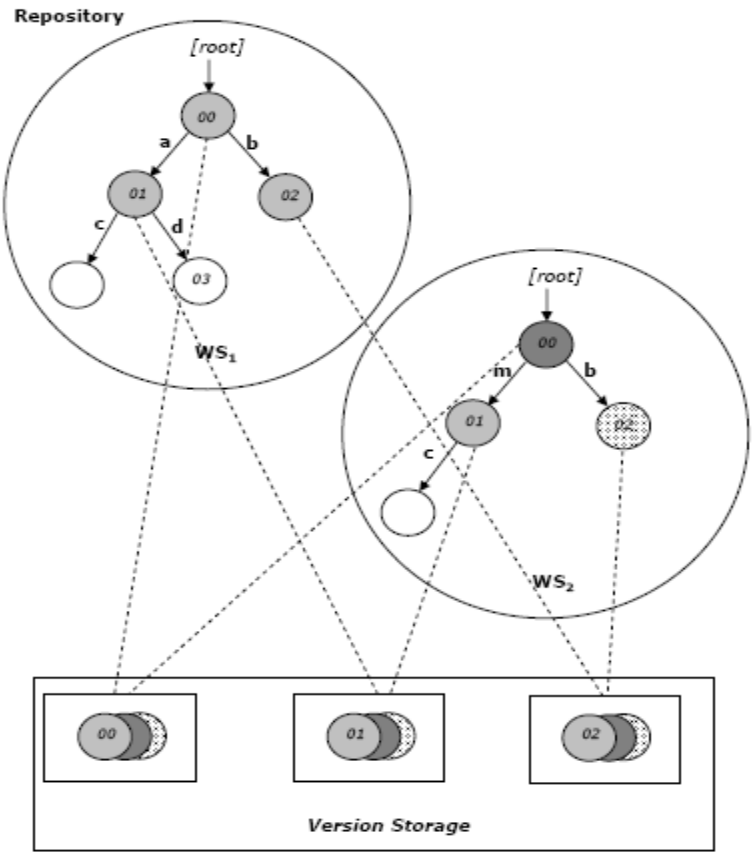
Če je vozlišče mogoče verzionirati, to pomeni, da je v vsakem trenutku mogoče shraniti njegovo stanje, ter vozlišče v prihodnosti povrniti nazaj v shranjeno stanje. Shranjeno stanje vozlišča se imenjue verzija (version), akcija shranjevanja verzije pa t.i. “checking in”.

Verzija obstaja kot del zgodovine verzij. Znotraj zgodovine verzij, verzija predstavlja graf verzij, ki opisuje relacije predhodnik/naslednike med verzija določenega verzioniranega vozlišča.

Zgodovine verzij so shranjene v pomnilniku verzij. Če repozitorij omogoča verzioniranje vsebuje tudi natanko en pomnilnik verzij, ki je dostopen znotraj vsakega delavnega prostora kot posebno zaščiteno poddrevo vozlišča /jcr:system/jcr:versionStorage.

Relacija med vozlišči in zgodovino verzij je narejena z uporabo UUID-jev. Podrobnosti te uporabe so naslenje:

- Vsaka množica korespondenčnih vozlišč (z enakim UUID-jem), si deli isto zgodovino verzij;
- V vsakem delavnem prostoru obstaja največ eno verzionirano vozlišče na zgodovino verzije;
- V delavnem prostoru lahko obstajajo zgodovine verzij, za katere ta delavni prostor ne vsebuje več korespondenčnih vozlišč;
- Delavni prostor lahko vsebuje vozlišča, ki jih ni mogoče verzionirati;
- Zgodovina verzij se avtomatsko ustvari v pomnilniku verzij ob kreiranju vozlišča, ki ga je mogoče verzionirati;
- Če je verzionirano vozlišče klonirano v drug delavni prostor, ohrani enak UUID, hkrati ostane zgodovina verzij za novo kreirano verzionirano vozlišče povezana z originalno zgodovino verzij;



Slika 17. Repozitorij, ki vsebuje dva delavna prostora. Pomnilnik verzij je prikazan kot pravokotnik na dnu slike. Vsebuje zgodovine verzij vseh verzioniranih vozlišč v repozitoriju. Verzionirane vozlišča so prikazana kot osenčena, neverzionira vozlišča pa so bele barve.

Zgodivina verzij je na sliki 17 prikazana kot sklad različno osenčenih krogov. Vsako verzionirano vozlišče si deli zgodovino verzij s korespondenčnimi vozlišči v drugih delavnih prostorih.V diagramu delavni prostor WS1 vsebuje svetlo siva vozlišča verzij vozlišč 00, 01 in 02. Delavni prostor WS2 pa vsebuje temno sivo vozlišče verzije 00, svetlo sivo vozlišče verzije 01 in črtkano vozlišče verzije 02.

Slika 17 predstavlja poenostavitev, ker vsaka zgodovina verzij vsebuje samo 3 verzije. V dejanskem sistemu lahko verzije različnih vozlišč razlikujejo, poleg tega pa lahko vsebujejo tudi relacije oče-otrok.

2.3 Specifikacija repozitorija vsebin JSR-283 (JCR-2.0)

Podatki in slike opisani v tem poglavju so povzeti glede na vir [3].

Specifikacija predstavlja razširitev specifikacije JSR-170 oz. predstavlja njeno nadgradnjo. Predmet specifikacije so predvsem naslednje razširitve :

- upravljanja kontrole dostopov;
- novi tipi vozilišč (nodetype) t.i. deljena vozlišča;
- registracija novih tipov vozlišč;
- vsako vozlišče ima identifikator;
- upravljanje življenskega cikla repozitorija;
- enostavno verzioniranje;
- abstraktni model povpraševanja;

2.3.1 Upravljanje kontrole dostopov

Specifikacija uvaja možnost upravljanja dostop do repozitorija vsebin z uporabo privilegijev. Privilegij predstavlja zmožnost opravljanja določene množice opravil nad elementi repozitorija.Vsak privilegij se identificira z imenom, ki je edinstven med vsemi privilegiji, ki jih podpira repozitorij. Prvilegij je lahko tudi agregiran oz. je lahko sestavljen iz množice privilegijev, ki so lahko tudi agregirani, vendar pa ne sme vsebovati cikličnih povezav. Repozitorij omogoča tudi abstraktne privilegije, ki jih ni mogoče uporabiti samostojno, ampak jih je mogoče uporabiti samo v agregiranih privilegijih. Privilegij je lahko hkrati abstrakten in agregiran.

2.3.2 Deljena vozlišča

Deljeno vozlišče je tip vozlišča, ki lahko deli svoje lastnosti in podrejena vozlišča z enim ali večimi vozlišči. Dve deljeni vozlišči imata enake lastnosti in podrejena vozlišča, ter različni imeni, indeks in pot. Namen deljivih vozlišč je možnost polnenja večih vozlišč na različnih poteh hkrati.

Deljena vozlišča so združena v t.i. deljeno množico (shared set) in jih je mogoče pridobiti s klicom metode javax.jcr.Node.getSharedSet(). Tip vozlišča mix:shareable označuje, da je vozlišče deljeno.

2.3.2.1 Podrejena vozlišča deljenih vozlišč

Vsa vozlišče v deljeni množici si delijo ista podrejena vozlišča. To pomeni, da dodajanje oz. odstranitev podrejenga vozlišča deljenemu vozlišču N , avtomatično doda oz. odstrani podrejeno vozlišče tudi vsem vozliščem v deljeni množici, ki vsebuje tudi deljeno vozlišče N.

2.3.2.2 Lastnosti deljenih vozlišč

Vsako deljeno vozlišče v deljeni množici si deli lastnosti in njihove vrednosti. Ob spremembi, odstranitvi ali dodajanju lastnosti je sprememba lastnosti takoj vidna na vseh vozliščih deljene množice.

2.3.2.3 Identifikatorji

Ker je deljeno vozlišče referenciabilno ima referenciabilen identifikator dostopen preko lastnosti jcr:uuid. To pomeni, da imajo vsa deljena vozlišča v deljeni množici prav tako isti identifikator. Zaradi slednjega mora klient, ki komunicira z repozitoriji, ki podpirajo deljena vozlišča, omogočati delo z več vozlišči z enakim identifikatorjem.

2.3.2.4 Prehodne (transient) spremembe vozlišč

Ko je narejena sprememba na deljenem vozlišču postane vrednost metode Node.isModified true. Ta sprememba je vidna na vseh vozliščih, ki pripadajo deljeni množici tega vozlišča. Ob shranitivi vozlišča postane vrednost metode Node.isModified false. Ta sprememba je prav tako vidna na vseh vozliščih, ki pripadajo deljeni množici tega vozlišča.

2.3.2.5 Deljeni cikli

Deljeni cikel nastane v primeru, ko je vozlišče v isti deljeni množici kot njegov predhodnik. Implementacija repozitorij lahko v tem primeru prepreči pojavitev deljenih ciklov in v primeru pojavitve slednjega dvigne napako ShareCycleException. V primeru, da se aplikacija sprehaja po drevesu repozitorija, ki ne preprečuje deljenih ciklov, je dolžnost aplikacije, da preverja ali se nahaja v ciklu.

2.3.3 Registracija tipov vozlišč

Repozitorij nivoja 2 lahko omogoča tudi registracijo tipov vozlišč. Tipe vozlišč je mogoče registrirati preko vmesnika NodeTypeManager oz. razreda, ki ga implementira. Edina omejitev pri definiranju novega tipa vozlišča je, da mora implemetacija preprečiti registracijo vsakega tipa vozlišč, ki uporablja rezerviran imenski prostor bodisi v imenu ali v katerem od njegovih definicij enot(item). Implemetacija lahko poleg omenjene omejitve dodatno omeji tipe vozlišč, ki jih je mogoče registrirati. Slednje je namenjeno predvsem primerom, ko je repozitorij zgrajen na vrhu obstoječega shranjevalnika (store) vsebin, ki ima notranje omejitve pri predstavitvi novih tipov vozlišč.

2.3.4 Identifikatorji

Vsako vozlišče ima svoj identifkator, ki je dostopen preko metode Node.getIdentifier(). Identifikator je niz, ki ni definiran v specifikacji JSR-283, vendar spoštuje naslednja pravila:

- Je edinstven znotraj delavnega prostora;
- Implementacija mora vzeti najbolj stabilen identifkator, ki ji je dostopen. Za preproste implemetacija je to lahko že pot do vozlišča;

Nekater implemetacije vsebujejo referencabilna vozlišča. Slednje pa morajo vsebovati referenciabilne identifikatorje, ki morajo spoštovati dodatna pravila:

- Morajo biti dodeljeni najkasneje, ko je vozlišče persistirano;
- So nespremenljivi celotno življenje vozlišča, vse do njegovega izbrisa;
- Še posebaj velja, da so nespremenljivi pri operacjah move in clone;
- Niso nespremenljivi pri operaciji copy;

2.3.5 Upravljanje življenskega cikla

Repozitorij lahko podpira osnove operacije za upravljanje z življenskim ciklom. Repozitorij, ki podpira te funkcionalnosti omogoča:

- Odkrivanje stanja vsebine znotraj življejnskega cikla;
- Odobritev oz. zavrnitev vsebine pri prehodu iz trenutnega stanja v novo stanje znotraj življenskega cikla;

Samo vozlišča, ki vsebujejo tip mix:lifecycle lahko sodelujejo v življenskem ciklu.

2.3.6 Enostavno verzioniranje

Enostavno verzioniranje omogoča linearno zgodovino verzij brez združevanja (merge) in vejanja (branch) verzij. Zgodovina verzij je dostopna preko vmesnika API, vendar se ne odraža v vsebini.

Mešani tip (mixin) mix:simpleVersionable označuje, da je vozlišče enostavno verziabilno. Tip predstavlja podtip tipa mix:referenceable, hkrati definira tip tudi lastnost jcr:isCheckedOut, ki je tipa BOOLEAN. Lastnost pove ali je vozlišče v checked-in ali checked-out stanju. Z verziabilnimi vozlišči je mogoče upravljati preko metod Node.checkin(), Node.checkout() in Node.checkpoint(). Node.checkpoint() je zaporedje klicov metod Node.checkin() in Node.checkout(), ki omogoča generiranje generiranje periodričnih verzij med obdelavo vsebine.

Shranjene verzije vozlišč je mogoče restavrirati s klicemom metode Node.restore() oz. N.restoreByLabel(). Klica teh dveh metod se izvedeta ne glede na to ali je vozlišče v stanju checkin oz. checkout. Preko klica metode Workspace.restore(Version[] versions, boolean removeExisting) je mogoče restavrirati tudi več vozlišč oz. njihovih shranjenih verzij na enkrat. Metoda Node.update(String srcWorkspace) zamenja vozlišče Node in njegovo poddrevo s klonom vozlišča in njegovega poddrevesa iz delavnega prosotora srcWorkspace.

Za razliko od polnega verzioniranja, enostavno verzioniranje ne zahteva, da se shramba verzij odraža tudi na vsebini. To pomeni, da zadostuje je, če je zgodovina verzij dostopna preko vmesnika API. Zaradi tega je enostavno verzioniranje popolnoma odvisno od implementacije.

2.3.7 Abstrakten model povpraševanja (query)

Popraševalni jezik javanskega repozitorija vsebin temelji na abstraktnem modelu povpraševanj (AQM), ki predstavlja strukturo in semantiko povpraševanja. JSR-283 vsebuje oz. definira dva konkretna jezika vezana na AQM:

- **JCR-SQL2,** v katerem je povpraševanje izraženo v obliki niza podobnega povpraševalnemu jeziku SQL;
- **JCR-JQOM** (JCR java query object model), v katerem je povpraševanje izraženo v obliki drevesa javanskih objektov;

Ker sta oba jezika neposredni preslikavi AQM-ja sta oba enako izrazna in ju je mogoče transformirati iz enega v drugega. Specifikacija zahteva, da mora implementacija podpreti oba povpraševalna jezika, če želi biti repozitorij kompatibiln s 1. nivojem specifikacije. Repozitorij lahko poleg teh dveh jezikov vsebuje tudi dodatne povpraševalne jezike, ki so bodisi preslikani v AQM ali pa so popolnoma neodvisni od tega modela.

JCR 1.0 je definiral SQL dialekt, ki je različen od JCR-SQL2 in povpraševanje preko dialekta jezika Xpath. Podpora pravkar omenjenima jezikoma je JCR 2.0 ustavljena in je vpisana v specifikacija samo zaradi kompatibilnost s specifikacijo JCR 1.0.

2.3.7.1 Osnove AQM

1. Selektorji(selectors): eno povpraševnje ima enega ali več selektorjev. Ko se povpraševanje obravnava, vsak selektor izbere podmnožico vozlišč iz delavnega prostora na osnovi njihovega tipa;
2. Združitve (joins): če ima povpraševanje več kot en selektor, potem ima tudi enega ali več združitev, ki transformirajo množice vozlišč, izbranih s selektorji, v eno množico dvo-terk vozlišč (node tuples);
3. Omejitve(constraints): povpraševanje lahko definira omejitve in tako filtrira množico dvo-terk vozlišč. Povpraševanje je možno omejti z uporabo absolutnih ali relativnh poti, imena vozlišča, vrednost lastnosti, dolžine lasnosti, obstoja lastnosti, full-text iskanja.
4. Urejanje vrstnega reda(orderings): povpraševanje lahko definira urejanje za potrebe sortiranja množice dvo-terk vozlišč preko vrednosti lastnosti.
5. Rezultat povpraševanja(query results): izvedba povpraševanja generira t.i rezultat povpraševanja, ki je lahko dveh formatov. Prvi format predstavlja seznam množice dvo-terk vozlišč, drug format pa je matrika množice dvo-terk vozlišč, katere vrstice so dvo-terke vozlišč, stolpci pa predstavljajo lastnosti vozlišč iz dvo-terk. Drugi način predstavlajo t.i. tabelarni pogled rezultata povpraševanja. Povpraševanje lahko definira stolpce in tako kontrolira, katere lastnosti se bodo prikazale v tabelarnem pogledu.

2.4 Specifikacija JavaServer Faces JSR-127 (JSF 1.0)

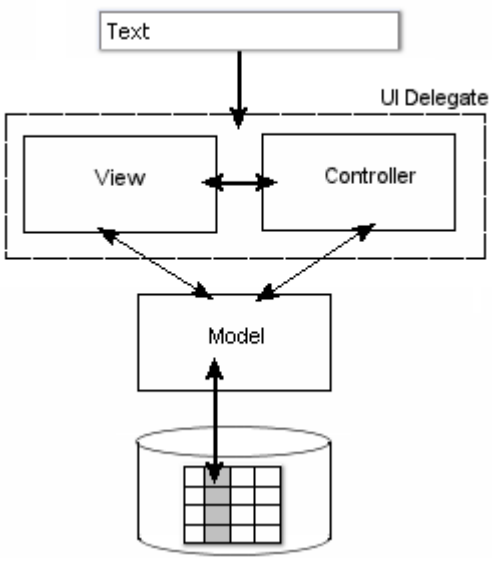
Podatki in slike opisani v tem poglavju so povzeti glede na vir [4].
JavaServer Faces (JSF) je ogrodje za generiranje uporabniških vmesnikov(UI) v javanskih spletnih aplikacijah. Ogrodje je narejeno z namenom olajšanja pisanja in vzdrževanja aplikacij, ki se izvajajo na javanskih aplikacijskih strežnikih in izrisujejo UI za prikaz na oddaljenih klientih. JSF oljaša delo programerjev na naslednjih področjih:

1. Olajša gradnjo UI s pomočjo množice ponovno uporabljivih komponent;
2. Olajša migracijo podatkov od in do uporabniškega vmesnika;
3. Pomaga pri upravljanju UI stanja med strežniškimi zahtevki (server requests);
4. Vsebuje enostaven model za prenos klinetovih dogodkov (events) do strežnika oz. kode spletne aplikacije;
5. Omogoča gradnjo novi UI komponent in njihovo ponovno uporabo;

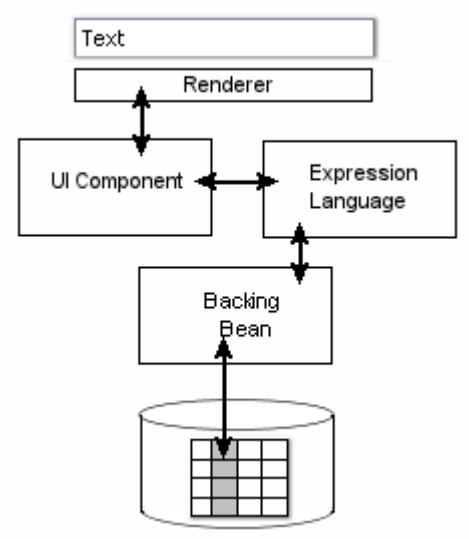
JSF ogrodje vzpostavlja standarde, ki so narejeni z namenom, da jih uporabijo orodja in s tem olajšajo razvijalcem razvoj spletnih aplikacij.

2.4.1.1 Primerjava z java swing

Java swing je del JFC (java foundation classes) in predstavlja komponente grafičnega uporabniškega vmesnik (GUI) za javanske (predvsem) namizne programe. Podatki v tem poglavju so povzeti po [37].



Slika 18. Arhitektura java swing.



Slika 19. Arhitektura JSF.

Swing je enonitni programski model, katerega arhitektura sledi programskemu vzorcu MVC (Model-View-Controller) . Naslednji seznam podaja razlike in podobonosti med tehnologijama JSF in swing:

- Razvoj: s stališča razvoja večina javanskih orodij za razvoj programov podpira tako razvoj swing aplikacij kot JSF aplikacij.
- Komponente: obe tehnologiji uporabljata JavaBeans komponente oz. javankse razrede za izgradnjo uporabniških vmesnikov. Drugače povedano obe tehnologiji temeljita na javanskih koponentah.
- Strani in postavitev: postavitev je v swing-u narejena z uporabo gnezednih swing plošč(panel). Pozicijo in obnašanje komponent ob povečavah je v swing-u narejeno z uporabo LayoutManagers. Podobno je postavitev v JSF-jih narejena z uporabo zabojnikov, ki imajo lahko več komponet otrok(lahko so tudi drugi zabojniki).
- Izgled (look and feel): swing podpira dinamične izglede z uporabo razredov, ki se nastavijo v aplikaciji ob njenem zagonu. Izgled aplikacije je mogoče zamenjati brez popravila kode aplikacije. JSF-ji uporablja prekrivne sloge (CSS) za spreminjanje izgleda aplikacij.
- Internacionalizacija: Obe tehnologiji omogočata uporabo prevodov nizov preko t.i. skupko resursov (resource bundles). JSF pridobijo priljubljen uporabniški jezik iz nastavitev klientovega brskalnika . Swing pridobijo priljubljen uporabniški jezik iz lokalne nastavitve na uporabnikovem operacijske sistemu javanskega izvajujočega (runtime) okolja.

- Dogodkovni model: komponente obeh tehnologij uporabljajo javanski JavaBean dogodkovni model za objavljanje informacij o spremembah v modelu ali komponentah. Komponente, ki jih zanimajo omenjeni dogodki lahko se lahko naročajo na dogodke z uporabo javanskih JavaBean poslušalcev(listeners).
- Integracija z namizjem: lokalno nameščene swing aplikacije imajo popolen dostop do klientovega namizja. JSF omogoča omejen dostop do klientovega namizja samo z uporabo nalaganja in snemanja (upload/download) datotek.
- Navigacija: swing omogoča samo programsko navigacijo oz. ne vsebuje nobenega zunanjega kontrolorja (controller), ki bi določal kater plošča naj se prikaže glede na izid določene akcije. JSF uporablja kontrolno komponento, ki jo je mogoče konfigurirati in s tem določiti navigacijske primere glede na izide določenih akcij.
- Kontrola aplikacije: swing omogoča večjo kontrolo aplikacije na klientu kot JSF. Uporabniki izvajajo swing aplikacije znotraj javanskega izvajajočega okolja (Java runtime environment) medtem, ko se JSF aplikacije izvajajo znotraj spletnih brskalnikov. Slednje predstavlja problem predvsem zaradi pomanjkanja standardov brskalnikov in skupnih API-jev.
- Varnost: JSF je integriran z deklerativno J2EE varnostjo in poenostavlja zaščito spletnih aplikaciji in enotno prijavo (single sign on). Swing je težje zaščititi in uporablja J2SE varnost, ki temelji na pravicah s kombinacijo javanskih avtentikacijskih in avtorizacijskih servisih (JAAS java authentication and authorization service).

2.4.1.2 JSF in splet

Osnovna arhitektura JSF ogrodja je narejena neodvisno od protokla, vendar je usmerjena direktno k reševanju problemov, ki nastajajo pri razvoju spletnih aplikacij, ki komunicirajo preko protokola HTTP. JSF rešuje naslednje težave povezane s spletnimi aplikacijami:

1. Upravljanje stanja UI komponent preko večih strežniških zahtevkov;
2. Enkapsulira razlike med prikazom označitvenih jezikov v različnih brskalnikih in klientih;
3. Podpira obdelavo večih spletnih form (več stranskih ali večih na eni strani);
4. Vključuje dogodkovni model, ki omogoča aplikaciji definiranje strežniških obdelovalnikov (handlers) za dogodke, ki se generirajo na strani klienta;
5. Validacija podatkov iz zahtevkov in ustrezno sporočanje napaka;
6. Koverzija tipov pri migraciji podatkov iz objektov v vrednosti označitvenega jezika (String) in obratno;
7. Obravnava napak in njihov sporočanje aplikacijskemu uporabniku v človeško berljivi obliki;

8. Obravnava navigacije med stranimi kot odgovor na UI dogodke;

2.4.2 Življenski cikel obravnavanja zahtevkov

Vsak zahtevek (request), ki vključuje JSF drevo komponent (t.i. view) potuje čez dobro definiran življenski cikel obdelave zahtevka, ki je sestavljen iz faz. Upoštevati je potrebno tri različne scenarije, vsakega s svojo kombinacijo faz in aktivnosti:

- Non-Faces zahtevek (request), ki generira Faces odgovor (response);
- Faces zahtevek (request), ki generira Faces odgovor (response);
- Faces zahtevek (request), ki generira Non-Faces odgovor (response);

Faces odgovor(response): Odgovor, ki je bil generiran z izvedbo Render Response faze življenksega cikla obdelave zahtevak.

Non-Faces odgovor (response): servlet ali JSP odgovor, ki ne vključuje JSF komponent

Faces zahtevek(request): Zahtevek, ki je bil poslan iz predhodno narejenega Faces odgovora.

Non-Faces zahtevek(request): Zahtevek, ki je bil poslan komponenti aplikacije npr. servletu ali JSP-ju in ni bil preusmerjen na Faces view.

Življenski cikel obdelave zahtevka je pri JSF ogrodju sestavljen iz 6-ih standardnih faz. Faza so Restore View, Apply Request Values, Process Validations, Update Model Values, Invoke Application in Render Response. Vsaka izmed faz opravi del procesiranja, ki je potreben za generiranje odgovora in njegovega pošiljanja klinetu kot odgovor na njegov zahtevek.

2.4.3 Komponenti model uporabniškega vmesnika (UI)

Komponentni uporabniški vmesnik ogrodja JSF predstavlja osnovne gradnike za izgradnjo uporaniškega vmesnika. Posamezna komponenta predstavlja konfigurabilen in ponovno uporabljiv element uporabniškega vmesnika, katerega kompleksnost se razteza od enostavnih elementov kot so npr. gumbi in tekstovna polja pa do sestavljenih elementov kot so npr. kontrola drevesa ali tabele. Komponente so lahko opcijsko povezane tudi z ustreznimi objekti v podatkovnem modelu aplikacije s pomočjo t.i »binding« izraza.

Poleg osnovnih gradnikov uporabniških vmesnikov vsebuje ogrodje JSF tudi pomožne gradnike :

- Converters(konverterji) so »pluggable« komponente, ki omogočajo pretvorbo vrednosti označitvnega jezika komponente v ustrezen tip na objekta;
- Events in Listeners(dogodki in poslušalci) so komponente, ki omogočajo razpošiljanje dogodkov in naročanje oz. registracijo na sprejemanje dogodkov(events);

- Validators(validatorji) so »pluggable« pomožni razredi, ki preverijo lokalne vrednosti komponente(prejete s klientovega zahtevka) ali v skladu s poslovnimi pravili zadovoljuejo pravila zapisana v validatorju. V primeru napaka lahko validatorji generirajo sporočilo o napaki in ga posredujejo uporabnku.

Uporabniški vmesnik za določeno stran spletne aplikacije zgrajene na osnovi JSF ogordja je narejena z združevanjem komponent uporabniškega vmesnika za določen zahtevek ali odgovor v t.i. pregled »view«. »View« je drevo razredov, ki implementirajo vmesnik UIComponent. Komponente v drevesu imajo relacijo starš-otrok z ostalimi komponentami. Drevo se začne s korenom, ki mora biti instanca vmesnika UIViewRoot. Komponente v drevesu so lahko anonimne ali pa jim je lahko dodeljen identifikator komponente. V bolj kompleksnih scenarijih je mogoče komponente dodati drugim komponentam kot t.i. »facets« oz. plati.

2.4.3.1 Vmesnika UIComponent in UIComponentBase

javax.faces.component.UIComponent je osnovni abstraktni razred za vse komponete uporabniških vmesnikov. Razred javax.faces.component.UIComponentBase predstavlja privzeto imlemetacijo skoraj vseh metod razreda UIComponent in s tem omogoča razvijalcem novih komponent, da se osredotočijo na edinstvene lastnosti komponente, ki jo razvijajo.

2.4.3.2 Komponentni vmesniki obnašanja

Komponente lahko poleg razširjanja razreda UIComponent implementiraja tudi enega ali več t.i. vedenjskih »behavioral« vmesnikov. Vmesnik ActionSource omogoča komponenti, da postane izvor dogodkov tipa ActionEvent. Komponenta z implemetacijo vmesnika NamingContainer doseže, da imajo vse njene komponente otroci, ki imajo identifikatorje različne od null, edinstvene vrednosti. Vmesnik StateHolder napoveduje, da mora razred, ki ga implementira, hraniti stanje preko več zahtevkov. ValueHolder je vmesnik, ki omogoča komponenti podporo za lokalne vrednosti. EditableValueHolder vmesnik razširja vmesnik ValueHolder in dodaja dodatne lastnosti, ki jih podpirajo komponente tako, da jih je mogoče urejati in s tem sprožati dogodeke kot je npr. tipa ValueChangeEvents, ter klicanje komponent tipa Validators ob spreminjanju vrednosti komponente.

2.4.3.3 Model konverzije

Model predstavlja način s pomočjo katerega JSF ogrodje pretvarja vrednosti zapisane v označevalnem jeziku v tipe, ki so definirane v javanskih razredih. Tipčno se v spletni aplikaciji

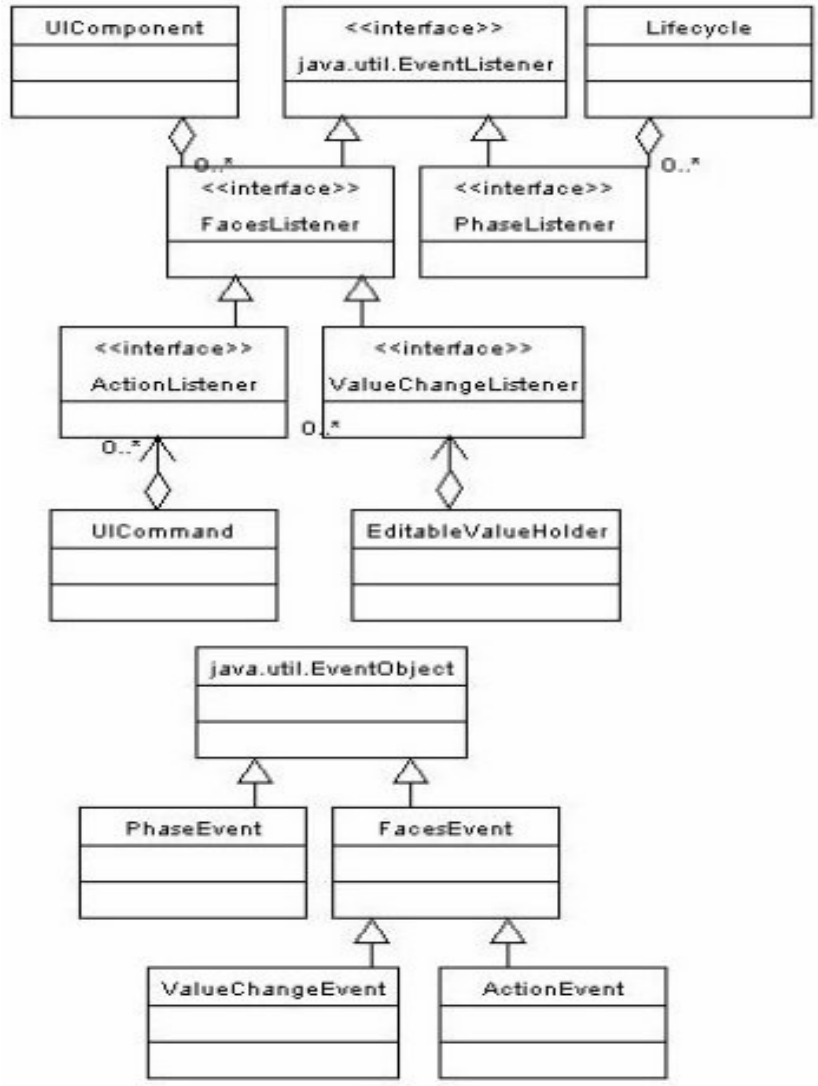
na strani uporabnika prikazujejo strani z uporabo t.i. označitvenih oz. markup jezikov npr. HTML. Pri taki predstavitvi so podatki na strani uporabnika predstavljeni kot nizi. Na strežniški strani pa so podatki običajno predstavljeni v obliki javanskih razredov, znotraj katerih so definirani kot lastnosti določenih javanskih tipov npr. java.util.Date. Ogrodje JSF ima, zato za potrebe pretvarjanja podatkov vmesnik javax.faces.convert.Converter. Vmesnik Converter definira naslednji metodi:

- `public Object getAsObject(FacesContext context, UIComponent component, String value) throws ConverterException;`
- `public String getAsString(FacesContext context, UIComponent component, Object value) throws ConverterException;`

Prva metoda se uporablja pri pretvorbi vrednosti tipa String v objekt, ki je definiran na razredu spletne aplikacije. Druga metoda pa se uporablja za pretvorbo v obratni smeri. Ogrodje JSF že vsebuje definirane naslednje pretvornike (vsi so del paketa javax.faces.convert): BigDecimalConverter, BigIntegerConverter, BooleanConverter, ByteConverter, CharacterConverter, DateTimeConverter, DoubleConverter, FloatConverter, IntegerConverter, LongConverter, NumberConverter, ShortConverter.

2.4.3.4 Model dogodkov in poslušalcev

Ogrodje JSF implementira model obvestil in registracijo poslušalcev dogodkov na osnovi načrtovalskih vzorcev, ki so definirani v JavaBean specifikaciji verzije 1.0.1. Podrazred razreda UIComponent se lahko odloči, da bo oddajal dogodke ob spremembah stanja vsem registriranim poslušalcem. Ob zaključkih večih faz v življenksem ciklu obdelave zahtevka bo JSF ogrodje razposlalo vse dogodke, ki so bili postavljeni v čakalno vrsto, zainteresiranim poslušalcem.



Slika 20. Prikazuje UML diagram glavnih razredov modela dogodkov in poslušalcev ogrodja JSF

2.4.3.5 Model validacije

JSF omogoča registracijo enega ali več validatorjev na vsako komponento tipa EditableValueHolder v drevesu component. Namen validatorja je, da preveri vrednosti komponente med fazo "Process Validations" življenskega cikla obdelave zahtevka. Dodatno lahko komponenta implementira interno preverjanje preko metodo validate, ki je del razreda komponenete.

Validator mora implementirati vmesnik javax.faces.validator.Validator, ki vsebuje metodo validate.

Vmesnik EditableValueHolder vsebuje metodi addValidator in removeValidator preko katerih je mogoče dodajati oz. odstranjevati validatorje za komponento. Poleg tega vmesnik omogoča tudi

dodajanje t.i. povezav metod MethodBinding, ki predstavljajo povezavo na metodo, ki upošteva podpis metode validate definirane v vmesniku Validator.

JSF ogrodje ima predefinirane naslednje validatorje : DoubleRangeValidator, LengthValidator, LongRangeValidator. DoubleRangeValidator preveri ali je število veljavno število tipa Double. Podobno preveri validator LongRangeValidator ali je število veljavno število tipa Long. LengthValidator pa preveri ali je komponenta pravilne dolžine npr. niz ali število.

2.4.4 Standardne komponente uporabniškega vmesnika

Kot napoveduje že naslov bom v tem razdelku opisal standardne komponentno za gradnjo uporabniških vmesnikov, ki jih vsebuje ogrodje JSF. Komponente predstavljajo osnovne gradnike za gradnjo uporabniških vmesnikov.

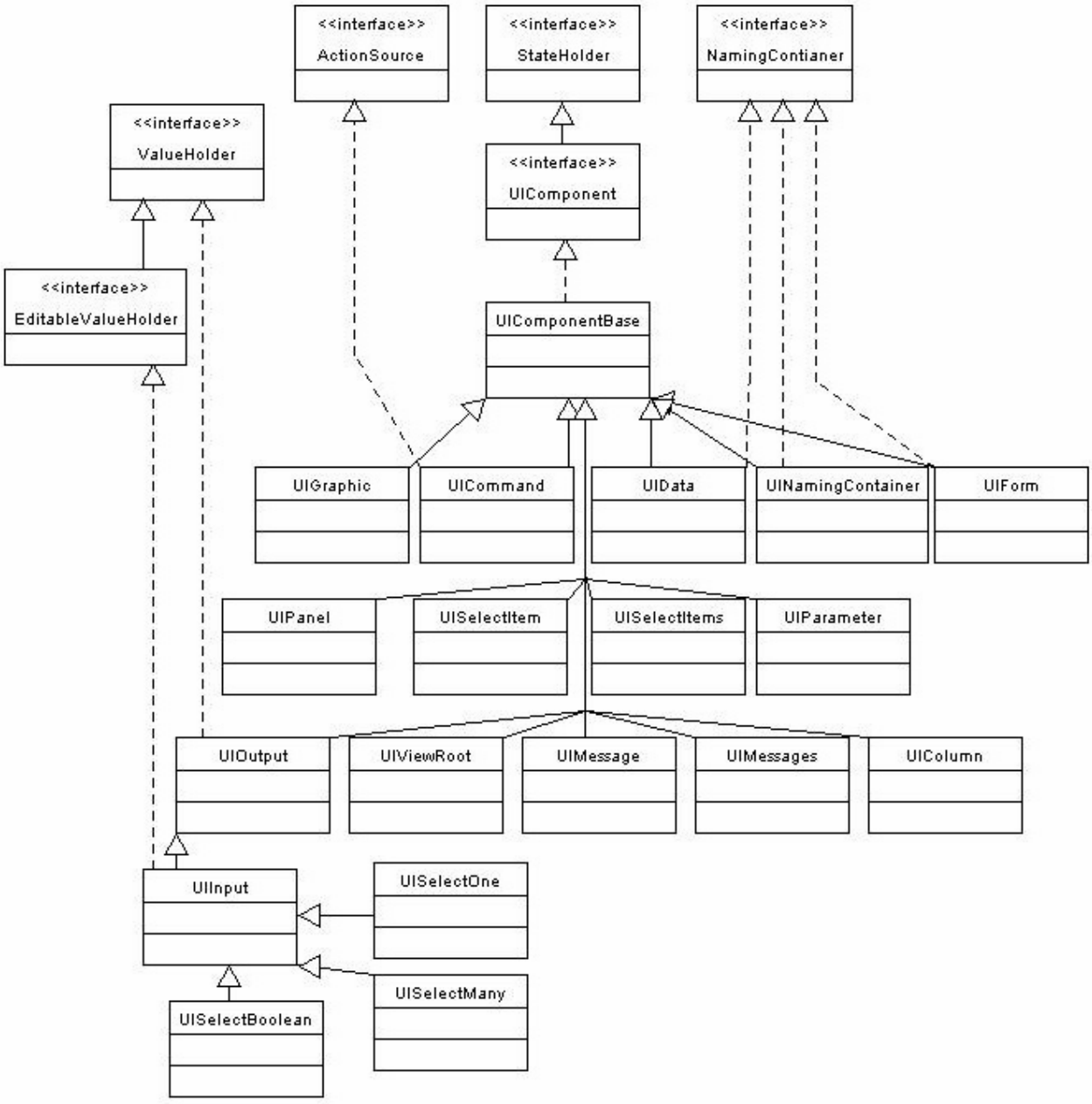
Vsaka standardna komponenta, ki implementira razred UIComponent mora definirati dve konstanti tipa static final String:

- COMPONENT_TYPE je konstanta predstavlja standarden tip identifikatorja komponente preko katerega je razred komponente registriran v objektu Application. Ta vrednost se lahko uporabi kot parameter pri klicu metode createComponent();
- COMPONENT_FAMILY je konstanta preko katere se izbere pravilni izrisovalnik (renderer) za to komponento;

Standardne komponente uporabniškega vmesnika ogrodja JSF so :

- UIColumn predstavlja en stolpec podatkov znotraj očetove komponenete UIData;
- UICommand predstavlja kontrolo, ki jo aktivira uporabnik in s tem sproži specifičen ukaz ali akcijo v aplikaciji. Komponenta tega tipa je običajno prikazana kot gumb, element menija ali hiperpovezava;
- UIData je komponenta, ki predstavlja podatkovno povezavo z zbirko podatkovnih objektov predstavljenih z instanco razreda DataModel;
- UIForm predstavlja vnosno formo(html forma), ki vsebuje komponente kot so vnosna polja itd.;
- UIGraphic se uporablja za prikaz slik uporabniku. Uporablja se samo za prikaz in je uporabike ne more manipulirati;
- UIInput je komponenta, ki hkrati prikaže trenutno vrednost uporabniku in obdela parametre zahteveka(request) pri zaporedni oddaji zahtevka s strani uporabnika, ki ga je potrebno dekodirati. Komponenta predstavlja vnosno polje oz. v HTML jeziku je to t.i. “input” polje;

- UIMessage predstavlja prikaz sporočila o napaki za določeno vnosno komponento;
- UIMessages predstavlja prikaz sporočil o napakah, ki niso povezane s posameznim vnosnim poljem oz. predstavlja vsa sporočila v čakalni vrsti (enqueued messages);
- UIOutput je komponenta, ki vsebuje vrednosti namenjeno samo za prikaz uporabniku;
- UIPanel je komponenta, ki ureja postavitev njenih elementov komponent otrokov;
- UIParameter predstavlja opcijski konfiguracijski parameter, ki vpliva na prikaz komponente očeta;
- UISelectBoolean predstavlja eno Boolean (true ali false) vrednost. Običajno je komponeta izrisana kot potrditevno polje (checkbox);
- UISelectItem je komponenta, ki je lahko gnezdena v komponentah UISelectMany in UISelectOne in predstavlja natančno en element instance SelectItem v seznamu možnih opcij v očetovi komponenti;
- UISelectItems je komponenta, ki je lahko gnezdena v komponentah UISelectMany in UISelectOne in predstavlja nič ali več elementov instance SelectItem za dodajanje v seznam možnih opcij v očetovi komponenti;
- UISelectMany predstavlja selekcijo enega ali več elemenotv iz seznama opcij, ki so na voljo. Običajno je prikazana kot kombinirano polje(combobox) oz. seznam potrditvenih polj (checkboxe);
- UISelectOne predstavlja selekcijo enega ali več elemenotv iz seznama opcij, ki so na voljo. Običajno je prikazana kot kombinirano polje(combobox) oz. seznam radijskih gumbov (radio button).
- UIViewRoot komponenta predsatvlja koren komponentnega drevesa.
- DataModel je abstraktni razred za generiranje ovojnica (wrapper) okrog poljubne podatkovo povezovalne (data binding) tehnologije. Lahko se uporabi za dostop do različnih izvorov podatkov znotraj component JSF, ki želijo podpirati množico podatkov (data set), ki jo je mogoče modelirati kot več vrstično(multiple rows).
- SelectItem je pomožen razred, ki predstavlja eno izbiro med vsemi izbirami dostopnimi uporabniku pri komponentah UISelectMany aliUISelectOne.
- SelectItemGroup je pomožni razred, ki razširaj razred SelectItem in predstavlja podrejene instance razreda SelectItem, ki jih je mogoče prikazati kot podmenije ali opcijske grupe.



Slika 21. Razredna hierarhija standardnih komponent uporabniškega vmesnika ogrodja JSF.

2.4.5 Kontekstne informacije pri obdelavi zahtevka

Abstrakten razred javax.faces.context.FacesContext predstavlja vse kontekstne inforamcije, ki so povezane z obdelavo zahtevka in generiranjem odgovora. Instanca razreda FacesContext je generirana s strani JSF implementacije pred začetkom obdelave zahtevka. Ko je življenski cikel obdelave zahtevak zaključen, JSF implementacija pokliče metodo release, ki omogoča sprostitev zavzvetih resursov in recikliranje instance razreda FacesContext.

Preko razreda FacesContext je mogoče dostopati do aplikacijske instance oz. spletne aplikacije, znotraj katerega je vključena JSF komponenta, zunanjega konteksta, ki je običajno servlet vmesnik API, ViewRoot oz. korena komponentnega drevesa, itd.

2.4.6 Model izrisovanja (Rendering Model)

JSF ogrodje omogoča dva programska načina za dekodiranje vrednosti komponent iz prihajajočih zahtevkov in enkodiranje vrednosti komponent v odhajajoče odgovore. Prvi način je t.i. direktni način (direct implementation) pri katerem morajo komponente samo opraviti dekodiranje in enkodiranje vrednosti. Drug način pa je t.i. delegiran način (delegated implementation) pri katerem pa se omenjeni operaciji delegirata instanci Renderer, ki je preko lastnosti rendererType povezana s komponento. Tak način izrisovanja omogoča aplikaciji, da upravlja s komponentami neodvisno od tega, kako so prikazane uporabniku in preko enostavne operacije (izbira določenega RenderKit) omogoči enostavno prilagoditev enkodiranja in dekodiranja posameznim napravam ali lokalizirani predstavitvi podatkov uporabniku.

Vsaka JSF implementacija mora vsebovati privzeto implementacijo RenderKit-a, ki se uporabi, če ni izbran noben drug RenderKit in je dostopen preko identifikatorja tipa String v razredu RenderKitFactory.HTML_BASIC_RENDER_KIT. Aplikacije, ki želijo večje zmožnosti, kot jih ponuja privzeta implementacija RenderKit-a, lahko ustvarijo svoje instance razreda Renderer v obstoječo instanco RenderKit-a. Običajno je, da aplikacija, ki potrebuje posebne komponentne razrede in Renderer-je, da jih registrira v standardnem RendererKit-u ob zagonu aplikacije.

Osnovni razredi za izrisovanje oz. renderiranje :

- RenderKit instanca podpira komponente, ki uporabljajo delegiran programski model za enkodiranje in dekodiranje komponentnih vrednosti;
- Renderer instanca implementira funkcionalnost dekodiranja in enkodiranja med fazama Apply Request Values in Render Response v življenskem ciklu obdelave zahtevka, v primeru, da komponenta nima lastnosti rendererType z vrednostjo null;
- ResponseStateManager je pomožni razred razredu javax.faces.application.StateManager, ki pozna specifično izrisovalno (rendering) tehnologijo, ki se uporablja za generiranje

odgovora. Razred pozna mehanizme za shranjevanje stanja ne gleda nato ali gre za skrita polja (hidden fields), sejo (session) ali kombinacjo obeh;

- RenderKitFactory je singelton razred, ki mora biti dostopen vsaki JSF spletni aplikaciji, ki teče v servlet ali portlet zabojniku. Kot je razvidno že iz imena razred omogoča upravljanje z razredi RenderKit.

Zaradi zagotovitve aplikacijske prenosljivosti mora JSF implementacija vsebovati RenderKit in z njim povezane Renderer-je za generiranje teksta kompatibilnega s specifikacijo HTML 4.01

TABLE 8-1 Concrete HTML Component Classes

javax.faces.component class	renderer-type	javax.faces.component.html class
UICommand	javax.faces.Button	HtmlCommandButton
UICommand	javax.faces.Link	HtmlCommandLink
UIData	javax.faces.Table	HtmlDataTable
UIForm	javax.faces.Form	HtmlForm
UIGraphic	javax.faces.Image	HtmlGraphicImage
UIInput	javax.faces.Hidden	HtmlInputHidden
UIInput	javax.faces.Secret	HtmlInputSecret
UIInput	javax.faces.Text	HtmlInputText
UIInjput	javax.faces.Textarea	HtmlInputTextarea
UIMessage	javax.faces.Message	HtmlMessage
UIMessages	javax.faces.Messages	HtmlMessages
UIOutput	javax.faces.Format	HtmlOutputFormat
UIOutput	javax.faces.Label	HtmlOutputLabel
UIOutput	javax.faces.Link	HtmlOutputLink
UIOutput	javax.faces.Text	HtmlOutputText
UIPanel	javax.faces.Grid	HtmlPanelGrid
UIPanel	javax.faces.Group	HtmlPanelGroup
UISelectBoolean	javax.faces.Checkbox	HtmlSelectBooleanCheckbox
UISelectMany	javax.faces.Checkbox	HtmlSelectManyCheckbox
UISelectMany	javax.faces.Listbox	HtmlSelectManyListbox
UISelectMany	javax.faces.Menu	HtmlSelectManyMenu
UISelectOne	javax.faces.Listbox	HtmlSelectOneListbox
UISelectOne	javax.faces.Menu	HtmlSelectOneMenu
UISelectOne	javax.faces.Radio	HtmlSelectOneRadio

Slika 22. Konkretne razrede implementiranih HTML komponent v ogrodju JSF

2.4.7 Integracija z JSP (java server pages)

Implementacija JSF-ja mora podpirati uporabo JavaServer Pages(JSP) kot opisni jezik za JSF strani. JSP podpora je narejena s pomočjo posebnih akcij tako, da je uporabniški vmesnik mogoče definirati znotraj JSP strani z uporabo posebnih akcij, ki predstavljajo JSF UI komponente. Posebne akcije JSF implementacije se lahko mešajo s standardnimi JSP akcijami in posebnimi akcijami z drugih knjižnic kot tudi tekstovnimi predlogami znotraj iste JSP strani.

Specifikacija JSF rezervira naslednja URI-ja za referenciranje standardne tag knjižnice pri uporabi posebnih akcij definiranih v JSF ogrodju:

- http://java.sun.com/jsf/core (URI JSF core (osnovne) tag knjižnice);
- http://java.sun.com/jsf/html (URI JSF standardne HTML RenderKit tag knjižnice);

JSF core knjižnica vsebuje osnovne razrede za preverjanje, pretvarjanje in spremljanje uporabnikovih vnos kot so razredi Converter, ValuChangeListener, ActionListener in Validator.

2.4.7.1 JSF osnovna (core) tag knjižnica

Vse JSF implementacije morajo implementirati tag knjižnico, ki vsebuje osnovne (core) akcije (opisane spodaj), ki so neodvisne od določenega RenderKit-a.

Osnove akcije JSF implementacije so:

- <f:actionListener type=”fully-qualified-classname”/> - registrira instanco ActionListener oz. poslušalca na razredu tipa UIComponent;
- <f:attribute /> - doda atribut na razredu tipa UIComponent;
- <f:convertDateTime> - registrira instanco DateTimeConverter na razredu tipa UIComponent. (uporablja se za pretvorbo datumov v različne formate);
- <f:convertNumber> - registrira instanco NumberConverter na razredu tipa UIComponent (uporablja se za pretvorbo števil v različne formate);
- <f:converter converterId=”converterId”/> - registrira instanco Converter na razredu tipa UIComponent.(omogoča registracijo poljubnega pretvornika);
- <f:facet name=”facet-name”/> - določene komponente imajo predefinirane dele, ki jih je mogoče upravljati oz. določati njihovo vsebino od zunaj npr. glava HTML tabele;
- <f:loadBundle> - naloži lokaliziran »resource bundle« za lokaliziran pogleda strani in omogoči dostop do njega preko atributov trenutnega zahtevka (requesta);
- <f:param> - doda komponento tipa UIParameter na razred tipa UIComponent;

- <f:selectItem> - doda komponento tipa UISelectItem na razred tipa UIComponent(en element izbirnega seznama);
- <f:selectItems> doda komponento tipa UISelectItems na razred tipa UIComponent(izbirni seznam);
- <f:subview> - pomožna akcija za vse JSF core in druge posebne akcije, ki se uporabljajo na gnezdenih straneh, ki so vključne preko <jsp:include> ali kakšne druge akcije, ki dinamično vključuje druge strani iste spletne aplikacije kot je npr. JSTL <c:import>;
- <f:validateDoubleRange> - registrira instanco DoubleRangeValidator na razred tipa UIComponent. (validator decimalnih števil);
- <f:validateLength> - registrira instanco LengthValidator na razred tipa UIComponent(validator dolžine vnosa uporabnika);
- <f:validateLongRange> - registrira instanco LongRangeValidator na razred tipa UIComponent(validator celih števil);
- <f:validator> registrira instanco Validator na razred tipa UIComponent (omogoča registracijo poljubnega validatorja);
- <f:valueChangeListener> - registrira instanco ValueChangeListener na razred tipa UIComponent;
- <f:verbatim> - registrira instanco UIOutput na razred tipa UIComponent (uporablja se za prikaz statičnega teksta znotraj JSF strani);
- <f:view> - osnovni JSF tag znotraj katerega se nahajajo vsi ostali JSF tag-i oz. komponente;

2.4.7.2 Standardna HTML RenderKit tag knjižnica

Poleg JSF osnovne (core) tag knjižnice mora implementacija JSF ogrodja vsebovati tudi implementacijo standardnih component uporabniškega vmesnika in razreda Renderer iz standardnega HTML RenderKit-a oz. izrisovalnika. Povedano drugače vsaka implementacija JSF ogrodja mora vsebovati tudi komponente in generatorje standardnih elementov uporabniškega vmesnika za generiranje HTML strani. Na sliki 23 so prikazani osnovni razredi in njihova imena s pomočjo katerih je mogoče znotraj JSP strani definirati uporabnški vmesnik, ki se izriše in prikaže uporabniku v ozančevalnem jeziku HTML.

TABLE 9-2 Standard HTML RenderKit Tag Library

getComponentType()	getRendererType()	custom action name
javax.faces.Column	(null)*	column
javax.faces.HtmlCommandButton	javax.faces.Button	commandButton
javax.faces.HtmlCommandLink	javax.faces.Link	commandLink
javax.faces.HtmlDataTable	javax.faces.Table	dataTable
javax.faces.HtmlForm	javax.faces.Form	form
javax.faces.HtmlGraphicImage	javax.faces.Image	graphicImage
javax.faces.HtmlInputHidden	javax.faces.Hidden	inputHidden
javax.faces.HtmlInputSecret	javax.faces.Secret	inputSecret
javax.faces.HtmlInputText	javax.faces.Text	inputText
javax.faces.HtmlSelectBooleanCheckbox	javax.faces.Checkbox	selectBooleanCheckbox
javax.faces.HtmlSelectManyCheckbox	javax.faces.Checkbox	selectManyCheckbox
javax.faces.HtmlSelectManyListbox	javax.faces.Listbox	selectManyListbox
javax.faces.HtmlSelectManyMenu	javax.faces.Menu	selectManyMenu
javax.faces.HtmlSelectOneListbox	javax.faces.Listbox	selectOneListbox
javax.faces.HtmlSelectOneMenu	javax.faces.Menu	selectOneMenu
javax.faces.HtmlSelectOneRadio	javax.faces.Radio	selectOneRadio

Slika 16. Standarne HTML komponente JSF ogrodja

TABLE 9-2 Standard HTML RenderKit Tag Library

getComponentType()	getRendererType()	custom action name
javax.faces.HtmlInputTextarea	javax.faces.Textarea	inputTextarea
javax.faces.HtmlMessage	javax.faces.Message	message
javax.faces.HtmlMessages	javax.faces.Messages	messages
javax.faces.HtmlOutputFormat	javax.faces.Format	outputFormat
javax.faces.HtmlOutputLabel	javax.faces.Label	outputLabel
javax.faces.HtmlOutputLink	javax.faces.Link	outputLink
javax.faces.HtmlOutputText	javax.faces.Text	outputText
javax.faces.HtmlPanelGrid	javax.faces.Grid	panelGrid
javax.faces.HtmlPanelGroup	javax.faces.Group	panelGroup

Slika 23. Standarne HTML komponente JSF ogrodja (nadaljevanje iz predhodne strani)

3 Liferay kot primer standardiziranega javanskega WCMS sistema

Glede na [10] in [6] je WCMS sistem Liferay vodilni javanski odprto-kodni WCMS. Trditev temelji na številnih nagradah, ki jih je prejel, kot tudi na številu prenešenih namestitev preko interneta(več kot 40000 prenosov na mesec in več kot miljon prenosov skupno).WCMS sistem Lifery je popularen, ker vsebuje vse funkcionalnosti, ki so potreben za uspešno spletno stran objavljeno na internetu, intranetu ali kje vmes. WCMS sistem predstavlja razširitev običajnih portalov s funkcionalnostmi za upravljanje vsebin (CMS fukcionalnosti). Drugače povedano sistem Liferay je javanski portal, ki vsebuje dodatne funkcionalnosti, ki omogočajo upravljanje z vsebino.

Arhitektura WCMS sistema Liferay je razširljiva in omogoča nadgradnjo z novimi javanskimi implementacijami specifikacij, ki jih morebiti potrebuje uporabnik. Slednje je prikazano tudi na primeru implementacije brskalnika po repozitoriju vsebin pri katerem je bilo potrebno v WCMS sistem Liferay vključiti knjižnico javanske specifikcije JSF.

Namestitev sistema Liferay vsebuje več kot 60 portletov(več kot katerikoli drugi portal na tržišču). Namestitev vključuje CMS portlete, portleti za spletno založništvo(web publishing), sodelovanje(collaboration) in socialna omrežja.

Sistem je razvit z uporabo odprte strategije SOA(service oriented architecture), zato omogoča integracijo obstoječih podatkov iz sistemov HR(human resources), računovodstva, prodaje in drugih virov.

Prijava v sistem omogoča t.i. SSO (single sign on), kar posebaj pride do izraze pri agregaciji in dostopu do vsebin iz različnih virov. V tem primeru lahko Liferay predstavlja integracijsko točko, ki prikazuje podatke iz različnih virov in hkrati skrbi za avtentikacijo uporabnikov preko sistema SSO.

Administratorji lahko dodeljujejo uporabnikom oz. skupini uporabnikov različne vloge oz. role in s tem nadozorujejo različne nivoje pravic za dostop in urejanje določenih skupnosti, datotek, aplikacij in orodij. Npr. direktor prodaje lahko pregleduje in ureje dokumente iz prodaje medtem, ko jih lahko pomočnik prodaje samo pregleduje. Drugače povedana sistem vsebuje granuliran model avtorizacije, ki temelji na vlogah oz. rolah.

Uporabniki se lahko grupirajo v hierarhijo »organizacij« ali med organizacijske skupnosti, ki omogočajo fleksibilno in enostavno administriranje. Npr. člani različnih geografskih področij(Amerika, Evropa) se lahko grupirajo v organizacije. Področja, ki pa niso vezana na geografska področja kot so prodaja, marketing in inžiniring, se lahko združijo v skupnosti. Vsaki skupini se lahko dodeli majhen portal s svojim naborom strani, sistemom za upravljanje vsebin, koledarjem in avtorizacijo. Uporabnik lahko pripada večim grupam in lahko enostavno navigira med njimi.

Liferay je bil eden izmed prvih javanskih spletnih portalov, ki je omogočil uporabnikom premikanje portletov na straneh portala z uporabo t.i. »drag&drop« akcije v fazi urejanja vsebine in urejanja strani portala. Omenjena funkcionalnost je dokaj običajen element uporabniških vmesnikov na operacijskih sistemih njena uporaba v spletnih aplikacijah pa je še vedno prej posebnost kot pravilo. Z uporabo »drag&drop« akcij je mogoče v portalu ustrezno postaviti portlete znotraj predefiniranih pogledov(layout). Poglede je mogoče nastaviti za vsako stran portala posebaj.

Liferay je mogoče povezati tudi z namizjem osebnega računalnika preko »Webdav« povezave. Na ta način je mogoče s prenosom datotek v splošno znano mapo »WebDAV« na namizju prenesti datoteke na portal.

Možno je tudi označevanje vsebin z meta podatki, dokumentov, niti sporočilnih desk (message board threads). Uporabniki lahko tako iščejo informacije znotraj posameznih portletov, skupnosti, znotraj celotnega portala in celo v zunanjih aplikacijah, ki so integrirane v portal.

Vsakemu uporabniku je dodeljen tudi osebni prostor, ki je lahko tudi prosto dostopen (uporabnik ga lahko objavi in je dostopen preko enolično določenega URL-ja). Prostor je mogoče prilagoditi in mu spremeniti izgled, orodja, aplikacije, ki so vključene v uporabnikov prostor ter kateri dokumenti se shranjujejo v knjižnico dokumentov in kdo lahko dostopa do njih.

Portal omogoča enostavno menjavo jezika s samo enimi klikom in ponuja enostavno dodajanje novih jezikov. Trenutno je podprih 22 jezikov med katerimi pa zaenkrat še ni slovenščine.

3.1 Tehnični podatki portala Liferay

Glede na podatke dostopne na spletni strani [6] je portal Liferay mogoče namestiti na operacijske sisteme tipa LINUX, UNIX in WINDOWS. Portal podpira podatkovne baze: Apache Derby, IBM DB2, Firebird, Hypersonic, Informix, InterBase, JdataStore, MySQL, Oracle, PostgresSQL, SAP, SQL Server, Sybase. Mogoče ga je namestiti na aplikacijske strežnike: Apache Geronimo, Sun GlassFish 2 UR1, JBoss, JOnAS, OracleAS, SUN JSAS, WebLogic, WebSphere. Portal omogoča napredno predpomnenje in predpomnjenje posameznih strani, ter distribuirano predpomnenje. Dodatno performančno optimizacijo predstavlja tudi izvoz statičnih strani. Hkrati omogoča replikacijio sej in izenačevanje obremenitve (load balancing). Pri izdelavi portala so bile uporabljene naslednje tehnologije: AJAX, Apache ServiceMix, Docbook, ehcache, Groovy, Hibernate, ICEfaces, Java J2EE/JEE, jBPM, JGroups, jQuery JavaScript Framework, Lucene, MuleSource ESB, PHP, Ruby, Seam, Spring & AOP, Struts & Tiles, Tapestry, Velocity. Portal uporablja implementacije knjižnic, ki implementriajo naslednje standarde: AJAX, iCalendar & Microformat, JSR-168, JSR-127, JSR-170, JSR-286 (portlet 2.0), JSF-314 (JSF 2.0), OpenSearch.

3.2 Portal Liferay kot WCMS

Liferay CMS predstavlja enoten prostor za agregiranje in urejanje vse vsebine. Vsaka skupnost znotraj portala dobi lastno knjižnico dokumentov in galerijo slik. Prav tako omogoča integracijo z aplikacijo Microsoft Office, kar pomeni, da se ob spremembi dokumentov na lokalnem disku lahko slednji avtomatsko prenesejo v repozitorij, ki je v portalu. Liferay CMS z uporabo prosto dostopnih SharePoint protokolov omogoča uporabnikom, da odprejo, shranijo, verzionirajo, zaklenejo in delijo dokumente med Liferay-om in Microsoft Office-om.

Možen je tudi prenos večih datotek hkrati oz. posameznih datotek v knjižnico dokumentov, galerijo slik in v ostala orodja, ki omogočajo pripenjanje datotek. V knjižnici dokumentov je prav tako mogoče pretvoriti format datoteke oz. ga izvoziti v drug format npr. iz word-a v pdf.

Poleg tega, da portal zbira vsebino iz različnih virov omogoča tudi objavo, kateregakoli dela vsebine znotraj portala preko pravil objave (publishing rules) ali z ročno izbiro. Če bi želeli objaviti vsebino z oznako (tag) nakup, bi to lahko dosegli z uporabo portleta Asset Publisher.V tem primeru bi se vsaka objava s to oznako, ne glede na to znotraj katerega portleta je objavljena, objavila tudi v portletu Asset Publisher.

Portal omogoča tudi enostavno postavitev portletov za navigacijo znotraj portala oz. t.i. »site-map« portletov.

Uporabniki lahko urejajo vse od enostavnih tekstovnih člankov in slik pa vse do celotnih spletnih strani. Spletno objavljanje deluje ob sami množici aplikaciji, ki jo vsebuje portal, kar pomeni, da je mogoče dodati vsebino praktično znotraj vsake aplikacije.

Portal avtomatsko popravlja t.i. »sitemap« podatek in s tem omogoča spletnim iskalnikom, da poiščejo na novo dodane strani skoraj istočasno z njihovim generiranje oz. dodajanjem.

Vključen je tudi t.i. bogati tekstovni urejevalnik, ki omogoča uporabnikom urejanje spletnih vsebin oz. strani brez predhodnega poznavanja jezika HTML. Urejanje spletnih strani tako postane enostavno, kot je npr. urejanje dokumentov v aplikaciji Microsoft Word.

Včasih je potrebno spremeniti objavljeno vsebino v živo. Portal Liferay omogoča popravljanje in predogled popravljene vsebine v realnem času, brez prikaza sprememb javnosti, vse dokler nismo popolnoma prepričani o objavi sprememb. Mogoče je urejati posamezne strani, skupnosti in celotne portale. Portal omogoča tudi planiranje objav v prihodnosti.

3.3 Portal Liferay kot orodje za sodelovanje (collaboration)

Poratl Liferay ponuja številke portlete, ki omogočajo prikaz vsebin in resursov, ki jih ponuja portal uporabnikom. Ti portleti so označeni s skupnim pojmom sodelovanje (collaboration).

Vsaka skupnost ima svoj »Wiki« s svojo množico avtorizacij. Vsakdo s pravicami za urejanje lahko hitro vpiše svoje informacije v te tematske spletne enciklopedije. Portlet »Wiki« omogča dokumentiranje pomembnih oz. zanimivih informacij.

Sporočilne deske (message boards) so odlična rešitev za pogovore o idejah znotraj oddelak ali projektne skupine. Hkrati so tudi rešitev za zajemanje in širjenje znanja znotraj delavne skupine. Liferay omogoča poglede statistike aktivnosti, zadnjih objav sporočilnih desk. Uporabniki se lahko naročijo na posamezne niti preko RSS ali elektronske pošte. Kot ostali portleti so tudi sporočilne deske zaščitene z granularnim sistemom avtorizacije, ki ga ponuja portal Liferay.

Portal prav tako ponuja »Blog«, ki omogoča uporabnikom pisanje »Bloga« z uporabo bogatega tekstovnega urejevalnika (rich text editor), obveščanja o novih objavah preko elektronske pošte in ocenjevalnega(rating) sistema objav. Na vse »Bloge« se je mogoče naročiti preko sistem RSS. Uporabniki lahko svoje objave označijo tako, da se objavijo ob določenem času.

Spremljanje zadnjih aktivnosti na portalu je mogoče preko portleta zadnjih aktivnosti (recent activity) in zidu aktivnosti (Activety Wall), ki je po svojem izgledu podben portalu Facbook. Mogoče je spremljati zadnje objave na »Blogih«, sporočilnih deskah, »Wikijih« in ostalih orodjih.

Pogovor s sodelavci oz. prijatelji je mogoče preko t.i. »Instant Message«(IM) sistema Sistem prikazuje imena vseh uporabnikov, ki so prijavljeni v sistem in omogoča dodajanje slike profilom uporabnikov, ter lastnih sporočil o statusu uporabnika.

Pošilanje elektronskih sporočil je mogče preko klienta za spletno pošto, ki temelji na tehnologiji AJAX in omogoča integracijo s popularnimi IMAP strežniki elektronske pošte.

Koledar s seznamom opravil omogoča uporabnikom dodajanje, upravljanje in iskanje dogodkov. Dogodke je mogoče deliti med skupnostmi. Možno je nastavljanje alarmov, ki obveščajo uporabnike na prihajajoče dogodke preko elektronske pošte, IM ali SMS-a. Microformats in druga podpora koledarju omogoča enostaven prenos koledarja in podatkov o uporabniku preko Web 2.0 standarda.

Preko obvestil in alarmov je mogoče obveščati različne skupine oz. razpošiljati e-novice. Vsak uporabnik lahko kontrolira na kakšen način je obveščen o alarmih. Obveščanje je lahko preko spletnega alarma portala, SMS-a, elektronskega sporočila ali katerega drugega mehanizma za dostavo sporočil.

Portal omogoča generiranje več-izbirnih anket preko orodja, ki hkrati tudi hrani sprejete glasove. Možno je upravljati več ločenih anket. Prikaz anket je mogoče narediti z ustrezno konfiguracijo večih portletov.

3.4 Sistemska zahtevnost in zmogljivost portala Lifery in portala Joomla

Joomla [14] je odprtokodni portal in sistem za urejanje vsebin s pomočjo katerega je mogoče graditi spletne strani. Portal je napisan v skriptnem programskem jeziku PHP [15], za shranjevanje podatkov pa uporablja odprtokodno podatkovno bazo MySQL[16]. Portal je posebno optimiziran za aplikacijski strežnik Apache [17]. Zaradi potreb testiranja zmogljivosti, sem portal Joomla namestil na aplikacijski strežnik Apache Tomcat [18]. Ker je slednji aplikacijski strežnik implementacija javanskih tehnologij [19] in [20], kot tak ni prilagojen za prikaz spletnih strani napisanih v skriptnem programskem jeziku PHP. Z uporabo postopka opisanega na spletni strani [21] mi je uspelo konfigurirati strežnik Apache Tomcat tako, da omogoča tudi izrisovanje spletnih strani napisanih v skriptnem programskem jeziku PHP oz. portala Joomla. Na strežnik je bilo potrebno namestiti javansko knjižnico, ki je vsebovala javanski servlet net.php.servlet, ki izvaja prevajanje skriptnega jezika PHP v označitveni jezik HTML. Prav tako je bilo potrebno v sistemsko spremenljivko PATH, operacijskega sistema windows XP dodati knjižnico php5servlet.dll, preko katere omenjeni servlet dejansko izvaja prevjanje skriptnega jezika PHP v HTML.

Za merjenje porabe sistemskih resursov obeh portalov sem uporabil naslednja orodja:

- JMeter [11] je javansko orodje narejeno z namenom testiranja in merjenja obremenitev sistema. Prvotno je bilo orodje zasnovano za testiranje spletnih aplikacij, vendar se je v vmesnem času nadgradilo s funkcijami za testiranje drugih funkcionalnosti. Trenutno je z orodjem JMeter mogoče testirati naslednje strežnike oz. protokole: HTTP, HTTPS, SOAP, JDBC, LDAP, JMS, POP3. Orodje izvaja teste z generiranjem zahtevkov in hkrati meri odzivne čase strežnika;
- JProfiler [12] je javansko orodje, ki omogoča merjenje, spremljanje porabe resursov javanskega virtualnega stroja v realnem času. Z orodjem je mogoče poiskati performančna ozka grla, spominska puščanja in odpraviti težave povezane z nitmi;
- Process Explorer [13] je orodje, ki omogoča natančne prikaz porabe resursov posameznega procesa na operacijskem sistemu windows;

Preverjanje sistemskih zmogljivosti je bilo narejeno na računalniškem sistemu z naslednjo specifikacijo:

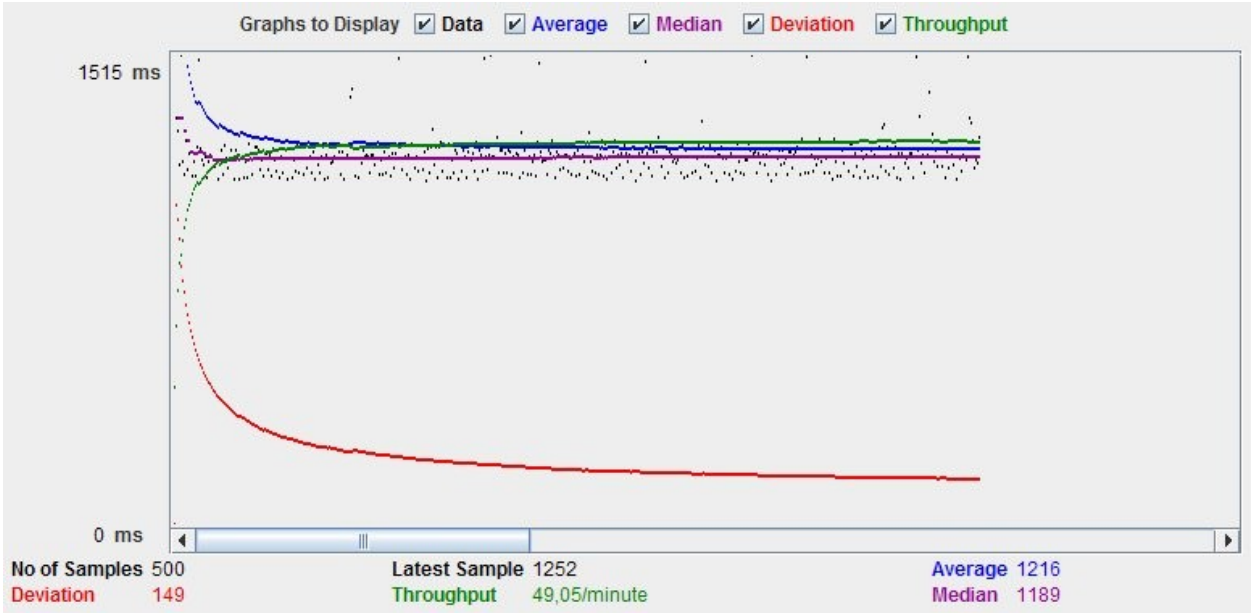
- Operacijski sistem Microsoft Windows XP Professional, Version 2002, Service Pack 3;
- Procesor Intel Pentium 4 CPU 2.40 GHz;
- Pomnilnik z naključnim dostopom 2 GB DDR;

- Aplikacijski strežnik Apache Tomcat 5.5.26;
- Verzija portala Lifery 5.1.2;
- Verzija portala Jooma 1.5.9;

Portala sta bila pred izvedbo tesiranja nameščena na ločena aplikacijska strežnika Apache Tomcat. Spremljanje porabe sistemskih resursov s strani aplikacijskega strežnika in spletnih portalov je bilo narjeno z uporabo orodij JProfiler in Process Explorer. HTTP zahtevki za strežbo strani portala so bili generirani z orodjem JMeter. JMeter je bil nastavljen tako, da je generiral 500 zahtevkov za enega uporabnika in je beležil čas potreben za izvedbo te naloge. Testirana je bila strežba vstopne strani portala Joomla, ki ni bila spremenjena. Prav tako je bila testirana vstopne strani portla Liferay, ki pa je bila spremenjeno in ji je bilo dodano 7 portletov. Dodani so bili naslendji portleti: prikaz novic (news), anketa (polls), dva prikaza vsebine žurnala (journal content), dva prikaza člankov žurnala (journal article), kazalo spletne strani (sitemap). Orodje JMeter je bilo nameščeno na drugem računalniku kot aplikacijska strežnika s portaloma. Računalnika sta bila med sabo povezana preko brezžičnega usmerjevalnika Asus WL-500g Premium. Računalnik z nameščenima portaloma je bila na usmerjevalnik priključen preko žične povezave. Računalnik z nameščenim orodjem JMeter pa je bil na usmerjevalnik priključen preko brezžične povezave.

3.4.1 Portal Joomla brez predpomnjenja

Portal Joomla brez vklopljenga predpomnenja je uspel obdelati 49,05 zahtevka na minuto. V povprečju je trajala obdelava enega zahtevak 1216 ms. Razlika med maksimalnim in minimalnim časom obdelave enega zahtevka je predvsem posledica začetne zakasnitve pri začetnih dostopih do vstopne strani pri katerih mora portal pridobiti tekstovne vrednosti, potrebe za prikaz prve strani, iz podatkovne baze. Hkrati je potrebno omeniti, da ima portal Joomla oz. javanski stroj precej manjšo zahtevnost po delavnem pomnilniku, pri strežbi vstopne strani maksimalno 50 MB, kot portal Liferay. Ima pa hkrati tudi precej več dostopov do V/I napravo, saj se glede na podatke iz programa Process Explorer prenosi podatkov iz V/I napravo konstantno gibljejo med 1,4 MB/s in 1,5 MB/s pri vsakem strežniškem zahtevku. Podatki iz programa JProfiler o porabi delavnega spomina kopice javanskega virtualnega stroja kažejo, da poraba ni presegla 5MB delavnega spomina, skupno pa je bilo rezerviranega 7MB delavnega spomina za kopice javanskega virtualnega stroja.



Slika 24. Graf hitrosti obdelave zahtevkov portala Joomla brez vklopljenega predpomnjenjem izmerjeno z orodjem JMeter.

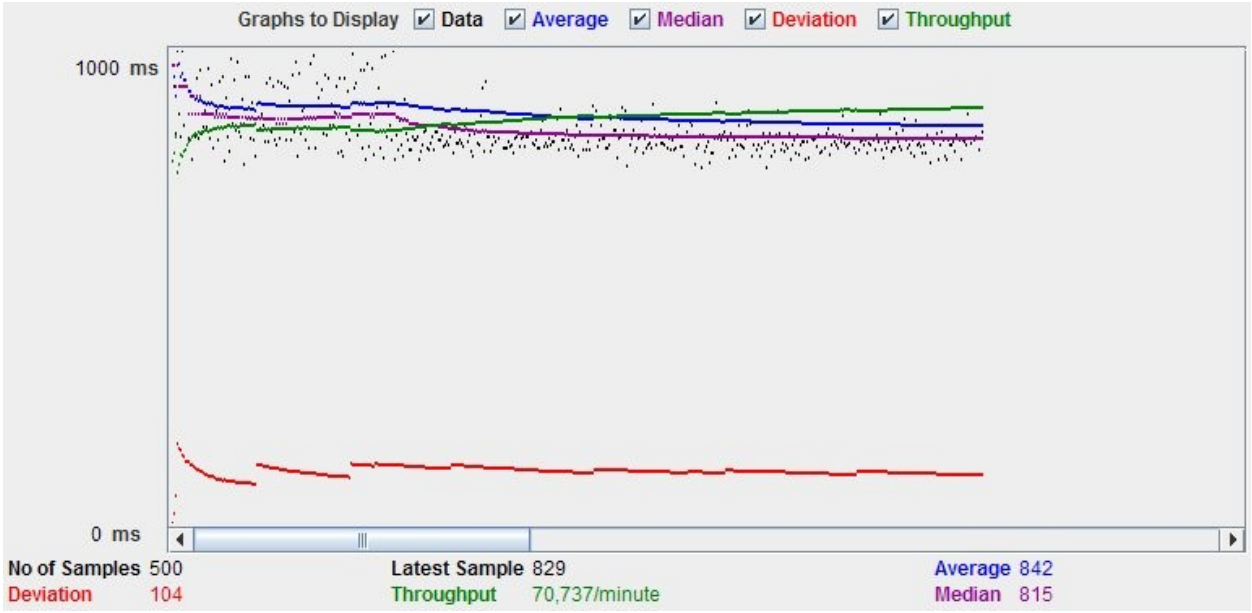
Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
Joomla req...	500	1216	1189	1299	1109	3384	0,00%	49,0/min	26,6

Slika 25. Tabela hitrosti obdelave zahtevkov portala Joomla brez vklopljenega predpomnjenjem izmerjeno z orodjem JMeter.

3.4.2 Portal Joomla s predpomnjenja

Z vklopljenim predpomnjenjem na portalu Joomla je bilo strežnik lahko obdelal 70,737 zahtevkov na minut, kar je nekoliko več kot v predhodnem primeru brez prepomnenja. Maksimalna poraba delavnega pomnilnika kopice javanskega stroja se je povzpela na 5,150 MB. Skupno pa je bilo rezerviranega 7,1 MB za kopice javanskega stroja. Celotna poraba delavnega pomnilnika je ostalo enaka kot v primeru brez vklopljenega predpomnenja brez večjih odstopanj.

Prav tako ni bilo večjih razlik pri V/I prenosih, ki so se konstantno gibali med 1,4 MB/s in 1,5 MB/s.



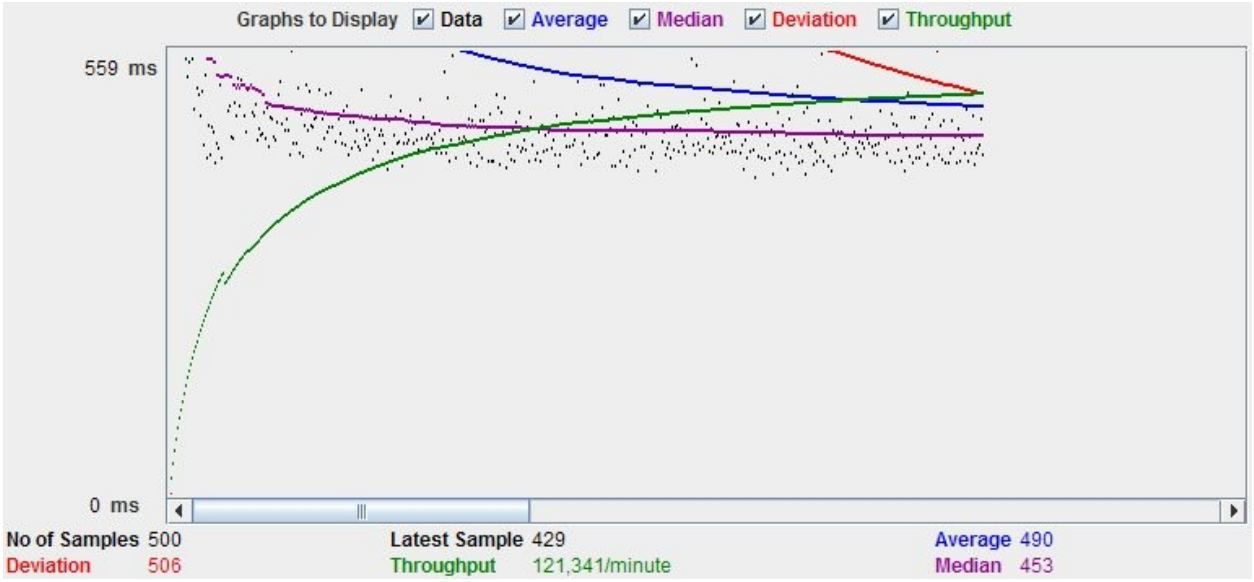
Slika 26. Graf hitrosti obdelave zahtevkov portala Joomla z vklopljenim predpomnjenjem izmerjeno z orodjem JMeter.

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
Joomla req...	500	842	815	932	753	1737	0,00%	1,2/sec	38,3

Slika 27. Tabela hitrosti obdelave zahtevkov portala Joomla z vklopljenim predpomnjenjem izmerjeno z orodjem JMeter.

3.4.3 Portal Liferay s predpomnjenjem

Portal Liferay je dosegel skoraj 2 krat večjo obdelavo zahtevkov kot portal Joomla z vklopljenim predpomnenjem. Razloge za toliko večjo zmožnost obdelave zahtevkov gre iskati predvsem v predpomnjenju, ki ga uporablja portal Liferay in hkrati s tem tudi porabi delavnega pomnilnika, ki je bila v tem primeru maksimalno skoraj 400 MB, povprečna poraba pa je bila okrog 350 MB. Razlika glede na portal Joomal je opazna tudi pri V/I dostopih, ki so pri portalu Liferay v povprečju med 0 KB/s pa do 500 KB/s. Maksimalen V/I prenos pa je bil okrog 3 MB. Maksimalna poraba kopice javanskega stroja je bila okrog 175 MB, skupno pa je bilo rezerviranega 245 MB delavnega prostora za kopico javanskega stroja. Kot je razvidno iz spodnjih dveh slik portal Liferay potrebuje ob zagonu nekaj več časa za obdelavo zahtekov maksimalno 11581 ms, ko pa je konfiguracija portala vzpostavljena in se hkrati vzpostavi tudi ustrezno stanje v portalskem prepomnilniku, postane tudi strežba zahtevkov hitrejša.



Slika 28. Graf hitrosti obdelave zahtevkov portala Liferay z vklopljenim predpomnjenjem izmerjeno z orodjem JMeter.

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
CMS tester:Liferay request	500	490	453	512	401	11581	0,00%	2,0/sec	51,6

Slika 29. Tabela hitrosti obdelave zahtevkov portala Lifery z vklopljenim predpomnjenjem izmerjeno z orodjem JMeter.

3.4.4 Sistemska zahtevnost in zmogljivost portala Lifery zaključek

Kot je razvidno iz testnih podatkov lahko z uporabo predpomnenja dosežemo večjo število postrežnih zahtevkov na določeno časovno enoto. Portal Liferay z večjo porabo delavnega pomnilnika in zamnjšano V/I aktivonostjo doseže večje število postreženih zahtevkov na časovno enoto. Ima pa hkrati ob prvih dostopih do posameznih strani portala večjo začetno zakasnitev, saj mora ob zagonu portal vzpostaviti ustrezne strukture in konfiguracijo, ter ustrezno napolniti predpomnilnik z generiranimi podatki, ki jih generirajo portleti in portal. Pri razlagi rezultatov je potrebno omeniti tudi, da portala uporabljata različni podatkovni bazi. Portal Joomla je pri testiranju uporabljal podatkovno bazo MySQL [16]. Portal Liferay pa je pri testiranju uporabljal bazo HSQLDB [22].

3.5 Predpomnenje v portalu Liferay

Portal Liferay uporablja za potrebe predpomnenja ogrodje Ehcache [23]. Ogrodje Ehcache je porazdeljen javanski predpomnilnik. Omogoča shranjevanje v delavni pomnilnik in disk, replikacijo s pomočjo kopiranja in razveljavitev, poslušalce, razširitve predpomnilnika, obdelovalce napak predpomnilnika in številne druge napredne zmožnosti. Portal ima definirani dve konfiguraciji za predpomnenje. Prva konfiguracija se nanaša na predpomnenje podatkov, ki se pojavljajo pri dostopih do podatkovne baze. Ta konfiguracija se uporablja pri ogrodju Hibernate, ki skrbi za dostop do baze. Definirane ima tri predpomnilniška področja oz. prepomnilnike. Prvi je t.i. privzet predpomnilnik, ki ga uporablja ogrodje Hibernate v primeru, da pravila ne določajo drugače. Drug je predpomnilnik, ki spremlja in shranjuje spremembe pri posodobljanju (update) podatkov v določenih baznih tabelah. Tretji predpomnilnik pa shranjuje podatke o prijavljenih uporabnikih preko razreda com.liferay.portal.model.impl.UserImpl.

Drugi predpomnilnik ima definirana štiri predpomnilniška področja. Kot v prejšnjem primeru je tudi tukaj definiran privzeti pomnilnik. Njegova vloga je enaka kot v predhodnem primeru. Definirana sta ločena predpomnilnika v katera se shranjujejo CSS in Javascript datoteke. Prav tako je definiran predpomnilnik v katerega se shranjujejo generirane vsebine spletnih strani in predstavlja glavni predpomnilnik za spletne vsebine.V zadnji predpomnilnik se shranjujejo datoteke (message.resource) v katerih so shranjena različna sporočila v obliki ključ=vrednost in se uporabljajo za prikaz sporočil uporabnikov, ter generiranje statičnih tekstov na spletnih straneh. Takšen način shranjevanja statičnih tekstov je posebno uporaben v primeru, ko gradimo večjezične spletne strani in nam zamenjava datoteke omogča hitri preklop iz prikaza strani v enem jeziku v prikaz strani v drugem jeziku.

4 Implementacija brskalnika po repozitoriju vsebin z uporabo portala Liferay

Kot primer razvoja standardnih javanskih portletov z uporabo sistema Liferay sem se odločil napisati portlet, ki omogoča brskanje po drevesu repozitorija vsebin. Za implementacijo repozitorija vsebin, javanske specifikacije JSR-168 [1], sem uporabil ogrodje Jackrabbit [24], ki je že vključeno v portal Liferay. Za izris oz. prikaz vnosne forme in prikaz rezultatov sem uporabil privzeto implementacijo ogrodja JSF [4] . Za izris drevesa rezultatov iskanja po repozitoriju vsebin in prikaz drevesa repozitorija vsebin pa sem uporabil Javascript knjižnico dtree [26].

Razvoj portleta sem razdelil v tri sklope, ki jih bom bolj podrobno opisal. V prvem sklopu sem ustrezno skonfiguriral dostop do repozitorija vsebin. Kodo, ki vzpostavi povezavo z repozitorijem in pridobi sejo je prikazana spodaj.

```
String home = "Mapa v katero se inicializiral repozitorij vsebin in v katero se bo shranjevala vsebina repozitorija.";

String confPath =  "Datoteka v kateri se nahaja  konfiguracija repozitorija vsebin.";

if (repository == null) {

        repository = new TransientRepository(confPath, home);

}

if ((session == null) || (!session.isLive())) {

                session = repository.login(new SimpleCredentials("none", "none"
        .toCharArray()));

}
```

S pomočjo omenjene kode aplikacija pridobi dostop do repozitorija vsebin, ki ji omogoča brskanje po repozitoriju, dodajanje novih vozlišč z vsebino, dodajanje novih verzij vozliščem, ponastavljanje verzij vozliščem, dodajanje novih lastnosti na vozlišča, itd. Sama konfiguracija ogrodja Jackrabbit je sestavljena iz treh delov:

1. V prvem delu je potrebno izbrati datotečni sistem v katere se bodo shranjevali podatki in konfiguracija repozitorija.V tem delu konfiguracije se definira tudi varnostni mehanizme, ki bo urejela in preverjal dostope do repozitorija vsebin. Primer konfiguracije:

   ```
   <!-- Konfiguracija datotečnega sistema za repozitorij vsebin -->

   <FileSystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">

           <param name="path" value="${rep.home}/repository" />

   </FileSystem>

   <!-- Konfiguracija urejevalnika dostopov  -->

   <Security appName="Jackrabbit">
   ```

```
<AccessManager class="org.apache.jackrabbit.core.security.SimpleAccessManager" />
    <LoginModule class="org.apache.jackrabbit.core.security.SimpleLoginModule">
            <param name="anonymousId" value="anonymous" />
        </LoginModule>
    </Security>
```

2. V drugem delu konfiguracije je potrebno izbrati datotečni sistem v katerega se bodo shranjevali podatki za posamezne delavne prostore. Konfiguracija predvideva definicijo privzetega delavnega prostora in korenske mape v katero se bodo shranjevali podatki iz vseh delavnih prostorov. Hkrati je potrebno izbrati tudi datotečni sistem v katere se bodo shranjevali podatki iz delavnih prostorov in urejevalnik, ki bo skrbel za način shranjevana podatko v datotečni sistem.

```
<Workspaces rootPath="${rep.home}/workspaces" defaultWorkspace="liferay" />
<Workspace name="${wsp.name}">
        <FileSystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
            <param name="path" value="${wsp.home}" />
        </FileSystem>
    <PersistenceManager
    class="org.apache.jackrabbit.core.persistence.bundle.BundleFsPersistenceManager" />
    </Workspace>
```

3. V tretjem delu je potrebno v konfiguraciji podobno kot v drugem primeru izbrati datotečni sistem in urejevalnik, ki bo skrbel za način shranjevana podatko v datotečni sistem, za verzioniranje vozlišč v repozitoriju.

```
<Versioning rootPath="${rep.home}/version">
        <FileSystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
            <param name="path" value="${rep.home}/version" />
        </FileSystem>
    <PersistenceManager
    class="org.apache.jackrabbit.core.persistence.bundle.BundleFsPersistenceManager" />
    </Versioning>
```

Iz opisane konfiguracije je razvidno, da ima ogrodje Jackrabbit tri, med sabo neodvisne konfiguracije, s pomočjo katerih je mogoče definirati ločena področja oz. mape za posamezne dele repozitorija. Možno je celo definirati shranjevanje različnih delov repozitorija v različne

datotečne sisteme in z uporabo različnih urejevalnikov shranjevanje vsebin v shranjevanje delov repozitorija v različnih formatih odvisno od potreb posameznega sistema oz. aplikacije.

V konfiguraciji ogrodja Jackrabbit je zapisan tudi primer uporabe gruče repozitorijev ogrodj Jackrabbit z uporabo podatkovne baze MySQL [16], ki je pa nisem bolj podrobno raziskal.

Za razvoj portletov ponuja sistem Liferay dva načina. Pri prvem načinu gre za razširjanje funkcionalnosti sistema Liferay. To je t.i. »extension« način preko katerega se nove funkcionalnosti portala vgradijo v obstoječo kodo portala in postanejo del osnovne množice datotek, ki sestavljajo sistem Liferay. Pri drugem načinu pa se nove funkcionalnosti, z uporabo določenih knjižnic, razvijejo kot manjši neodvisni paketi (WAR[27]), ki jih je mogoče dodajati že delujočemu sistemu Liferay (hotdeploy). Ta način razvoja portletov za sistem Liferay je t.i. »plugin« način razvoja portletov. Pri razvoju portleta za sistem Liferay sem uporabil prvi način razvoja. Portlet sem razvil z uporabo orodja Eclipse[28].

Za vključitev portleta v portal Liferay je potrebno vpisati portlet v naslednji konfiguracijski datoteki:

1. Portlet je potrebno vpisati v datoteko liferay-portlet-ext.xml, preko katere se določijo vloge (role), preko katerih mogoče upravljati oz. dostopati do portleta. Izsek iz datoteke:

   ```
   <portlet>
     <portlet-name>JCR_ BROWSER</portlet-name>
    <instanceable>true</instanceable>
   </portlet>
   <role-mapper>
         <role-name>administrator</role-name>
         <role-link>Administrator</role-link>
    </role-mapper>
   ```

2. Portlet je prav tako potrebno vpisati v datoteko liferay-display.xml, preko katerej je portlet umeščen v ustrezno kategorijo in je kot tak dostopen preko aplikacije za dodajanje portletov sistema Liferay in ga je mogoče umestiti v portal. Izsek iz datoteke:

   ```
   <category name="category.JCR_browser">
    <portlet id="JCR_BROWSER" />
   </category>
   ```

3. Definicija portleta oz. razreda, ki predstavlja portlet se zapiše v datoteko portlet-ext.xml. V datoteki se definira tudi razred, ki bo obdeloval zahtevke usmerjene na portlet. Ker sem razvil portlet, ki bo uporabil JSF knjižnico, sem za portlet izbral že implementiran razred

com.sun.faces.portlet.FacesPortlet, ki omogoča strežbo JSF portlet zahtevkov. Izsek iz datoteke:

```
<portlet>
    <portlet-name>JCR_BROWSER</portlet-name>
    <display-name>JCR_ BROWSER</display-name>
    <portlet-class>com.sun.faces.portlet.FacesPortlet</portlet-class>
     <init-param>
          <name>com.sun.faces.portlet.INIT_VIEW</name>
          <value>/html/portlet/ext/repbrowser/index.jsp</value>
    </init-param>
    <supports>
       <mime-type>text/html</mime-type>
     </supports>
    <portlet-info>
          <title>Brskalnik po repozitoriju vsebin</title>
           <short-title>Brskalnik</short-title>
           <keywords>brskalnik,repozitorij vsebin</keywords>
     </portlet-info>
```

4. Poleg omenjene konfiguracije je bilo potrebno ustrezno konfiguriratu tudi datoteko web.xml. V njej je bilo potrebno definirati servlet[19], ki skrbi za obdelavo JSF zahtevkov in ustrezno definirati poslušalce, ki obvestijo sistem Liferay o novem portletu. Hkrati je bilo potrebno vpisati servlet[19] com.liferay.portal.kernel.servlet.PortletServlet, preko katerega se sistemu Liferay z ustreznim parametrom sporoči ime razreda portleta, ki bo sprejemal in obdeloval zahtevke.

5. Nazadnje je bilo potrebno v datoteko faces-config.xml vpisati ime razreda v katerega se bodo preko spletne forme shranjevale vrednosti, ki jih uporabnik vpiše v spletni brskalnik. Omenjena datoteka predstavlja konfiguracjo ogrodja JSF. V njej je bilo potrebno definirati tudi delovni tok oz. prehode med posameznimi spletnimi stranmi, ki jih sproža uporabnik s klikom na ustrezne gumbe. Enostavnost uporabe ogrodje JSF je opazna pri klicih metod javanskih razredov. Metode razredov, definiranih v datoteki faces-config.xml, ki imajo ustrezen podpis je mogoče klicati direktno iz JSP-jev[20. Vračati morajo namreč vrednost tipa String in ne smejo sprejemati parametrov. Izsek iz datoteke:

```
<!-- Izsek prikazuje definicijo razreda na katerem je mogoce klicati ustrezno napisane metode in v katerega se vpisujejo podatki iz spletnih form -->
<managed-bean>
<managed-bean-name>repbrowser</managed-bean-name>
<managed-bean-class>com.ext.portlet.diploma.RepositoryBrowser</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
</managed-bean>
<!-- Navigacijsko pravilo prdstavlja delavni tok aplikacije oz. prehod uporabnika iz neke strani aplikacije na neko drugo stran, glede na to, kateri gumb oz. akcija je bila izvedena s strani uporabnika -->
<navigation-rule>
    <from-view-id>/html/portlet/ext/repbrowser/index.jsp</from-view-id>
    <navigation-case>
        <from-outcome>prikazidrevo</from-outcome>
        <to-view-id>/html/portlet/ext/repbrowser /prikaziDrevo.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

Z uporabo ogrodja JSF postane prenašanje vrednosti iz spletnega brskalnika ali celo klicanje javanskih metod zelo enostavno. Primer uporabe JSF ogrodja:

1. Z naslednjim označitvenim elementom je mogoče prenesti, vrednost vnosnega polja iz spletnega brskalnika v javanski razred brez vsakršne dodatne kode. Prav tako je mogoče definirati polje kot obvezno tako, da ga mora uporabnik obvezno izpolniti. Vrednost atributa value predstavlja ime lastnosti razreda v katero se prenese vrednost vnosnega polja.

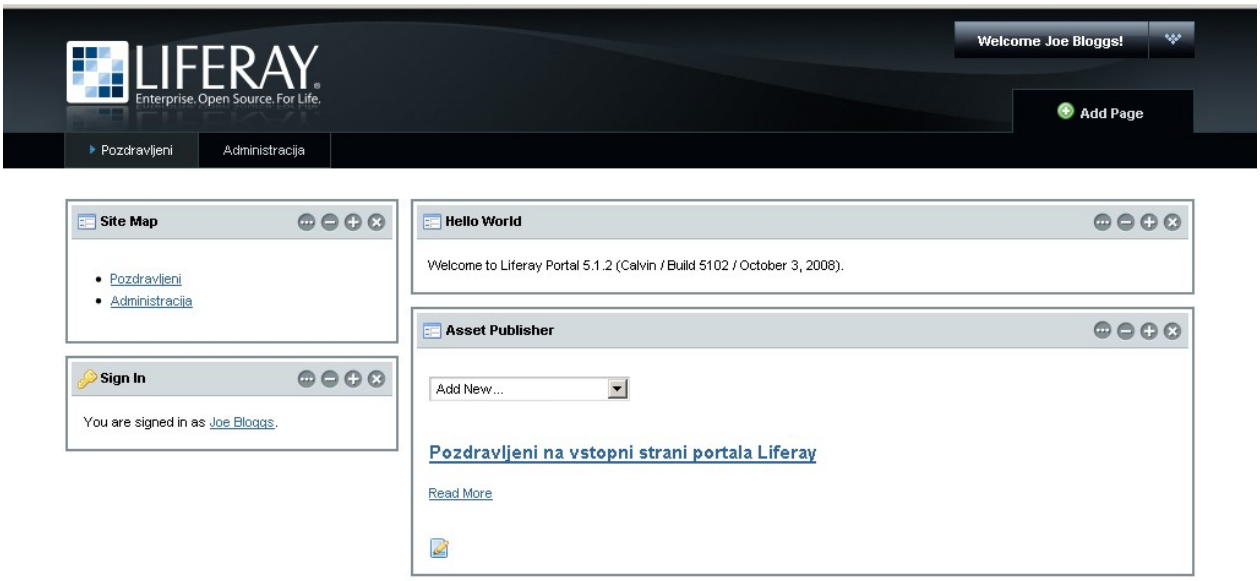
   ```
   <h:inputText id="nodeName" required="false"
   value="#{repbrowser.nodeName}" />
   ```

2. Z naslednjim označitvenim elementom je mogoče poklicata metodo na javanskem razredu in s tem uporabnika usmeriti na neko drugo stran aplikacije oz. sprožiti kakšno drugo obdelavo. Vrednost atributa action predstavlja ime metode, ki se pokliče na javanskem razredu. Metoda mora vračati vrednost tipa String in ne sme sprejemati parametrov.

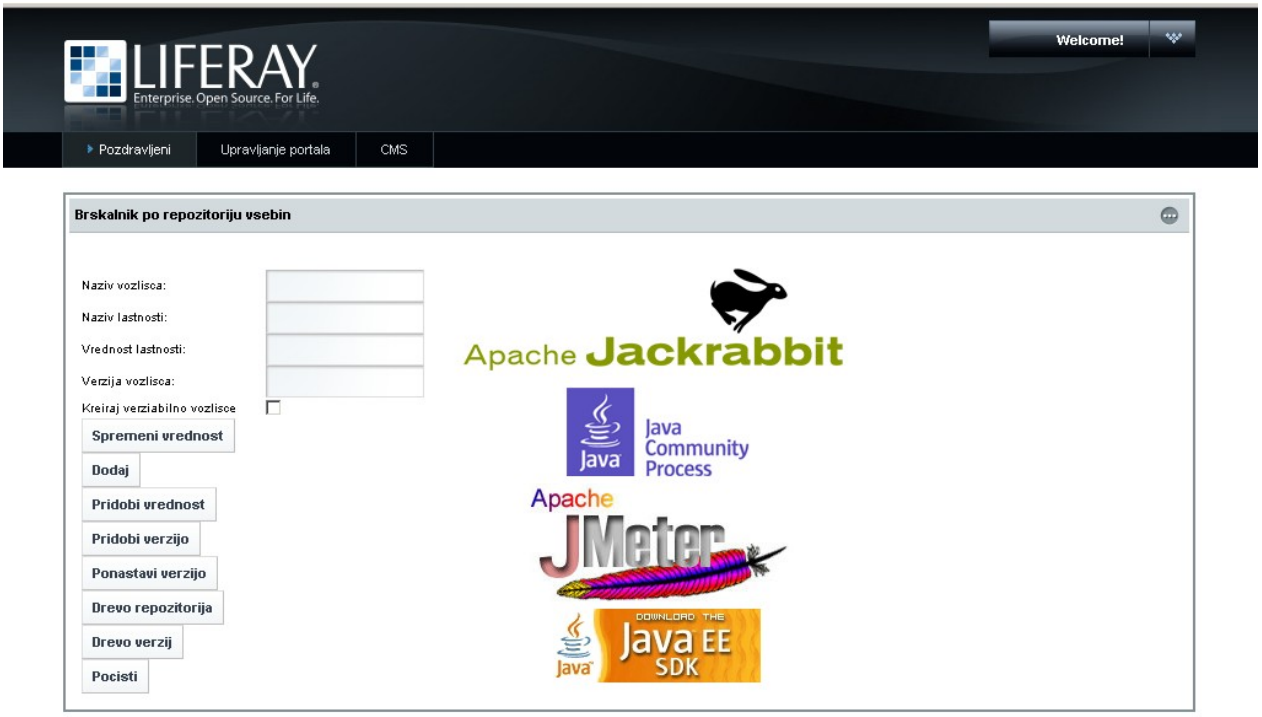
   ```
   <h:commandButton action="#{repbrowser.forwardVerzija}"
   ```

value="Spremeni vrednost" />

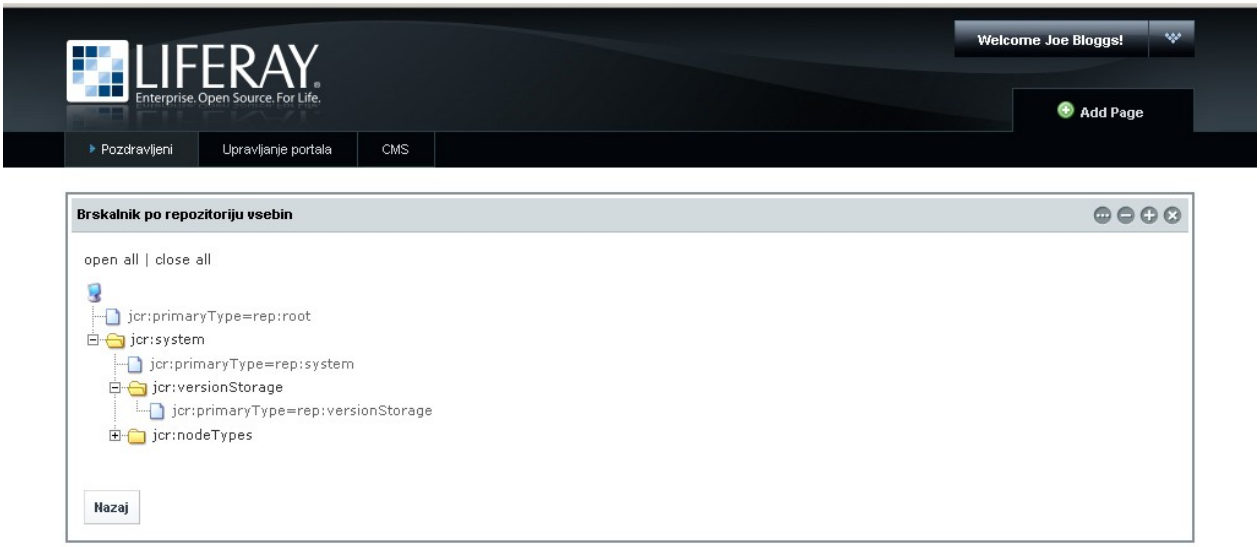
Z uporabo vseh omenjenih ogrodij in konfiguracij mi je uspelo narediti javanski portlet, ki omogoča upravljanje z vozlišči repozitorija vsebin. Portlet je vključen v portal Liferay in ga je mogoče poljubno dodajati oz. odstranjevati s posameznih strani portala. Portlet omogoča tudi izris celotnega drevesa repozitorija vsebin, ter prikaz samo določenega vozlišča ali zgodovine verzij določenega vozlišča v obliki drevesa.



Slika 30. Vstopno stran WCMS sistema Liferay.



Slika 31. Portlet brskalnika po repozitoriju vsebin.



Slika 32. Prikaz drevesa repozitorija vsebin.

5 Sklepne ugotovitve

Standardizirane specifikacije omogočajo programerjem in razvijalcem programske opreme pisanje programov in programske kode, ki ni neposredno vezana na implementacijo posamezne knjižnice oz. ogrodja posameznega proizvajalca oz. implementatorja. Javanske specifikacije občajno definirajo množico vmesnikov, preko katerih razvijalec programske opreme uporablja funkcionalnost, ki mu jih nudi implementacija določene specifikacije. Tak način programiranja omogoča menjavo implementacije določene knjižnice oz. ogrodja znotraj programa brez večjih sprememb oz. posegov v kodo, ki izvaja klice metod ali funkcij preko vmesnikov določene implementacije. Javanske specifikacije običajno, poleg vmesnikov definirajo tudi natančna navodila funkcionalnosti oz. delovanja posameznih metod, ter t.i komplet kompatibilnosti tehnologije (TCK - Technology Compatibility Kit), ki glede na [29] predstavlja množico testov, orodij in dokumentov preko katerih je mogoče ugotoviti ali določena implementacija zadošča zahtevam podanim v specfikaciji oz. ali je skladna z zahtavmi zapisanimi v specifikaciji. Slaba stran specifikacij pa je vsekakor nezmožnost popolne kontrole nad implementacijo posameznega proizvajalca, ki lahko privede do težav oz. nekompatibilnosti ob zamenjavah implementacij. Komplet kompatibilnosti tehnologoje in sama podrobnost s katero specifikacija opisuje posamezne funkcinalnost deloma odpravlja omenjeno težavo.

Standardne javanske specifikacije omogočajo javanskim portalom kot je Liferay zamenjavo (brez večjih sprememb kode) aplikacijskih strežnikov, podatkovnih baz, knjižnic potrebnih za izris spletnih strani, knjižnic za upravljanja z vsebinami, itd. Omenjena fleksibilnost vpliva tudi na razširljivost in uporabo takšnih produktov oz. sistemov. Hkrati standardne javanske specifikacije niso omejene samo na programski jezik Java in jih je mogoče implementirati tudi v drugi programskih jezikih npr. PHP, kar je razvidno tudi iz primera implementacije javanskega repozitorija vsebin v jeziku PHP oz. WCMS sistemu Typo3 [9].

Specifikacija javanskih portletov JSR-168 omogoča pisanje portletov, ki so neodvisni od posameznega portal ali aplikacijskega strežnika. Zadostno zagotovilo za takšno prenosljivost je kompatibilnost aplikacijskega strežnika s specifikacijo JSR-168. Portleti omogočajo generiranje delov vsebine, ki jih je mogoče kasneje agregirati in s tem generirati vsebino spletne strani.

Specfikacija repozitorija vsebin JSR-170 in njena novejša različica JSR-283 omogočata razvijalcem upravljanje z vsebino in njeno verzioniranje. Repozitorij vsebin je predstavljen kot množica delavnih prostorov. Vsak delavni prostor je drevesna struktura, ki je sestavljena iz vozlišč in lastnosti. Lastnosti so deli vozlišč v katere se shranjuje vsebina. Implementacija specifikacije Jackrabbit omogoča podrobno konfiguracijo repozitorija in je dokaj enostavna za vključitev in uporabo v aplikaciji.

Specifikacija JavaServer Faces JSR-127 omogoča javanskim razvjalcem izdelavo spletnih strani skoraj brez uporabe označitvenega jezika HTML z uporabo posebnih označitvenih knjižnic (tag library). Hkrati specifikacija omogoča direktno klicanje javanskih metod, ob izvedbah različnih

dogodkov(Javascript events), ki ji proži uporabnik v spletnem brskalniku. Specifikacija predstavlja nivo abstrakcije, ki je postavljen nad označitveni jezik HTML, ki skrije podrobnosti uporabe obeh omenjen tehnologije in s tem zmanjša zahtevnost oz. potrebno znanje za pisanje spletnih aplikacij.

Specifikacija portletov predstavlja osnovno za vsak javanski portal, ki omogoča izvajanje portletov oz. vsebuje portlet zabojnik. WCMS sistem Liferay je v osnovi množica portletov razdeljenih v kategorije glede na njihov namen uporabe. Implemetacijo javanskega repozitorija vsebin predstavlja knjižnica Jackrabbit, ki je prav tako že vključena v WCMS sistem Liferay in služi za shranjevanje vsebin oz. se uporablja v vseh portletih, ki se nahajajo v kategoriji portletv CMS. Preko njih je mogoče vnašati tekstovne vsebine, datoteke in slike, ki jih je mogoče verzionirati. Implementacija javanske specifikacije JSF ni vključena v WCMS sistem Liferay, vendar jo je možno brez večjih težav vključiti v portal. Slednje je bilo prikazana z implementacijo brskalnika po repozitoriju vsebin. Implementacija javanske specifikacije JSF omogoča hitrejši razvoj spletnih aplikacij saj s svojim abstraktnim UI modelom olajša pisanje uporabniških vmesnikov in omogoča razvoj spletnih strani skoraj popolnoma brez znanja označitvenega jezika HTML, skriptnega jezika javascript Javascript.

Portal Liferay je visoko zmogljiv portal, ki omogoča napredno upravljanje z vsebinami, dinamično izdelavo spletnih strani, njegovo razširanje z dodajanjem novih Portlevo in servisov in uporabo preko 60 portletov, ki so že vključeni portal. Portal uporablja napredne tehnike predpomnjenja, ki je sestavljeno iz dveh nivojev. Prvi del predpomnenja se izvaja ob dostopih do baze. Drug del pa se uporablja za predpomnenje generiranih spletnih strani. Razlika v uporabi predpomnenja se je pokazala tudi v primerjavi s portalom Joomla, kjer je portal Liferay dosegal skoraj dvakrat več postreženih zahtevkov kot drugi portal. Cena za večjo hitrost obdelave zahtevkov pa je bila vsekakot večja poraba delavnega pomnilnika. Znotraj portala Liferay sem razvil/implementiral portlet za brskanje po repozitoriju vsebin. Razvoj portleta je bil po začetnih težava z vzpostavitvijo razvojnega okolja, dokaj enostaven in je potekal brez večjih težav. Portal Liferay obsega veliko količino tehnologij in implementacij različnih specifikacij, zato bi vsekakor predlagal, da bi portal tudi v prihodnje ostal predmet nadaljnih študij in raziskav.

6 Dodatek A (koda in konfiguracija portleta brskalnika po repozitoriju vsebin)

6.1 Javanski razred

```
package com.ext.portlet.repositorybrowser;

import java.io.IOException;
import java.util.Iterator;

import javax.jcr.AccessDeniedException;
import javax.jcr.InvalidItemStateException;
import javax.jcr.ItemExistsException;
import javax.jcr.LoginException;
import javax.jcr.Node;
import javax.jcr.NodeIterator;
import javax.jcr.PathNotFoundException;
import javax.jcr.Property;
import javax.jcr.PropertyIterator;
import javax.jcr.Repository;
import javax.jcr.RepositoryException;
import javax.jcr.Session;
import javax.jcr.SimpleCredentials;
import javax.jcr.UnsupportedRepositoryOperationException;
import javax.jcr.Value;
import javax.jcr.ValueFormatException;
import javax.jcr.lock.LockException;
import javax.jcr.nodetype.ConstraintViolationException;
import javax.jcr.nodetype.NoSuchNodeTypeException;
import javax.jcr.nodetype.NodeType;
import javax.jcr.version.Version;
import javax.jcr.version.VersionException;
import javax.jcr.version.VersionHistory;
import javax.jcr.version.VersionIterator;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.jackrabbit.core.TransientRepository;

public class RepositoryBrowser {

	private Log log = LogFactory.getLog(RepositoryBrowser.class);
	private String nodeName = "";
	private String nodeVersion = "";
	private String nodeProperty = "";
	private String propertyValue = "";
	private String drevo = "";
	private String message = null;
	private boolean versionable;
	private Repository repository = null;
	private Session session = null;
	private int zapSt = -1;
	private String messageStyle = "display: none;";
```

```
	public String forward() {
		setMessage("");
		String forward = "submit";
		return forward;
	}

	public String pocisti() {
		nodeName = "";
		nodeVersion = "";
		nodeProperty = "";
		propertyValue = "";
		drevo = "";
		message = "";
		return "";
	}

	public String forwardVerzija() {
		setMessage("");
		log
				.info("Spreminjam vrednost property-a. Dodajam novo
verzijo vozlisca.");
		spremeniVrednost(nodeName, nodeProperty, propertyValue);
		getSession().logout();
		return "";
	}

	public String getVrednostProperty() {
		setMessage("");
		log.info("Pridobi verzijo vozlišča.");
		Node n = getNode(nodeName);
		try {
			setMessage("Ime vozlisca: " + nodeName + " Ime lasnosti: "
					+ nodeProperty + " Vrednost: "
					+ n.getProperty(nodeProperty).getString());
		} catch (PathNotFoundException e) {
			log.error("Vrednost propertya : " + e);
			setMessage(e.getMessage());
		} catch (RepositoryException e) {
			log.error("Vrednost propertya : " + e);
			setMessage(e.getMessage());
		}

		getSession().logout();
		return "";
	}

	public String getVerzija() {
		setMessage("");
		log.info("Pridobi verzijo vozlišča.");
		versioningHistory(nodeName, nodeProperty, nodeVersion);
		getSession().logout();
		return "";
	}

	public String revertNodeVersion() {
		setMessage("");
		log.info("Vracam vozlisce na verzijo: " + nodeVersion);
```

```
            setMessage("Vozlisce uspesno prestavljeno na verzijo " +
nodeVersion);
            Node n = getNode(nodeName);
            try {
                  n.restoreByLabel(nodeVersion, false);
            } catch (RepositoryException e) {
                  log.error("Ponastavljenje verzije ni uspelo: " +
e.getMessage());
                  setMessage(e.getMessage());
            }
            getSession().logout();
            return "";
      }

      public String dodajVozlisce() {
            setMessage("");
            setMessage("Dodajam vozlisce: ");
            try {
                  String[] s = nodeName.split("/");
                  Node n = getSession().getRootNode();
                  for (int i = 0; i < s.length; i++) {
                        n = n.addNode(s[i], "nt:unstructured");
                        if (isVersionable()) {
                              n.addMixin("mix:versionable");
                        }
                        message += s[i] + " ";
                        if (i == (s.length - 1)) {
                              String[] pNamesA = nodeProperty.split(",");
                              String[] pValuesA = nodeProperty.split(",");
                              if (pNamesA.length != pValuesA.length) {
                                    message = "Stevilo lastnosti in stevilo
vrednosti lastnosti se ne ujema.";
                                    log
                                                .error("Stevilo lastnosti in
njihovih vrednosti se ne ujema.");
                                    return "";
                              }
                              for (int j = 0; j < pNamesA.length; j++) {
                                    message += pNamesA[j] + "=" + pValuesA[j]
+ " ";
                                    n.setProperty(pNamesA[j], pValuesA[j]);
                              }
                        }
                  }
                  setMessage(message);
                  getSession().save();
            } catch (RepositoryException e) {
                  log.error("Dodajanje vozlisca: " + e);
                  setMessage(e.getMessage());
            }

            getSession().logout();
            return "";
      }

      public String getNodeTree() {
            setMessage("");
```

```
		Node n = null;
		try {
			n = getSession().getRootNode();
		} catch (RepositoryException e) {
			log.error("Izris drevesa vozlisc: " + e);
		}
		StringBuffer sb = new StringBuffer();
		initTree(sb);
		getNodeTree(sb, n, "", -1);
		closeTree(sb);
		drevo = sb.toString();
		getSession().logout();
		return "prikazidrevo";
	}

	public String getVersionTree() {
		setMessage("");
		StringBuffer sb = new StringBuffer();
		initTree(sb);
		getVersionHistoryTree(sb, getNode(nodeName));
		closeTree(sb);
		drevo = sb.toString();
		getSession().logout();
		return "prikazidrevo";
	}

	private Node getNode(String nodePath) {
		Node node = null;
		try {
			node = getSession().getRootNode();
			node = node.getNode(nodePath);
		} catch (PathNotFoundException e) {
			log.error("Pridobivam node: " + e);

		} catch (RepositoryException e) {
			log.error("Pridobivam node: " + e);
		}
		return node;
	}

	public void spremeniVrednost(String nodePath, String nodeProperty,
			String value) {
		log.info("Kreiram novo verzijo vozlisca: " + nodePath + " "
				+ nodeProperty + " " + value);
		Node n = getNode(nodePath);
		try {
			boolean isVer = n.isNodeType("mix:versionable");
			if (isVer) {
				n.checkout();
			}
			n.setProperty(nodeProperty, value);
			getSession().save();
			if (isVer) {
				n.checkin();
			}
			setMessage("Uspesno sem naredil novo verzijo vozlisca");
		} catch (UnsupportedRepositoryOperationException e) {
```

```
                log.error("Kreiram novo verzijo vozlisca error: " + e);
            } catch (LockException e) {
                log.error("Kreiram novo verzijo vozlisca error: " + e);
            } catch (RepositoryException e) {
                log.error("Kreiram novo verzijo vozlisca error: " + e);
            }

        }

        public void versioningHistory(String nodePath, String property,
                String verzija) {
            setMessage("");
            message = "Verzija lastnosti: " + verzija + " " + property + "=";
            try {
                Node n = getNode(nodePath);
                VersionHistory history = n.getVersionHistory();
                history.addVersionLabel(verzija, verzija, false);
                Version v = history.getVersion(verzija);
                n = v.getNode("jcr:frozenNode");
                log.info(n.toString());
                Property p = n.getProperty(property);
                if (p.getDefinition().isMultiple()) {
                    Value[] vv = p.getValues();
                    for (int i = 0; i < vv.length; i++) {
                        message += vv[i].getString() + ",";
                    }
                } else {
                    message += p.getString();
                }
                setMessage(message);
            } catch (UnsupportedRepositoryOperationException e) {
                log.error("Pridobi zgodovino: " + e);
            } catch (RepositoryException e) {
                log.error("Pridobi zgodovino: " + e);
            }
        }

        private Session getSession() {
            String home = "C:/jackrabbit-repository";
            String confPath = home + "/config/repository.xml";
            try {
                log.info("Pridobivam session");
                if (repository == null) {
                    repository = new TransientRepository(confPath, home);
                }

                if ((session == null) || (!session.isLive())) {
                    session = repository.login(new
SimpleCredentials("none", "none"
                            .toCharArray()));
                }
            } catch (LoginException e) {
                log.error("pridobivanje sessiona: " + e);

            } catch (IOException e) {
                log.error("pridobivanje sessiona: " + e);
```

```
            } catch (RepositoryException e) {
                  log.error("pridobivanje sessiona: " + e);

            }
            return session;
      }

      private void getVersionHistoryTree(StringBuffer sb, Node n) {
            String imeVozlisca = "";
            try {
                  zapSt = 0;
                  int level = -1;
                  imeVozlisca = n.getName();
                  sb.append("\nd.add("  +  zapSt  +  ","  +  level  +  ",'"  +
imeVozlisca
                              + "');\n");
                  level = zapSt;
                  VersionHistory history = n.getVersionHistory();
                  for   (VersionIterator   it   =   history.getAllVersions();
it.hasNext();) {
                        Version version = (Version) it.next();
                        log.info(version.getName() + " "
                                    + version.getCreated().getTime());
                        zapSt++;
                        sb.append("\nd.add(" + zapSt + "," + level + ",'"
                                    + version.getName() + " "
                                    +     version.getCreated().getTime()     +
"');\n");
                        PropertyIterator pi = version.getProperties();
                        int currZapSt = zapSt;
                        properties2String(sb, pi, currZapSt);
                        nodes2String(sb, version.getNodes(), currZapSt);
                  }
            } catch (UnsupportedRepositoryOperationException e) {
                  log.error("Napaka pri branju verzije vozlisca: " + e + " "
                              + imeVozlisca);
            } catch (RepositoryException e) {
                  log.error("Napaka pri branju verzije vozlisca: " + e + " "
                              + imeVozlisca);
            }
      }

      private void nodes2String(StringBuffer sb, NodeIterator ni,
                  int level) {
            try {
                  while (ni.hasNext()) {
                        zapSt++;
                        Node n = ni.nextNode();
                        sb.append("\nd.add(" + zapSt + "," + level + ",'" +
n.getName()
                                    + "');\n");
                        int currZapSt = zapSt;
                        properties2String(sb, n.getProperties(),currZapSt);
                        NodeIterator nii = n.getNodes();
                        if(nii.hasNext()){
                              nodes2String(sb,nii,currZapSt);
                        }
```

```
                }
            } catch (UnsupportedRepositoryOperationException e) {
                log.error("Napaka pri branju verzije vozlisca: " + e);
            } catch (RepositoryException e) {
                log.error("Napaka pri branju verzije vozlisca: " + e);
            }
        }

        private void properties2String(StringBuffer sb, PropertyIterator pi,
                int level) {
            try {
                while (pi.hasNext()) {
                    zapSt++;
                    Property p = pi.nextProperty();
                    if (p.getDefinition().isMultiple()) {
                        Value[] vA = p.getValues();
                        for (int i = 0; i < vA.length; i++) {
                            Value v = vA[i];
                            sb.append("\nd.add(" + zapSt + "," + level
+ ",'"
                                    +  p.getName()  +  "  "  +
v.getString() + "');\n");
                            zapSt++;
                        }
                    } else {
                        sb.append("\nd.add("  +  zapSt  +  ","  +  level  +
",'"
                                + p.getName() + " " + p.getString()
+ "');\n");
                    }

                }
            } catch (UnsupportedRepositoryOperationException e) {
                log.error("Napaka pri branju verzije vozlisca: " + e);
            } catch (RepositoryException e) {
                log.error("Napaka pri branju verzije vozlisca: " + e);
            }
        }

        private void getNodeTree(StringBuffer sb, Node n, String prefix, int
level) {
            try {
                boolean oneNode = true;
                if (nodeName != null && !nodeName.equals("")) {
                    oneNode = n.getName().equals(nodeName);
                    if (oneNode&&zapSt!=0) {
                        level = -1;
                        zapSt = 0;
                    }else{
                       level = 0;
                    }
                }
                if (oneNode) {
                    sb.append("\nd.add("  +  zapSt  +  ","  +  level  +  ",'"  +
n.getName()
                            + "');\n");
```

```
                }
                level = zapSt;
                log.info(n.getName());
                PropertyIterator pi = n.getProperties();
                while (pi.hasNext()) {
                      Property p = pi.nextProperty();
                      if (oneNode) {
                            if (p.getDefinition().isMultiple()) {
                                  log.info("Vozlisce je tipa mixin");
                                  Value[] v = p.getValues();
                                  for (int i = 0; i < v.length; i++) {
                                        zapSt++;
                                        sb.append("d.add(" + zapSt + "," +
level + ",'"
                                                    + p.getName() + "=" +
v[i].getString()
                                                    + "');\n");
                                  }
                            } else {
                                  log.info("Vozlisce ni tipa mixin");
                                  zapSt++;
                                  sb.append("d.add(" + zapSt + "," + level +
",'"
                                              + p.getName() + "=" +
p.getString() + "');\n");
                            }
                      }
                }
                NodeIterator ni = n.getNodes();
                while (ni.hasNext()) {
                      zapSt++;
                      getNodeTree(sb, ni.nextNode(), prefix, level);
                }
          } catch (UnsupportedRepositoryOperationException e) {
                log.error("Napaka pri branju verzije vozlisca: " + e);
          } catch (RepositoryException e) {
                log.error("Napaka pri branju verzije vozlisca: " + e);
          }
    }


    public void initTree(StringBuffer sb) {
          zapSt = 0;
          sb.append("<link                                    rel=\"StyleSheet\"
href=\"../../html/js/tree/dtree.css\" type=\"text/css\" />");
          sb.append("<script                              type=\"text/javascript\"
src=\"../../html/js/tree/dtree.js\"></script>");
          sb.append("<div class=\"dtree\">");
          sb.append("<p><a href=\"javascript: d.openAll();\">open all</a> |
<a href=\"javascript: d.closeAll();\">close all</a></p>");
          sb.append("<script type=\"text/javascript\">");
          sb.append("<!--\n");
          sb.append("d = new dTree('d');\n");

    }
```

```
public void closeTree(StringBuffer sb) {
	sb.append("document.write(d);\n");
	sb.append("//-->");
	sb.append("</script>");
	sb.append("</div>");
}


public String getDrevo() {
	return drevo;
}

public void setDrevo(String drevo) {
	this.drevo = drevo;
}

public String getNodeName() {
	return nodeName;
}

public void setNodeName(String nodeName) {
	this.nodeName = nodeName;
}

public String getNodeVersion() {
	return nodeVersion;
}

public void setNodeVersion(String nodeVersion) {
	this.nodeVersion = nodeVersion;
}

public String getNodeProperty() {
	return nodeProperty;
}

public void setNodeProperty(String nodeProperty) {
	this.nodeProperty = nodeProperty;
}

public String getPropertyValue() {
	return propertyValue;
}

public void setPropertyValue(String propertyValue) {
	this.propertyValue = propertyValue;
}

public String getMessage() {
	return message;
}

public void setMessage(String message) {
	if((message == null) ||(message.equals(""))){
		messageStyle="display: none;";
	}else{
		messageStyle="display: block;";
```

```
            }
            this.message = message;
      }

      public boolean isVersionable() {
            return versionable;
      }

      public void setVersionable(boolean versionable) {
            this.versionable = versionable;
      }

      public String getMessageStyle() {
            return messageStyle;
      }

      public void setMessageStyle(String messageStyle) {
            this.messageStyle = messageStyle;
      }

}
```

6.2 Konfiguracijska datoteka Java Server faces faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer Faces
Config 1.1//EN" "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config xmlns="http://java.sun.com/JSF/Configuration">
      <managed-bean>
            <managed-bean-name>repbrowser</managed-bean-name>
            <managed-bean-
class>com.ext.portlet.repositorybrowser.RepositoryBrowser</managed-bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
      <navigation-rule>
            <from-view-id>/html/portlet/ext/repbrowser/index.jsp</from-view-
id>
<navigation-case>
                  <from-outcome>prikazidrevo</from-outcome>
                  <to-view-
id>/html/portlet/ext/repbrowser/prikaziDrevo.jsp</to-view-id>
            </navigation-case>
      </navigation-rule>
      <navigation-rule>
            <from-view-id>/html/portlet/ext/repbrowser/prikaziDrevo.jsp</from-
view-id>
            <navigation-case>
                  <from-outcome>back</from-outcome>
                  <to-view-id>/html/portlet/ext/repbrowser/index.jsp</to-view-
id>
       </navigation-case>
      </navigation-rule>
</faces-config>
```

6.3 Konfiguracijska datoteka za vključitev portleta v portal Liferay liferay-portlet-ext.xml

```
<?xml version="1.0"?>
<!DOCTYPE liferay-portlet-app PUBLIC "-//Liferay//DTD Portlet Application
5.1.0//EN" "http://www.liferay.com/dtd/liferay-portlet-app_5_1_0.dtd">

<liferay-portlet-app>
      <!-- Custom Portlets -->
      <portlet>
      <portlet-name>JCR_BROWSER</portlet-name>
      <instanceable>true</instanceable>
   </portlet>
      <role-mapper>
            <role-name>administrator</role-name>
            <role-link>Administrator</role-link>
      </role-mapper>
      <role-mapper>
            <role-name>guest</role-name>
            <role-link>Guest</role-link>
      </role-mapper>
      <role-mapper>
            <role-name>power-user</role-name>
            <role-link>Power User</role-link>
      </role-mapper>
      <role-mapper>
            <role-name>user</role-name>
            <role-link>User</role-link>
      </role-mapper>
</liferay-portlet-app>
```

6.4 Konfiguracijska datoteka javanskega portleta portlet-ext.xml

```
<?xml version="1.0"?>
<portlet-app>
      <portlet>
      <portlet-name>JCR_ BROWSER </portlet-name>
      <display-name>JCR_ BROWSER </display-name>
      <portlet-class>com.sun.faces.portlet.FacesPortlet</portlet-class>
      <init-param>
                  <name>com.sun.faces.portlet.INIT_VIEW</name>
                  <value>/html/portlet/ext/repbrowser/index.jsp</value>
            </init-param>
      <supports>
         <mime-type>text/html</mime-type>
      </supports>
      <portlet-info>
         <title>Brskalnik po repozitoriju vsebin</title>
         <short-title>Brskalnik</short-title>
         <keywords>brskalnik,repozitorij vsebin</keywords>
      </portlet-info>
      <security-role-ref>
         <role-name>guest</role-name>
```

```
      </security-role-ref>
      <security-role-ref>
         <role-name>power-user</role-name>
      </security-role-ref>
      <security-role-ref>
         <role-name>user</role-name>
      </security-role-ref>
       <security-role-ref>
         <role-name>administrator</role-name>
      </security-role-ref>
   </portlet>
</portlet-app>
```

6.5 Konfiguracijska datoteka javanske spletne aplikacije web.xml

```
<?xml version="1.0"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
<display-name>diploma</display-name>
      <context-param>
            <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
            <param-value>client</param-value>
      </context-param>
      <context-param>
            <param-name>javax.faces.application.CONFIG_FILES</param-name>
            <param-value>/WEB-INF/faces-config.xml</param-value>
      </context-param>
      <context-param>
            <param-name>com.sun.faces.validateXml</param-name>
            <param-value>false</param-value>
      </context-param>
      <listener>
            <listener-
class>com.liferay.util.bridges.jsf.sun.LiferayConfigureListener</listener-
class>
      </listener>
      <listener>
            <listener-
class>com.liferay.portal.kernel.servlet.PortletContextListener</listener-
class>
      </listener>
      <servlet>
            <servlet-name>FacesServlet</servlet-name>
            <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
      </servlet>
      <servlet>
            <servlet-name>diploma</servlet-name>
            <servlet-
class>com.liferay.portal.kernel.servlet.PortletServlet</servlet-class>
            <init-param>
                  <param-name>portlet-class</param-name>
                  <param-value>com.sun.faces.portlet.FacesPortlet</param-
value>
```

```
            </init-param>
            <load-on-startup>0</load-on-startup>
      </servlet>
      <servlet-mapping>
            <servlet-name>FacesServlet</servlet-name>
            <url-pattern>/faces/*</url-pattern>
      </servlet-mapping>
      <servlet-mapping>
            <servlet-name>repbrowser</servlet-name>
            <url-pattern>/repbrowser/*</url-pattern>
      </servlet-mapping>
      <jsp-config>
            <taglib>
                  <taglib-uri>http://java.sun.com/portlet_2_0</taglib-uri>
                  <taglib-location>/WEB-INF/tld/liferay-portlet.tld</taglib-
location>
            </taglib>
            <taglib>
                  <taglib-uri>http://liferay.com/tld/portlet</taglib-uri>
                  <taglib-location>/WEB-INF/tld/liferay-portlet-
ext.tld</taglib-location>
            </taglib>
            <taglib>
                  <taglib-uri>http://liferay.com/tld/security</taglib-uri>
                  <taglib-location>/WEB-INF/tld/liferay-security.tld</taglib-
location>
            </taglib>
            <taglib>
                  <taglib-uri>http://liferay.com/tld/theme</taglib-uri>
                  <taglib-location>/WEB-INF/tld/liferay-theme.tld</taglib-
location>
            </taglib>
            <taglib>
                  <taglib-uri>http://liferay.com/tld/ui</taglib-uri>
                  <taglib-location>/WEB-INF/tld/liferay-ui.tld</taglib-
location>
            </taglib>
            <taglib>
                  <taglib-uri>http://liferay.com/tld/util</taglib-uri>
                  <taglib-location>/WEB-INF/tld/liferay-util.tld</taglib-
location>
            </taglib>
      </jsp-config>
      <security-constraint>
            <web-resource-collection>
                  <web-resource-name>Page Sources</web-resource-name>
                  <url-pattern>*.jsp</url-pattern>
                  <url-pattern>*.jspx</url-pattern>
                  <url-pattern>*.xhtml</url-pattern>
            </web-resource-collection>
            <auth-constraint>
                  <role-name>nobody</role-name>
            </auth-constraint>
      </security-constraint>
      <security-role>
            <role-name>nobody</role-name>
      </security-role>
```

```
</web-app>
```

6.6 Datoteka za prikaz javanskega portleta index.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<f:view><f:loadBundle basename="Language" var="msgs" />
      <h:form><table><tr><td><table border="0">
                  <tr><td colspan="2" align="center">
                        <h:outputText id="errorMessage"
value="#{repbrowser.message}" style="font-size: medium; font-weight: bold;
border: medium solid red; color: red;#{repbrowser.messageStyle}"
styleClass="portlet-font" /></td></tr><tr><td width="150px"></td>
                        <td width="150px"> </td> </tr> <tr>
                        <td ><font class="portlet-font" style="font-size: x-
small;"><h:outputText value="#{msgs.node_name}" />
</font></td><td><h:inputText id="nodeName" required="false"
                              value="#{repbrowser.nodeName}" /></td></tr>
                  <tr>
                        <td><font class="portlet-font" style="font-size: x-
small;">
                        <h:outputText value="#{msgs.property_name}" />
</font></td><td><h:inputText id="nodeProperty" required="false"
                              value="#{repbrowser.nodeProperty}" /></td></tr>
                  <tr>
                        <td><font class="portlet-font" style="font-size: x-
small;">    <h:outputText value="#{msgs.property_value}" /> </font></td>
                        <td><h:inputText id="propertyValue" required="false"
                              value="#{repbrowser.propertyValue}" /></td>
                  </tr> <tr>
                        <td><font class="portlet-font" style="font-size: x-
small;"><h:outputText value="#{msgs.node_version}" /> </font></td>
                        <td><h:inputText id="nodeVersion" required="false"
                              value="#{repbrowser.nodeVersion}" /></td> </tr>
                  <tr><td><font class="portlet-font" style="font-size: x-
small;">    Kreiraj verziabilno vozlisce</font></td>
                        <td><h:selectBooleanCheckbox
value="#{repbrowser.versionable}" /></td></tr><tr>
                        <td><h:commandButton
action="#{repbrowser.forwardVerzija}"
                              value="Spremeni vrednost" /></td>
                        <td> </td></tr><tr>
                        <td><h:commandButton
action="#{repbrowser.dodajVozlisce}"
                              value="Dodaj" /></td>
                        <td> </td></tr><tr>
                        <td><h:commandButton
action="#{repbrowser.getVrednostProperty}"
                              value="Pridobi vrednost" /></td>
                        <td> </td></tr><tr>
                        <td><h:commandButton action="#{repbrowser.getVerzija}"
                              value="Pridobi verzijo" /></td>
                        <td> </td></tr><tr>
<td><h:commandButton action="#{repbrowser.revertNodeVersion}"
                              value="Ponastavi verzijo" /></td>
```

```
                        <td> </td></tr><tr>
                        <td><h:commandButton
action="#{repbrowser.getNodeTree}"
                              value="Drevo repozitorija" /></td>
                        <td><font class="portlet-font" style="font-size: x-
small;">          </font></td></tr> <tr>
                        <td><h:commandButton
action="#{repbrowser.getVersionTree}"                value="Drevo verzij"
/></td>
<td><font class="portlet-font" style="font-size: x-small;"> </font></td></tr>
      <tr>
                        <td><h:commandButton action="#{repbrowser.pocisti}"
                              value="Pocisti" /></td>
                        <td><font class="portlet-font" style="font-size: x-
small;">          </font></td></tr></table></td>
            <td align="center"><h:graphicImage id="jackrabbit"
value="http://localhost:8080/html/portlet/ext/repbrowser/images/jlogo.gif"/>
      <br /><h:graphicImage id="jcr"
value="http://localhost:8080/html/portlet/ext/repbrowser/images/logo_jcp.gif"/
>     <br /><h:graphicImage id="jmeter"
value="http://localhost:8080/html/portlet/ext/repbrowser/images/jmeter_logo.jp
g"/>  <br />
      <h:graphicImage id="javasdk"
value="http://localhost:8080/html/portlet/ext/repbrowser/images/javaeesdk.gif"
/></td></tr></table></h:form>
</f:view>
```

6.7 Datoteka za prikaz drevesa repozitorija vsebin prikaziDrevo.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<f:view><h:form>
      <h:outputFormat escape="false" value="#{repbrowser.drevo}" />
      <br /><br /><h:commandButton action="back" value="Nazaj" />
      </h:form>
</f:view>
```

6.8 Konfiguracijska datoteka repozitorija vsebin Jackrabbit repository.xml

```
<?xml version="1.0"?>
<!DOCTYPE Repository PUBLIC "-//The Apache Software Foundation//DTD Jackrabbit
1.4//EN" "http://jackrabbit.apache.org/dtd/repository-1.4.dtd">
<Repository>
    <FileSystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
        <param name="path" value="${rep.home}/repository"/>
    </FileSystem>
    <Security appName="Jackrabbit"> <AccessManager
class="org.apache.jackrabbit.core.security.SimpleAccessManager">
        </AccessManager> <LoginModule
class="org.apache.jackrabbit.core.security.SimpleLoginModule">
           <param name="anonymousId" value="anonymous"/>
```

```
        </LoginModule> </Security>
    <Workspaces rootPath="${rep.home}/workspaces" defaultWorkspace="default"/>
     <Workspace name="${wsp.name}"><FileSystem
class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
            <param name="path" value="${wsp.home}"/>
        </FileSystem> <PersistenceManager
class="org.apache.jackrabbit.core.persistence.bundle.BundleFsPersistenceManage
r"><param name="schemaObjectPrefix" value="${wsp.name}_"/>
</PersistenceManager> </Workspace><Versioning rootPath="${rep.home}/version">
  <FileSystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
            <param name="path" value="${rep.home}/version" />
</FileSystem> <PersistenceManager
class="org.apache.jackrabbit.core.persistence.bundle.BundleFsPersistenceManage
r"> <param name="schemaObjectPrefix" value="version_"/></PersistenceManager>
    </Versioning></Repository>
```

7 Viri

[1] JSR 168: Portlet Specification. Dostopno na: http://jcp.org/en/jsr/detail?id=168

[2] JSR 170: Content Repository for JavaTM technology API. Dostopno na: http://jcp.org/en/jsr/detail?id=170

[3] JSR 283: Content Repository for JavaTM Technology API Version 2.0. Dostopno na: http://jcp.org/en/jsr/detail?id=283

[4] JSR 127: JavaServer Faces. Dostopno na: http://jcp.org/en/jsr/detail?id=127

[5] Upravljanje z vsebinami. Dostopno na: http://en.wikipedia.org/wiki/Content_management

[6] Odprtokodni portal Lifery. Dostopno na: http://www.liferay.com/

[7] Java community process. Dostopno na: http://jcp.org/en/home/index

[8]Prednosti standardizacije. Dostopno na: http://wiki.answers.com/Q/Describe_the_advantages_and_disadvantages_of_standardization

[9] Implementacija specifikacija JSR-283 v PHP-ju oz. WCMS sistemu Typo3. Dostopno na: http://typo3.org/fileadmin/teams/5.0-development/PHPQC08-JSR-283.pdf

[10] Liferay administration guide. Dostopno na: http://docs.liferay.com/portal/5.1/official/liferay-administration-guide-5.1.pdf

[11] JMeter. Dostopno na : http://jakarta.apache.org/jmeter/

[12] JProfiler. Dostopno na : http://www.ej-technologies.com/products/jprofiler/overview.html

[13]Process Explorer. Dostopno na: http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx

[14] Spletni portal Joomla. Dostopno na: http://www.joomla.org/

[15] Skriptni jezik PHP . Dostopno na: http://www.php.net/

[16] Odprtokodna podatkovna baza MySQL. Dostopno na: http://www.mysql.com

[17] Aplikacijski strežnik Apache. Dostopno na: http://httpd.apache.org/

[18] Aplikacijski strežnik Apache Tomcat. Dostopno na: http://tomcat.apache.org/

[19] Java Servlet technology. Dostopno na: http://java.sun.com/products/servlet/index.jsp

[20] JavaServer Pages Technology. Dostopno na: http://java.sun.com/products/jsp/

[21] Uporaba PHP-ja na strežniku Apache Tomcat. Dostopno na: http://blog.taragana.com/index.php/archive/running-php-5x-on-windows-using-tomcat-4x-or-5x/

[22] Podatkovna baza HSQLDB. Dostopno na: http://hsqldb.org/

[23] Javansko ogrodje za predpomnenje Ehcache: Dostopno na: http://ehcache.sourceforge.net/

[24] Implementacija javanske specifikacije JSR-168 Apache Jackrabbit. Dostopno na: http://jackrabbit.apache.org/

[25] Implementacija javanske specifikacije JSR 127 JSF. Dostopno na: http://java.sun.com/javaee/javaserverfaces/

[26] Javascript knjižnica za izris dreves. Dostopno na: http://destroydrop.com/javascripts/tree/

[27] Datoteke WAR. Dostopno na: http://en.wikipedia.org/wiki/Sun_WAR_(file_format)

[28] Orodje Eclipse. Dostopno na: http://www.eclipse.org/

[29] TCK. Dostopno na: http://jcp.org/en/resources/tdk

[30] WCMS sistem Joomla. Dostopno na: http://www.joomla.si/

[31] Seznam WCMS sistemov. Dostopno na: http://en.wikipedia.org/wiki/List_of_content_management_systems

[32] FLOW3 ogrodje za razvoj aplikaciji. Dostopno na: http://flow3.typo3.org/documentation/reference/

[33] Stran, ki ponuja implemetacije portletov za javanske portale. Dostopno na: http://www.portlets.com/Default.html

[34] Predstavitev portlet zabojnika. Dostopno na http://java.sun.com/developer/technicalArticles/J2EE/sdk_portletcontainer/

[35] Uvod v JSF. Dostopno na: http://www.oracle.com/technology/tech/java/newsletter/articles/introjsf/index.html

[36] Uvod v servlete. Dostopno na: http://faculty.inverhills.mnscu.edu/speng/cs1127/Notes/Ch26/Overview.htm

[37] Primerjava java swing in JSF. Dostopno na: http://www.oracle.com/technology/pub/articles/nimphius-mills-swing-jsf.html