Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

Matjaž Auflič

VARNE SPLETNE STORITVE NA MOBILNI PLATFORMI

Univerzitetno diplomsko delo

Mentorica: doc.dr. Mojca Ciglarič

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

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

Št. naloge: 01541/2009
Datum: 15.01.2009

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

Kandidat: **MATJAŽ AUFLIČ**

Naslov: **VARNE SPLETNE STORITVE NA MOBILNI PLATFORMI**
SECURE WEB SERVICES ON A MOBILE PLATFORM

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Izdelajte aplikacijo za mobilni telefon, ki bo uporabniku omogočala iskanje in plačilo parkirnih mest. Aplikacija naj za komunikacijo uporablja spletne storitve, varnost naj zagotavlja po specifikaciji WS-Security. Ostale uporabljene tehnologije naj bodo standardne. V delu analizirajte glavne probleme pri programiranju varnih aplikacij za mobilne naprave ter uporabi spletnih storitev z varnostnimi mehanizmi WS-Security, opišite temeljne značilnosti izbranih tehnologij in navedite argumente za njihovo izbiro.
Zasnujte sistem za informiranje uporabnikov o prostih parkirnih mestih in plačevanje parkirnine, opišite njegovo arhitekturo ter opišite aplikacijo za mobilno napravo. Aplikacijo implementirajte, preizkusite ter kritično ovrednotite. Navedite tudi možnosti nadaljnjega razvoja ter morebitne pomanjkljivosti, ki bi jih bilo potrebno preseči pred produkcijsko uporabo sistema.

Mentor:

doc. dr. Mojca Ciglarič

Dekan:

prof. dr. Franc Solina

Povzetek

Java ME ogrodje podpira izvajanje aplikacij na prenosnih in vgradnih napravah kot so na primer prenosni telefoni. Vsebuje majhen podniz Java API funkcij, kar posledično pomeni omejeno funkcionalnost aplikacij. Obstajajo odprtokodne knjižnice za klice spletnih storitev v Java ME ogrodju, ki omogočajo povečanje funkcionalnosti aplikacij. Knjižnica, ki bi podpirala klice varnih spletnih storitev, pa še ne obstaja, zato se diplomska naloga loteva implementacije protokola *WS-Security*, ki skrbi za varno uporabo spletnih storitev v Java ME ogrodju. V ta namen smo implementirali aplikacijo *Parkiranje*, ki vsebuje uporabo varnih spletnih storitev. Aplikacija *Parkiranje* je vrste MIDlet, ki temelji na Java ME ogrodju. Diplomska naloga predstavi rešitev uporabe protokola *WS-Security* v Java ME ogrodju, kjer so najprej predstavljene tehnologije, s katerimi smo se srečali pri izdelavi aplikacije *Parkiranje.* Sledi praktični del, v katerem je prikazana implementacija aplikacije. Parkiranje v večjih mestih nemalokrat predstavlja problem. Iskanje prostega parkirnega mesta ob vse pogostejši časovni stiski velikokrat predstavlja izziv. Na drugi strani pa uporaba in funkcionalnost prenosnih telefonov narašča. Slednje lahko izkoristimo za prej omenjeni problem. Implementirali smo aplikacijo *Parkiranje*, ki bo omogočala iskanje najbližjih prostih parkirnih mest ter rezervacijo in plačevanje le-teh.

Abstract

Java ME framework supports running of applications on mobile and embedded devices like mobile phones. It contains small subset of Java API functions what consequently means limited application functionality. Open-source libraries for calling web services can be utilised in Java ME framework to increase application functionality. The library for support of secure web service calls does not exist yet. The thesis undertakes the development of the *Parkiranje* application which utilises *WS-security* protocol for secure handling of web services in Java ME framework. The application *Parkiranje* is a MIDlet and is based on Java ME framework. The thesis presents the application of *WS-Security* protocol in Java ME framework and the technologies used for programming of *Parkiranje* application. The thesis also demonstrates the *Parkiranje* application in a practical part. Street parking in big cities can be a tricky and time consuming exercise. There is an opportunity to use a mobile phone to speed up and simplify the search of parking space and to facilitate the payment transaction. In the thesis we have developed and implemented *Parkiranje* application, which will help the driver to find the closest free parking spot, facilitate parking spot reservation and payment transaction.

Zahvala

Zahvaljujem se mentorici, doc. dr. Mojci Ciglarič za nasvete in vodenje izdelave diplomskega dela. Predvsem bi se rad zahvalil asistentu Andreju Krevlu za številne nasvete, strokovno pomoč in ideje pri izdelavi praktičnega dela naloge.

Zahvaljujem se svoji družini za podporo izkaženo v času študija. Zahvalil bi se tudi Katarini za lektoriranje diplomskega dela.

Posebna zahvala gre »Mateji« za vso moralno podporo, da me je motivirala in spodbujala, da sem končno prišel do zaključka študija, ter da je vedno verjela vame.

Markusu

Kazalo

1. UVOD ... 1
2. OPIS PROBLEMA ... 3
3. TEHNOLOGIJA ... 4
3.1 Java ME [11] ... 4
3.1.1 Java ME ogrodje ... 5
3.1.2 Konfiguracija za majhne naprave - Connected Limited Device Configuration (CLDC) 6
3.1.3 Konfiguracija za bolj zmogljive naprave in pametne telefone - Connected Device Configuration (CDC) ... 7
3.1.4 MIDP ... 8
3.2 MIDlet ... 8
3.3 Spletne storitve ... 10
3.3.1 Definicija spletne storitve [1] ... 10
3.3.2 Spletne storitve v Java ME ogrodju ... 12
3.3.3 Tehnologije spletnih storitev ... 15
3.3.4 WSO2 WSAS strežnik za spletne storitve ... 16
3.4 Varne spletne storitve ... 18
3.4.1 Sklad tehnologij spletnih storitev [2] ... 18
3.4.2 Web Service Security [2,5] ... 19
3.4.3 WS Policy ... 24
3.5 KRIPTOGRAFIJA ... 26
3.5.1 Uvod v kriptografijo ... 26
3.5.2 Simetrična kriptografija ... 26
3.5.3 Asimetrična kriptografija ... 26
3.5.4 Digitalni podpis dokumentov XML ... 28
3.5.5 Knjižnica Bouncy Castle ... 31
4. PREDSTAVITEV PROGRAMA ... 33
4.1 Primeri uporabe aplikacije Parkiranje ... 33
4.1.1 Prikaz najbližje parkirne hiše in pregled rezervacij ... 34
4.1.2 Iskanje najbližjih parkirnih mest ... 35
4.1.3 Izbira, rezervacija in plačilo parkirnega mesta ... 35
4.1.4 Pregled rezervacij in plačil parkirnih mest ... 36
4.1.5 Iskanje rezerviranih parkirnih mest po parkirnih hišah ... 36

4.1.6 Iskanje najbližje parkirne hiše 37
4.1.7 Iskanje rezerviranih parkirnih mest uporabnika 37
4.1.8 Iskanje najbližjih parkirnih mest 37
4.1.9 Rezerviranje parkirnega mesta 38
4.2 Arhitektura sistema rešitve 38
4.2.1 Arhitektura spletnih storitev 39
4.2.2 Arhitektura aplikacije 47
4.3 Aplikacija za iskanje prostih parkirnih mest Parkiranje 49
4.3.1 Implementacija aplikacije 49
4.3.2 Delovanje aplikacije 53
4.4 Možnosti nadaljnjega razvoja aplikacije 57
4.4.1 Tehnični del 57
4.4.2 Uporabniški del 58
5. SKLEP 59
6. VIRI 61

Seznam slik

Slika 1: Java mobilni ekosistem 5
Slika 2: Java ogrodje 6
Slika 3: CLCD brezžično ogrodje 6
Slika 4: Digitalno medijsko ogrodje 7
Slika 5: Življenjski cikel MIDlet-a 9
Slika 6: Arhitektura spletne storitve 12
Slika 7: Java ME spletne storitve v tipični arhitekturi spletne storitve 13
Slika 8: Tipična JSR 172 aplikacija 13
Slika 9: Tipični koraki kreiranja stub Java razredov 14
Slika 10: Generiranje stub razredov v Sun Wireless Toolkit orodju 14
Slika 11: Primer uporabe knjižnice kSOAP 15
Slika 12: Struktura SOAP sporočila 15
Slika 13: Jedro WSO2 WSAS aplikacijskega strežnika 17
Slika 14: Sklad tehnologij spletnih storitev 18
Slika 15: Varnostni žeton z uporabniškim imenom in geslom 20
Slika 16: Varnostni žeton s Kerberos listkom 20
Slika 17: Varnostni žeton s certifikatom X.509 21
Slika 18: Uporaba XML Signature s SOAP sporočilu 22
Slika 19: Primer časovne znamke 23
Slika 20: Primer uporabe standarda WS-Policy 25
Slika 21: Digitalno podpisovanje 27

Slika 22: Preverjanje digitalnega podpisa 27
Slika 23: Aplikacija Parkiranje in strežnik za spletne storitve kot del življenjske situacije 33
Slika 24: Diagram primera uporabe 34
Slika 25: Arhitekturni diagram 39
Slika 26: Parkirne hiše 40
Slika 27: Parkirna mesta 40
Slika 28: XML datoteka s podatki o uporabniku 41
Slika 29: XML datoteka s podatki o plačilih. 41
Slika 30: Razredni diagram za razred FreePHImpl 42
Slika 31: Razredni diagram za razred FreePPImpl 42
Slika 32: Razredni diagram za razred BookedPPImpl 43
Slika 33: Razredni diagram za razred BookPPImpl 43
Slika 34: WS-Policy za spletno storitev rezervacije parkirnega mesta 44
Slika 35: Določitev varnostnega scenarija za spletno storitev 45
Slika 36: Razredni diagram za razred ReservedPPImpl 45
Slika 37: Spletne storitve objavljene na WSO2 WSAS strežniku 46
Slika 38: Administracija privatnih in javnih ključev v WSO2 WSAS 47
Slika 39: Razredni diagram aplikacije 48
Slika 40: Koda za zgoščevanje in podpisovanje 50
Slika 41: Koda za kreiranja SOAP sporočila po meri 51
Slika 42: Klic varne spletne storitve 52
Slika 43: Pridobitev podatkov o trenutni lokaciji 53
Slika 44: Glavni obrazec aplikacije 54
Slika 45: Obrazec za iskanje prostih parkirnih mest 55
Slika 46: Diagram zaporedja scenarija rezervacije parkirnega mesta 55
Slika 47: Postopek rezervacije/plačila parkirnega mesta 57

Kratice in simboli

API *ang. Application Programming Interface*
Niz funkcij, procedur, metod, razredov ali protokolov, ki ga nudi operacijski sistem, knjižnica ali storitev za podporo računalniškemu programu.

BPEL *ang. Business Process Execution Language*
Jezik za definiranje poslovnih procesov. Uporablja strukturo XML.

BREW *ang. Binary Runtime Environment for Wireless*
Platforma za razvoj aplikacij za prenosne telefone podjetja Qualcomm.

CA *ang. Certificate Authority*
Overitelj. Fizična ali pravna oseba, ki izdaja potrdila ali opravlja druge storitve v zvezi z overjanjem ali elektronskimi podpisi.

DES *ang. Data Encryption Standard*
Enkripcijski standard pri simetrični kriptografiji.

ETSI *European Telecommunications Standards Institute*
ETSI je neprofitna organizacija, katere poslanstvo je priprava standardov in priporočil s področja telekomunikacij in informacijske tehnologije za področje Evrope.

HTTP *ang. HyperText Transfer Protokol*
Protokol, ki se uporablja za prenos podatkov od spletnih strežnikov do brskalnika na strani uporabnika.

IPSec *ang. Internet Protocol Security*
Internetni protokol, ki uvaja celovit pristop k zaščiti prenosa podatkov v omrežjih IP. Poleg zaščite pred napadalci in prisluškovalci omogoča tudi gradnjo navideznih zasebnih omrežij.

JAXP *ang. Java API for XML Processing*
Vmesnik za dodajanje in uporabljanje XML procesorjev v Java aplikacijah (podpora za različne XML procesorje).

JAX-RPC *ang. Java API for XML-based Remote procedure call*
Vmesnik za vključitev klicev za oddaljeni postopek v aplikacijo.

JSR *ang. Java Specification Request*
Dokument specifikacij za razvoj programov v Java ME, Java SE in Java EE ogrodju.

MIDP *ang. Mobile Information Device Profile*
Specifikacija za uporabo Jave na vgradnih napravah kot so to prenosni telefoni ali pa dlančniki.

MMS *ang. Multimedia Messaging System*

Sporočilo MMS je večpredstavnostno oz. multimedijsko sporočilo, ki se ga lahko pošlje s prenosnim telefonom. Sestavljeno je lahko iz kombinacije teksta, slike, videoposnetka in melodije.

OEM *ang. Original Equipment Manufacturer*
Originalni proizvajalec opreme. Tipično je to podjetje, ki uporablja komponente drugih podjetij v svojih produktih, ali pa prodajajo produkte drugih podjetij pod lastno blagovno znamko.

OTA *ang. Over-The-Air*
Ena izmed metod distribuiranja programske opreme na prenosni telefon.

PKCS *ang. Public Key Cryptography Standards*
Niz kriptografskih specifikacij in standardov.

PKI *ang. Public Key Infrastucture*
Infrastruktura javnih ključev.

RSA Ime algoritma tvorijo začetnice avtorjev Rivest, Shamir in Adleman. Algoritem spada v družino algoritmov za šifriranje z javnim ključem in se uporablja tako za šifriranje kot za podpisovanje.

SAML *ang. Security Assertion Markup Language*
Označevalni jezik za varnostne trditve. Temelji na XML ogrodju, ki omogoča izmenjavo varnostnih informacij med domenami.

SHA *ang. Secure Hash Algorithm*
Družina zgoščevalnih funkcij.

SOAP *ang. Simple Object Access Protocol*
SOAP predstavlja XML osnovan protokol za izmenjavo informacij med komponentami, ki je neodvisen od spodaj ležečega transportnega protokola. Večina SOAP implementacij uporablja protokol HTTP.

STB *ang. Set Top Box*
Set-top box (STB) vmesnik je naprava, ki jo uporabnik postavi pred TV, da digitalni signal spremeni v ustrezno obliko. Ta naprava je pretvornik iz digitalnega v analogni TV signal.

SMS *ang. Short Message Service*
Kratka sporočila so tehnologija na prenosnih telefonih. Omogočajo prenos besedilnih sporočil do 160 znakov.

SSL *ang. Secure Sockets Layer*
Protokol za varen prenos podatkov prek spletnih strani. Naslovi spletnih strani URL, ki uporabljajo protokol SSL, se začnejo s *https://*.

URI *ang. Uniform Resource Identifier*

URI je definiran kot niz znakov, ki predstavlja abstrakten ali fizičen vir in je navadno uporabljen za naslavljanje fizičnih virov, kot so spletne strani ali datoteke, v standardu XML pa se največkrat uporablja za referenciranje abstraktnih virov, največkrat imenskih prostorov.

W3C *World Wide Web Consortium*
Mednarodna organizacija za razvoj in vzdrževanje specifikacij za splet (World Wide Web).

XadES *ang. Advanced Electronic Signatures*
Elektronski podpis, ki osnovni specifikaciji XML digitalnega podpisa dodaja elemente za varen digitalni podpis ter elemente za dolgotrajno in legitimno hranjenje digitalno podpisanih dokumentov.

XML *ang. eXtensible Markup Language*
Razširljivi označevalni jezik.

XMLDSIG Standard za digitalni podpis v XML dokumentih.

1. UVOD

Java ME ogrodje predstavlja majhno, robustno okolje za aplikacije, ki se izvajajo na prenosnih in vgradnih napravah. Prenosne in vgradne naprave imajo omejene vire kot so to procesna moč, pomnilnik ali pa prikazovalnik. Nabor API funkcij, ki jih ogrodje Java ME vsebuje, je majhen. Na internetu obstaja veliko odprtokodnih knjižnic za to ogrodje, saj je skupnost razvijalcev v tem ogrodju zelo velika. Aplikacije imajo tako z uporabo teh knjižnic možnost, da se njihova uporabnost in funkcionalnost poveča. Eden izmed načinov obogatitve funkcionalnosti aplikacije je uporaba spletnih storitev. Uporaba spletnih storitev mora biti pri zahtevnejših klicih varna. Za varen klic spletne storitve skrbi protokol WS-Security. Majhen nabor API funkcij Java ME ogrodja ne podpira uporabe tega varnostnega protokola. Protokol WS-Security skrbi s pomočjo avtentikacije in šifriranja ter zagotavljanja integritete spletne storitve za njeno varno uporabo. Knjižnica za klic spletnih storitev že obstaja, vendar pa ne podpira varnega klica. Pričujoča diplomska naloga se bo ob vseh že poznanih tehnologijah osredotočila na protokol WS-Security in njegovo implementacijo v Java ME ogrodju.

Parkiranje v večjih mestih predstavlja vse večji problem. Število avtomobilov, ki dnevno tranzitirajo v večja mesta, vsako leto narašča in tako se problem s parkiranjem samo še povečuje. Iskanje prostega parkirnega mesta zahteva od posameznika vse več časa, potrpljenja in seveda sreče.

Dandanes ima že skoraj vsak prenosni telefon. Prenosni telefoni postajajo vse cenejši, njihova uporabnost pa z vsako novo generacijo prenosnih telefonov narašča. Skoraj ni prenosnega telefona, ki ne bi podpiral podatkovnega prenosa. Cena uporabe interneta na prenosnem telefonu pada in kar sama kliče po večji uporabnosti te storitve.

Prenosni telefon (tudi mobilni telefon, mobitel, mobilec ali mobilnik) je elektronska telekomunikacijska naprava z osnovnimi zmožnostmi, enakimi običajnemu stacionarnemu telefonu, poleg tega pa je popolnoma prenosna in ne potrebuje žične povezave s telefonskim omrežjem. Večina sodobnih prenosnih telefonov se v omrežje povezuje z oddajanjem (in sprejemanjem) radijskih valov. Prenosni telefon komunicira prek omrežja baznih postaj, ki so povezane z običajnim telefonskim sistemom.

Poleg zvočnega pogovora, osnovne funkcije telefona, prenosni telefoni podpirajo tudi številne dodatne storitve kot so SMS za pošiljanje kratkih besedilnih sporočil, paketni prenos podatkov za dostop do interneta in MMS za sprejemanje in pošiljanje fotografij in videa. Nekateri sodobni telefoni so zmožni opravljati naloge, za katere so bile do nedavnega potrebne posebne naprave. Z njimi lahko, denimo, predvajamo glasbene posnetke, poslušamo radio, fotografiramo, ...[9]

Prenosni telefoni so tako postali neločljivi del sodobne družbe. Njihova funkcionalnost se veča iz dneva v dan. Pričujoča diplomska naloga se navezuje na povečano funkcionalnost prenosnih telefonov. Najprej bom predstavil tehnologije, ki so bolj ali manj povezane s prenosnimi telefoni, potem pa predstavil aplikacijo za prenosni telefon, ki išče prosta parkirna mesta glede na trenutno lokacijo. Zaradi zanimivosti problema iskanja prostih parkirnih mest

in želje po spoznavanju novih tehnologij sem se odločil za implementacijo aplikacije za prenosne telefone. Prenosni telefoni pa imajo vseeno še omejitve predvsem glede pomnilnika, zato je bilo potrebno poiskati rešitev v sodelovanju s spletnimi storitvami, za kar pa je potrebna povezava z internetom, kar ne predstavlja problema, saj se stroškovno gledano, strošek interneta na prenosnih telefonih vsako leto manjša. Aplikacije na prenosnih telefonih morajo biti čim manj potratne z viri, kar pomeni, da si ne moremo privoščiti izdelave programa z bogato funkcionalnostjo. In ravno tu več kot prav pridejo spletne storitve, ki predstavljajo dostopno funkcionalnost na oddaljenih računalnikih.

Zadali smo si torej nalogo, da izdelamo aplikacijo za prenosni telefon, ki preko spletnih storitev nudi ustrezno funkcionalnost za iskanje prostih parkirnih mest. Osredotočili se bomo na rešitev, ki izkoristi povečano funkcionalnost prenosnih telefonov za lažje parkiranje v večjih mestih. Izmed vseh tehnologij, s katerimi se bomo srečali, bo glavni cilj implementacija protokola WS-Security v Java ME ogrodju. Vse ostale izbrane tehnologije so široko poznane in uporabljene, tako da se bomo osredotočili, kako realizirati WS-Security za mobilno napravo.

2. OPIS PROBLEMA

Naj najprej predstavimo vsakdanjo situacijo. Janez Novak se pripelje po opravkih v Ljubljano in več časa išče prosto parkirno mesto, kot pa trajajo njegovi opravki. Avtomobilov, ki se vozijo vsak dan po Ljubljani, je vse več, naraščanje števila parkirnih mest pa poteka v manjšem tempu, kot je tempo naraščanja mobilne pločevine. Danes ima že vsak prenosni telefon, medtem ko za navigacijsko napravo tega ne moremo trditi. Torej bi Janezu še kako prav prišla kakšna aplikacija za prenosni telefon, ki bi ga potegnila iz zagate. Vendar mora takšna aplikacija nuditi naslednje funkcionalnosti: iskanje parkirnih mest, plačevanje rezervacije in pregled rezervacij.

Pri izdelavi aplikacije za iskanje se pojavi več izzivov:

- Rešiti je treba problem, kako dobiti podatke o trenutni lokaciji, ki je potrebna za iskanje najbližjih parkirnih mest, ker podatek o prostem parkirnem mestu na drugem koncu mesta, nasproti, kjer se uporabnik nahaja, nima nobene vrednosti.
- Izdelati je treba aplikacijo, katero bi lahko podpiralo oziroma izvajalo čim več prenosnih telefonov.
- Plačevanje storitve rezervacije preko prenosnega telefona mora biti varno.
- Potrebno je izdelati učinkovit algoritem za iskanje najbližjih parkirnih mest.

Odgovori na zgornje izzive nam torej nudijo rešitev zadane naloge, se pravi izdelavo aplikacije za iskanje parkirnih mest.

3. TEHNOLOGIJA

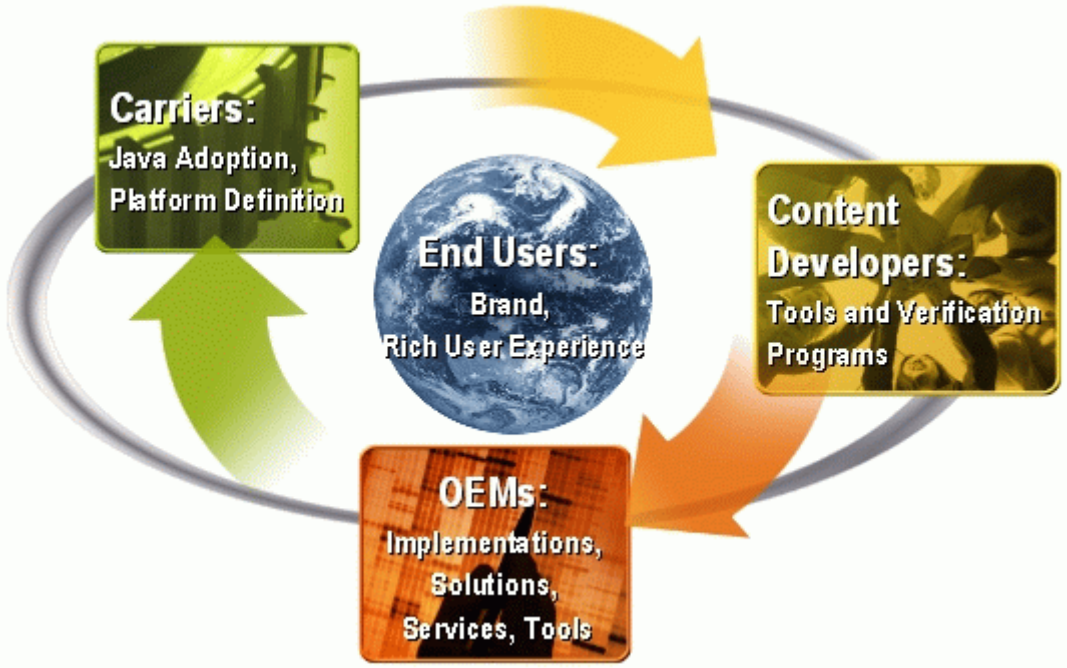
V tem poglavju bomo predstavili tehnologije, s katerimi smo se srečevali pri izdelavi diplomske naloge. Na koncu vsakega podpoglavja bomo utemeljili izbiro opisane tehnologije.

3.1 Java ME [11]

Java ogrodje, Micro Edition (Java ME) zagotavlja robustno in prilagodljivo okolje za izvajanje aplikacij na mobilnih in drugih vgradnih napravah kot so prenosni telefoni, dlančniki, STB (set top box) vmesniki in tiskalniki. Java ME vključuje prilagodljive uporabniške vmesnike, robustno varnost, vgrajene mrežne protokole in podporo za mrežne in »offline« aplikacije, ki se lahko dinamično nalagajo. Aplikacije, ki bazirajo na Java ME, so prenosljive preko mnogih naprav in hkrati izkoriščajo izvorne zmožnosti naprave.

Ogrodje Java ME predstavlja edino pravo odprto rešitev za implementacijo mobilnih aplikacij za industrijo. Tehnologija dopušča prenosljivost aplikacij med platformami in investicije so minimalizirane zaradi možnosti ponovne uporabe. Nenehni razvoj ogrodja narekuje povečujoče zahteve po zmožnostih in performanci v industriji in je zagotovljen z definicijo komponent ogrodja in API funkcij v Java Community Process procesu. Zaradi dejstva, da je tehnologija odprta za uporabo za kogarkoli, je skupnost razvijalcev aplikacij za ogrodje velika in raste. To zagotavlja nenehno izboljševanje in dostopnost aplikacij za ogrodje, ki s tem posel za vse vpletene vključuje v ekosistem.

Ekosistem Java ME tehnologije nastaja okrog številčnih različnih igralcev v industriji, ki vsi sodelujejo in vplivajo na nenehno izboljšavo tehnologije in ogrodja. Končni uporabniki (*end users*) konstantno zahtevajo nove funkcionalnosti in zmožnosti njihovih storitev. Razvijalci vsebine (*content developers*) sprejmejo zahteve uporabnikov in kreirajo nove storitve z novimi zmožnostmi. OEM (*originalni proizvajalci opreme*) kreirajo nove sposobne naprave, ki lahko gostijo nove storitve in tudi s predstavitvijo novih zmožnostih končnim uporabnikom kreirajo nove zahteve. Ponudniki mobilne telefonije (*carriers*) kreirajo mobilno okolje za gostitev in razvoj storitev ter vodijo raziskavo novih poslovnih storitev h končnim uporabnikom. Ta stalna evolucija zahtev in zmožnosti je edini najbolj pomemben razlog za uspeh ogrodja Java in zagotavlja, da se bo nadaljevala z razvojem v prihodnje potrebe vseh sodelujočih v ekosistemu.



Slika 1: Java mobilni ekosistem

3.1.1 Java ME ogrodje

Tehnologija Java ME je bila originalno izdelana zaradi omejitev, ki se pojavljajo pri implementaciji aplikacij za majhne naprave. Za ta namen je Sun definiral osnove za Java ME tehnologijo za prilagoditev takemu omejenemu okolju in s tem omogočil izdelavo Java aplikacij, ki se izvajajo na majhnih napravah z omejenim pomnilnikom, prikazovalnikom in napajalno zmogljivostjo.

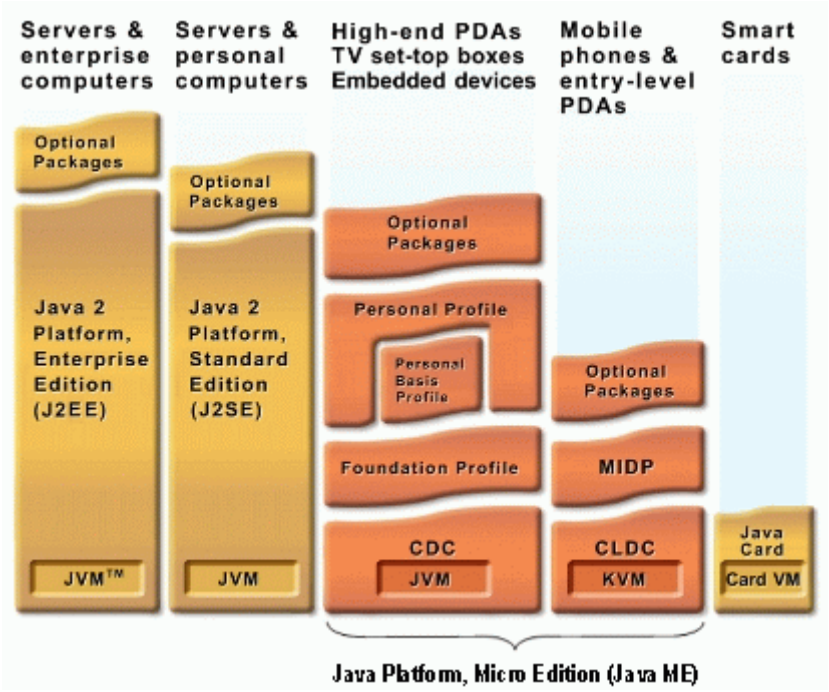
Tehnologija Java ME temelji na treh elementih:

- **konfiguracija** (configuration) zagotavlja najbolj osnovni niz knjižnic in zmožnosti virtualne naprave za širok spekter naprav,
- **profil** (profile) je niz API funkcij, ki podpira bolj ožji spekter naprav in
- **paket možnosti** (option package) je niz API funkcij, ki so specifične glede na tehnologijo.

Skozi čas se je Java ME ogrodje razvilo v dve osnovni konfiguraciji; eno za majhne mobilne naprave in eno, ki cilja na bolj zmogljive mobilne naprave kot so pametni telefoni in STB vmesniki.

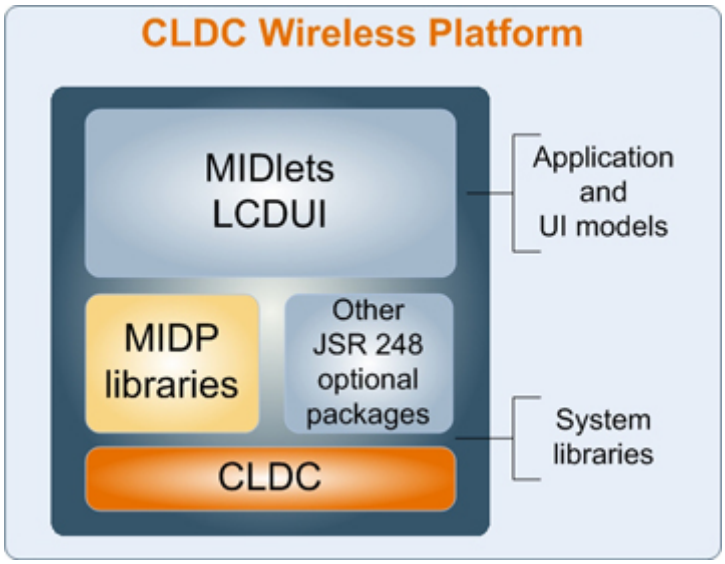
Konfiguracija za majhne naprave se imenuje Connected Limited Device Configuration (CLDC), medtem ko se bolj zmogljiva konfiguracija imenuje Connected Device Profile (CDC).

Slika št. 2 prikazuje pregled komponent Java ME in kako se Java ME nanaša na ostalo Java tehnologijo.



Slika 2: Java ogrodje

3.1.2 Konfiguracija za majhne naprave - Connected Limited Device Configuration (CLDC)



Slika 3: CLCD brezžično ogrodje

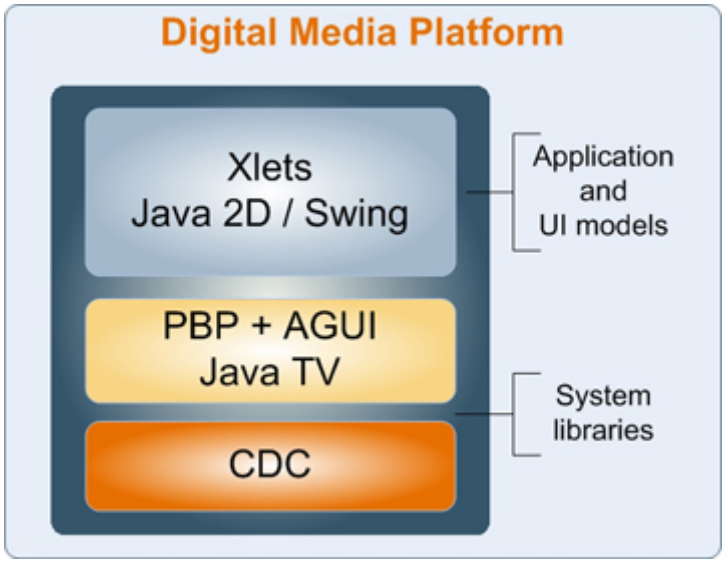
Konfiguracija za naprave z omejenimi viri kot so prenosni telefoni, se imenuje Connected Limited Device Configuration (CLDC). Je specialno načrtovana, da lahko zadošča potrebam ogrodju Java, da se le-ta lahko izvaja na napravah z omejenim pomnilnikom, procesno močjo in grafičnimi zmožnostmi. Na vrhu različnih konfiguracij ogrodje Java ME prav tako specificira številne profile, ki definirajo niz API funkcij na višjih ravneh, ki tako dalje definirajo aplikacijo. Široko privzeta praksa je, da se s kombinacijo CLDC-ja skupaj z Mobile Information Device Profile (MIDP) zagotavlja dovršeno okolje Java aplikacij za mobilne telefone in ostale naprave s podobnimi zmožnostmi.

Dejanska aplikacija s konfiguracijo in profili razpolaga z uporabo različnih razpoložljivih API funkcij v profilu. Za CLDC in MIDP okolje, s katerim je danes večina mobilnih naprav implementirana, je potem kreiran MIDlet. MIDlet je aplikacija, ki jo kreira Java ME razvijalec programske opreme kot je npr.: igra, poslovna aplikacija ali druge mobilne aplikacije. Ti MIDlet-i so lahko enkrat napisani in se lahko izvajajo na vseh razpoložljivih napravah, ki ustrezajo specifikacijam Java ME tehnologiji. MIDlet je na voljo v repozitoriju nekje v eko-sistemu, medtem pa lahko končni uporabnik išče specifični tip aplikacije in jo s pomočjo »over the air« (OTA) standarda naloži na svojo napravo.

3.1.2.1 Sun Java Wireless Toolkit za CLDC

Sun Java Wireless Toolkit (prej poznan kot Java 2 Platform, Micro Edition (J2ME) Wireless Toolkit) je najmodernejše orodje za razvoj brezžičnih aplikacij, ki temeljijo na CLDC-ju in MIDP-ju, ki se izvaja na prenosnih telefonih, dlančnikih in drugih majhnih mobilnih napravah. Orodje vsebuje emulator, funkcije za optimizacijo zmogljivosti in nastavitev značilnosti aplikacije, dokumentacijo ter primere, s pomočjo katerih lahko razvijalci hitro ponudijo trgu zmogljivejše in uspešnejše brezžične aplikacije.

3.1.3 Konfiguracija za bolj zmogljive naprave in pametne telefone - Connected Device Configuration (CDC)



Slika 4: Digitalno medijsko ogrodje

Konfiguracija za podporo večjim napravam z večjimi zmogljivostmi in s stalno mrežno povezavo kot so dlančniki in STB vmesniki, se imenuje Connected Device Configuration (CDC). CDC konfiguracija omogoča uporabo ogrodja Java SE in razvojnih orodij za Java SE v napravah z omejenimi sistemskimi viri.

Upoštevajoč prednosti CDC konfiguracije, ki jih prinaša različnim skupinam, se lahko predpostavi naslednje:

- Podjetja pridobijo na račun uporabe mrežno baziranih aplikacij, ki širijo doseg poslovne logike mobilnim strankam, partnerjem in zaposlenim.
- Uporabniki bodo pridobili na račun kompatibilnosti in varnosti Java tehnologije.
- Razvijalci pridobijo na račun varnosti in produktivnosti Java programskega jezika in bogatih API funkcij v Java ogrodju.

V CDC konfiguraciji so definirani trije različni profili:

- Temeljni profil (The Foundation Profile) (JSR 219),
- Osebni temeljni profil (The Personal Basis Profile) (JSR 217) in
- Osebni profil (The Personal Profile) (JSR 216).

3.1.4 MIDP

Mobile Information Device Profile (MIDP) je specifikacija, ki je izdana za uporabo Jave na vgradnih napravah kot so to prenosni telefoni in dlančniki. MIDP je del Java ME ogrodja in sedi na vrhu CLDC-ja, niz programskih vmesnikov na nižji ravni. MIDP so razvili pod Java Community Process-om kot JSR 37 (MIDP 1.0) in JSR 118 (MIDP 2.0). Od leta 2007 naprej se MIDP 3.0 razvija pod JSR 271. Prve MIDP naprave so bili modeli i80s in i50sx, znamke Motorola, ki so se pojavile na tržišču aprila 2001.

Za izdelavo aplikacije v Java ME ogrodju smo se odločili, ker večina prenosnih telefonov podpira to ogrodje in ker lahko na internetu poiščemo veliko odprtokodnih knjižnic, ki še dodatno razširjajo funkcionalnost izdelave aplikacij v tem ogrodju. Glavne prednosti Java ME ogrodja so v dinamičnosti (nove aplikacije so lahko nameščene na napravah kadarkoli), varnosti, široki skupnosti razvijalcev, objektni orientiranosti in zanesljivosti.

3.2 MIDlet

MIDlet je Java program za vgradne naprave, bolj specifično za Java ME navidezni stroj. Splošno gledano so to igre in aplikacije, ki se izvajajo na prenosnih telefonih. Midlet je MIDP aplikacija. Tako kot applet je tudi MIDlet upravljana aplikacija. Namesto, da jo upravlja brskalnik, je MIDlet upravljan s posebno programsko opremo, ki je vgrajena v napravo, najpogosteje prenosni telefon. Zunanje upravljanje aplikacije je zelo smiselno, saj je potem lahko aplikacija prekinjena kadarkoli s strani zunanjega dogodka kot je to lahko vhodni telefonski klic. [12]

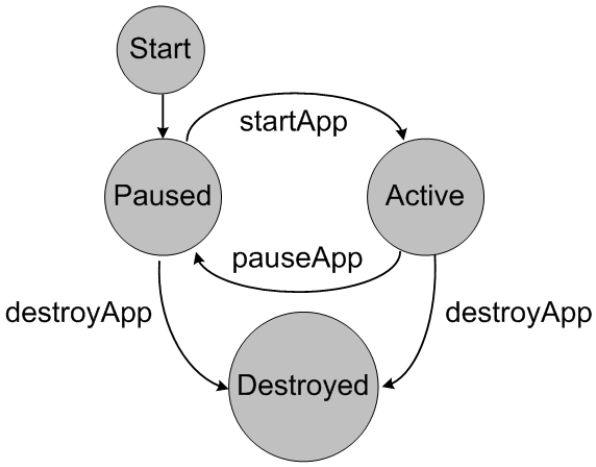
MIDlet zahteva za izvajanje napravo, ki implementira vsaj Java ME, CLDC in MIDP. Kot tudi drugi Java programi imajo MIDlet-i zmožnost »compile once, run anywhere«. Glavna distribucijska datoteka za MIDlet je datoteka tipa .jar, toda MIDlet distribucije lahko sestojijo tudi iz datotek tipa .jad, ki vsebuje lokacijo in opis vsebine .jar datoteke. Implementacija MIDlet-a lahko ali pa ne zahteva prisotnost .jad datoteke.

MIDlet mora izpolniti naslednje zahteve, da se lahko izvaja na mobilnem telefonu:

- Glavni razred mora bit podrazred javax.microedition.midlet.MIDlet,
- MIDlet mora biti zapakiran znotraj .jar datoteke,
- .jar datoteka mora biti pred-preverjena z uporabo »*preverifier*« orodja in
- v nekaterih primerih mora biti .jar datoteka podpisana s strani mobilne družbe.

Vse aplikacije za MID profil morajo biti izpeljane iz posebnega razreda MIDlet. MIDlet razred upravlja življenjski cikel aplikacije. Nahaja se v paketu javax.microedition.midlet.

MIDlet lahko obstaja v štirih različnih stanjih: *loaded* (naložen), *active* (aktiven), *paused* (ustavljen) in *destroyed* (uničen). Slika 5 prikazuje življenjski cikel MIDlet-a. Ko je MIDlet naložen na napravo in je poklican konstruktor, se nahaja v stanju *loaded*. To se lahko zgodi ob kateremkoli času preden programski upravitelj zažene aplikacijo s klicem metode *startApp()*. Po klicu *startApp()* se MIDlet nahaja v aktivnem stanju dokler aplikacijski upravitelj ne pokliče *pauseApp()* ali *destroyApp()*. *PauseApp()* začasno ustavi delovanje MIDlet-a, medtem ko *destroyApp()* ustavi MIDlet. Vsi povratni klici, ki spreminjajo stanje, naj se končajo hitro, ker se stanje ne spremeni popolnoma, dokler metoda ne vrne rezultata.



Slika 5: Življenjski cikel MIDlet-a

Med delovanjem *pauseApp()* metode naj bi aplikacije prekinile z animacijami in sprostijo vire, ki so nepotrebni, medtem ko se aplikacija začasno ustavi. Tako obnašanje se izogiba konfliktom virov med izvajanjem aplikacije v ozadju in nepotrebno potrošnjo baterije. Midlet se lahko ob določenem pogoju izogne prehodu v stanje *destroyed* s klicem točno določene izjeme *MIDletStateChangeException*. MIDlet lahko zahteva ponovno izvajanje aktivnosti s klicem *resumeRequest()*. Če se MIDlet odloči za ustavljeno stanje, naj obvesti aplikacijskega upravitelja s klicem *notifyPaused()*. MIDlet lahko pokliče *notifyDestroyed()* za ustavitev. Klic *System.exit()* ni podprt v MIDP in bo povzročil klic izjeme, namesto da bi ustavil izvajanje aplikacije.

MIDlet-i so lahko aplikacije, ki se popolnoma izvajajo v ozadju ali aplikacije za interakcijo z uporabnikom. Interaktivne aplikacije lahko dobijo dostop do prikazovalnika s pridobitvijo instance razreda *Display*. MIDlet lahko dobi *Display* instanco s klicem *Display.getDisplay* (MIDlet midlet), kjer je MIDlet sam podan kot parameter.

Razred *Display* in vsi ostali MIDP uporabniško vmesniški razredi se nahajajo v paketu javax.microedition.lcdui. Razred *Display* priskrbi metodo *setCurrent()*, ki nastavi trenutno vsebino MIDlet prikazovalnika. Ni zahtevano, da aktualni zaslon naprave nemudoma prikaže MIDlet prikazovalnik, ker metoda *setCurrent* samo vpliva na notranje stanje MIDlet prikazovalnika in obvesti aplikacijeskega upravitelja, da bi MIDlet rad imel dani *Displayable* objekt prikazan. Razlika med *Display* in *Displayable* razredom je ta, da *Display* razred predstavlja strojno opremo prikazovalnika, medtem ko je *Displayable* nekaj, kar je lahko prikazano na prikazovalniku. MIDlet lahko pokliče metodo *isShown()* razreda *Displayable*, da lahko tako ugotovi, če je vsebina dejansko prikazana.

Za izdelavo MIDlet-a smo se odločili, ker je izdelava MIDlet-a dokaj hitra ter učinkovita in ker obstaja veliko brezplačnih orodij, ki podpirajo izdelavo MIDlet-ov. Obstaja več alternativ, in sicer so to lahko Symbian aplikacije, katere je možno izvajati na Symbian operacijskem sistemu, ki so ga skupaj razvili večji razvijalci prenosne opreme. Symbian aplikacije se razvijajo v programskem jeziku C++, kar v našem primeru predstavlja pomanjkljivost, ker obstaja več odprtokodnih knjižnic, ki so nam predstavljale pomoč v razvoju aplikacije, za ogrodje Java ME, kot za operacijski sistem Symbian. Aplikacije za prenosne telefone je možno izdelovati tudi s pomočjo Python programskega jezika, saj obstaja orodje za razvoj Python skript, ki se izvaja na prenosnih telefonih Nokia serija S60. Omejitev, da se skripte izvajajo samo na prenosnih telefonih Nokia serija S60, je največja pomanjkljivost Python skript. Alternativa je tudi BREW aplikacija, kajti BREW (Binary Runtime Environment for Wireless) predstavlja platformo za razvoj aplikacij za prenosne telefone. BREW platforma je zakonsko zaščitena programska oprema, podpira pa jo le majhno število prenosnih telefonov, kar predstavlja slabost razvoja BREW aplikacij. Za BREW tehnologijo stoji podjetje Qualcomm iz San Diega, ZDA. Omenimo pa tudi i-Appli aplikacije podjetja DoCoMo iz Japonske. i-Appli aplikacije prav tako kot MIDlet-i uporabljajo konfiguracijo CLCD podjetja SUN. Japonsko podjetje je razvilo svoj niz API funkcij in ni uporabilo MIDP specifikacije. Pomanjkanje navodil za razvoj i-Appli aplikacij v angleškem jeziku je glavna njihova slabost. Vsaka omenjena alternativa ima pomanjkljivost, ki se ji z odločitvijo za izdelavo MIDlet-a izognemo. MIDlet razvijamo v ogrodju Java ME, za katerega obstaja veliko odprtokodnih knjižnic, ki predstavljajo pomoč v razvoju, in pa veliko prenosnih telefonov podpira ogrodje Java ME. Kot že omenjeno, obstaja veliko brezplačnih orodij za izdelavo MIDlet-ov. Skupnost razvijalcev v Java ME ogrodju je velika, kar predstavlja tudi veliko podporo izdelavi MIDlet-ov.

3.3 Spletne storitve

3.3.1 Definicija spletne storitve [1]

Spletna storitev je programska storitev v spletu, ki jo je mogoče poklicati s standardnimi spletnimi protokoli. Spletna storitev je sestavljena iz ene ali več spletnih metod ali funkcij. Spletna metoda je spletna funkcija, ki ne vrača odgovora. Spletne storitve za komunikacijo z odjemalcem navadno uporabljajo protokol SOAP. Arhitekturo spletnih storitev prikazuje slika 6.

Spletne storitve so kot splošna tehnologija neodvisne od:

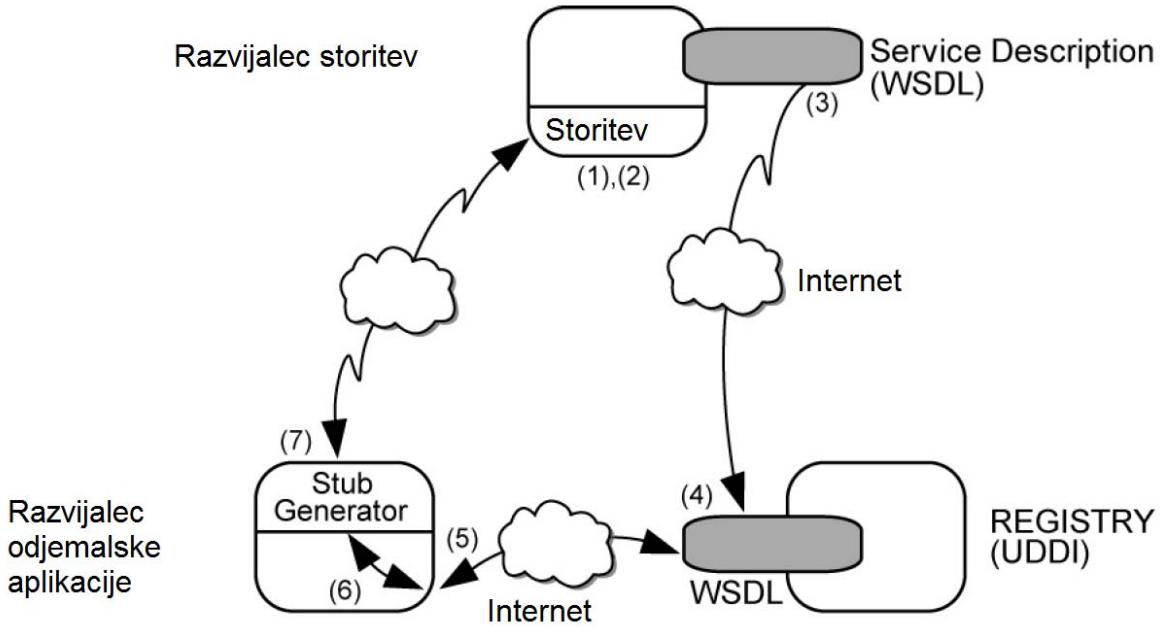
- **Programskega jezika**
  Spletne storitve lahko pišemo v poljubnem programskem jeziku, ne glede na platformo operacijskega sistema.
- **Vrste aplikacije**
  Narava aplikacije ne vpliva na možnost razvoja ali uporabe spletne storitve. Tako jih je mogoče uporabiti v namiznih aplikacijah, spletnih aplikacijah, na spletnih straneh ali pri drugih spletnih storitvah.
- **Operacijskega sistema**
  Za vsak operacijski sistem obstajajo že izdelani objektni modeli, ki omogočajo pošiljanje sporočil SOAP in posledično uporabo tehnologije spletnih storitev. Tako lahko spletne storitve razvijamo in uporabljamo na vseh operacijskih sistemih podjetja Microsoft, prav tako pa tudi na vseh operacijskih sistemih UNIX in preostalih.
- **Strojne platforme**
  Glede na neodvisnost tehnologije od operacijskih sistemov je mogoče sklepati, da je uporaba prav tako neodvisna od spodaj ležeče strojne platforme.

Osnovna ideja spletnih storitev je, da jih je mogoče uporabiti in izdelati na poljubni strojni platformi, operacijskem sistemu, s poljubnim programskim jezikom. Možnost uporabe ni omejena na namizne ali spletne aplikacije. Odjemalec spletne storitve je tako lahko spletna stran, namizna aplikacija, aplikacija mobilne naprave ali druga spletna storitev.

3.3.1.1 Lastnosti spletnih storitev

- **Funkcionalnost črne škatle**
  Spletne storitve in metode, podobno kot objekti, ponujajo možnost funkcionalne abstrakcije. Celotna funkcionalnost spletnih metod je določena z njihovim opisom in programskim vmesnikom. Konkretna izvedba je razvijalcu skrita.
- **Tipično brez stanja**
  Na spletne storitve je treba gledati s stališča izmenjave sporočil, saj gre za sporočilno osnovano arhitekturo.
- **Neodvisnost**
  Spletne storitve so neodvisne od uporabljene tehnologije razvoja ali strojne platforme. Ponudniki izvedb obstajajo praktično na vseh platformah.
- **Šibko sklopljene (povezane)**
  Komunikacija med spletnimi storitvami poteka z izmenjavo standardiziranih sporočil protokola SOAP. Kot ponudnik spletne storitve, ne morete predvidevati, kakšna je platforma na drugi strani žice ali kakšno tehnologijo uporablja.
- **»Brezobrazna« internetna oz. intranetna tehnologija**
  Spletne storitve nimajo uporabniškega vmesnika in vedno ponujajo samo oddaljeno funkcionalnost. Opredelimo jih lahko tudi kot **predstavitveni nivo brez uporabniškega vmesnika**. S stališča večnivojske arhitekture spletne storitve sodijo v nivo uporabniškega vmesnika in predstavljajo **programski uporabniški vmesnik** za nivo poslovne logike.

Osnovna lastnost spletnih storitev je **neodvisnost** od uporabljene strojne in programske platforme. Spletne storitve predstavljajo sporočilno osnovan programski način komunikacije in so namenjene komunikaciji med dvema ali več programskimi sistemi. Pomembno je, da spletne storitve ne razumemo kot uporabniški mehanizem, ker niso namenjene komunikaciji med uporabnikom in računalnikom, pač pa medračunalniški komunikaciji.

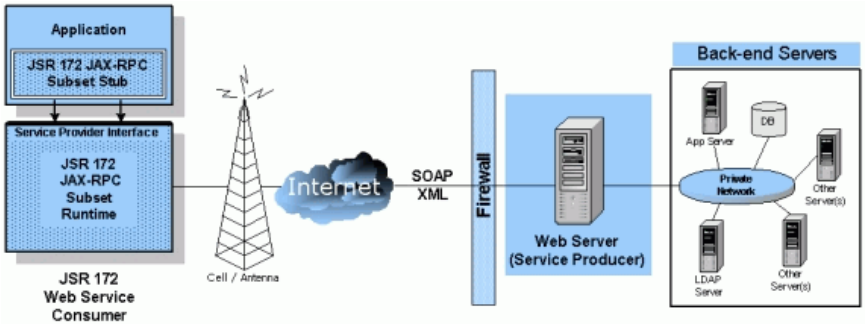


Slika 6: Arhitektura spletne storitve

3.3.2 Spletne storitve v Java ME ogrodju

3.3.2.1 Java ME Web service API funkcije (WSA)

Specifikacija JSR 172, pogosto poznana kot WSA, opisuje možnost povezovanja mobilnega klienta s spletno storitvijo. Namen specifikacije je omogočiti SOAP/XML klice spletnih storitev v mobilni Javi in omogoča tudi osnovno razčlenjevanje XML dokumentov. Implementacija te API funkcije v mobilni napravi je nujen pogoj, ki omogoča dostop do spletnih storitev na tej mobilni napravi. Možna je uporaba *stub generatorja*, ki zgenerira niz razredov potrebnih za dostop do spletne storitve. Omogoča klic metode oddaljene storitve na strani klienta, kot da je lokalni objekt. Generator je del vseh novejših Java ME orodij, kot je to Sun Wireless Toolkit orodje. Kreira tudi vse potrebne uporabniške tipe iz danega WSDL dokumenta. Razredi kreirani po tej poti so lahko preprosto vključeni v mobilno aplikacijo in uporabljeni. Uporaba *stub* rezultira v dejstvu, da ne obstaja nobena vidna sled komunikacije z oddaljeno spletno storitvijo [7].

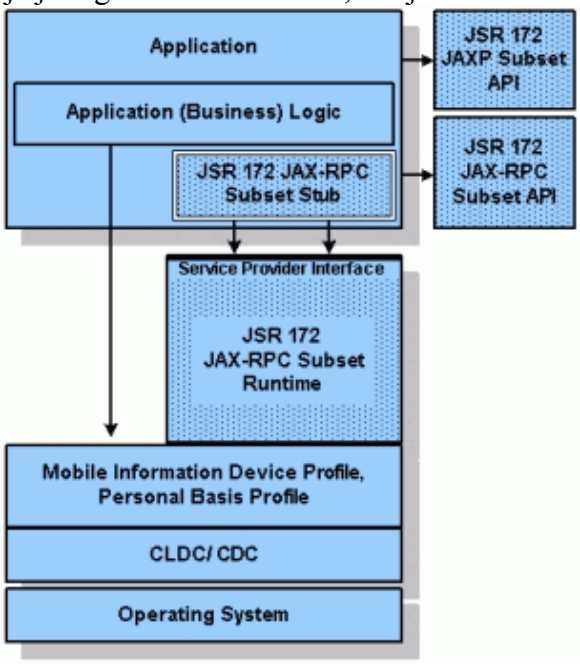


Slika 7: Java ME spletne storitve v tipični arhitekturi spletne storitve

Na visoki ravni ima ta arhitektura prikazana na sliki 7 tri elemente:

- Mrežna aplikacija, ki se nahaja na brezžični napravi, ki podpira WSA. Aplikacija vsebuje JSR 172 stub, ki uporablja JSR 172 okolje za komunikacijo z mrežo.
- Brezžično omrežje in internet ter ustrezne komunikacijske in kodirne podatkovne protokole, vključujoč binarne protokole, HTTP in SOAP/XML.
- Spletni strežnik, ki deluje kot ponudnik storitve, zadaj pa tipično eden ali več požarnih zidov in posredniških prehodov (*proxy gateway*). Spletni strežnik pogosto nudi dostop do izhodnih aplikacij in strežnikov na privatni mreži.

Tipična JSR 172 aplikacija je organizirana na način, kot je viden na sliki 8.

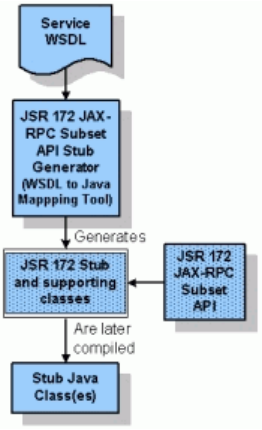


Slika 8: Tipična JSR 172 aplikacija

Aplikacija sama je pametni odjemalec, ki temelji na MIDP-u s poslovno specifično logiko, uporabniškim vmesnikom, vztrajnostno logiko in življenjskim ciklom ter upravljanjem stanja aplikacije. Za obravnavanje XML dokumentov lahko aplikacija uporabi JAXP podniz API funkcij. Za dostop do spletnih storitev pa lahko uporablja podniz JAX-RPC API funkcij.

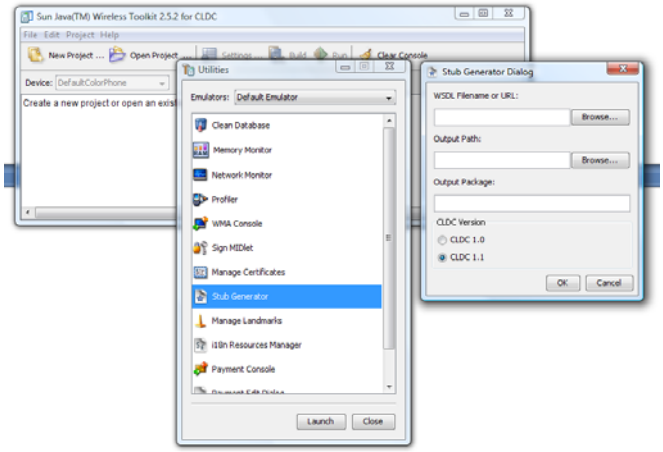
Tipični koraki za kreiranje JSR 172 JAX-RPC klienta (prikazano na sliki 9) so:

1. Generiranje JSR 172 JAX-RPC stub razredov na podlagi WSDL XML dokumenta, ki opisuje oddaljeno spletno storitev.
2. V kodi izdelava primerka generiranega stuba.
3. Po namestitvi klicanje metod generiranega stuba. Te metode ustrezajo storitveni *endpoint wsdl:operation* elementu v WSDL XML dokumentu.



Slika 9: Tipični koraki kreiranja stub Java razredov

Na sliki 10 je prikazano orodje Sun Wireless Toolkit in njegova funkcionalnost generiranja stub Java razredov.



Slika 10: Generiranje stub razredov v Sun Wireless Toolkit orodju

3.3.2.2 Knjižnica kSOAP 2.0

kSOAP je knjižnica, ki omogoča kreiranje aplikacije v mobilni Javi, ki neposredno komunicira s spletno storitvijo in uporablja SOAP protokol. Ena osnovnih prednosti kSOAP knjižnice je, da je lahko uporabljena tudi v starejših napravah, ki uporabljajo starejši MIDP1 profil in v napravah, ki ne ponujajo WSA podpore. Namesti se v obliki dodatne knjižnice za mobilno aplikacijo v mobilni napravi. Knjižnica implementira osnovne razrede za sodelovanje z XML in SOAP protokolom, vendar ne uporablja statičnih podatkovnih tipov, zato je potrebna uporaba HashMap razreda ali implementacija *KVMSerializable* vmesnika, ki je del te knjižnice, za dostop do strukturiranih uporabniških tipov. Trenutno je na voljo druga

revidirana verzija knjižnice [3]. Na sliki 11 je prikaz primer uporabe kSOAP knjižnice v Java ME okolju.

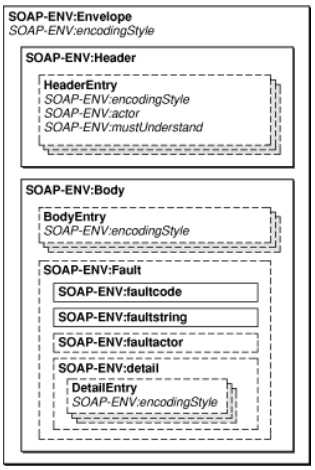
```
try {
//kreiranje SOAP objekta, definiranje imenskega prostora
SoapObject so = new SoapObject("http://ws.crm/","getCustomer");
//nastavitev parametrov
so.addProperty("index",Integer.valueOf(tfCustNo.getString()));
SoapSerializationEnvelope env = new SoapSerializationEnvelope(SoapEnvelope.VER11);
env.setOutputSoapObject(so);
HttpTransport transport = new HttpTransport("http://my:8084/CRM_WS/CRMService" );
//klic metode
transport.call("",env);
//pridobivanje vrednosti rezultata klica metode
SoapObject res=(SoapObject) env.getResponse();
striName.setText(res.getProperty("name").toString());
striSurname.setText(res.getProperty("surname").toString());
}
catch (Exception ex) {ex.printstackTrace();}
```

Slika 11: Primer uporabe knjižnice kSOAP

3.3.3 Tehnologije spletnih storitev

3.3.3.1 SOAP protokol [4,10]

SOAP je lahek protokol za izmenjavo informacij v decentraliziranem, porazdeljenem okolju. Protokol temelji na XML formatu in se sestoji iz treh delov: ovojnice, ki definira ogrodje za opis, kaj je v sporočilu in kako ga obdelati; niza kodirnih pravil za izražanje primerov aplikacijsko definiranih podatkovnih tipov; konvencije za predstavitev oddaljenih proceduralnih klicev in odgovorov. SOAP je lahko potencialno uporabljen tudi v kombinaciji z drugimi protokoli.



Slika 12: Struktura SOAP sporočila

SOAP sporočilo, katerega struktura je prikazana na sliki 12, je XML dokument, ki sestoji iz obvezne ovojnice (SOAP:envelope), opcijskega zaglavja (SOAP:header), telesa (SOAP:body), napak (SOAP:Fault) in predstavitve podatkov (SOAP:encodingStyle):

- Ovojnica (envelope) je korenski element dokumenta XML, ki predstavlja sporočilo. Njegovi atributi definirajo imenske prostore (namespaces), ki jih sporočilo uporablja.

Ima lahko dodatne poljubne atribute. Če jih vsebuje, se morajo nanašati na nek imenski prostor. Lahko vsebuje največ en zaglavni element in en telesni element.

- Zaglavje (header) predstavlja mehanizem za dodajanje novih možnosti v sporočilo. Zaglavje vsebuje informacijo o sporočilu ali o kontekstu, v katerem je sporočilo poslano, ali pa karkoli drugega, za kar pošiljatelj sporočila misli, da je smiselno vstaviti v zaglavje namesto v telo sporočila. Očitno pa je, da se morata pošiljatelj in prejemnik sporočila strinjati glede formata zaglavja. Slednje na primer pri varnih spletnih storitvah določa standard WS-Policy. Zaglavje je element XML, ki je neposredno pod korenskim elementom in vsebuje poljubno število naslednikov, ki sprejemniku povedo, kako naj obdeluje sprejeto informacijo. Ta informacija je zapisana v telesu sporočila. Vsi podelementi morajo pripadati nekemu imenskemu prostoru.
- Telo (body) mora slediti elementu ovojnice. Ta element vsebuje samo informacijo, ki si jo komunicirajoči strani izmenjujeta. Lahko vsebuje neomejeno število podelementov, ki jim pravimo telesni elementi. Ti lahko pripadajo imenskemu prostoru. Privzeto nad njim ne veljajo kakršnakoli pravila kodiranja.
- Napake (faults) so del telesa in se pojavijo le enkrat. Kadar pri obdelavi sporočila pride do kakršnekoli napake, ki je preprečila uspešno obdelavo, je potrebno to napako sporočiti odjemalcu. Protokol SOAP je pri tem zelo natančen, saj dobro predpisuje obliko sporočila, s katerim odjemalcu povemo kaj in morda zakaj je šlo narobe. To storimo z napakami SOAP, ki nosijo status ter informacijo o napaki. Definiramo lahko tudi svoje napake.
- Predstavitev podatkov. V preteklosti je veliko težav pri povezovanju med različnimi platformami povzročala različna predstavitev podatkov. Predstavitev SOAP sporočila v XML formatu, ki omogoča enostavno preverjanje, pretvorbe podatkov, enostavno gradnjo podatkovnih struktur ter zmanjšanje podatkovnega prostora.

3.3.3.2 WSDL

Za uspešno implementacijo spletnih storitev je potreben tudi neke vrste opis funkcionalnosti, ki odjemalcu poda zmožnosti spletne storitve. Trenutno je to podano v obliki standarda WSDL (Web Service Description Language), ki definira prisotne metode, njihove parametre in tipe parametrov, na katere se spletna storitev odziva. WSDL je zapisan v standardu XML, za sistem tipov pa uporablja standard »XML Schema«.

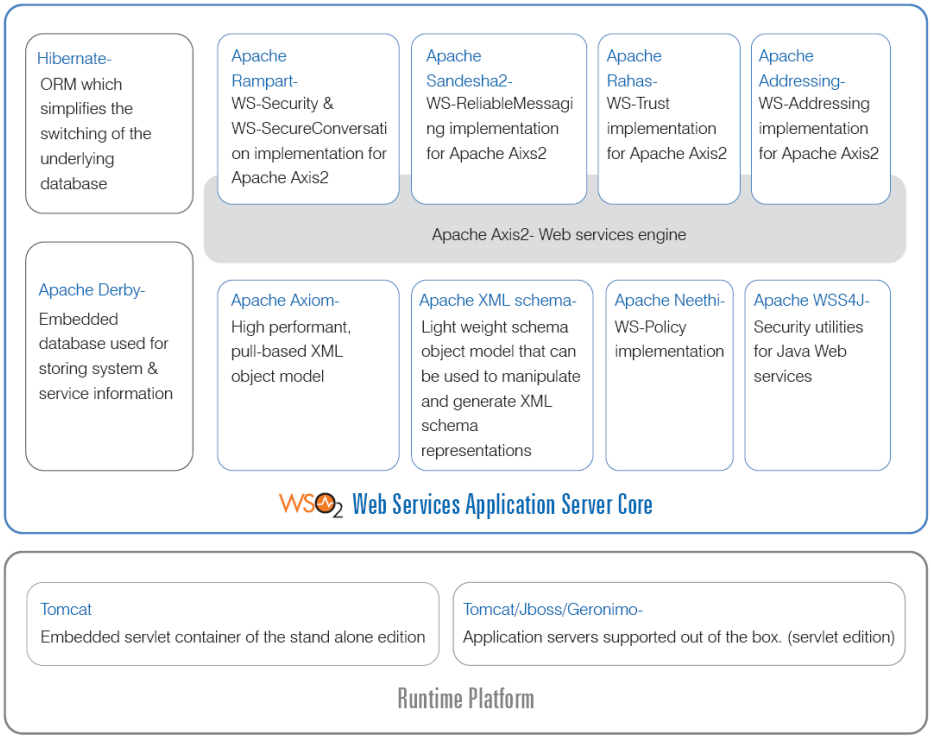
3.3.3.3 UDDI

UDDI pomeni Universal Description, Discovery and Integration. To je imenik spletnih storitev. Neke vrste Google za iskanje spletnih storitev. Hrani podatke o ponudniku storitev, kako je implementirana, kje se nahaja, namen, kateremu služi ter kako se povežemo s spletnimi storitvami.

3.3.4 WSO2 WSAS strežnik za spletne storitve

WSO2 WSAS je Java aplikacijski strežnik narejen za spletne storitve. Je odprtokodni projekt, distribuiran pod licenco Apache Software License. WSAS vsebuje orodja za vsako nalogo, ki mora biti izvršena s spletno storitvijo, vključujoč njeno kreiranje, razvijanje, upravljanje itd.

Ponuja posebno učinkovita orodja, ki omogočajo mnoge funkcionalnosti v modernih spletnih storitvah kot so na primer tudi različni WS-* protokoli, specifikacije itd. WSAS je relativno nov produkt, saj je bila prva verzija, pod imenom WSO2 Tungsten, izdana maja 2006. Zgrajen je na že dokazani odprtokodni tehnologiji, ki vključuje Axis2 in Apache Tomcat. Jedro strežnika je prikazano na sliki 13.



Slika 13: Jedro WSO2 WSAS aplikacijskega strežnika

Z vgrajenim visoko zmogljivim spletnim strežnikom, polno podporo XML, SOAP, WSDL in podporo za zanesljive in varne komunikacije WSO2 Web Services Application Server nudi preprosto in učinkovito ustvarjanje, uporabo in upravljanje spletnih storitev.

WSO2 WSAS je trenutno na voljo kot samostojna aplikacija kot tudi v verziji za gostovanje v Tomcat in Java EE aplikacijskih strežnikih.

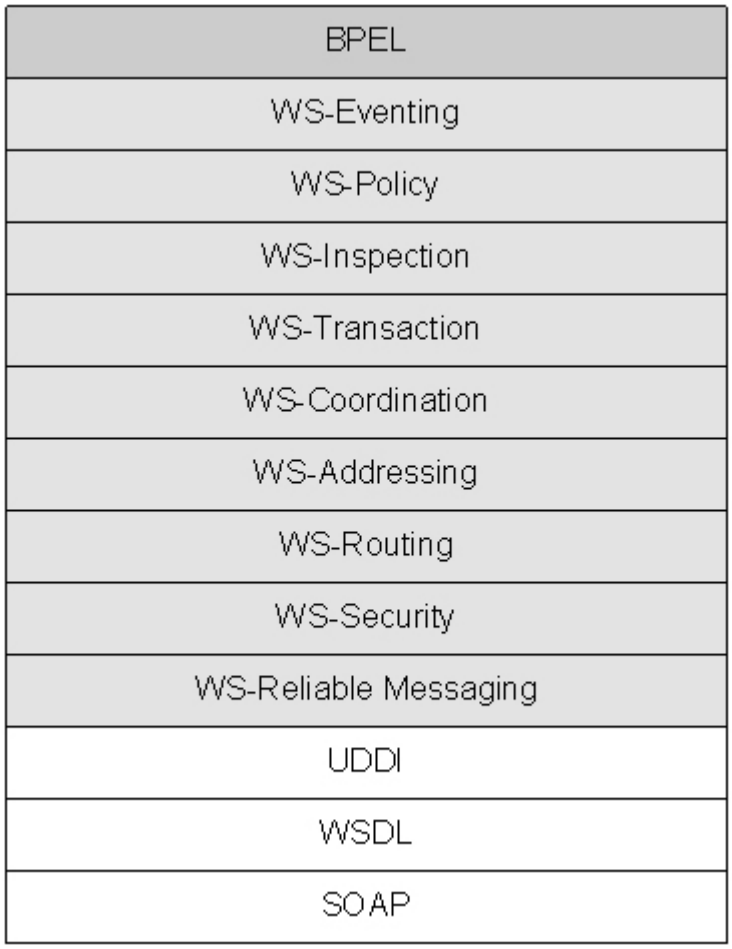
Za spletne storitve smo se odločili, ker predstavljajo dodatno funkcionalnost, ki so za MIDlet-e dosegljive preko uporabe KSoap2 knjižnice. MIDlet-u, kot zmogljivostno zelo omejenem programu, predstavlja klic in uporaba spletne storitve dragoceno širitev njegove funkcionalnosti.

3.4 Varne spletne storitve

3.4.1 Sklad tehnologij spletnih storitev [2]

Ključni cilj storitvenih arhitektur je zagotoviti integracijo aplikacij z namenom doseganja čim višje stopnje avtomatizacije poslovanja. Storitvena arhitektura predstavlja osnovo, da podjetja in druge organizacije avtomatizirajo poslovanje do te mere, da dokumenti v papirni obliki ne bodo več potrebni. To enostavno dejstvo pa zahteva korenite spremembe. Dokler dokumenti namreč obstajajo v fizični obliki, so ljudje (zaposleni) tisti, ki upravljajo tok dokumentacije – z drugimi besedami, zaposleni upravljajo oziroma nadzirajo poslovne procese.

Kakor hitro pa dokumenti v papirni obliki niso več potrebni, pa postane informacijski sistem tisti, ki upravlja s poslovnim procesom. Tedaj postane očitno, da potrebujemo namesto klasičnih aplikacij, ki so naslavljale določeno problemsko področje, integriran informacijski sistem, ki direktno podpira poslovne procese in ki ga lahko razmeroma enostavno in hitro prilagodimo spremembam ter dopolnitvam v poslovanju. Podporo celotnim poslovnim procesom pa lahko dosežemo le z integracijo posameznih aplikacij na nivoju storitev. Vizija spletnih storitev in storitvene arhitekture temelji na izpostavitvi funkcionalnosti v obliki storitev in njihovo kompozicijo v agregirane storitve ter orkestracijo le-teh na nivoju poslovnih procesov. V ta namen pa je potrebno zagotoviti ustrezne tehnologije, ki bodo omogočale dosego zastavljenih ciljev na standardiziran, deklarativni način.



Slika 14: Sklad tehnologij spletnih storitev

Slika 14 prikazuje sklad tehnologij spletnih storitev, ki temelji na znanem protokolu SOAP, jeziku za opis spletnih storitev WSDL in registru UDDI. Nad temi pa igrajo pomembno vlogo še naslednje tehnologije: WS-Reliable Messaging, WS-Security, WS-Routing, WS-Addressing, WS-Coordination, WS-Transaction, WS-Inspection, WS-Policy in WS-Eventing. Vse omenjene tehnologije predstavljajo osnovo za orkestracijo in koreografijo spletnih storitev v smislu podpore poslovnih procesov. To vlogo v naboru tehnologij spletnih storitev odigra jezik BPEL (Business Process Execution Language for Web Services, tudi BPEL4WS).

3.4.2 Web Service Security [2,5]

Standard WS-Security obravnava področje varovanja sporočil SOAP, izmenjanih med spletnimi storitvami. Zaradi svoje narave so sporočila SOAP enostavno čitljiva za človeka, enostavno pa je tudi njihovo spreminjanje. WS-Security tako naslavlja področja zagotavljanja integritete, zaupnosti, avtentikacije in šifriranja sporočil ter zagotavlja varnost od točke oblikovanja SOAP sporočila do končnega prejemnika. V ta namen uporablja WS-Security tehnologijo digitalnega podpisa XML dokumentov, šifriranje XML dokumentov ter časovno žigosanje. Omogoča navezavo sporočil na ustrezne varnostne žetone. Le-ti so lahko binarni, certifikati X.509, Kerberos listek, ipd. Ponuja tudi delno šifriranje oziroma šifriranje delov dokumenta z različnimi ključi.

Namen standarda WS-Security je zagotoviti mehanizme, s pomočjo katerih je mogoče izvesti varno izmenjavo sporočil SOAP. Standard ne podaja novih šifrirnih postopkov in njihove uporabe, temveč predstavlja ogrodje, ki omogoča vzpostavitev varnosti z vsemi znanimi in morebitnimi novimi, še nerazvitimi varnostnimi protokoli. Tako omogoča uporabo: infrastrukture javnih ključev (Public Key Infrastructure - PKI), Kerberos in SSL (Secure Socket Layer) varnostnih modelov. Z WS-Security razširimo protokol SOAP z varnostnimi elementi. Dodane so možnosti za: pošiljanje varnostnih žetonov (Security Tokens) kot dela sporočil, zagotovitev integritete sporočil (Message Integrity) in zagotovitev zaupnosti sporočil. Samo WS-Security ne zagotavlja polne varnostne rešitve za spletne storitve, ampak gre za gradnik, ki ga uporabljajo nadaljnje razširitve spletnih storitev in višje nivojski programsko specifični protokoli, kot so: WS-Policy, WS-SecurityPolicy, WS-Trust, WS-Addressing, WS-SecureConversation, SAML, XrML in drugi.

3.4.2.1 Varnostni žetoni

Za naslovnika oziroma ponudnika storitev je zelo pomembno, da ve, kdo je pošiljatelj sporočila – poznati mora njegovo identiteto. Prejemnik ne more biti prepričan glede pošiljateljeve identitete, če je ne more na nek način preveriti, saj bi se lahko pošiljatelj predstavil za nekoga drugega, o čemer govori načelo nezatajljivosti. Tako je najpomembnejši namen varnostnih žetonov ta, da prejemnik sporočila lahko preveri istovetnost pošiljatelja. Varnostni žeton je lahko sestavljen npr. iz uporabniškega imena in gesla, Kerberos listka (Ticket), certifikata X.509 ali česa drugega. V primeru uporabniškega imena je istovetnost potrjena z geslom, pri Kerberos listku prejemnik preveri veljavnost s ključem izdajatelja, pri certifikatu lahko podamo referenco na izdajatelja (CA – Certificate Authority).

WS-Security ne določa, kako izvesti preverjanje istovetnosti, temveč definira, kako je mogoče prejemniku prenesti nabor različnih varnostnih žetonov, ki jih lahko nato uporabi kakor hoče. Čeprav je mogoče uporabiti poljuben varnostni žeton, se največkrat uporabljajo že omenjene vrste varnostnih žetonov. Žeton z uporabniškim imenom in geslom je najenostavnejša izbira in je definiran z elementom <UsernameToken>. Primer je prikazan na sliki 15.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Envelope
xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
        <s:Header>
                <wsse:Security>
                        <wsse:UsernameToken>
                                <wsse:Username>Peter</wsse:Username>
                                <wsse:Password>mY5ecRet</wsse:Password>
                        </wsse:UsernameToken>
                        ...
                </wsse:Security>
        </s:Header>
        <s:Body>
        •••
        </s:Body>
</s:Envelope>
```

Slika 15: Varnostni žeton z uporabniškim imenom in geslom

Kot vidimo, sta poleg XML sheme za SOAP ovojnico podani še shemi secext in utility, določeni s WS-Security. Glavni element v shemi secext (wsse) je <Security>. Takšnega dokumenta (z geslom) v praksi seveda ne smemo poslati v omrežje nezavarovanega, zato moramo uporabiti SSL ali IPSec.

Druga specificirana izbira za posredovanje identitete je varnostni žeton, v katerem je Kerberos listek. Tudi v tem primeru se žeton pošlje v glavi SOAP sporočila znotraj elementa <Security> - glej slika 16.

```
...
<wsse:Security>
        <wsse:BinarySecurityToken
        ValueType="wsse:Kerberosv5ST" EncodingType="wsse:Base64Binary">
        QMwcAG ...
        </wsse:BinarySecurityToken>
</wsse:Security>
...
```

Slika 16: Varnostni žeton s Kerberos listkom

Vidimo, da se Kerberos listek prenaša z elementom <BinarySecurityToken>, da gre za Kerberos, je določeno z atributom ValueType, določen je še atribut EncodingType, ki podaja kodirno shemo, vsebina pa je listek sam (podano je le nekaj prvih znakov).

Tretja, zaenkrat zadnja specificirana izbira za varnostni žeton, je certifikat X.509. Podana je zelo podobno kot predhodna, razlikuje se le v atributu ValueType (in seveda vsebini) – glej slika 17.

```
...
<wsse:Security>
        <wsse:BinarySecurityToken ValueType="wsse:X509v3" EncodingType="wsse:Base64Binary">
        KkFPle ...
        </wsse:BinarySecurityToken>
</wsse:Security>
...
```

Slika 17: Varnostni žeton s certifikatom X.509

Poleg teh treh vrst neposrednega vstavljanja varnostnih žetonov v samo sporočilo, WSSecurity podaja tudi <SecurityTokenReference> element, s katerim lahko povemo, kje se nahaja varnostni žeton. To je lahko še vedno kje znotraj SOAP sporočila, lahko pa tudi kjerkoli v medmrežju. Ta element lahko vsebuje več različnih vrst sklicev. Najbolj specifična je neposredna – z elementom <Reference>, ki jo podamo z atributom URI, ki kot ime pove, podaja URI (Uniform Resource Identifier), na katerem se nahaja varnostni žeton. Naslednja definirana vrsta sklica na žeton je oznaka ključa (Key Identifier), ki je podana z zakrito vrednostjo (Opaque Value). Z njo lahko nedvoumno identificiramo žeton, in je običajno predstavljena z zgoščeno vrednostjo (Hash) ustreznih delov žetona. Ustrezen element v sporočilu je <KeyIdentifier>. Naslednji način sklicevanja je z oznako ključa (Key Name). To omogoča sklicevanje žetona z nizom, ki ustreza imenu identitetne trditve (Identity Assertion) vsebovane v žetonu. Takšnemu sklicu lahko ustreza več žetonov. Zadnja vrsta sklicevanja je izvedena z vstavljeno referenco (Embedded Reference), ki omogoča vstavljanje žetonov, v nasprotju s kazalci na žetone, ki se nahajajo nekje drugje. Podamo jo z elementom <Embedded>. Primer takšnega sklica je npr. vstavljena SAML trditev (SAML Assertion).

3.4.2.2 Celovitost sporočil

Z zagotovitvijo celovitosti (Integrity) sporočil dosežemo, da dobi prejemnik točno takšno sporočilo, kakršno je pošiljatelj poslal. V ta namen enako kot v drugih podobnih primerih uporabljamo elektronski podpis (Digital Signature). Ker gre v bistvu za XML dokument, je najprimernejša oblika že standardizirana XML Signature tehnologija, ki je le rahlo dopolnjena za uporabo v SOAP sporočilih.

Primer uporabe XML Signature v SOAP sporočilu je na sliki 18.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Envelope
        xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
        xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
        <s:Header>
                <wsse:Security>
                        <wsse:BinarySecurityToken ValueType="wsse:X509v3"
                        EncodingType="wsse:Base64Binary" wsu:Id="X509Cert">
                        KkFPle ...
                        </wsse:BinarySecurityToken>
                <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                <ds:SignedInfo>
                <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14N"/>
                <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                <ds:Reference URI="#TeloSporocila">
                        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <ds:DigestValue>aOb4Luuk...</ds:DigestValue>
                </ds:Reference>
                </ds:SignedInfo>
                <ds:SignatureValue>A9qqIrtE3xZ...</ds:SignatureValue>
                <ds:KeyInfo>
                        <wsse:SecurityTokenReference>
                                <wsse:Reference URI="#X509Cert"/>
                        </wsse:SecurityTokenReference>
                </ds:KeyInfo>
                </ds:Signature>
                </wsse:Security>
        </s:Header>
        <s:Body wsu:Id="TeloSporocila">•••</s:Body>
</s:Envelope>
```

Slika 18: Uporaba XML Signature s SOAP sporočilu

Na začetku imamo varnostni žeton s certifikatom X.509, ki je predstavljen v elementu <BinarySecurityToken>, ki poleg ostalega vsebuje tudi atribut Id (iz sheme utility), s katerim ga označimo, da ga je mogoče sklicevati. Sledi mu element <Signature> in njegovi podelementi, s katerimi uporabimo XML Signature elektronski podpis. Podana je metoda za standardizacijo (CanonicalizationMethod), metoda za podpis (SignatureMethod), s katero šifriramo izvleček (Digest). Nato imamo element <Reference>, s katerim podamo referenco na vsebino, ki jo podpisujemo. Za tem imamo metodo, s katero iz vsebine naredimo izvleček. V našem primeru je izvleček narejen iz telesa sporočila (kot vsebina elementa <DigestValue>). Sledi mu šifrirana vrednost tega izvlečka (kot vsebina elementa <SignatureValue>) in predstavlja bistvo podpisa. Naslednji element je <KeyInfo>, v katerem podamo ključ za dešifriranje vsebine v elementu <SignatureValue>. V našem primeru je podan neposredni sklic na varnostni žeton, ki je v našem sporočilu takoj na začetku elementa <Security>. WS-Security omogoča podajanje več podpisov, v okviru istega sporočila, ki so lahko namenjeni različnim vmesnim prejemnikom na poti sporočila do končnega prejemnika. Z WS-Security lahko elektronsko podpišemo tudi varnostne žetone same.

3.4.2.3 Šifriranje sporočil

Pogosto nam ni dovolj, da ne more nihče spremeniti sporočila med njegovo potjo, ne da bi to opazili, ampak tudi zahtevamo, da ga nihče nepooblaščen ne more razumeti, hočemo torej, da je tajno. Tako kot pri celovitosti, tudi tukaj uporabljamo že obstoječo varnostno tehnologijo; to je XML Encryption. To ni posebej zapleten standard, delno uporablja že znan XML Signature, v WS-Security uporabljamo le elemente <EncryptedData>, <EncryptedKey> in <ReferenceList>. Od teh je najpomembnejši <EncryptedData>, ki vsebuje <CipherData> in lahko nadalje vsebuje tudi podatek o uporabljenem šifrirnem algoritmu, uporabljenem ključu in seveda šifrirane podatke.

3.4.2.4 Časovni žig

Za prejemnika SOAP sporočila je pogosto pomemben podatek o času nastanka sporočila. V primeru, da je starejše od nekega intervala, se ob procesiranju zavrže, saj ga lahko nepooblaščena oseba uporabi za ponovitveni napad (Replay Attack). Sama sinhronizacija časa, med vpletenimi stranmi ni del specifikacije. Priporočeno je, da so časovne znamke izražene s tipom XML Schema dateTime in izražene v univerzalnem času (UTC Time), ki naj ne bo natančnejši od tisočinke sekunde. Če že uporabimo drugačen časovni format, ga moramo podati z ValueType atributom. Na sliki 19 je podan primer časovne znamke.

```
<S:Envelope xmlns:S="..." xmlns:wsse="..." xmlns:wsu="...">
        <S:Header>
                <wsse:Security>
                        <wsu:Timestamp wsu:Id="CasovnaZnamka">
                                <wsu:Created>2008-09-13T08:42:00Z</wsu:Created>
                                <wsu:Expires>2008-10-13T09:00:00Z</wsu:Expires>
                        </wsu:Timestamp>
                        ...
                </wsse:Security>
                ...
        </S:Header>
        <S:Body>
        ...
        </S:Body>
</S:Envelope>
```

Slika 19: Primer časovne znamke

Kot vidimo je časovna znamka podana z elementom <Timestamp>. Element <Created> kaže čas, ko je bilo sporočilo pripravljeno za prenos (Serialized for Transmission), element <Expires> pove, kdaj varnostni elementi sporočila prenehajo veljati. Po tem času je najbolje zavreči celotno sporočilo.

3.4.2.5 Varnostna tveganja

Elektronski podpis ne jamči za varovanje sporočil pred napadom ponovnega pošiljanja (Replay Attack). Tretja oseba lahko posname podpisano sporočilo in ga pozneje ponovno pošlje. Zato je priporočljivo, da se v sporočilo vključi elektronsko podpisane elemente, ki prejemniku omogočajo zaznavanje tega napada. Najbolj znani so: uporaba časovnih znamk, zaporedna številka, elementi, ki prenehajo veljati, in korelacija sporočil.

3.4.2.6 Sklep

Web Services Security (WS-Security) ne vpeljuje novih varnostnih tehnologij, ampak samo tvori ogrodje, s katerimi omogoča uporabo obstoječih tehnologij. Ker so SOAP sporočila v bistvu XML dokumenti, sta za njihovo varnost najpomembnejša standarda XML Signature, s katerim zagotavljamo integriteto ter XML Encryption, s katerim zagotavljamo zaupnost sporočil. Tretji pomembni del WS-Security so varnostni žetoni, s katerimi prejemniki sporočil preverijo identiteto pošiljatelja. SOAP sporočila pogosto potujejo preko več posrednikov (ki sami procesirajo del sporočila), zato imamo lahko več varnostnih žetonov ter XML Signature in XML Encryption elementov.

3.4.3 WS Policy

Pomembna lastnost spletnih storitev in storitvenih arhitektur v splošnem je zmožnost kompozicije storitev. Pri kompoziciji pa mnogokrat agregirana storitev želi postaviti določene omejitve na vpletene storitve kot npr. podpora ustreznim varnostnim standardom, oblikam vmesnikov, tipu komunikacije, ipd. Način definicije takih omejitev specificira standard WS-Policy, ki poleg tega določa mehanizme specifikacije pravil in načine preverjanja podpore določenih pravil. Slika 20 prikazuje primer uporabe standarda WS-Policy.

```
<wsp:Policy Id="SigOnly">
 - <wsp:ExactlyOne>
   - <wsp:All>
     - <sp:AsymmetricBinding>
       - <wsp:Policy>
         - <sp:InitiatorToken>
           - <wsp:Policy>
             - <sp:X509Token sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRecipient">
               - <wsp:Policy>
                   <sp:WssX509V3Token10/>
                 </wsp:Policy>
               </sp:X509Token>
             </wsp:Policy>
           </sp:InitiatorToken>
         - <sp:RecipientToken>
           - <wsp:Policy>
             - <sp:X509Token sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/Never">
               - <wsp:Policy>
                   <sp:WssX509V3Token10/>
                 </wsp:Policy>
               </sp:X509Token>
             </wsp:Policy>
           </sp:RecipientToken>
         - <sp:AlgorithmSuite>
           - <wsp:Policy>
               <sp:Basic256/>
             </wsp:Policy>
           </sp:AlgorithmSuite>
         - <sp:Layout>
           - <wsp:Policy>
               <sp:Strict/>
             </wsp:Policy>
           </sp:Layout>
           <sp:IncludeTimestamp/>
           <sp:OnlySignEntireHeadersAndBody/>
         </wsp:Policy>
       </sp:AsymmetricBinding>
     - <sp:Wss10>
       - <sp:Policy>
           <sp:MustSupportRefKeyIdentifier/>
           <sp:MustSupportRefIssuerSerial/>
         </sp:Policy>
       </sp:Wss10>
     - <sp:SignedParts>
         <sp:Body/>
       </sp:SignedParts>
     </wsp:All>
   </wsp:ExactlyOne>
</wsp:Policy>
```

Slika 20: Primer uporabe standarda WS-Policy

Aplikacija za iskanje prostih parkirnih mest mora vsebovati tudi obliko plačilne transakcije (rezervacija in plačilo parkirnega mesta), ki jo izvedemo preko spletne storitve. To pa pomeni, da mora biti klic spletne storitve varen. Varen klic pomeni, da se mora natančno vedeti, kdo kliče oziroma uporablja spletno storitev, torej da se točno identificira in avtenticira plačnika storitve rezervacije in plačila. Obenem pa je tudi potrebno zagotoviti integriteto klica spletne storitve, kar pomeni da se podatki (kdo, kaj in za koliko časa se rezervira) med pošiljanjem ne spreminjajo. To pa nam omogoča protokol WS-Security s pomočjo standarda WS-Policy.

3.5 KRIPTOGRAFIJA

3.5.1 Uvod v kriptografijo

Kriptografija ali skrivnopisje je znanstvena veda o tajnem, nerazumljivem pisanju sporočil in njihovemu prebiranju, ko je poznan spremembni postopek. Enkripcija (šifriranje) je transformacija podatkov v obliko, ki je praktično nemogoča za branje brez pravega znanja (ključa). Njen namen je zagotavljanje tajnosti podatkov pred tistimi, katerim ti podatki niso namenjeni, tudi če imajo šifrirano sporočilo. Dekripcija (dešifriranje) je nasprotna operacija šifriranju – to je transformacija šifriranih podatkov nazaj v razumljivo obliko. Šifriranje in dešifriranje zahtevata uporabo neke tajne informacije, to je ključa. Nekateri algoritmi uporabljajo en ključ tako za šifriranje kot za dešifriranje, pri drugih pa je ključ za šifriranje različen od ključa za dešifriranje.

Obstajata dva tipa kriptografije, simetrična in asimetrična. Pri simetrični kriptografiji se uporablja isti ključ tako za šifriranje kot za dešifriranje. Pri teh sistemih je pomembna varna distribucija ključa. Danes je najbolj znan algoritem DES (Data Encryption Standard). Pri asimetrični kriptografiji ima vsak uporabnik dva ključa, privatnega in javnega. Javni ključ je javen, do njega ima vsak dostop in se uporablja za šifriranje. Privatni ključ pa mora ostati tajen in se uporablja za dešifriranje. Asimetrična kriptografija odpravlja problem varne distribucije ključev pri simetrični kriptografiji. Danes se največ uporablja algoritem RSA, ki je primeren tako za podpisovanje kot za šifriranje.

3.5.2 Simetrična kriptografija

Glavna značilnost simetrične kriptografije je uporaba enakega ključa za šifriranje in dešifriranje. Primer iz fizičnega sveta bi bil uporaba ključavnic, kjer za zaklepanje in odklepanje uporabljamo identične ključe. Ključ simetrične kriptografije imenujemo tudi »skupna skrivnost« (shared secret). Prednost simetrične kriptografije je v hitrosti algoritmov in preprostosti uporabe. Največji problem simetrične kriptografije pa je v varni distribuciji ključev. Simetrični ključ mora vedno ostati skrit, poleg tega ga moramo periodično zamenjevati, saj obstajajo metode kriptoanalitičnih napadov, kjer lahko z analizo dovolj velike količine šifriranega materiala dobimo ključ.

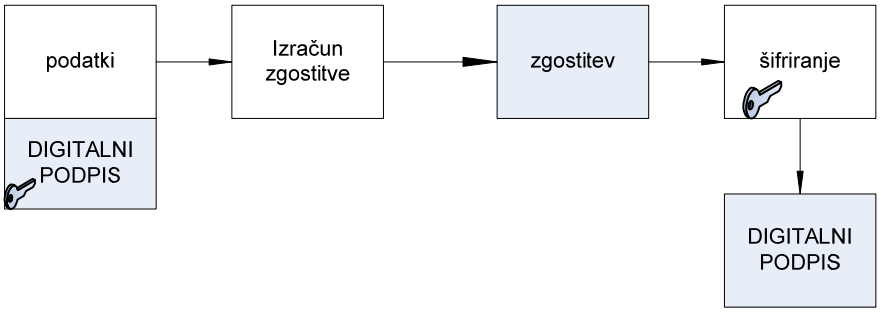
3.5.3 Asimetrična kriptografija

Pri asimetrični kriptografiji uporabljamo dva ključa, ki sta med seboj matematično povezana, nista pa enaka. Prav zaradi matematične povezanosti ključev so možni napadi na asimetrično kriptografijo, saj je vedno možno iz javnega ključa dobiti privatni ključ. Obramba pred temi napadi je, da naredimo pridobivanje privatnega ključa iz javnega ključa kar se da težko. Nekateri algoritmi so zasnovani na matematičnem problemu razbijanja faktorjev velikih praštevil. Vse šifriramo s privatnim ključem, dešifriramo pa z javnim. Javni ključ lahko objavimo, privatni ključ pa mora ostati skrit. S tem je odpravljena potreba po »skupni skrivnosti« in varni distribuciji ključev. Vsakdo, ki nam hoče poslati šifrirano sporočilo, ga šifrira z našim javnim ključem. To sporočilo je možno dešifrirati samo z našim privatnim ključem. Asimetrična kriptografija se uporablja tako za šifriranje kot za digitalni podpis.

Edina slabost asimetričnih algoritmov je v njihovi počasnosti v primerjavi s simetričnimi algoritmi.

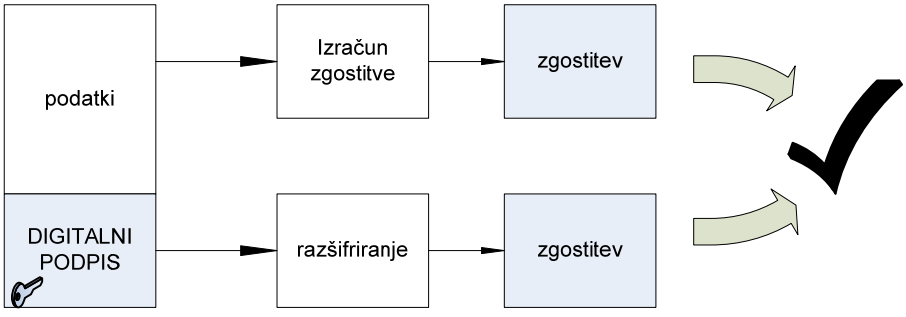
Poglejmo si, kako poteka šifriranje. Ana želi Borutu poslati skrivno sporočilo. Poiskala bo njegov javni ključ, z njim šifrirala sporočilo in mu ga poslala. Borut bo to sporočilo dešifriral s svojim privatnim ključem in ga prebral. Šifrirano sporočilo lahko dobi vsakdo, toda le Borut ga bo lahko dešifriral in prebral, saj ima edino on ustrezen privatni ključ, potreben za dešifriranje sporočila. Vsakdo mu lahko pošlje šifrirano sporočilo, toda le on ga bo lahko dešifriral.

Omenili smo že, da se asimetrična kriptografija uporablja tudi za digitalni podpis. Pa si poglejmo na kratko, kako poteka podpisovanje sporočila. Ana podpiše izvorno sporočilo s svojim privatnim ključem. Elektronski podpis skupaj s sporočilom združi v podpisano sporočilo in ga pošlje Borutu (slika 21).



Slika 21: Digitalno podpisovanje

Borut s pomočjo Aninega javnega ključa preveri veljavnost podpisa, s čimer potrdi njegovo celovitost. Najprej izračuna zgostitev sporočila, nato s pomočjo Aninega javnega ključa dešifrira originalno zgoščeno vrednost, nakar obe vrednosti primerja. Če se ujemata, sporočilo ni bilo spremenjeno (slika 22).



Slika 22: Preverjanje digitalnega podpisa

Digitalni podpis se uporablja za zagotavljanje integritete sporočila. Tako kot tradicionalni podpis na papirju, se tudi digitalni podpis uporablja za avtentikacijo podpisnika. Poleg tega digitalni podpis zagotavlja, da se podatki po podpisu niso spremenili. Digitalni podpis je

šifrirani izvleček sporočila. Potrebujemo ustrezno zgoščevalno funkcijo in asimetrični postopek šifriranja. Zelo pogosto je uporabljena naslednja kombinacija:

- zgoščevalna funkcija SHA-1
- asimetrični šifrirni algoritem RSA

Pošiljatelj najprej izračuna zgostitev ter jo šifrira s svojim privatnim ključem. Dobljeno vrednost skupaj s svojim javnim ključem pripne k sporočilu kot digitalni podpis. Prejemnik digitalni podpis preveri tako, da najprej izračuna zgostitev sporočila. Nato s pomočjo pošiljateljevega javnega ključa dešifrira originalno zgoščeno vrednost in obe vrednosti primerja. Če se ujemata, je sporočilo verodostojno in potrjena je identiteta pošiljatelja. V tem primeru pošiljatelj ne more zanikati sporočila.

3.5.4 Digitalni podpis dokumentov XML

Osnovna oblika interakcije v medposlovni komunikaciji je izmenjava sporočil med udeleženci v poslovnem procesu. Pri tem so sporočila pogosto v obliki XML. Pri medposlovni izmenjavi podatkov v XML sta bistvenega pomena neokrnjenost podatkov in nezmožnost zanikanja, kar zagotavlja digitalni podpis. V tem poglavju bom predstavil XML podpis, standarde, oblike in tudi težave, ki se pojavljajo pri XML podpisu.

3.5.4.1 Razširljivi označevalni jezik XML

XML je razširljivi označevalni jezik, ki omogoča zapisovanje strukturiranih dokumentov. Uveljavlja se kot dejanski sintaktični standard pri različnih načinih prenosa podatkov v medposlovnem povezovanju. Dokumenti XML lahko predstavljajo vsebino poslovnih dokumentov. V obliki XML je lahko tudi vsebina, namenjena nastavitvam programov (npr. inicializacijske datoteke, stanje objektov ipd.). XML pa je tudi temeljna oblika izmenjave sporočil pri spletnih storitvah (npr. sporočila XML v ovojnici SOAP). XML je zelo uporaben pri izmenjavi podatkov med različnimi aplikacijami ter med različnimi sistemi, povezanimi v internet. Odlike jezika XML so preprostost, razširljivost, povezljivost in odprtost.

3.5.4.2 Kanonska oblika XML

Standarda »XML 1.0« in »Imenski prostori v XML« dopuščata sintaktične ohlapnosti, zaradi katerih imamo težave pri digitalnem podpisovanju dokumentov XML. Standard XML med drugim dopušča poljubni vrstni red atributov, poljubno število presledkov v definiciji elementa, različne načine opredelitve imenskih prostorov, tudi navedbo privzete kodne strani (UTF-8) lahko izpustimo. S stališča XML sta naslednja dva dokumenta identična:

```
<knjiga avtor='Saadat Malik' isbn="1-58705-025-0" />
<knjiga   isbn='1-58705-025-0'    avtor="Saadat Malik"  ></knjiga>
```

Za vse dokumente XML, ki so logično ekvivalentni (in nosijo isto informacijo) pravimo, da imajo skupen »XML Information Set« [14]. XML Information Set (ali XML Infoset) je abstrakten podatkovni model dokumenta XML, ki opisuje vse njegove pomembne značilnosti, opusti pa vse nepomembne razlike v sintaksi, ki jih XML dopušča. XML Infoset je rezultat razčlenjevanja dokumenta XML, kjer se opustijo vse površinske lastnosti dokumenta, ki so nepomembne, kadar na XML gledamo skozi njegovo sintakso. Ko v namenskih programih

delamo z dokumenti XML, običajno uporabljamo razčlenjevalnik XML, npr. Microsoft XML Parser. Ob branju dokumenta XML razčlenjevalnik v spominu ustvari podatkovno strukturo enakovredno XML Infosetu, nad katero izvaja operacije. S tem pri branju opusti vse nepomembne površinske lastnosti, ki jih sintaksa XML 1.0 dopušča. Ko ta infoset shranimo nazaj v niz znakov ali v binarno zaporedje, se na določen način izvede serializacija podatkov v skladu s sintaktičnimi pravili XML in izbrano kodno stranjo. Težava je v tem, da različni razčlenjevalniki XML ali celo posamezne različice istega razčlenjevalnika dokument serializirajo na različne načine, na samo pretvorbo pa nimamo vpliva. Če so podatki pred preverjanjem podpisa bili serializirani na drugačen način kot pri podpisovanju (zaradi drugačnega razčlenjevalnika XML), bo digitalni podpis neveljaven, čeprav ni prišlo do spremembe podatkov v XML. Seveda takšno obnašanje digitalnega podpisa ni zaželeno. Kako lahko torej pravilno implementiramo digitalni podpis? Ena možnost je, da uporabimo isti razčlenjevalnik. Vendar lahko takoj vidimo, da takšna rešitev ni praktična, saj smo odvisni od razčlenjevalnika in s tem precej omejimo prenosljivost svoje programske rešitve. Druga možnost je, da obravnavamo dokument XML kot binarne podatke. Pri tem scenariju podpisnik podatke XML na poljuben način serializira in podpiše. Pri preverjanju podpisa dokument XML ne naložimo v razčlenjevalnik, ampak preverimo podpis kar na binarnih podatkih. Če je podpis pravilen, podatke naložimo v razčlenjevalnik in jih uporabimo. S takšno implementacijo omejimo veljavnost digitalnega podpisa le na binarne podatke. V preprostih primerih ta pristop zadošča. Slabost tega pristopa je, da lahko podpišemo le cel dokument (ker dokument XML obravnavamo kot celoto) – ne moremo podpisovati samo odlomkov XML ter zunanjih podatkov. Tretji, najbolj praktičen in najustreznejši scenarij je uporaba kanonizacije. Podpisovanje XML v kanonski obliki predpisuje tudi standard »XML Signature« [13]. Kanonska oblika in proces kanonizacije podatkov XML sta opredeljena v standardu »Kanonski XML« [6]. Kanonska oblika je vnaprej predpisana, normalizirana oblika dokumenta XML, serializacija v takšno obliko pa kanonizacija. Nekaj takšnih predpisanih značilnosti je:

- znaki so kodirani po UTF-8
- CDATA odseki se zamenjajo z ustreznimi entitetnimi oznakami (>, <, itd.)
- vrednosti atributov so zapisane v dvojnih narekovajih
- atributi in imenski prostori se v okviru elementa uredijo po abecednem vrstnem redu
- presledki izven elementa se normalizirajo
- prazni elementi se iz <knjiga /> pretvorijo v <knjiga></knjiga>

Omenjen dokument XML v kanonizirani obliki izgleda tako:

```
<knjiga avtor="Saadat Malik" isbn="1-58705-025-0"></knjiga>
```

Kanonska oblika XML nam omogoča, da vpeljemo ekvivalenco med dokumenti XML. Logično ekvivalenti dokumenti XML imajo v kanonski obliki do bajta enak binarni zapis. Dokument XML moramo kanonizirati tako pred podpisovanjem kot pred preverjanjem podpisa. Digitalni podpis bo veljaven, tudi če bo po podpisu prišlo do kakršnekoli sintaktične

spremembe dokumenta XML. Kanonska oblika dokumenta XML je gotovo najboljša izbira pri podpisovanju dokumentov XML.

3.5.4.3 XmlDSig, XadES

World Wide Web Consortium (W3C) je mednarodna organizacija, katere poslanstvo je razvoj tehnologij, ki omogočajo medsebojno povezljivost aplikacij v spletnem okolju. V W3C deluje tudi delovna skupina za XML podpis (XML Signature ali XMLDSIG), ki je pripravila priporočila za sintakso in procesiranje XML podpisa. V Evropski uniji pripravlja standarde in priporočila s področja telekomunikacij in informacijske tehnologije European Telecommunications Standards Institute (ETSI). ETSI je neprofitna organizacija, katere poslanstvo je priprava standardov in priporočil za področje Evrope, ki bodo služili kot smernice razvoja telekomunikacijskih storitev in informacijske tehnologije v prihodnje. V Sloveniji področje sintakse in procesiranja XML podpisa ni posebej urejeno, zato se na tem področju zgledujemo po mednarodnih standardih in priporočilih.

W3C XML-Signature priporočilo za digitalni podpis je razvito za uporabo pri izmenjavi XML dokumentov. Priporočilo določa sintakso in pravila procesiranja digitalnega podpisa v XML dokumentih. XML-Signature, podobno kot ostali standardi za shranjevanje podpisanih podatkov (npr. PKCS #/7) zagotavlja avtentifikacijo podpisnika, integriteto podatkov in podporo nezatajljivosti. Za razliko od ostalih standardov XML-Signature upošteva lastnosti in prednosti Interneta in XMLja. XML-Signature lahko uporabimo za podpis podatkov v XML obliki, podatkov v tekstovni obliki (npr. HTML), podatkov v binarni obliki (npr. JPG), ali pa le del podatkov v XML obliki. Podpisani podatki so lahko vključeni v XMLSignature strukturo preko sklica na URI, so del istega dokumenta kot XML-Signature, so vsebovani v XML-Signature strukturi ali pa je XML-Signature del podatkovne strukture.

Glede na lokacijo, kje so podatki, ki jih želimo podpisati, razlikujemo tri različne XML podpise:

- **ovit podpis:** celoten XML dokument se nahaja znotraj elementa Signature, ki je v tem primeru korenski element podpisa in našega dokumenta
- **vsebovan podpis**: dokument in podpis sta v istem dokumentu, le da je element Signature del XML dokumenta
- **zunanji podpis**: imamo samostojen XML dokument, podpis se nahaja v drugi datoteki

Struktura <ds:Signature> elementa je sledeča:

```
<element name="Signature" type="ds:SignatureType"/>
<complexType name="SignatureType">
  <sequence>
    <element ref="ds:SignedInfo"/>
    <element ref="ds:SignatureValue"/>
    <element ref="ds:KeyInfo" minOccurs="0"/>
    <element ref="ds:Object" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
</complexType>
```

<ds:Signature> element vsebuje naslednje elemente:

- <ds:SignedInfo> - podatki, ki smo jih podpisali
- <ds:SignatureValue> - dejanski biti digitalnega podpisa
- opcijski <ds:KeyInfo> - podatki o javnem ključu
- opcijski <ds:Object> - dodatne informacije o podpisu

<ds:SignedInfo> element vsebuje naslednje elemente:

- <ds:SignatureMethod> - kateri algoritem za zgoščevanje in podpisovanje je bil uporabljen
- <ds:CanonicalizationMethod> - kako kanonizirati dokument
- <ds:Reference> - ena ali več referenc na dejanske podatke

3.5.4.4 Podpisovanje odlomkov XML

Pomembna lastnost dokumentov XML je, da imajo urejeno strukturo in da imajo tudi posamezni deli dokumenta določen pomen. Zato pogosto želimo podpisati le del dokumenta XML. Npr. pri poslovnih dokumentih je posamezni podpisnik pogosto odgovoren le za del njegove vsebine. Zato je pravilno, da je tudi v XML podpisan le tisti odlomek XML, za katerega je posameznik odgovoren. Podpisani odlomek lahko vstavimo tudi v drugi XML. Npr. podpisane podatke pred transportom vstavimo v ovojnico SOAP.

Pri podpisovanju odlomkov XML je potrebno paziti, kako jih prenašamo med dokumenti. V standardu XML je namreč določeno, da posamezni element podeduje vse deklaracije imenskih prostorov od svojih prednikov (tudi privzeti imenski prostor), če jih sam ne opredeli drugače. Ko odlomek prenesemo v drug dokument XML, bo njegova definicija vsebovala poleg obstoječih tudi deklaracije podedovanih imenskih prostorov. Digitalni podpis, ustvarjen v izvornem dokumentu, bo zato neveljaven.

Standard XML Signature omogoča tudi podpisovanje delov dokumentov XML. Težavo z dedovanjem imenskih prostorov lahko najenostavneje rešimo tako, da se dedovanju imenskih prostorov odpovemo. Standard XML Signature svetuje uporabo takšnih kanonizacijskih metod, ki v teh primerih ne prenašajo konteksta starševskih vozlišč na otroke.

3.5.5 Knjižnica Bouncy Castle

Za ogrodje Java ME obstaja knjižnica Bouncy Castle, ki vsebuje niz lahkih kriptografskih API funkcij. Uporabili smo funkcije za zgoščevanje s pomočjo razreda *SHA1Digest* in podpisovanje s pomočjo razreda *RSADigestSigner*. Uporaba te knjižnice oziroma njenih funkcij je prikazana na sliki 40.

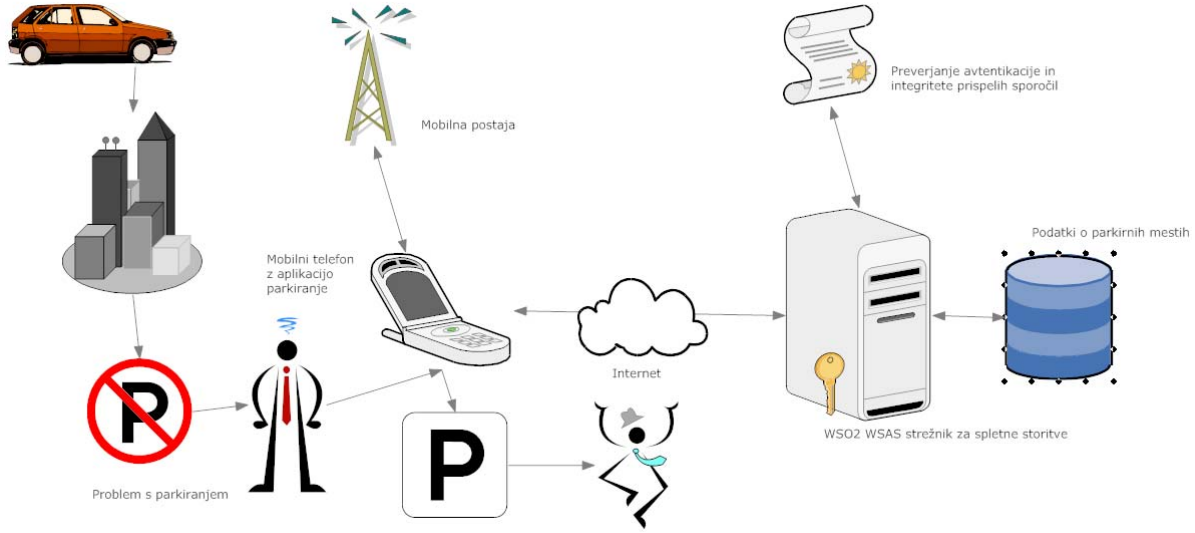
Za uspešno varno rezervacijo parkirnega mesta je potrebna uporaba protokola WS-Security, ki zagotavlja popolno varnost od začetka do konca rezervacije. Omenjeni protokol zagotavlja avtentikacijo in integriteto poslanega sporočila, zato smo pri izdelavi le-tega uporabili zahtevano asimetrično šifriranje, digitalno podpisovanje sporočila in zapis dela sporočila v

kanonsko obliko. Odločili smo se za uporabo knjižnice Bouncy Castle, ker nudi uporabo funkcij (zgoščevalna, podpisovanje sporočil), ki so nujno potrebne za pravilno uporabo WS-Security protokola.

4. PREDSTAVITEV PROGRAMA

V tem poglavju bomo predstavili praktični del diplome, sestavljene iz aplikacije za mobilno napravo Parkiranje, ki smo jo implementirali v okolju Java ME s pomočjo orodja Eclipse ter spletnih storitev, ki smo jih sprogramirali v okolju Java SE ter objavili na aplikacijskem strežniku za spletne storitve WSO2 WSAS. Celotna arhitektura sistema za iskanje prostih parkirnih mest je prikazan na sliki 23.

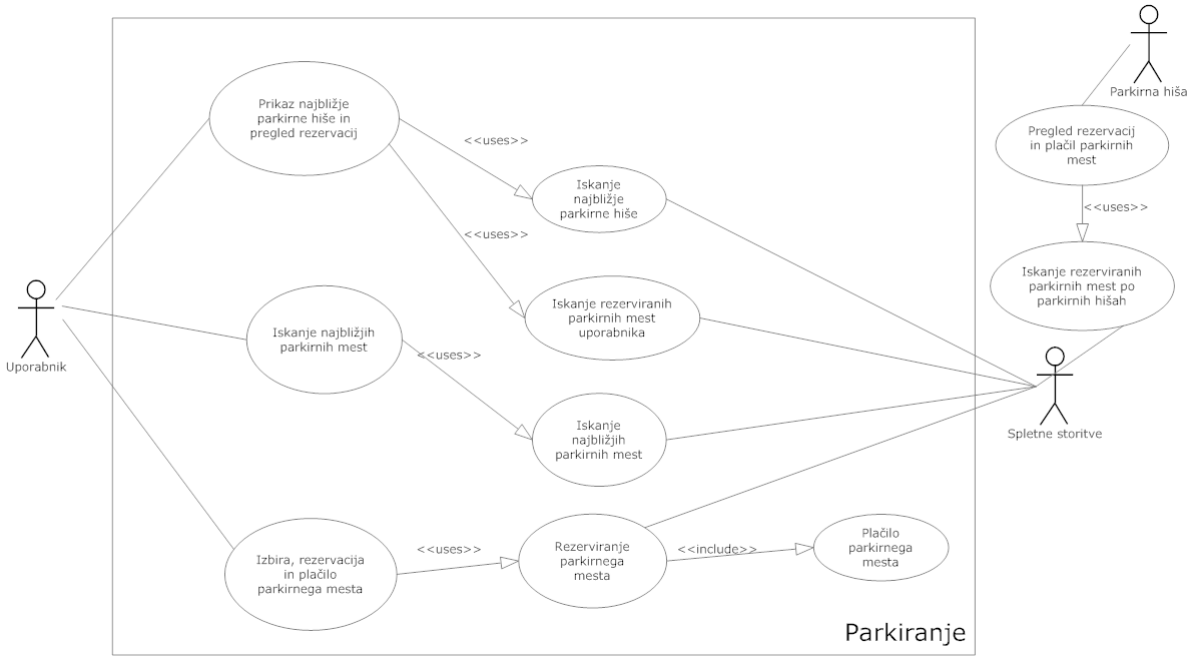
Aplikacija Parkiranje, ki predstavlja del praktičnega dela diplomske naloge, služi namenu iskanja najbližjih parkirnih mest in rezervacijo le-teh. Uporabnik s pomočjo mobilnega telefona, na katerem je naložena aplikacija, poišče najbližje parkirno mesto in ga rezervira za določen čas. Vsaka rezervacija se zabeleži in parkirna hiša preko spletne storitve poizve po uspešnih rezervacijah in na podlagi le-teh na koncu meseca izstavi račun uporabniku.



Slika 23: Aplikacija Parkiranje in strežnik za spletne storitve kot del življenjske situacije

4.1 Primeri uporabe aplikacije Parkiranje

Funkcionalnost aplikacije je moč videti preko primerov uporabe, ki so razvidni na sliki 24, njihov podrobnejši opis pa sledi na naslednjih straneh. Za vsak primer uporabe so opisani predpogoji za njegovo izvršitev in vsaj en osnovni tok dogodkov.



Slika 24: Diagram primera uporabe

4.1.1 Prikaz najbližje parkirne hiše in pregled rezervacij

Uporabniku se po zagonu aplikacije preko spletne storitve izpiše najbližja parkirna hiša in njegova rezervirana parkirna mesta oziroma časovni rok rezervacij. Aplikacija uporabniku v meniju preko ukaza *Išči* ponuja iskanje najbližjih parkirnih mest, preko ukaza *Osveži* ponovno iskanje najbližje parkirne hiše in osvežitev podatkov o njegovih rezervacijah ter preko ukaza *Pomoč* razlago ukaza Išči.

Predpogoj

1. Prenosni telefon, na katerem se izvaja aplikacija, mora biti povezan z internetom za dostop do spletnih storitev.

Osnovni tok dogodkov

1. Aplikacija na podlagi trenutne lokacije poišče najbližjo parkirno hišo. Za uporabnika aplikacije se izpišejo tudi vsa rezervirana parkirna mesta.
2. Uporabnik z ukazom *Išči* iz menija sproži iskanje najbližjih prostih parkirnih mest glede na trenutno lokacijo.
3. Uporabnik z ukazom *Osveži* iz menija na podlagi trenutne lokacije ponovno sproži iskanje najbližje parkirne hiše in podatke o njegovih rezerviranih parkirnih mestih.
4. Uporabniku je z ukazom *Pomoč* prikazan opis ukaza Išči.

Alternativni tok dogodkov

Prenosni telefon nima dostopa do interneta, zato aplikacija ob zagonu in pri ukazih, ki uporabljajo spletne storitve, izpiše sporočilo o napaki.

4.1.2 Iskanje najbližjih parkirnih mest

Aplikacija preko spletne storitve ponuja uporabniku iskanje prostih parkirnih mest. Uporabniku je omogočena rezervacija/plačilo izbranega najdenega prostega parkirnega mesta s pomočjo ukaza *Rezerviraj* v meniju. Preko ukaza *Osveži* v meniju aplikacija ponovno sproži iskanje najbližjih parkirnih mest ter preko ukaza *Pomoč* razlago ukaza Rezerviraj.

Predpogoj

1. Prenosni telefon, na katerem se izvaja aplikacija, mora biti povezan z internetom za dostop do spletnih storitev.

Osnovni tok dogodkov

1. Ko uporabnik iz glavnega menija izbere ukaz Išči, aplikacija na podlagi trenutne lokacije poišče najbližja prosta parkirna mesta.
2. Uporabnik izbere posamezno najdeno parkirno mesto.
3. Uporabnik izbrano najdeno parkirno mesto z izbiro ukaza Rezerviraj poskuša rezervirati.

Alternativni tok dogodkov 1

Prenosni telefon nima dostopa do interneta, zato aplikacija ob ukazu, ki uporablja spletno storitev, izpiše sporočilo o napaki.

Alternativni tok dogodkov 2

Uporabnik izbere ukaz Rezerviraj, čeprav ni bilo najdeno oziroma izbrano nobeno parkirno mesto. Aplikacija izpiše opozorilo, da rezervacija parkirnega mesta ni možna.

4.1.3 Izbira, rezervacija in plačilo parkirnega mesta

Aplikacija preko spletne storitve ponuja uporabniku rezervacijo in plačilo izbranega najdenega prostega parkirnega mesta. Uporabniku je omogočena dokončna rezervacija/plačilo izbranega najdenega prostega parkirnega mesta s pomočjo ukaza *Plačaj* v meniju. Preko ukaza *Pomoč* v meniju aplikacija prikaže razlago ukaza Plačaj.

Predpogoj

1. Prenosni telefon, na katerem se izvaja aplikacija, mora biti povezan z internetom za dostop do spletnih storitev.

Osnovni tok dogodkov

1. Ko uporabnik v obrazcu za iskanje prostih parkirnih mest v meniju izbere ukaz rezerviraj, aplikacija odpre obrazec za plačilo rezervacije.
2. Uporabnik izbere časovno omejitev rezervacije z vpisom število ur rezervacije.
3. Uporabnik izbrano najdeno parkirno mesto z izbiro ukaza Plačaj rezervira.
4. Aplikacija potrdi rezervacijo tako, da izpiše časovni rok, do katerega velja rezervacija izbranega parkirnega mesta.

Alternativni tok dogodkov 1

Prenosni telefon nima dostopa do interneta, zato aplikacija ob ukazu, ki uporablja spletno storitev, izpiše sporočilo o napaki.

Alternativni tok dogodkov 2

Uporabnik izbere ukaz Plačaj, vmes pa je bilo najdeno oziroma izbrano parkirno mesto že rezervirano s strani drugega uporabnika. Aplikacija izpiše opozorilo, da rezervacija parkirnega mesta ni možna.

4.1.4 Pregled rezervacij in plačil parkirnih mest

Lastniku parkirne hiše je omogočen pregled izvedenih transakcij rezervacij za vsa njegova parkirna mesta.

Predpogoj

1. Računalnik na katerem se izvaja aplikacija, mora biti povezan z internetom za dostop do spletnih storitev.

Osnovni tok dogodkov

Aplikacija prikaže vse izvedene rezervacije parkirnih mest v lasti parkirne hiše.

Alternativni tok dogodkov

Računalnik nima dostopa do interneta, zato aplikacija ob ukazu, ki uporablja spletno storitev, izpiše sporočilo o napaki.

4.1.5 Iskanje rezerviranih parkirnih mest po parkirnih hišah

Aplikacija ob zagonu in ob izbiri ukaza Osveži iz glavnega menija poizvede po rezerviranih parkirnih mest uporabnika.

Predpogoj

1. Prenosni telefon, na katerem se izvaja aplikacija, mora biti povezan z internetom za dostop do spletnih storitev.

Osnovni tok dogodkov

1. Uporabnik iz glavnega menija izbere ukaz Osveži.
2. Aplikacija izpiše rezervirana parkirna mesta.

Alternativni tok dogodkov

Prenosni telefon nima dostopa do interneta, zato aplikacija ob ukazu, ki uporablja spletno storitev, izpiše sporočilo o napaki.

4.1.6 Iskanje najbližje parkirne hiše

Aplikacija ob zagonu in ob izbiri ukaza Osveži iz glavnega menija poizvede po najbližji parkirni hiši.

Predpogoj

1. Prenosni telefon, na katerem se izvaja aplikacija, mora biti povezan z internetom za dostop do spletnih storitev.

Osnovni tok dogodkov

1. Uporabnik iz glavnega menija izbere ukaz Osveži.
2. Aplikacija izpiše najbližjo parkirno hišo.

Alternativni tok dogodkov

Prenosni telefon nima dostopa do interneta, zato aplikacija ob ukazu, ki uporablja spletno storitev, izpiše sporočilo o napaki.

4.1.7 Iskanje rezerviranih parkirnih mest uporabnika

Aplikacija ob zagonu in ob izbiri ukaza Osveži iz glavnega menija poizvede po rezerviranih parkirnih mest uporabnika.

Predpogoj

1. Prenosni telefon, na katerem se izvaja aplikacija, mora biti povezan z internetom za dostop do spletnih storitev.

Osnovni tok dogodkov

1. Uporabnik iz glavnega menija izbere ukaz Osveži.
2. Aplikacija izpiše rezervirana parkirna mesta.

Alternativni tok dogodkov

Prenosni telefon nima dostopa do interneta, zato aplikacija ob ukazu, ki uporablja spletno storitev, izpiše sporočilo o napaki.

4.1.8 Iskanje najbližjih parkirnih mest

Aplikacija ob zagonu in ob izbiri ukaza Išči iz glavnega menija poizvede po najbližjih prostih parkirnih mestih.

Predpogoj

1. Prenosni telefon, na katerem se izvaja aplikacija, mora biti povezan z internetom za dostop do spletnih storitev.

Osnovni tok dogodkov

1. Uporabnik iz glavnega menija izbere ukaz Išči.
2. Aplikacija odpre nov obrazec in izpiše najbližja prosta parkirna mesta.

Alternativni tok dogodkov

Prenosni telefon nima dostopa do interneta, zato aplikacija ob ukazu, ki uporablja spletno storitev, izpiše sporočilo o napaki.

4.1.9 Rezerviranje parkirnega mesta

Aplikacija preko spletne storitve uporabniku rezervira in naroči plačilo izbranega najdenega prostega parkirnega mesta. Uporabniku je omogočena dokončna rezervacija/plačilo izbranega najdenega prostega parkirnega mesta s pomočjo ukaza *Plačaj* v meniju.

Predpogoj

1. Prenosni telefon, na katerem se izvaja aplikacija, mora biti povezan z internetom za dostop do spletnih storitev.
2. Uporabnik mora imeti v aplikaciji naložen veljavni certifikat za avtentikacijo in preverjanje integritete poslane zahteve po naročilu rezervacije in plačila izbranega prostega parkirnega mesta.

Osnovni tok dogodkov

1. Uporabnik izbrano najdeno parkirno mesto z izbiro ukaza Plačaj rezervira.
2. Aplikacija potrdi rezervacijo tako, da izpiše časovni rok, do katerega velja rezervacija izbranega parkirnega mesta.

Alternativni tok dogodkov 1

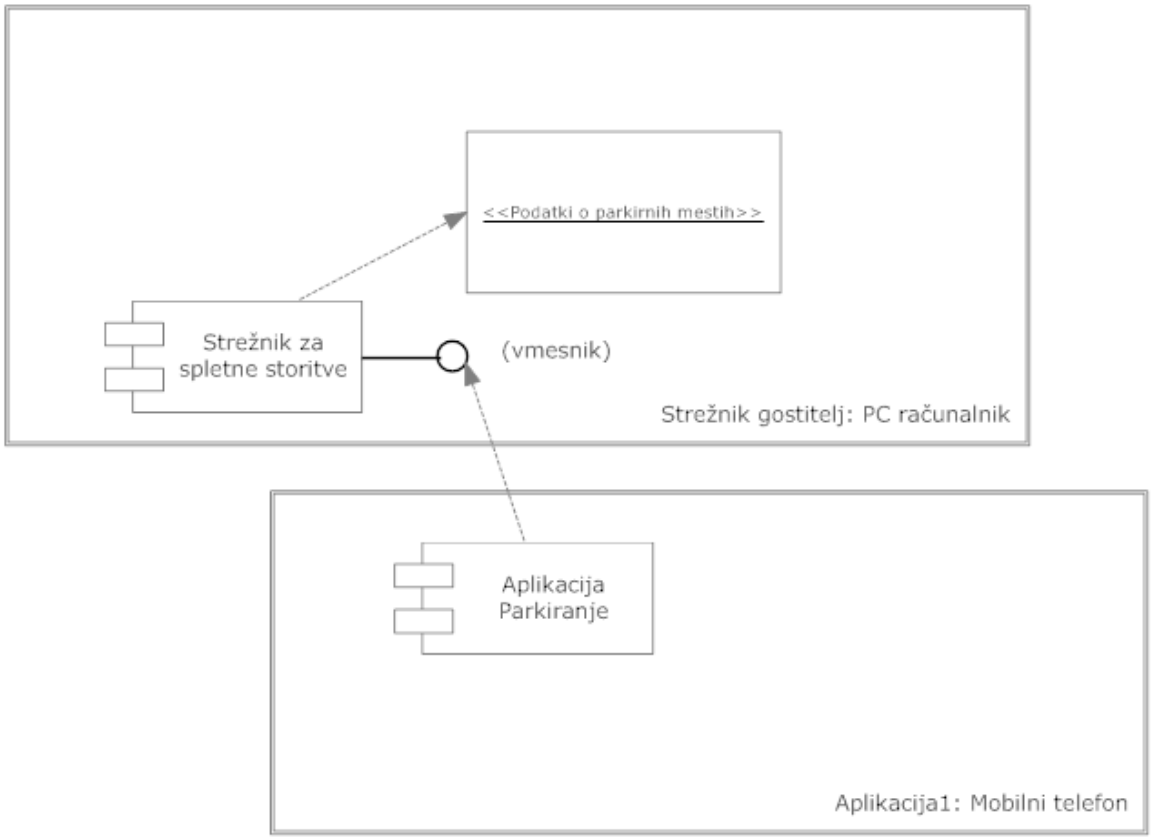
Prenosni telefon nima dostopa do interneta, zato aplikacija ob ukazu, ki uporablja spletno storitev, izpiše sporočilo o napaki.

Alternativni tok dogodkov 2

Uporabnikova aplikacija ne vsebuje veljavnega certifikata za avtentikacijo in preverjanje integritete poslanih zahtev, zato rezervacija/plačilo ni možna in aplikacija v tem primeru izpiše napako.

4.2 Arhitektura sistema rešitve

Za rešitev problema podanega v drugem poglavju smo implementacijo rešitve razdelili na dva dela in sicer implementacijo aplikacije za mobilne naprave in implementacijo spletnih storitev ter objavo le-teh na strežniku za spletne storitve. Diagram na sliki 25 prikazuje arhitekturo celotnega sistema.



Slika 25: Arhitekturni diagram

4.2.1 Arhitektura spletnih storitev

Prvi korak k implementaciji rešitve je bila definicija in implementacija spletnih storitev, katere se bodo klicale iz aplikacije Parkiranje. Rešitev mora vsebovati tudi podatke o parkirnih hišah, parkirnih mestih, uporabnikih in izvedenih rezervacijah. Zaradi preprostosti smo se odločili, da podatke shranimo v XML datotekah.

4.2.1.1 Shranjevanje podatkov

Podatki o parkirnih hišah se vzdržujejo v XML datoteki *PH.xml*. V datoteki se hranijo podatki o posamezni parkirni hiši (park_hisa). Parkirna hiša pa vsebuje še enolični identifikator (id_park_hisa), podatka o geografski legi (x_latitude, y_longitude) ter opis oziroma naziv parkirne hiše (opis). Na sliki 26 je prikazan primer XML datoteke, ki vsebuje podatke o parkirnih hišah. Podatki iz te datoteke se uporabijo pri spletni storitvi, ki bi glede na trenutno lokacijo podala najbližjo parkirno hišo.

```
<?xml version="1.0" encoding="UTF-8"?><parkhouse>
	<park_hisa>
		<id_park_hisa>1</id_park_hisa>
		<x_latitude>12</x_latitude>
		<y_longitude>12</y_longitude>
		<opis>Hiša 1 - Tržaška cesta 25</opis>
	</park_hisa>
	<park_hisa>
		<id_park_hisa>2</id_park_hisa>
		<x_latitude>22</x_latitude>
		<y_longitude>24</y_longitude>
		<opis>Hiša 2 - Jadranska cesta 9</opis>
	</park_hisa>
	<park_hisa>
		<id_park_hisa>3</id_park_hisa>
		<x_latitude>4</x_latitude>
		<y_longitude>2</y_longitude>
		<opis>Hiša BTC - Šmartinska cesta 152</opis>
	</park_hisa>
	<park_hisa>
		<id_park_hisa>4</id_park_hisa>
		<x_latitude>66</x_latitude>
		<y_longitude>88</y_longitude>
		<opis>Hiša Študent - Dunajska cesta 22</opis>
	</park_hisa>
</parkhouse>
```

Slika 26: Parkirne hiše

```
<?xml version="1.0" encoding="UTF-8"?><parkirisce>
	<parkirno_mesto>
		<id_parkirno_mesto>1</id_parkirno_mesto>
		<x_latitude>12</x_latitude>
		<y_longitude>12</y_longitude>
		<prosto_do>2008-11-08 21:16:04.208</prosto_do>
		<id_lastnik>1</id_lastnik>
		<opis>Tržaška cesta 25 - 01</opis>
		<uporabnik>Matjazz</uporabnik>
	</parkirno_mesto>
	<parkirno_mesto>
		<id_parkirno_mesto>2</id_parkirno_mesto>
		<x_latitude>22</x_latitude>
		<y_longitude>24</y_longitude>
		<prosto_do>Empty</prosto_do>
		<id_lastnik>1</id_lastnik>
		<opis>Tržaška cesta 25 - 02</opis>
		<uporabnik>Matjazz</uporabnik>
	</parkirno_mesto>
</parkirisce>
```

Slika 27: Parkirna mesta

Podatki o parkirnih mestih se vzdržujejo v XML datoteki *PP.xml*. Datoteka hrani podatke o posameznem parkirnem mestu (parkirno_mesto). Parkirno mesto vsebuje enolični identifikator glede na vsa parkirna mesta (id_parkirno_mesto), podatka o geografski legi (x_latitude, y_longitude), časovni rok rezervacije (prosto_do), opis oziroma naziv parkirnega mesta (opis) ter podatek o uporabniku parkirnega mesta (uporabnik). Podatki iz te datoteke se uporabljajo pri spletni storitvi, ki poizveduje o najbližjih prostih parkirnih mestih ter pri

spletni storitvi, ki poizveduje o rezervacijah parkirnih mest za določenega uporabnika. Za prosto parkirno mesto se smatra tisto parkirno mesto, ki ima v polju *prosto_do* vrednost »Empty« ali pa vsebuje čas, ki ima vrednost manjšo od trenutnega časa. Na sliki 27 je prikazan primer XML datoteke, ki vsebuje podatke o parkirnih mestih.

Podatki o uporabnikih se vzdržujejo v XML datoteki *Uporabniki.xml*. V datoteki se hranijo posamezni uporabniki (uporabnik). Uporabnik vsebuje enolični identifikator (sifra), podatke o naslovu uporabnika (naslov, kraj), davčni številki (davcna) in telefonski številki (telefon). Na sliki 28 je prikazan primer XML datoteke, ki vsebuje podatke o uporabniku.

```
<?xml version="1.0" encoding="UTF-8"?><uporabniki>
      <uporabnik>
            <sifra>Matjazz</sifra>
            <ime>Matjaž</ime>
            <priimek>Auflič</priimek>
            <davcna>86435663</davcna>
            <naslov>Maistrova ulica 5</naslov>
            <kraj>1270 Litija</kraj>
            <telefon>+38631373923</telefon>
      </uporabnik>
</uporabniki>
```

Slika 28: XML datoteka s podatki o uporabniku

Podatki o plačilih se vzdržujejo v XML datoteki *Placila.xml*. V datoteki se hranijo posamezna plačila (placilo). Posamezno plačilo vsebuje podatke o parkirni hiši (id_lastnik), parkirnem mestu (id_parkirno_mesto), uporabniku parkirnega mesta (uporabnik), številu ur parkiranja (stevilo_ur), časovnem roku parkiranja (koncni_cas) ter podatek o tem, ali je parkirna hiša že prevzela podatek o plačilu (prevzet). Zadnje polje je namenjeno spletni storitvi, ki bi posamezni parkirni hiši vrnila podatke o uspešnih rezervacijah in se tako po uspešni poizvedbi zabeleži, da je podatek o plačilu prevzet. Slednje pa služi namenu, da parkirna hiša ne bi prevzela podatke o posameznem plačilu več kot enkrat. Na sliki 29 je prikazan primer XML datoteke, ki vsebuje podatke o plačilu.

```
<?xml version="1.0" encoding="UTF-8"?><placila>
      <placilo>
            <id_lastnik>1</id_lastnik>
            <id_parkirno_mesto>1</id_parkirno_mesto>
            <uporabnik>"Matjazz"</uporabnik>
            <stevilo_ur>1</stevilo_ur>
            <koncni_cas>2008-11-08 21:16:04.208</koncni_cas>
            <prevzet>0</prevzet>
      </placilo>
</placila>
```

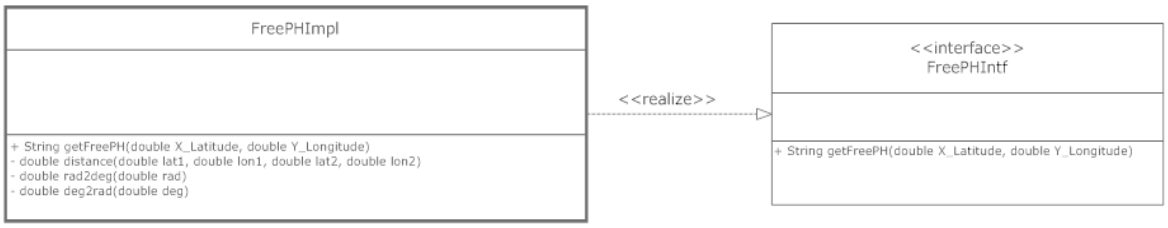
Slika 29: XML datoteka s podatki o plačilih.

4.2.1.2 Spletne storitve

Definirali smo naslednje spletne storitve:

1. Iskanje najbližje parkirne hiše
   Spletna storitev ima dva vhodna parametra in sicer X in Y koordinato, ki predstavljata trenutno lokacijo uporabnika, vrne pa podatek o najbližji parkirni hiši. Spletna storitev vsebuje tudi privatno funkcijo *distance*, ki vrne absolutno razdaljo med dvema koordinatama. Slednja funkcija se uporabi za izračun najbližje parkirne hiše. Funkcija izračuna neposredno zračno razdaljo med dvema točkama in ne dejansko cestno razdaljo med dvema točkama, zato je možno, da spletna storitev ne poda dejansko najbližje parkirne hiše, vendar smo za potrebe diplomske naloge predpostavili kot dovolj dobro rešitev. Na sliki 30 je prikazan razredni diagram za razred *FreePHImpl*.

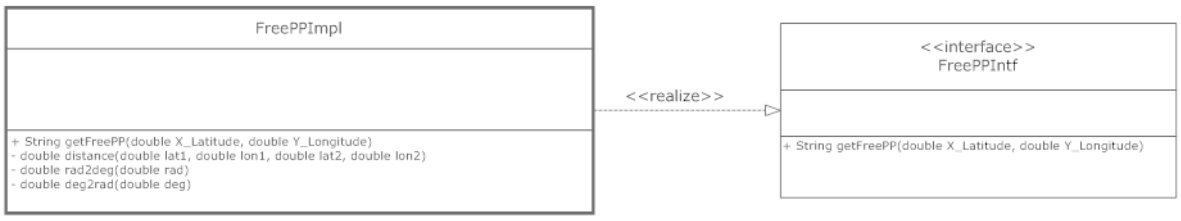
   Definicija: public String getFreePH(double X_Latitude, double Y_Longitude)



Slika 30: Razredni diagram za razred FreePHImpl

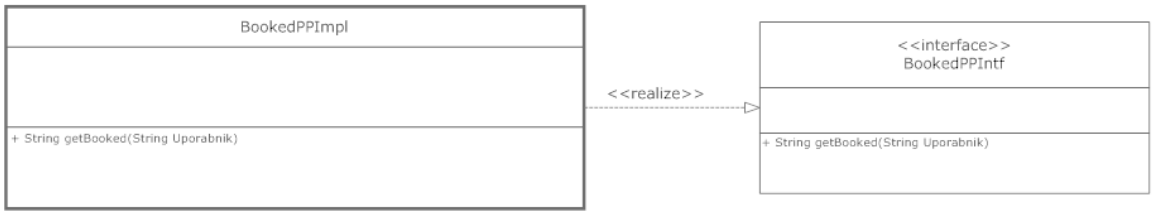
2. Iskanje najbližjih parkirnih mest
   Spletna storitev ima dva vhodna parametra in sicer X in Y koordinato, ki predstavljata trenutno lokacijo uporabnika, vrne pa podatek o najbližjih prostih parkirnih mestih. Spletna storitev vsebuje tudi privatno funkcijo *distance*, ki vrne absolutno razdaljo med dvema koordinatama. Na sliki 31 je prikazan razredni diagram za razred *FreePPImpl*.

   Definicija: public String getFreePP(double X_Latitude, double Y_Longitude)



Slika 31: Razredni diagram za razred FreePPImpl

3. Pregled rezervacij za uporabnika
   Spletna storitev ima vhodni parameter uporabnik, ki predstavlja uporabnika aplikacije, kateri uporablja to spletno storitev, vrne pa podatek o rezerviranih parkirnih mestih za tega uporabnika. Na sliki 32 je prikazan razredni diagram za razred *BookedPPImpl*.
   Definicija: public String getBooked(String Uporabnik)

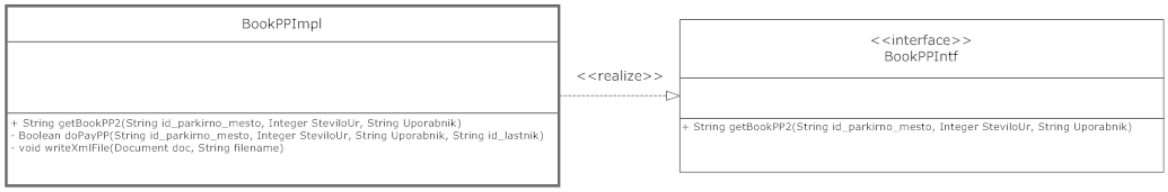


Slika 32: Razredni diagram za razred BookedPPImpl

4. Rezervacija parkirnega mesta

Spletna storitev ima več vhodnih parametrov in sicer enolični identifikator za parkirno mesto *id_parkirno_mesto*, število ur rezervacije parkirnega mesta in uporabnika, ki rezervira parkirno mesto. Storitev vsebuje tudi privatno funkcijo za izvedbo plačila, ki v datoteko *Placila.xml* zapiše podatke o uspešni rezervaciji za kasnejši obračun plačil uporabe parkirnih mest. Spletna storitev torej izvede rezervacijo parkirnega mesta. Na sliki 33 je prikazan razredni diagram za razred *BookPPImpl*.

Definicija: public String getBookPP2(String id_parkirno_mesto, Integer SteviloUr, String Uporabnik)

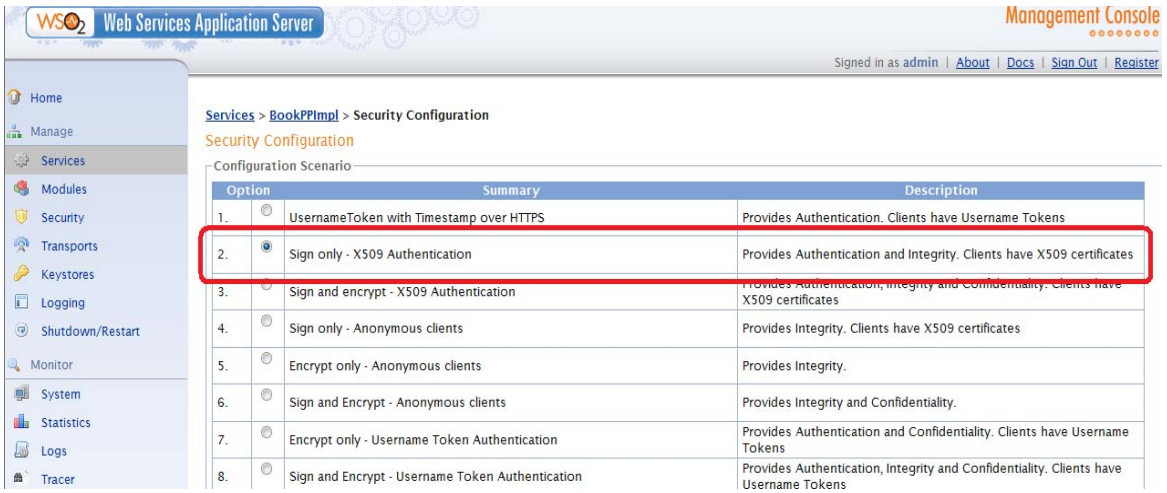


Slika 33: Razredni diagram za razred BookPPImpl

Za razliko od ostalih definiranih in implementiranih spletnih storitev, pa ta storitev vsebuje neke vrste plačilno transakcijo, ki zahteva določeno stopnjo varnosti. Potrebno je poskrbeti za pristno avtentikacijo uporabnika in pa integriteto SOAP sporočila, ki se pošlje s strani uporabnikove aplikacije. Slednje pomeni, da je SOAP sporočilo varno pred spreminjanjem podatkov o rezervaciji (kaj, kdo in koliko časa se rezervira). V primeru naknadnega spreminjanja podatkov je rezervacija neuspešna in s tem zagotovljena varnost. Varnost spletne storitve zagotavlja komunikacijski protokol WS-Security, katerega podpira WSO2 WSAS strežnik. Na WSO2 WSAS strežniku objavljene storitve smo izbrali varnostni scenarij za to spletno storitev, kot je to prikazano na sliki 35. Na sliki 34 pa je prikazan standard WS-Policy za to spletno storitev, ki vsebuje potrebne podatke o uporabi modula Rampart, ki je pri strežniku WSO2 WSAS zadolžen za varnost spletnih storitev.

```
<wsp:Policy wsu:Id="SigOnly" xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:AsymmetricBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
        <wsp:Policy>
          <sp:InitiatorToken>
            <wsp:Policy>
              <sp:X509Token sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRecipient">
                <wsp:Policy>
                  <sp:WssX509V3Token10 />
                </wsp:Policy>
              </sp:X509Token>
            </wsp:Policy>
          </sp:InitiatorToken>
          <sp:RecipientToken>
            <wsp:Policy>
              <sp:X509Token sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/Never">
                <wsp:Policy>
                  <sp:WssX509V3Token10 />
                </wsp:Policy>
              </sp:X509Token>
            </wsp:Policy>
          </sp:RecipientToken>
          <sp:AlgorithmSuite xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
            <wsp:Policy>
              <sp:Basic256 />
            </wsp:Policy>
          </sp:AlgorithmSuite>
          <sp:Layout>
            <wsp:Policy>
              <sp:Strict />
            </wsp:Policy>
          </sp:Layout>
          <sp:IncludeTimestamp />
          <sp:OnlySignEntireHeadersAndBody />
        </wsp:Policy>
      </sp:AsymmetricBinding>
      <sp:Wss10 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
        <sp:Policy>
          <sp:MustSupportRefKeyIdentifier />
          <sp:MustSupportRefIssuerSerial />
        </sp:Policy>
      </sp:Wss10>
      <sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
        <sp:Body />
      </sp:SignedParts>
    </wsp:All>
  </wsp:ExactlyOne>
  <rampart:RampartConfig xmlns:rampart="http://ws.apache.org/rampart/policy">
    <rampart:user>wso2wsas</rampart:user>
    <rampart:encryptionUser>useReqSigCert</rampart:encryptionUser>
    <rampart:timestampTTL>300</rampart:timestampTTL>
    <rampart:timestampMaxSkew>300</rampart:timestampMaxSkew>
    <rampart:encryptionCrypto>
      <rampart:crypto provider="org.wso2.wsas.security.ServerCrypto">
        <rampart:property name="org.wso2.wsas.security.wso2wsas.crypto.keystore">wso2wsas.jks</rampart:property>
        <rampart:property name="org.apache.ws.security.crypto.provider">org.wso2.wsas.security.ServerCrypto</rampart:property>
        <rampart:property name="org.wso2.wsas.security.wso2wsas.crypto.alias">wso2wsas</rampart:property>
        <rampart:property name="rampart.config.user">wso2wsas</rampart:property>
      </rampart:crypto>
    </rampart:encryptionCrypto>
    <rampart:signatureCrypto>
      <rampart:crypto provider="org.wso2.wsas.security.ServerCrypto">
        <rampart:property name="org.wso2.wsas.security.wso2wsas.crypto.keystore">wso2wsas.jks</rampart:property>
        <rampart:property name="org.apache.ws.security.crypto.provider">org.wso2.wsas.security.ServerCrypto</rampart:property>
        <rampart:property name="org.wso2.wsas.security.wso2wsas.crypto.alias">wso2wsas</rampart:property>
        <rampart:property name="rampart.config.user">wso2wsas</rampart:property>
      </rampart:crypto>
    </rampart:signatureCrypto>
  </rampart:RampartConfig>
</wsp:Policy>
```

Slika 34: WS-Policy za spletno storitev rezervacije parkirnega mesta

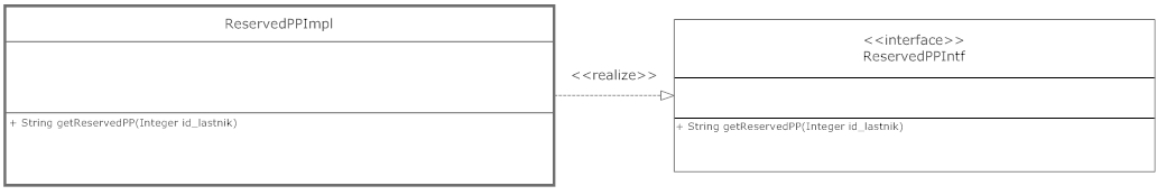


Slika 35: Določitev varnostnega scenarija za spletno storitev

5. Pregled rezervacij po parkirnih hišah

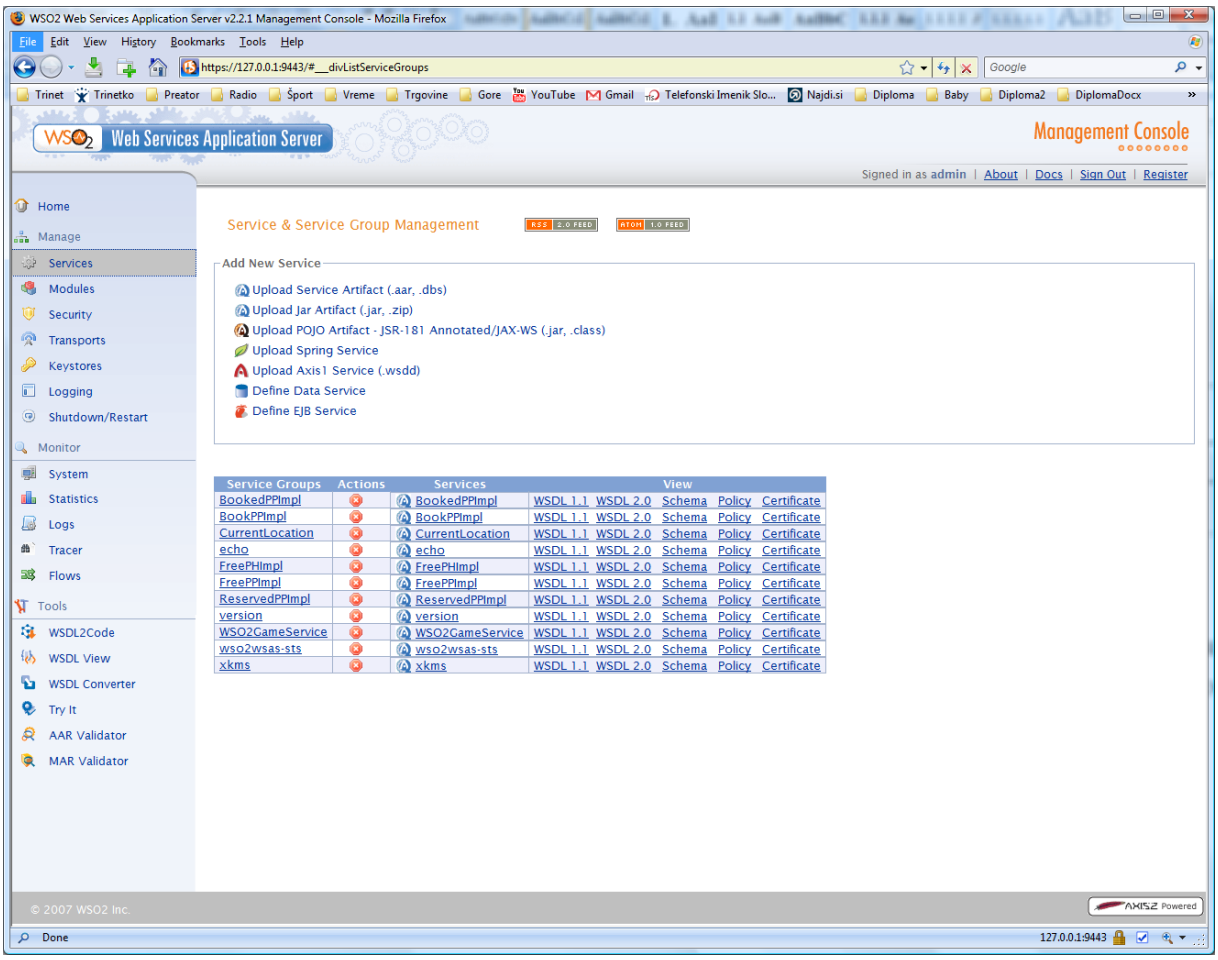
   Spletna storitev ima za vhodni parameter enolični identifikator id_lastnik, ki predstavlja posamezno parkirno hišo. Storitev vrne podatke o uspešnih rezervacijah parkirnih mest za obračun plačil. Ta spletna storitev je edina, ki se ne uporablja v aplikaciji. Njen namen je ta, da parkirne hiše preko svojih aplikacij, preko spletne storitve dostopajo do želenih podatkov. Slika 36 predstavlja razredni diagram za razred *ReservedPPImpl*.

   Definicija: public String getReservedPP(Integer id_lastnik);

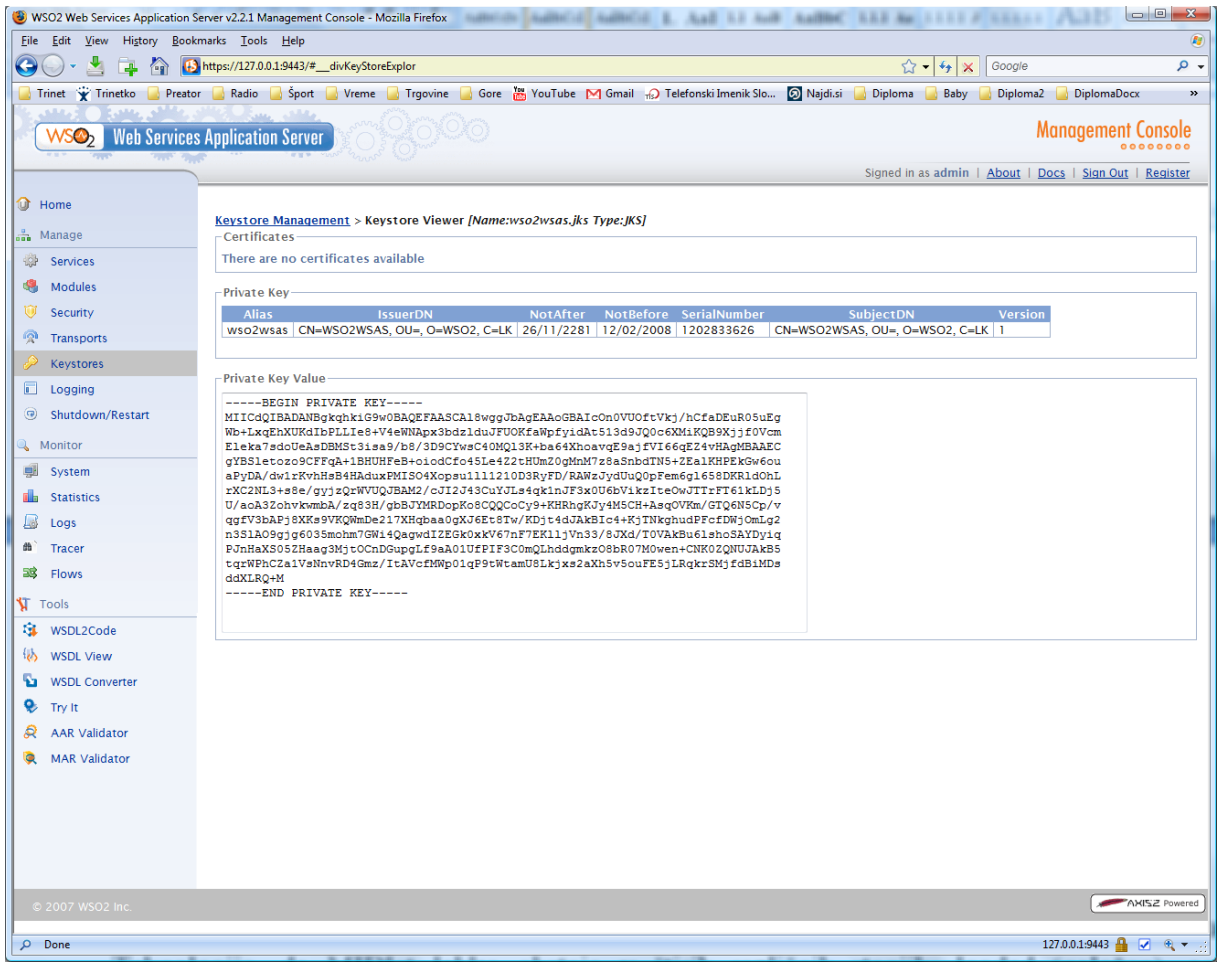


Slika 36: Razredni diagram za razred ReservedPPImpl

Za objavo spletnih storitev smo se odločili za WSO2 WSAS strežnik za spletne storitve zaradi učinkovite administracije spletnih storitev (slika 37), podpore protokolu WS-Security in WS-Policy standardu, administracijo privatnih in javnih ključev (slika 38) ter preproste objave.



Slika 37: Spletne storitve objavljene na WSO2 WSAS strežniku



Slika 38: Administracija privatnih in javnih ključev v WSO2 WSAS

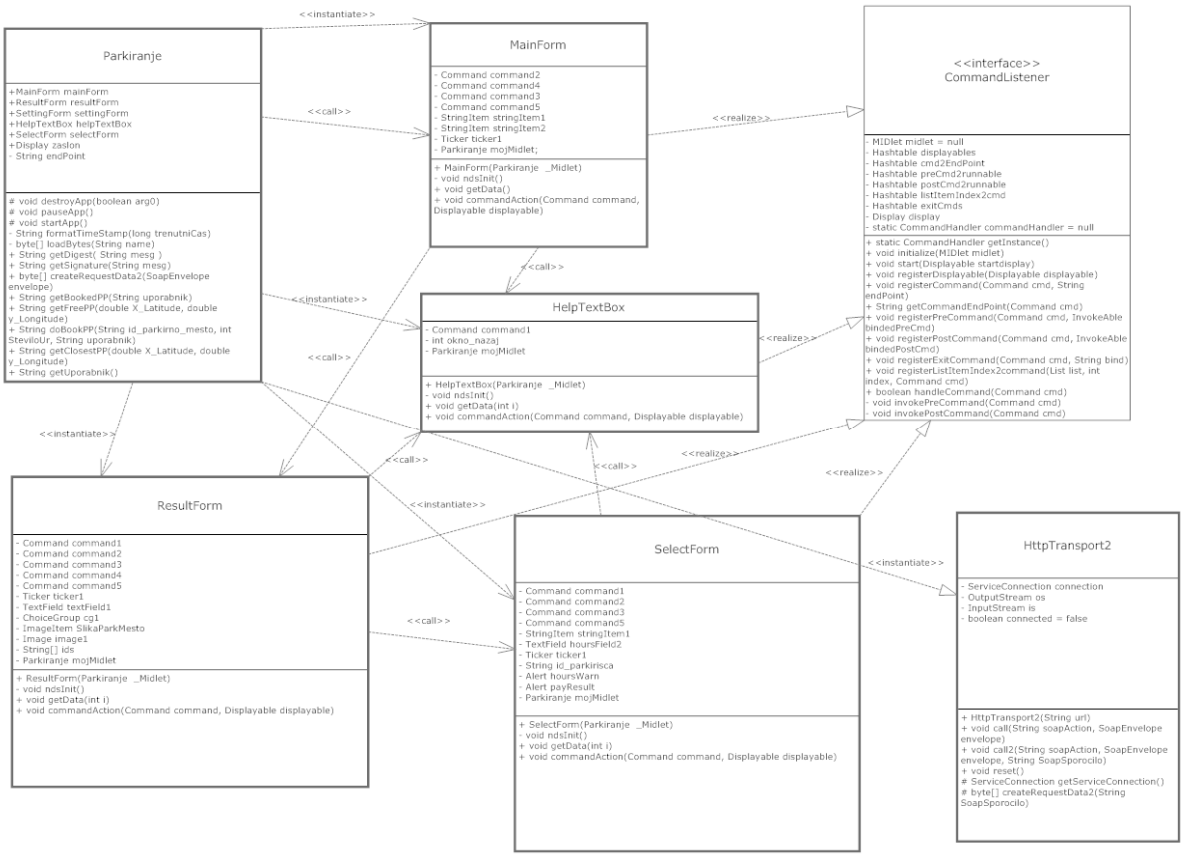
4.2.2 Arhitektura aplikacije

Aplikacija Parkiranje je MIDlet. Za MIDlet pa je značilno, kot je bilo to omenjeno v poglavju Tehnologije, da je MIDlet lahko v štirih različnih stanjih: *loaded* (naložen), *active* (aktiven), *paused* (ustavljen) in *destroyed* (uničen). Struktura razredov aplikacije je razvidna na sliki 39. Osnovna zaslonska maska aplikacije neposredno uporablja razred *MainForm*. Uporabnik uporablja unikatno aplikacijo, saj vsebuje uporabnikovo unikatno šifro za identifikacijo v spletnih storitvah ter uporabnikov javni in privatni ključ.

4.2.2.1 Razredi aplikacije

1. *Parkiranje* je razred tipa MIDlet, torej osnovni razred, ki vzdržuje stanja MIDlet-a in poskrbi za inicializacijo le tega.
2. *MainForm* je razred tipa Form in je glavni obrazec v osnovni zaslonski maski.
3. *ResultForm* je razred tipa Form in je obrazec za predstavitev najbližjih prostih parkirnih mest.
4. *SelectForm* je razred tipa Form in je obrazec za rezervacijo parkirnega mesta.
5. *HelpTextBox* je razred tipa TextBox in služi za prikaz pomoči uporabniku.

6. *HttpTransport2* je razred tipa Transport in je predelana verzija razreda HttpTransport iz knjižnice kSoap2. Razred smo implementirali za realizacijo protokola WS-Security v MIDlet-u.
7. *CommandHandler* je razred, ki skrbi za interakcijo zgoraj naštetih obrazcev tipa Form in TextBox.



Slika 39: Razredni diagram aplikacije

4.2.2.2 Dodatne knjižnice

Za implementacijo MIDlet aplikacije je potrebna namestitev orodja Sun Java Wireless Toolkit, ki doprinese osnovne knjižnice za razvoj MIDlet-ov v okolju Java ME. Rešitev opisanega problema v drugem poglavju zahteva klic spletne storitve po komunikacijskem protokolu WS-Security, kar osnovni paket knjižnic ne podpira. Pri tem smo se poslužili knjižnice **kSoap2**, ki podpira kreiranje lastnega SOAP sporočila z zaglavjem po meri protokola WS-Security, ki je bistveni del spletne storitve. Ker tudi ta knjižnica ne podpira popolne izdelave SOAP sporočila po meri, smo za ta namen implementirali razred HttpTransport2. Izdelava SOAP sporočila po protokolu WS-Security pa zahteva uporabo kriptografskih funkcij in za ta namen smo v projekt rešitve ustreznega odjemalca spletnih storitev MIDlet-a vključili še knjižnico **BouncyCastle**.

4.2.2.3 Javni in privatni ključ

Klicanje varne spletne storitve zahteva kreiranje SOAP sporočila z zaglavjem po meri protokola WS-Security, ki pa vsebuje uporabo kriptografskih funkcij. Za slednje se pa še potrebuje uporabnikov privatni in javni ključ, ki sta posledično dodana MIDlet-u. Podrobnejša izdelava SOAP sporočila in potrebni kriptografski koraki v tej smeri so že opisani v poglavju Tehnologije.

4.3 Aplikacija za iskanje prostih parkirnih mest Parkiranje

4.3.1 Implementacija aplikacije

Pri razvoju smo si pomagali z emulatorjem za mobilne naprave, ki je del orodja Sun Java Wireless Toolkit.

Osnovna knjižnica v okolju Java ME ne podpira klicanja varnih spletnih storitev oziroma standarda WS-Security ter podpisovanja SOAP sporočil s certifikatom, zato smo si pri implementaciji pomagali z dvema dodatnima knjižnicama. S pomočjo knjižnice Bouncy Castle smo lahko realizirali zgoščevalno funkcijo in podpisovanje SOAP sporočila, ki predstavljata realizacijo varne spletne storitve poznano pod Web Service Security. Koda prej opisane funkcionalnosti je prikazana na sliki 40. Druga knjižnica, s katero smo si pomagali, se imenuje kSoap2. Slednja omogoča kreiranje lastnih sporočil SOAP, vendar ni mogoče kreirati zaglavja SOAP sporočila po meri, ki bi ustrezal WS-Security tehnologiji. Problem smo lahko rešili, ker je ta knjižnica odprtokodna in smo lahko sami predelali funkcije za oblikovanje ustreznega SOAP sporočila. Na sliki 41 je prikazana spremenjena koda, ki omogoča zadnjo opisano funkcionalnost. Opisani problem je predstavljal največji izziv pri implementaciji rešitve. Na sliki 42 pa je prikazan izsek iz kode, kjer se kliče **varna** spletna storitev. Pri kreiranju SOAP sporočila po meri smo pri računanju vrednosti DigestValue in kreiranju digitalnega podpisa Signature, da so bili deli SOAP sporočila kot vhodni parameter v kanonski obliki.

```
private byte[] loadBytes(String name) throws IOException {
            InputStream in = getClass().getResourceAsStream(name);
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            int c;
            while((c = in.read()) != -1)
                out.write(c);
            byte[] raw = out.toByteArray();
            out.close();
            return raw;
        }

public String getDigest( String mesg ) throws Exception {
         SHA1Digest digEng = new SHA1Digest();
         byte [] mesgBytes = mesg.getBytes();
         digEng.update( mesgBytes, 0, mesgBytes.length );
         byte [] digest = new byte[digEng.getDigestSize()];
         digEng.doFinal(digest, 0);
         // Encode the digest into ASCII format for XML
         return (new String(Base64.encode(digest)));
    }

public String getSignature(String mesg) throws Exception {
         String keyFile = "/wso2wsas.pem";
         InputStream keyIn = getClass().getResourceAsStream(keyFile);
         byte[] certificateBytes = loadBytes(keyFile);
         String b64encoded = new String(certificateBytes);
         byte [] asn1PrivateKeyBytes = Base64.decode(b64encoded.getBytes());
         ASN1InputStream asn1is = new ASN1InputStream(asn1PrivateKeyBytes);
         DERObject obj = asn1is.readObject();
         keyIn.close();

        RSAPrivateKeyStructure rsa = new RSAPrivateKeyStructure((ASN1Sequence)obj);
                BigInteger crtCoef = rsa.getCoefficient();
                BigInteger mod = rsa.getModulus();
                BigInteger pubExp = rsa.getPublicExponent();
                BigInteger privExp = rsa.getPrivateExponent();
                BigInteger p1 = rsa.getPrime1();
                BigInteger p2 = rsa.getPrime2();
                BigInteger exp1 = rsa.getExponent1();
                BigInteger exp2 = rsa.getExponent2();

        RSAPrivateCrtKeyParameters privKey = new RSAPrivateCrtKeyParameters(mod,
                pubExp, privExp, p1, p2, exp1, exp2, crtCoef);
                SHA1Digest digest = new SHA1Digest();
                RSADigestSigner eng2 = new RSADigestSigner(digest);
                eng2.init(true,privKey);
                byte[] mbytes2 = mesg.getBytes();
                eng2.update(mbytes2, 0, mbytes2.length);
                byte[] signature2 = eng2.generateSignature();
                String result = new String(Base64.encode(signature2))
                return result;
    }
```

Slika 40: Koda za zgoščevanje in podpisovanje

```
public void call2(String soapAction, SoapEnvelope envelope, String SoapSporocilo) throws
IOException, XmlPullParserException {
        if (soapAction == null)
            soapAction = "\"\"";
        byte[] requestData = createRequestData2(SoapSporocilo);
        requestDump = debug ? new String(requestData) : null;
        responseDump = null;
        try {
            connected = true;
            connection = getServiceConnection();
            connection.setRequestProperty("SOAPAction", soapAction);
            connection.setRequestProperty("Content-Type", "text/xml");
            connection.setRequestProperty("Content-Length", "" + requestData.length);
            connection.setRequestProperty("User-Agent", "kSOAP/2.0");
            connection.setRequestMethod(HttpConnection.POST);
            os = connection.openOutputStream();
            os.write(requestData, 0, requestData.length);
            os.close();
            requestData = null;
            is = connection.openInputStream();
            if (debug) {
                ByteArrayOutputStream bos = new ByteArrayOutputStream();
                byte[] buf = new byte[256];
                while (true) {
                    int rd = is.read(buf, 0, 256);
                    if (rd == -1)
                        break;
                    bos.write(buf, 0, rd);
                }
                bos.flush();
                buf = bos.toByteArray();
                responseDump = new String(buf);
                is.close();
                is = new ByteArrayInputStream(buf);
            }

            parseResponse(envelope, is);
        } finally {
            if (!connected)
                throw new InterruptedIOException();
            reset();
        }
        if (envelope.bodyIn instanceof SoapFault)
            throw ((SoapFault) envelope.bodyIn);
    }
```

Slika 41: Koda za kreiranja SOAP sporočila po meri

```
.......
String wsh =
          "<soapenv:Envelope xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\">"+nl+
          "<soapenv:Header>"+nl+
          "<wsse:Security          xmlns:wsse=\"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd\"
soapenv:mustUnderstand=\"1\">"+nl+
          "<ds:Signature xmlns:ds=\"http://www.w3.org/2000/09/xmldsig#\" Id=\"Signature-8543475\">"+nl+
          "<ds:SignedInfo>"+nl+nl+
          "<ds:CanonicalizationMethod Algorithm=\"http://www.w3.org/2001/10/xml-exc-c14n#\"></ds:CanonicalizationMethod>"+nl+
          "<ds:SignatureMethod Algorithm=\"http://www.w3.org/2000/09/xmldsig#rsa-sha1\"></ds:SignatureMethod>"+nl+
          "<ds:Reference URI=\"#"+referenceURI+"\">"+nl+
          "<ds:Transforms>"+nl+
          "<ds:Transform Algorithm=\"http://www.w3.org/2001/10/xml-exc-c14n#\"></ds:Transform>"+nl+
          "</ds:Transforms>"+nl+
          "<ds:DigestMethod Algorithm=\"http://www.w3.org/2000/09/xmldsig#sha1\"></ds:DigestMethod>"+nl+
          "<ds:DigestValue>"+digestvalue+"</ds:DigestValue>"+nl+
          "</ds:Reference>"+nl+
          "</ds:SignedInfo>"+nl+
          "<ds:SignatureValue>"+signaturevalue+"</ds:SignatureValue>"+nl+
          "<ds:KeyInfo Id=\"KeyId-10779843\">"+nl+
          "<wsse:SecurityTokenReference      xmlns:wsu=\"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd\" wsu:Id=\"STRId-16443279\">"+nl+
          "<wsse:KeyIdentifier         EncodingType=\"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-
1.0#Base64Binary\"                         ValueType=\"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509v3\">"+publicKey+"</wsse:KeyIdentifier>"+nl+
          "</wsse:SecurityTokenReference>"+nl+
          "</ds:KeyInfo>"+nl+
          "</ds:Signature>"+nl+
          "<wsu:Timestamp          xmlns:wsu=\"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd\"
wsu:Id=\"Timestamp-18034294\">"+nl+
          "<wsu:Created>"+createTime+"</wsu:Created>"+nl+
          "<wsu:Expires>"+expireTime+"</wsu:Expires>"+nl+
          "</wsu:Timestamp>"+nl+
          "</wsse:Security>"+nl+
          "</soapenv:Header>"+nl+
          "<soapenv:Body         xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"         xmlns:wsu=\"http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd\" wsu:Id=\""+referenceURI+"\"></soapenv:Body>"+nl+
          "</soapenv:Envelope>";

                    String SoapSporocilo = wsh;

                    // Call the WebService
                    try
                    {
                              t2.call2(method, envelope, SoapSporocilo);
                    }
                    catch(XmlPullParserException io) {
                              return "Napaka pri klicu na strežnik: "+io.getMessage();
                    }
                    String result = envelope.getResponse().toString();
                    return result;


........
```

Slika 42: Klic varne spletne storitve

Aplikacija pogosto uporablja koordinate trenutne lokacije, ki smo jo pridobili na naslednji način. Vsak prenosni telefon ima dostop do koordinat najbližje bazne postaje in tako smo v našem primeru uporabili to rešitev. Natančnost te rešitve je odvisna od velikosti celice, ki jo posamezna bazna postaja pokriva. Celice so zelo različnih velikosti, od nekaj 100 m² do nekaj 100 km². Celice so večje tam, kjer je gostota uporabnikov nizka, denimo na ruralnih območjih, manjše pa tam, kjer je gostota uporabnik visoka, predvsem v mestih. [8] V našem primeru, kjer bi se aplikacija uporabljala v večjih mestih, se lahko zagotovi natančnost do 150 m. Za Java ME ogrodje obstaja opcijski paket JSR 179, ki vsebuje niz lokacijskih API funkcij. Na sliki 43 je prikazana koda za pridobitev trenutne lokacije uporabnika. Najprej je treba določiti kriterije za lokacijsko metodo s pomočjo razreda *Criteria,* s katerim se lahko določi natančnost rezultata, odzivni čas, potreba po nadmorski višini in hitrost. Ko aplikacija pridobi primerek razreda *LocationProvider,* ki zadostuje zahtevanim kriterijem, lahko uporabimo ta objekt za pridobitev želenih podatkov s pomočjo razreda *Location.*

```
Criteria cr= new Criteria();
  cr.setHorizontalAccuracy(500);

  //pridobimo inštanco
  LocationProvider lp= LocationProvider.getInstance(cr);

  //zahtevamo lokacjo
  Location l = lp.getLocation(60);
  Coordinates c = l.getQualifiedCoordinates();

  if (c != null ) {
  // uporabi informacijo koordinat
  double lat = c.getLatitude();
  double lon = c.getLongitude();

  try {
          String s = mojMidlet.getClosestPP(lat,lon);
          int i=s.indexOf("\n");
          String tokens = s.substring(0, i);
          stringItem1.setText("\n"+tokens);
      }
      catch (Exception e) {
         stringItem1.setText("Napaka: "+e.getMessage());
      } }
```

Slika 43: Pridobitev podatkov o trenutni lokaciji

4.3.2 Delovanje aplikacije

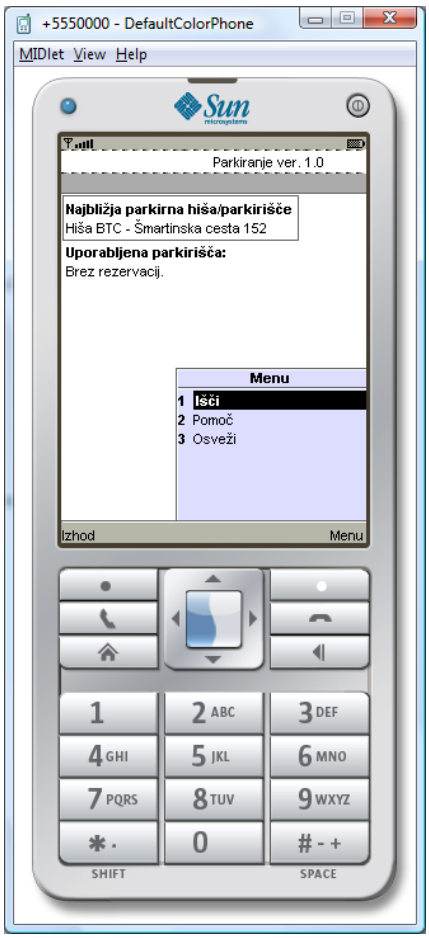
V tem poglavju bomo predstavili delovanje mobilne aplikacije iz uporabniškega vidika. Predstavili bomo tudi osnovne zaslonske maske aplikacije.

4.3.2.1 Inicializacija

Ko uporabnik zažene aplikacijo na mobilni napravi, se v ozadju nemudoma prične odvijati inicializacija. Preko spletnih storitev se na podlagi trenutne lokacije, ki se preko mobilne naprave prebere od najbližje bazne postaje, poizve po najbližji parkirni hiši in pa rezerviranih parkirnih mestih uporabnika. V primeru, da mobilna naprava zaradi nepovezanosti z internetom nima dostopa do spletnih storitev, aplikacija izpiše napako.

4.3.2.2 Prikaz glavnega obrazca aplikacije

V glavnem obrazcu programa na sliki 44 se po uspešni poizvedbi s pomočjo spletnih storitev izpiše najbližja parkirna hiša in pa rezervirana parkirna mesta uporabnika. Glavni obrazec v meniju ponuja uporabniku ukaze *Išči*, *Pomoč* in *Osveži* ter na levi strani spodaj ukaz *Izhod*. Z ukazom *Išči* uporabnik sproži poizvedbo o najbližjih prostih parkirnih mestih. Ukaz *Osveži* uporabnik uporabi za ponovno poizvedbo o najbližji parkirni hiši v primeru, da se je lokacija uporabnika spremenila ter za ponovno poizvedbo o rezerviranih parkirnih mestih v primeru, da je minilo že veliko časa od zadnje poizvedbe. Skratka, uporabnik lahko ponovno poizve o najbližji parkirni hiši ter stanju njegovih rezervacij, s pomočjo katerih lahko preveri, kako dolgo lahko rezervirano parkirno mesto še uporablja. Ukaz *Izhod* služi za ustavitev izvajanja aplikacije, ukaz *Pomoč* pa za razlago ukaza *Išči*.

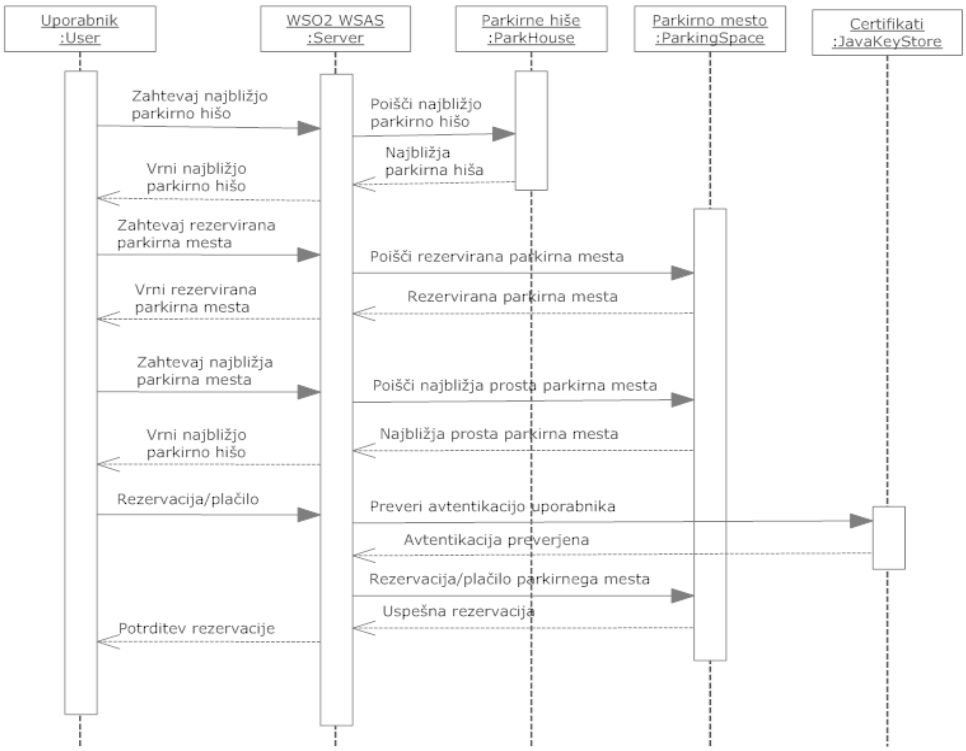


Slika 44: Glavni obrazec aplikacije

4.3.2.3 Iskanje prostih parkirnih mest

V glavnem obrazcu programa po uspešni izvedbi ukaza *Išči* se s pomočjo spletnih storitev prikaže obrazec prostih parkirnih mest (slika 45) in izpišejo se najbližja prosta parkirna mesta. Obrazec v meniju uporabniku ponuja ukaze *Nazaj*, *Rezerviraj*, Pomoč in *Osveži* ter na levi strani spodaj ukaz *Izhod*. Z ukazom *Rezerviraj* uporabnik sproži rezervacijo izbranega prostega parkirnega mesta. Ukaz *Osveži* uporabnik uporabi za ponovno poizvedbo o najbližjih prostih parkirnih mestih v primeru, da se je lokacija uporabnika spremenila. Skratka uporabnik lahko ponovno poizve o najbližjih prostih parkirnih mestih. Ukaz *Izhod* služi za ustavitev izvajanja aplikacije, ukaz *Pomoč* za razlago ukaza *Rezerviraj*, medtem ko ukaz *Nazaj* popelje uporabnika nazaj v glavno okno.

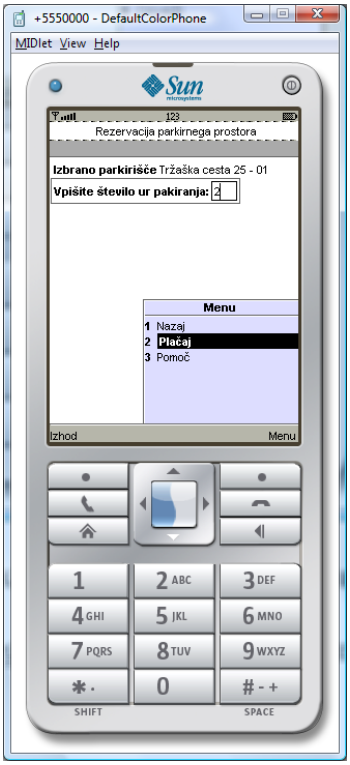
Slika 45: Obrazec za iskanje prostih parkirnih mest



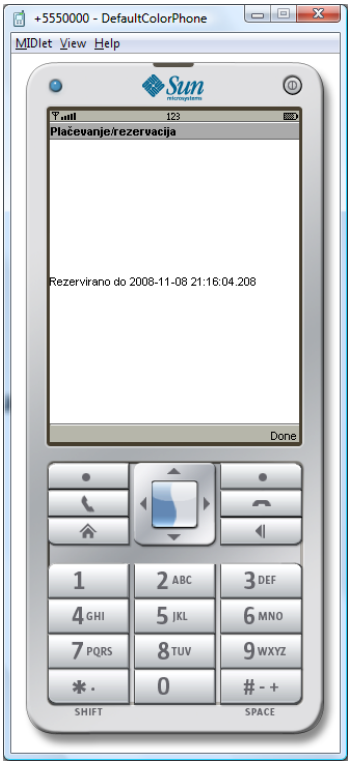
Slika 46: Diagram zaporedja scenarija rezervacije parkirnega mesta

4.3.2.4 Rezervacija/plačilo izbranega prostega parkirnega mesta

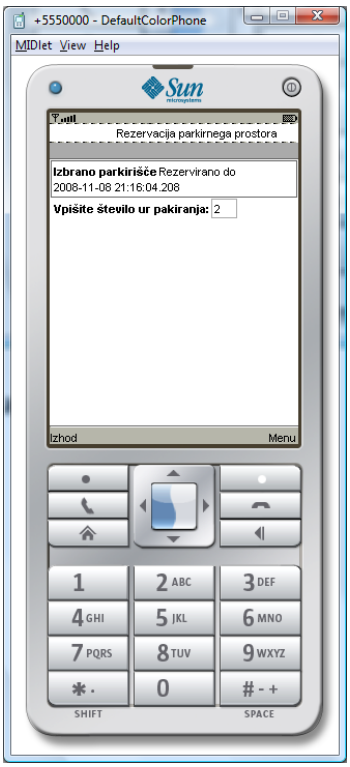
V obrazcu prostih parkirnih mest, po uspešni izvedbi ukaza *Rezerviraj, se* prikaže obrazec za rezervacijo (slika 47). Obrazec v meniju ponuja uporabniku ukaze *Nazaj*, *Plačaj* in *Pomoč* ter na levi strani spodaj ukaz *Izhod*. V polje *Število ur* pakiranja uporabnik vpiše želeno časovno obdobje rezervacije. Z ukazom *Plačaj* uporabnik rezervira oziroma naroči plačilo izbranega prostega parkirnega mesta. Za uspešno izvedbo slednjega ukaza mora aplikacija vsebovati veljavni certifikat za uspešno avtentikacijo uporabnika in ohranitev integritete poslanega sporočila, ki se pošlje spletni storitvi s tem ukazom. V primeru uspešne izvedbe ukaza se potrdi rezervacija in posledično osvežijo podatki o rezervacijah uporabnika v glavnem obrazcu, kakor je to razvidno na sliki 47, kjer je prikazan postopek rezervacije prostega izbranega parkirnega mesta. Ukaz *Izhod* služi za ustavitev izvajanja aplikacije, ukaz *Pomoč* za razlago ukaza *Plačaj*, medtem ko ukaz *Nazaj* popelje uporabnika nazaj v prejšnje okno. Na sliki 46 je prikazan diagram zaporedja za scenarij primera rezervacije prostega izbranega parkirnega mesta.



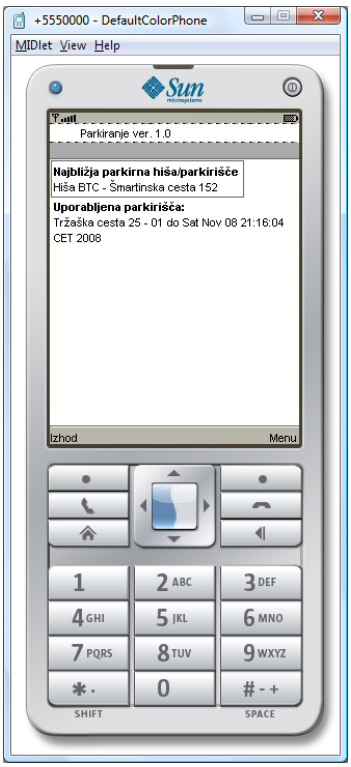
1. korak



2. korak



3. korak



4. korak

Slika 47: Postopek rezervacije/plačila parkirnega mesta

4.4 Možnosti nadaljnjega razvoja aplikacije

V tem poglavju bomo opisali nadaljnje možnosti razvoja aplikacije in celotnega sistema rešitve. Nadaljnji razvoj bomo predstavili v dveh delih, in sicer v tehničnem, kjer bomo opisali izboljšavo na strojni in programerski ravni, ter v uporabniškem delu, kjer bomo opisali možne funkcionalne nadgradnje aplikacije.

4.4.1 Tehnični del

Glavni problem trenutne aplikacije je nenatančno poizvedovanje po trenutni lokaciji uporabnika. Za potrebe diplomske naloge smo poiskali rešitev, ki za trenutno lokacijo vzame koordinati najbližje bazne postaje, ki oddaja signal za mobilne telefone. Asistent Andrej Krevl, ki je bedel nad implementacijo rešitve za diplomsko nalogo, je za svojo diplomsko nalogo realiziral modul v Java ME okolju, ki podpira branje GPS koordinat. Torej bi povezovanje s tem modulom doprineslo natančnejšo trenutno lokacijo uporabnika in posledično boljše ter natančnejše iskanje prostih parkirnih mest.

Spletne storitve uporabljajo za izračun najbližjega parkirnega prostora funkcijo, ki vzame koordinati trenutne lokacije uporabnika in koordinati parkirnega mesta ter na podlagi teh koordinat izračuna zračno razdaljo med uporabnikom in parkirnim mestom. V realni situaciji pa se lahko zgodi, da izračunana najmanjša razdalja ne predstavlja tudi najbližjo iskano parkirno mesto, ker pot po ulicah in cestah do parkirnega mesta seveda ne poteka po zračni

liniji med iskalcem in iskanim. Potrebna bi bila podrobna baza koordinat mesta, v katerem iščemo parkirno mesto, in tudi baza možnih poti med posameznimi koordinatami. Spletna storitev bi tako v bazi koordinat poiskala lokacijo uporabnika in parkirnega mesta, nato pa bi s pomočjo baze možnih poti izračunala razdaljo med uporabnikom in parkirnim mestom.

Izboljšanje aplikacije je tudi implementacija spletne storitve, ki rezervira parkirno mesto, hkrati pa le-ta sama obvesti parkirno hišo o izvedeni rezervaciji. Za to je seveda potrebna realizacija povezave spletne storitve z informacijskim sistemom parkirne hiše. Vendar bi tako ta izboljšava pripomogla k večji naravnanosti celotnega sistema k »real-time« rešitvi. »Real-time« programiranje zagotavlja določen rezultat v predvidenem časovnem roku. Torej v našem primeru pomeni, da bi parkirna hiša imela takoj ažurne podatke o rezervacijah parkirnih mest.

4.4.2 Uporabniški del

Uporabnik ima v aplikaciji trenutno na voljo relativno malo funkcionalnosti. Išče lahko najbližja parkirna mesta, jih rezervira in naknadno pregleduje svoje rezervacije. Možne dodatne funkcionalnosti so naslednje:

- Uporabnik predčasno prekine rezervacijo. Uporabnik bi v osnovnem oknu izbral rezervirano parkirno mesto in prekinil rezervacijo. Tako bi uporabnik plačal toliko, kolikor časa bi rezervacija trajala.
- Uporabnik podaljša rezervacijo. Uporabnik bi v osnovnem oknu izbral rezervirano parkirno mesto in podaljšal rezervacijo. Tako bi uporabnik lahko podaljšal rezervacijo in preprečil morebitno kazen ob neplačilu uporabe parkirnega mesta.
- Uporabnik izbere poljubni čas rezervacije. Trenutno se čas rezervacije ponuja v urah. Uporabnik bi tako lahko izbiral manjšo časovno enoto rezervacije parkirnega mesta in temu primerno plačal manj.
- Pri tehničnem delu smo omenili »real-time« rešitev celotnega sistema. Parkirna hiša bi lahko izdala kartice s čipom za identifikacijo pri vhodu in izhodu iz parkirne hiše, ki so omejene z zapornico. Pri tej rešitvi bi uporabnik na izhodu iz parkirne hiše, le-ta že imela podatek o opravljeni rezervaciji in preko kartice bi uporabnik imel omogočen izhod iz parkirne hiše. Druga rešitev bi namesto kartice s čipom lahko bila uporaba mobilnega telefona s klicem na določeno številko, kakor tudi deluje sistem Moneta. Torej »real-time« rešitev celotnega sistema opisuje možno realno rešitev problema opisanega v drugem poglavju.
- Uporabnik ima trenutno na voljo pregled nad samo trenutno veljavnimi rezervacijami. Nadgradnja bi bila, da bi v aplikaciji pregledoval mesečno porabo in tako imel pregled nad izstavljenim računom.

5. SKLEP

V diplomski nalogi smo predstavili rešitev za lažje parkiranje v večjih mestih. Dotaknili smo se tehnologij za izvedbo rešitve, v praktičnem delu pa prikazali implementacijo aplikacije Parkiranje.

Parkiranje je uporabniku enostavna in koristna aplikacija, ki nudi hitro iskanje in varno plačevanje storitve parkiranja. V določenih primerih, ko uporabnik prenosnega telefona že ima naročnino na prenosni telefon, ki vsebuje uporabo interneta, rešitev ne predstavlja dodatnega stroška za uporabnika te aplikacije oziroma storitve. Aplikacija je bila implementirana na uporabniku prijazen način, ki omogoča hitro in varno uporabe le-te. Izbrali smo razvoj aplikacije tipa MIDlet v okolju Java ME zaradi enostavnosti in dostopnosti razvoja v tem okolju. Poslužili smo se spletnih storitev, ki so obogatile funkcionalnost MIDlet-a, za varno uporabo spletnih storitev pa smo poskrbeli z uporabo protokola WS-Security, ki skrbi za varnost uporabe spletne storitve od začetka do konca.

Diplomska naloga poda tudi rešitev za uporabo protokola WS-Security v ogrodju Java ME, kar je bil tudi njen glavni namen. Pri tem smo si pomagali z dvema odprtokodnima knjižnicama za ogrodje Java ME. Knjižnica *Bouncycastle* je bila uporabna pri potrebnih šifrirnih postopkih, medtem ko smo knjižnico KSoap2 uporabili za izdelavo in pošiljanje SOAP sporočila, ki je v zaglavju vsebovalo potrebne elemente za realizacijo WS-Security protokola. Žal smo bili pri kreiranju sporočila SOAP omejeni, zato je bila potrebna predelava knjižnice KSoap2, kar pa ni predstavlja večjega problema, saj je knjižnica odprtokodna. Pri izdelavi SOAP sporočila je bilo potrebno dele SOAP sporočila tudi zapisati v pravilni kanonski obliki, kakor to zahteva protokol WS-Security.

Rešitev plačevanja parkirnine s pomočjo prenosnega telefona je poznano tako pri nas kot tudi v tujini. Hrvaški operater Tele2 je svojim uporabnikom omogočil plačevanje parkirnine preko SMS sporočil v 20 hrvaških mestih. Pred tem so plačilo take vrste že uvedli v Zagrebu. Postopek plačila je sledeč: uporabnik pošlje SMS sporočilo z registrsko označbo na številko, ki je napisana na parkirnem aparatu. Dokaj preprost postopek, ki pa je pri nas že poznan občanom Kranja, Ljubljane in Nove Gorice, kjer je plačevanje parkirnine z mobilniki že dolgo uveljavljena praksa. Parkirnino v Kranju in Novi Gorici se lahko enostavno plača s klicem na posebno telefonsko številko in sicer na parkirnem avtomatu se pritisne tipka GSM in uporabnik mora poklicati številko, ki se izpiše na zaslonu, parkomat izda listek oz. potrdilo o plačani parkirnini. Vendar za razliko od naše rešitve predstavlja pisanje SMS in klicanje eventuelno dodaten strošek in možnost napake pri uporabi te storitve, na primer, če napišemo napačno vsebino SMS-a. V Ljubljani se plačuje parkiranje s pomočjo plačilnega sistema Moneta. Rešitev v teh mestih ni univerzalna, ker je rešitev v Kranju vezana na ponudnika mobilnih storitev Simobil, medtem ko je v Ljubljani rešitev vezana na ponudnika mobilnih storitev Mobitel. Zgoraj opisane rešitve predstavljajo samo plačevanje parkirnine, ne pa tudi iskanja prostih pakirnih mest, kar nudi naša rešitev.

Aplikacija Parkiranje ima ob nadgradnji funkcionalnosti celotnega sistema, katere smo omenjali v poglavju o nadaljnjih možnostih, možnost za praktično uporabo. Celotna rešitev

seveda presega rešitev diplomske naloge, vendar pa kljub temu diplomska naloga nakazuje v smer, v kateri bi lahko poiskali primerno rešitev, ki bi z izkoriščeno dodatno uporabnostjo prenosnih telefonov olajšala parkiranje v mestih.

6. VIRI

[1] Gačnik Matevž, Arhitektura spletnih storitev, USMERITVE ZA NAČRTOVANJE, marec 2003

[2] Jurič Matjaž B., Pušnik Maja, SKLAD TEHNOLOGIJ ZA SPLETNE STORITVE, http://cot.uni-mb.si/cotl/pomlad2005/juric1.html, pomlad 2005

[3] Kozel Tomas, Slaby Antonin, Mobile Devices and Web Services, 7th WSEAS International Conference on APPLIED COMPUTER SCIENCE, Venice Italy, stran 323-324, november 21-23 2007

[4] Štimac Uroš, ”Zagotavljanje varnosti spletnih storitev”, Diplomska naloga, Univ Lj, FRI, 2003

[5] Verdonik Ivan, Varne spletne storitve, http://www.freeweb.siol.net/ivanver1/WSS.pdf, november 2004

[6] Canonical XML (Version 1.0), http://www.w3.org/TR/2001/REC-xml-c14n-20010315, februar 2007

[7] Introduction to J2ME Web Services, http://developers.sun.com/mobility/apis/articles/wsa, oktober 2008

[8] Odvečen strah pred sevanjem, http://www.dobrojutro.net/index.php?stran=arhiv&tip=8&id=9893, oktober 2008

[9] Prenosni telefon, http://sl.wikipedia.org/wiki/Prenosni_telefon, november 2008

[10] Simple Object Access Protocol (SOAP) 1.1, http://www.w3.org/TR/2000/NOTE-SOAP-20000508, oktober 2008

[11] Sun Developer Network, http://java.sun.com/javame/index.jsp, oktober 2008

[12] Web and XML glossary, Midlet, http://dret.net/glossary/midlet, november 2008

[13] XML-Signature Syntax and Processing, http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/, februar 2007

[14] XML Information Set, http://www.w3.org/TR/2001/REC-xml-infoset-20011024, februar 2007

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

diplomskega dela

Spodaj podpisani/-a Matjaž Auflič,

z vpisno številko 63960004,

sem avtor/-ica diplomskega dela z naslovom:

VARNE SPLETNE STORITVE NA MOBILNI PLATFORMI

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek) doc. dr. Mojca Ciglarič

  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 22.1.2009 Podpis avtorja/-ice: _____