

UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – praktična matematika (VSŠ)

Anže Germovšek

Priprava izmenjave podatkov med sistemom IMS in programom Microsoft Navision

Diplomska naloga

Ljubljana, 2006

Zahvala

Zahvaljujem se vsem, ki so mi na kakršenkoli način pomagali pri nastanku moje diplomske naloge. Posebna zahvala mentorju Matiji Lokarju za koristne napotke in predloge pri diplomski nalogi. Posebna zahvala gre tudi mojemu dekletu Tjaši Ciperle za vso podporo, ki mi jo je nudila.

Diplomsko nalogo posvečam svojim staršem in mojemu dekletu Tjaši Ciperle.

Fakulteta za matematiko in fiziko

Vsebinsko kazalo

1. UVOD	7
1.1. OPIS PROBLEMA	7
1.2. OZNAČEVANJE IN JEZIK SGML	9
1.3. KRATEK OPIS JEZIKA XML	10
1.3.1. Dobro oblikovani opisi	11
1.3.2. Pravilni opisi – DTD (Document Type Definition)	12
1.4. PROBLEMI RAZLIČNIH OBLIK DOKUMENTOV	16
1.5. SISTEM IMS	17
1.6. IZVAJANJE OBRAČUNA V SISTEMU IMS	23
1.7. APLIKACIJA MICROSOFT BUSINESS SOLUTIONS – NAVISION	24
2. IMPLEMENTACIJA PROGRAMA	25
2.1. NAČRTOVANJE PROGRAMA	25
2.2. RAZRED TREE MAP	39
2.3. BRANJE DATOTEK XML	42
2.3.1. Manjkajoča datoteka DTD	42
2.3.2. Nezaključene končne značke	43
2.3.3. Obdelava večjega števila datotek	44
2.3.4. Prebiranje datoteke XML	45
2.4. RAZRED RACUN	48
2.4.1. Komponente razreda Racun	50
2.4.2. Metode razreda Racun	52
ZAKLJUČEK	58
LITERATURA IN VIRI	59

Kazalo slik

Slike 1: Podatki o naslovu v sistemu IMS	19
Slike 2: Glavno okno sistema IMS	20
Slike 3: Dodajanje nove storitve v sistem IMS	22
Slike 4: Storitve, na katere je naročen naročnik.....	22
Slike 5: Spletna stran obračuna	23
Slike 6: Glavno okno programa Microsoft Navision	24

Program diplomske naloge

V diplomski nalogi predstavite način, kako pripravljamo podatke za izmenjavo med programoma IMS in Microsoft Navision. Pri tem na kratko povzemite osnove standardnega jezika za izmenjavo podatkov XML in opišite probleme, ki se dogajajo pri preurejanju (pregledovanju) datotek v formatu XML. Diplomska naloga naj zajema tudi konkreten primer branja datoteke v formatu XML s pomočjo programskega jezika Java.

mentor

mag. Matija Lokar

Povzetek

V svoji diplomski nalogi sem predstavil praktičen primer pretvorbe dokumentov iz ene oblike dokumentov v drugo.

Na kratko sem povzel standard XML. Ta se vse bolj uporablja za izmenjavo podatkov, ker ni odvisen od operacijskega okolja, niti od programskih jezikov. Na primeru, kjer sem urejal objekte imenovane Racun, sem prikazal primer obdelave datotek XML s pomočjo programskega jezika Java. Te objekte sem iz podatkov v datotekah XML, kakršne dobim iz sistema IMS, priredil obliki za uvoz v program za obračunavanje Microsoft Navision.

Pri tem sem obravnaval probleme, ki nastanejo pri preoblikovanju (parsanju) datotek v formatu XML ter probleme različnih oblik dokumentov, ki jih zahtevajo določeni programi. Tako je oblika podatkov, ki smo jo potrebovali za izdelavo uvoznih datotek z računi v program MIT, opremljena z vsemi podatki, tako imeni, priimki, naslovi, podatkih o storitvah in ostalih pomembnih podatkih. Program Microsoft Navision, s katerim smo nadomestili program MIT, pa uporablja preprostejšo obliko uvozne datoteke z računi, kjer poleg ključa uporabnika povemo samo podatke o storitvah.

Math. Subj. Class. (2000): 68N15, 68N19, 68P05, 68T20

Computing Review Class. System (1998): D.1.5, D.2.4, D.3.3, D.3.4, D.4.3, D.4.4, E.1, E.2

Ključne besede: XML, DTD, sistem IMS, aplikacija Microsoft Business Solutions - Navision

Keywords: XML, DTD, system IMS, application Microsoft Business Solutions - Navision

1. Uvod

1.1. Opis problema

V podjetju Voljatel Telekomunikacije d.d. je moja naloga izdelovanje datotek z neknjiženimi prodajnimi računi v obliki, primerni za uvoz v program Microsoft Business Solutions Navision.

Iz obračunskega sistema IMS vsak drugi dan v mesecu dobim datoteke v formatu XML. V teh datotekah so podatki, ki zajemajo naročnika in storitve, na katere je naročen. Take storitve so lahko naročnine, ki se zaračunavajo za mesečno obdobje, ki prihaja ali pa za obdobje, ki je že mimo, različne priključnine, ki se morajo zaračunati naročniku, poraba telefonskih klicev, minutna internetna poraba, ... Storitve so v sistemu IMS poimenovane Produkti, v programu Microsoft Navision pa Viri.

Moja naloga je, da iz teh datotek XML izdelam datoteke za uvoz v obračunski program Microsoft Business Solutions Navision. Ta ima za uvoz podatkov točno določeno obliko. Iz obračunskega sistema IMS poberem datoteke XML. Te se nahajajo v točno določeni mapi, ki nastane pri vsakem obračunu. V posamezni datoteki XML je ponavadi 50 računov ali manj. V datotekah XML so za posameznega naročnika zabeleženi podatki o naročnikovih naslovih, naročenih paketih storitev in opravljenih posameznih storitvah. S svojim programom se sprehodim čez vse datoteke XML. Iz datotek XML preberem le določene podatke, ki so pomembni za oblikovanje uvoznih datotek za program Microsoft Navision. Zaradi lažjega uvoza računov pri zapisovanju računov preverim določene kriterije, ki olajšajo delo osebi, ki račune dejansko uvozi v program Microsoft Navision. Ti kriteriji med drugim preverjajo, če je mogoče naročnik, na katerega se račun nanaša, zaradi neplačil ali drugih razlogov blokiran v programu Microsoft Navision. Pri dejanskem uvozu takega računa v program Microsoft Navision se izpiše sporočilo, da računov blokiranim naročnikom ne moremo zaračunati. Kot drug kriterij preverim, ali v programu Microsoft Navision naročniška številka obstaja in račune, ki pripadajo neobstoječim naročniškim številkam, razvrstim v posebno datoteko. Nato preverjam še določene druge kriterije in razvrščam račune po prvi storitvi, ki se zaračuna naročniku. To je pomembno tudi zato, ker obstaja nekaj storitev, ki se ne smejo zaračunati. Na ta način jih lahko zapišem v posebno datoteko.

Največji problem se pojavi pri obračunavanju telefonskih klicev, ker je v datotekah XML premalo podatkov za pripravo podatkov v obliki, kot jih zahteva Navision. Zato moram za vsakega naročnika, ki je uporabljal telefonske storitve, za dodatne podatke o njegovih klicih pogledati v bazo podatkov Oracle.

Problem je tudi v tem, da se pri obračunavanju uporabljajo različni ceniki. Občasno se izvajajo različne promocijske akcije s posebnim načinom obračunavanja, ki lahko trajajo del obračunskega obdobja. Primer takih akcij sta na primer:

- Akcija z imenom »naj traja dlje«. Pri tej akciji se za posamezen klic prve 3 minute pogovora zaračunavajo po normalni ceni, druge 3 minute po znižani tarifi1, vse ostale minute po znižani tarifi2.
- IP telefonija, kjer je prvih 120 minut klica na določeno destinacijo brezplačnih.

Pri izdelavi obračuna je potrebno vse te akcije upoštevati, upoštevati obdobje, v katerem veljajo in podobno. Zato je sam obračun lahko dokaj zapleten.

Pri urejanju in dopolnjevanju podatkov se lahko pojavijo določene napake. Te napake v podatkih moram odkriti. Take napake so:

- Manjkajoče šifre in imena storitev v datoteki »cs_id_name.txt«, kjer so zapisana prava imena storitev. Do težav pride, ko s pomočjo šifre storitve poskušam najti ime storitve, ki mora biti izpisana na računih, ta storitev pa še ni vnešena v omenjeno datoteko. Velikokrat se zgodi, da je odgovorna oseba v sistem IMS dodala novo storitev in mi tega ni sporočila. Zato v datoteki storitve ni. Napako sem začasno rešil na ta način, da pred glavnim obračunom iz datotek XML preberem vse šifre storitev, ki v njih nastopajo. Imena storitev, katerih šifre ne obstajajo v datoteki, dobim tako, da ime paketa storitev in opis storitve preberem iz sistema IMS. Na ta način dobim v datoteko vse šifre in imena storitev. Vendar imena pogosto niso primerne oblike za izpis na računih, saj so velikokrat zapisana angleška imena kot npr. Montly fee namesto naročnina. Zato pred obračunom odprem datoteko in ročno preoblikujem imena v primerna.
- Klici v nove destinacije. Problem se pojavi, ko urejam telefonske klice. V datoteki »IMS_slo_imena_drzav.txt« so shranjena angleška imena destinacij in njihov prevod v slovenska imena. Administrator, ki ureja telefonske destinacije, občasno doda novo destinacijo in mi tega ne sporoči. Zaradi načina organizacije dela se je pokazalo, da je najenostavnejša rešitev ta, da na račun enostavno zapišem angleško ime in na zaslon izpišem opozorilo. Nato po končanem obračunu ročno popravim morebitne napake (napačna imena) v računih in v datoteko dodam destinacije, ki so izpadle. Tako naslednjič s temi destinacijami ne bo več problema.
- Napačni datumi za določene storitve. Vedno, ko se pojavijo novi paketi storitev, moram ročno pregledati izdelane račune in preveriti, ali so se storitve zaračunale za pravilno obdobje. Paketi storitev v sistemu IMS namreč občasno niso narejeni pravilno. Običajna težava je v odločitvi, ali se storitev zaračuna za preteklo ali tekoče obdobje. Težavo rešim tako, da po končanem obračunu odprem eno izmed datotek, v kateri so nove storitve in pregledam, za katero obdobje so zaračunane. V primeru napak jih ponovno ročno odpravim. Vendar moram v programu poskrbeti, da so storitve primerno razvrščene v datoteke, saj na ta način lažje najdem take na novo vpeljane storitve.
- Večkratno zaračunavanje. Določeni naročniki imajo naročenih več enakih telefonskih storitev. Posamezni klici se jim zaračunajo za vsako od teh storitev, čeprav bi se morali računati le ob eni storitvi. Iz izdelanih datotek je potrebno odstraniti odvečne postavke na računih. Iz datotek XML dobim podatek o končni vsoti zaračunane storitve za posameznega naročnika. Vsakič, ko pregledujem klice za določeno storitev v bazi Oracle, zapišem skupen znesek v izhodno datoteko »telefonija.csv«. V tej so torej po končanem obračunu podatki za vsakega naročnika, ki je opravil kakšen telefonski klic v sklopu posamezne telefonske storitve. Iz te datoteke preberem, katerim naročnikom so se obračunale večkratne storitve in podvojene postavke ročno odstranim iz izvoznih računov. Tega nisem storil programsko, ker je dejansko največ pet takih strank in je na pol ročna rešitev dejansko hitrejša.
- V datoteki XML se za določenega naročnika pojavi previsok ali pa prenizek znesek za storitev. Zaradi različnega načina zaračunavanja storitev, blokiranja in ponovnega aktiviranja računov, ter načina, kako se te zadeve beležijo v sistemu IMS, se lahko zgodi, da se znesek računa ne ujema s pričakovanim (ki je običajen za določenega naročnika). V takem primeru tovrstne račune zapišem posebej, kjer se potem še enkrat pregledajo.

Pri izdelavi računov je pomembno tudi razporejanje računov v ustrezne datoteke. Tako denimo zapišemo v eno datoteko račune, ki se nanašajo na storitve ADSL, v drugo račune, ki

so povezani s telefonskimi storitvami. V tretjo zapišemo račune za kabelske storitve, v četrto pa račune, ki se nanašajo na tiste naročnike, ki so blokirani v programu Microsoft Navision. Spet v peto datoteko damo take račune, kjer podatki o naročniku v programu Microsoft Navision ne obstajajo in tako naprej. Na ta način omogočimo lažji uvoz računov v program Microsoft Navision. Tu moram opozoriti, da je to le delno razporejanje. Račun za enega naročnika, če ima naročnik več storitev, se razporedi le glede na prvo storitev, ki je bila dodana naročniku. Tako razporejanje je ključnega pomena za lažje uvažanje računov, ker za uvoz v program Microsoft Navision skrbita dva skrbnika.

Ob izdelavi datotek z računi se v posebne datoteke pišejo opombe. Ko so datoteke z računi izdelane, pregledam vse opombe, ki so se izpisale med izvajanjem obračuna in popravim morebitne napake. Nato vsako od datotek z računi še ročno pregledam. S tem poskrbim, da ni prišlo do kritične programske napake tako pri datumih kot pri imenih.

V sistemu IMS in v datotekah XML se uporabljajo angleški izrazi, kot npr. `account_id`, `cs_id`, `product_id` in tudi drugi. Da pri pisanju programov ne bomo imeli zmede, bomo tudi mi največkrat uporabljene parametre v programu poimenovali tako, kot so poimenovana ta polja. Ostala imena spremenljivk, metod in drugega pa bodo slovenska, kot bodo slovenski tudi komentarji. Razmišljal sem, da bi vse poslovenil ali pisal vse v angleščini, a menim, da je zgoraj opisana rešitev najbolj smiselna in uporabna.

Tako se v programih uporabljajo naslednji angleški izrazi:

- `account_id` - naročniška številka
- `cs_id` - šifra storitve
- `product_id` - ključ paketa storitev

Ko so računi pregledani, je potrebno na podlagi teh računov pridobiti tudi ustrezne statistične podatke. Prav tako je potrebno pregledati vse izpadle račune ter ugotoviti razloge, zaradi katerih računi niso bili izdelani.

1.2. Označevanje in jezik SGML

En od pomembnejših pojmov pri delu s podatki postaja označevanje. Dele besedila oklenemo z značkami, ki določajo, kaj dani del besedila predstavlja. Pogosto se označevanje uporabi tudi za to, da se na enoten način oblikuje enakovrstne dele podatkov (npr. besedila). Čeprav vsi opisi oblikovanega besedila temeljijo na nekakšnem označevanju, predstavlja prelomnico pojav označevalnega jezika SGML (Standard Generalized Markup Language). Ta je bil standardiziran kot standard ISO leta 1986. SGML omogoča hranjenje, izmenjavo in lažjo obdelavo podatkov. Namenjen je opisu zgradbe podatkov (slovnica, sintaksa), ki je ločen od opisa oblike (prikaza oziroma drugih učinkov). Oblika je določena s slogi.

Vendar je SGML prekompleksen, da bi se dejansko uporabljal kot tak. Običajno se uporablja kot sestav za pripravo definicij bolj specializiranih označevalnih jezikov. Ti se omejujejo na določeno ožje področje podatkov.

Primeri takih jezikov so:

- HTML – Hyper Text Markup Language,
- ISO 12083 – Electronic Manuscript Standard,
- TEI – Text Encoding Initiative,
- CALS/JTA – Computer-aided Acquisition and Logistic Support,

- NITF – News Industry Text Format
- DDI – Data Documentation Initiative
- CML – Chemical Markup Language

Z razvojem svetovnega spleta se je pojavila potreba, da lahko uporabnik razširi HTML s svojimi značkami. Prirejena izpeljanka SGMLja je XML – Extensible Markup Language. XML je ohranil pomembnejše zmogljivosti in glavne značilnosti SGMLja.

1.3. Kratek opis jezika XML

Ker aplikacija, ki jo opisujem, temelji na prebiranju datotek v formatu XML, je smiselno, da si na kratko ogledamo nekaj osnov tega jezika.

XML je standardiziran jezik za opis strukturiranih podatkov. Njegova uporaba hitro narašča, saj omogoča skupno rabo podatkov v spletu. Zaradi standardiziranosti in formata, ki je vedno berljiv, odpade skrb za nezdružljive programe, računalniška omrežja, podatkovne strukture in operacijske sisteme. XML torej olajšuje skupno rabo podatkov.

Datoteka v formatu XML je običajna tekstovna datoteka. Zato je berljiva v vseh operacijskih sistemih.

XML uporablja standardne oznake za določanje strukture in vsebine datoteke. Ker so v vseh datotekah oznake XML enake, to omogoča, da učinkovito indeksiramo, iščemo, združujemo in ponovno uporabljamo podatke, ki temeljijo na besedilu.

XML nam omogoča, da dokument označimo z množico značk ("tags"). Te so zelo podobne značkam iz HTML. Na datoteko, zapisano v jeziku HTML, namreč lahko (če odmislimo določene nepravilnosti, ki jih dopušča HTML, kot so nezaključene značke in podobno) gledamo kot na datoteko, zapisano v XML. Vendar v nasprotju s HTML, ki uporablja omejen, standarden nabor značk, v jeziku XML značke določamo kar sami.

Značke so lahko:

- **Strukturne** – te nam povejo, kako je dokument strukturiran. Tako označimo del besedila, ki je naslov, odstavek, razdelek, ... Primer strukturnih značk so dobro poznane značke iz jezika HTML <H1>, <P> in <DIV>.
- **Oblikovne** – te nam povejo, kako naj bo dokument oblikovan. Oblikovne značke so na primer <I>, in <SMALL>.
- **Semantične** - te nam povejo nekaj o vsebini besedila. Primera semantičnih značk v HTML-ju sta <TITLE> in <CODE>.

Kot smo omenili, je HTML nekakšen »podjezik« XML-ja. Vendar je njun namen precej različen. Jezik HTML je namenjen oblikovanju in prikazovanju dokumentov v spletu. Ni pa uporaben, da bi z njim razbirali podatke iz spletnih strani.

Največ poslovnih dokumentov je takšne narave, da je vsebino dokumenta potrebno razbrati, jo obdelati ter nazadnje še shraniti. Potrebno je vedeti, kateri del dokumenta predstavlja naslov, kateri datum, kateri znesek računa in podobno. Kaj takšnega je s HTML-jem skoraj nemogoče narediti, v XML pa je to prav enostavno.

Kakor v HTML, se v XML značke lahko pojavijo:

- v parih – oklepajne značke
- posamično – samostojne značke

Del opisa, ki ga nek par značk oklepa, je vsebina te značke. Vsebina značk je lahko:

- prazna,
- enostavna (niz znakov, ki ne vsebuje drugih značk),
- sestavljena.

XML dopušča poljubno izbiro imen značk, razen tistih, ki so rezervirana imena. S tem omogočimo lažjo razumljivost in pomen značk. Imena značk so nizi znakov, ki ne vsebujejo presledkov ali dvopičij in se ne začnejo s številko, ločilom ali podnizom xml (Xml, XML, ...).

Značka ima lahko tudi lastnosti:

`<ime lastnost1 = "vrednost1" lastnost2 = "vrednost2" ... lastnostk = "vrednostk">`

Z lastnostmi značk določimo, kakšne informacije vsebuje značka, itd.

1.3.1. Dobro oblikovani opisi

Označeno besedilo je dobro oblikovano, če zadošča naslednjim zahtevam:

- oklepajne značke nastopajo v parih `<ime>` in `</ime>`; samostojne značke pa imajo obliko `<ime />` ali enakovredno `<ime></ime>`,
- v opisu podatkov so značke gnezdene – zaporedje `<a>...<b>...</a>...</b>` ni dovoljeno,
- v imenih značk se upošteva velikost črk,
- vsak opis mora oklepiti glavna, oziroma kot ji tudi rečemo, korenska, značka,
- vrednosti lastnosti so v navednicah " ali '.

Večkratni presledki se ne skrčijo v enega kot v HTML. Nova vrstica je predstavljena z znakom LF. Kot v HTML pojasnila oklenemo v `<!--...-->`.

Dobro oblikovan opis lahko prestavimo kot drevo – DOM (Document Object Model). Podatke lahko v opisu predstavimo kot vsebino neke značke ali kot vrednosti njenih lastnosti. Priporočilo pravi, da prave podatke predstavimo kot vsebino. Lastnosti pa naj povedo podatke o podatkih.

Novejše izdaje spletnih pregledovalnikov Internet Explorer (vsaj 5.5) in Mozilla/Netscape (vsaj 7) omogočajo preverjanje dobre oblikovanosti opisov in ličen (določen z prekrivnimi slogi) prikaz dobro oblikovanih opisov.

Primer dobro oblikovanega opisa

```
- <SeznamStudentov>
- <Student vpisna_st="12345678" ime="Anže" priimek="Germovšek">
  - <Zivljenjepis>
    Rodil se je dne 18.10.1982 v Kranju. Osnovno šolo je obiskoval v OŠ Križe. Po uspešni končani OŠ se
    je vpisal na Srednjo biotehniško šolo v Kranju in jo tudi uspešno končal. Po končani srednji šoli se je
    vpisal na Fakulteto za Matematiko in Fiziko.
  </Zivljenjepis>
</Student>
<Student vpisna_st="8765431" ime="Tjaša" priimek="Ciperle"/>
</SeznamStudentov>
```

1.3.2. Pravilni opisi – DTD (Document Type Definition)

Zaenkrat še nismo povedali, kateri opisi so 'slovnično' pravilni – v kakšnih medsebojnih odnosih so lahko posamezne značke, katere lastnosti imajo in podobno.

Slovnico označevanega jezika, s katerim označimo določen dokument, določimo z DTD (Document Type Definition). Z njim povemo, kako so posamezne značke, med seboj povezane. Pri zapisu slovnice uporabljamo naslednje stavke:

➤ ELEMENT

Značko vpeljemo s stavkom `<!ELEMENT ime_elementa kontekst>`, kjer je `ime_elementa` ime vpeljane značke in je `kontekst` določen z eno od naslednjih možnosti:

- **EMPTY** – samostojna značka, primer je značka `<hr>` iz jezika HTML
- **CDATA** – večinoma se uporablja za elemente, ki predstavljajo programsko kodo.
- **ANY** – poljubno zaporedje znakov
- **(#PCDATA)** – niz znakov – zaporedje znakov (brez značk);
- (členi) določilo – členi sestavljajo bodisi zaporedje bodisi izbiro. Zaporedje sestavljajo takrat, ko so ločeni z vejico. Tudi v opisu se morajo pojaviti v istem vrstnem redu. Med členi izbiramo, kadar so ločeni s črtico |. Pri izbiri se mora v opisu pojaviti le en izmed njih. Določilo je ali prazen znak ali pa eden izmed znakov ?, + ali *. Posamezni člen je ime podrejene značke (otroka). Če mu sledi ? (na primer `ime?`) je dovoljena največ ena pojavitev značke. Oblika `ime+` pove, da je obvezna vsaj ena pojavitev; oblika `ime*` pa označuje poljubno (tudi nič) pojavitev.

Posamezni znački ustreza natanko en stavek ELEMENT. Oglejmo si nekaj primerov definicij značk. Pri vseh definiramo značko A:

- `<!ELEMENT A (B, C, D)>`
Značka A mora vsebovati elemente B, C, D v tem vrstnem redu
- `<!ELEMENT A (B | C | D)>`
Značka A mora vsebovati natanko enega izmed elementov B, C ali D
- `<!ELEMENT A (B?)>`
Značka B se mora pojaviti enkrat ali nikoli
- `<!ELEMENT A (B*)>`
Značka B se lahko pojavi ničkrat ali večkrat
- `<!ELEMENT A (B+)>`
Značka B se mora pojaviti vsaj enkrat

Preprost del primera XML

```
<datum>
  <dan>9</dan>
  <mesec>2</mesec>
  <leto>2000</leto>
</datum>
```

Njegov DTD opis

```
<!ELEMENT datum (dan, mesec, leto)>
<!ELEMENT dan (#PCDATA)>
<!ELEMENT mesec (#PCDATA)>
<!ELEMENT leto (#PCDATA)>
```

V DTD-ju smo določili:

- XML značka `<datum>` je sestavljena iz značk `<dan>`, `<mesec>`, `<leto>`.
- Značke `<dan>`, `<mesec>`, `<leto>` znotraj značke `<datum>` si sledijo v natančno takšnem zaporedju, kot so zapisane.
- Značka `<dan>` je niz znakov.
- Značka `<mesec>` je niz znakov.
- Značka `<leto>` je niz znakov.

Na nekaj primerih si oglejmo še združevanje operatorjev:

- `<!ELEMENT A (B?, C)>`
Značka A ima lahko B, ki mu mora slediti C.
- `<!ELEMENT A (B+, C+)>`
Značka A mora imeti vsaj en B in vsaj en C (toda ne mešano).
- `<!ELEMENT A (B?, C+)>`
Značka A lahko ima B, ki mu sledi en ali več C-jev.
- `<!ELEMENT A (B, C)*>`
Značka A lahko ima poljubno število B-jev, za vsakim B pa sledi C.
- `<!ELEMENT A (B | C)*>`
Značka A lahko ima poljubno število B-jev in C-jev (v poljubni kombinaciji).

➤ ATTLIST

Vsaka značka ima lahko poljubno število lastnosti. Poglejmo primer XML značke z dvema lastnostima.

```
<značka lastnost1 = "vrednost lastnosti1" lastnost2 = "vrednost lastnosti2">
vrednost značke
</značka>
```

Definicija lastnosti je preprosta. Povedati je potrebno:

- kateri znački lastnost pripada,
- ime lastnosti,
- tip lastnosti in
- vrsto lastnosti

Ko lastnost definiramo, uporabimo naslednji stavek

```
<!ATTLIST ime_značke ime_lastnosti tip_lastnosti vrsta_lastnosti>
```

Zaradi preglednosti lastnosti praviloma definiramo takoj za definicijo značke, ki ji pripadajo in jih zamaknemo. Kot primer si oglejmo deklaracijo lastnosti `naslov`, ki pripada znački `skok`.

```
<!ELEMENT skok (#PCDATA)>
  <!ATTLIST skok naslov CDATA #REQUIRED>
```

Drug način deklariranja lastnosti pa je ob sami deklaraciji značk. Takrat znotraj značke deklariramo novo značko, ki predstavljajo lastnosti. Teh značk je lahko več, tako lahko definiramo več lastnosti. Tega prijema se nasplošno izogibamo in za lastnosti uporabljamo zgornjo definicijo.

Primer deklariranja lastnosti v XML

Deklaracija lastnosti znotraj značke

```
<Anze koncuje_diplomo="Ja">  
</Anze>
```

Deklaracija lastnosti, kot notranja značka

```
<Anze>  
    <koncuje_diplomo>Ja</koncuje_diplomo>  
</Anze>
```

IME LASTNOSTI

Pri imenih lastnosti moramo biti pozorni na naslednja pravila:

- Lahko je sestavljeno iz celotnega nabora znakov in števk.
- Prvi znak lastnosti ne sme biti števka ali podčrtaj.
- Ne sme se začeti z nizom XML.
- V imenu ne sme biti presledkov.
- Lahko vsebuje tudi neangleške črke (primer č, š in ž), a to ni priporočljivo zaradi različnih sistemov.

TIP LASTNOSTI

Lastnost lahko pripada enemu izmed množice tipov:

- **ID** – vrednost je identifikacijska značka
- **CDATA** – vrednost vsebuje niz znakov
- **IDREF** – vrednost je identifikacijska oznaka neke druge značke
- **IDREFS** – vrednost je seznam identifikacijskih oznak drugih značk
- **ENTITY** – vrednost je entiteta
- **ENTITIES** – vrednost je seznam entitet
- **NMTOKEN** – vrednost je veljavno XML ime
- **NMTOKENS** – vrednost je seznam veljavnih XML imen

Običajno uporabljamo tip CDATA, s katerim povemo, da se kot vrednost lahko uporabi poljuben niz znakov ali pa lastnosti vnaprej predpišemo veljavne vrednosti. To naredimo tako, da jih preprosto naštejemo.

Primer lastnosti, katere možne vrednosti so določene že vnaprej

```
<!ATTLIST izhod enota (zaslon | tiskalnik | datoteka) "zaslon">
```

Značka izhod ima torej lastnost enota. Ta lahko zavzame vrednosti zaslon, tiskalnik ali pa datoteka. Privzeta vrednost (glej razlago v nadaljevanju) je zaslon.

VRSTA LASTNOSTI

Vrsta lastnosti je zelo pomembna, saj z njo med drugim lastnosti določimo, ali je obvezna ali ne. Možnosti so sledeče:

- **DEFAULT** – določimo privzeto vrednost lastnosti. Če uporabnik v XML dokumentu vrednosti te lastnosti ne določi, se privzame zapisana vrednost (besedico #DEFAULT lahko izpustimo)

```
<!ELEMENT pismo>
  <!ATTLIST pismo posiljatelj CDATA #DEFAULT "Anže Germovšek">
```

- **REQUIRED** – lastnost v znački je obvezna

```
<!ELEMENT pismo>
  <!ATTLIST pismo prejemnik CDATA #REQUIRED>
```

- **IMPLIED** – lastnost v znački ni obvezna. To pomeni, da jo lahko pri znački navedemo, ali pa ne.

```
<!ELEMENT pismo>
  <!ATTLIST pismo datum CDATA #IMPLIED>
```

- **FIXED** – lastnosti v znački določimo fiksno (nespremenljivo) vrednost

```
<!ELEMENT diploma>
  <!ATTLIST diploma mentor CDATA #FIXED "mag. Matija Lokar">
```

Definicija značk in njihovih lastnosti predstavlja večino stavkov slovnice. Srečamo pa tudi definicije entitet.

➤ ENTITY

Na entitete lahko gledamo kot na okrajšave (kratice). Entitete so posebni znaki, kot so < in & in podobni. S pomočjo kratic lahko v XML zapišemo posebna znaka < in &, ki ju sicer ne bi mogli. To sta le posebna primera entitet, saj lahko entiteta predstavlja poljuben niz znakov. Poglejmo si primer.

Definicija entitete

```
<!ENTITY avtor "Anže Germovšek">
```

Uporaba v XML

```
... Avtor teksta je &avtor; ...
```

Lahko pa entiteta vsebuje tudi poljuben, dobro formuliran niz znakov v obliki XML.

```
<!ENTITY avtor "<poudari>Anže Germovšek</poudari>">
```

Entiteta lahko vsebuje povezavo na datoteke. Datoteke na računalniku ali na spletu deklariramo z naslednjo deklaracijo

```
<!ENTITY kratica vrsta "naslov" >.
```

Pri tem kot vrsto navedemo

- **SYSTEM**, če gre za lastne datoteke,
- **PUBLIC** pa uporabimo, kadar gre za javno datoteko.

Pri kodiranih datotekah na koncu stavka ENTITY dodamo še par NDATA oblika in jo s stavkom <!NOTATION oblika SYSTEM "program"> povežemo z ustreznim prikazovalnim programom. Tako z

```
<!ENTITY slikaBika PUBLIC "bull.jpg" NDATA jpgDat>
<!NOTATION jpgDat SYSTEM "irfanView.exe">
```

povemo, da oznaka slikaBika predstavlja datoteko bull.jpg, ki jo odpiramo s programom irfanView.exe.

1.4. Problemi različnih oblik dokumentov

Različni programi, ki obdelujejo določene skupne podatke, zahtevajo, da so ti podatki pripravljeni v različnih oblikah. Tako smo pri izdelavi datotek uvoznih prodajnih računov sprva uporabljali obliko, primerno za uporabo v programu Mit. Ta je zahteval, da so bili kot vhodni podatki pri vsakem posameznem računu navedeni vsi podatki: imena, naslovi, opisi, cene, obdobja, za katere naj bi izstavili račune in podobno.

Nato smo zaradi novih potreb knjiženja in statistik začeli uporabljati Microsoft Navision. Tu se je vnos podatkov poenostavil. Pri vhodnih podatkih ni bilo več potrebno navajati vseh podrobnosti o naročniku. Program namreč s pomočjo ključa (polja `account_id` iz sistema IMS) v svoji bazi naročnikov poišče ustrezne podatke za izpis na računih. Potrebno je le podati postavke računov, torej storitve, na katere je naročnik naročen in jih lahko uporablja.

Tako je definicija računa za uvoz v program Microsoft Navision videti takole:

```
1|ISP koda|IMS account id|
2|številka vira|opis vira|začetni datum|končni datum
2|številka vira|opis vira|začetni datum|končni datum
3|začetno obdobje|končno obdobje|datum izdelave računa
4|opis obdobja|cena brez ddv|količina|merska enota|
4|opis obdobja|cena brez ddv|količina| merska enota|
5|šifra artikla|lokacija|opis|cena brez ddv|količina|merska enota|popust(%)
```

Opis zgornje definicije

Ločitveni znak med podatki v posamezni vrstici datoteke, ki jo izdelujemo pri obračunu, je ravna navpična črta »|«.

Ker je v datoteki, ki jo uvažamo, lahko večje število računov, imamo na začetku vrstic ločitvene znake. Prvo polje je rezervirano za oznako vrstice. Oznaka je prvi znak na začetku vrstice. Oznaka je obvezna in ima lahko vrednosti 1, 2, 3, 4 ali 5. Vse ostale vrstice, ki se začnejo s kakršnimkoli drugim znakom, so izpuščene.

Vrstica, ki se začne z znakom 1, nam pove podatke glave računa

- ISP koda - Internet services provider (ponudnik internetnih storitev). V račune vedno vpišem konstanto »23«, ki predstavlja šifro podjetja Voljatel. Zaenkrat uvažamo le račune, ki jih izdaja podjetje Voljatel.
- IMS account id - ključ naročnika v sistemu IMS in hkrati povezovalni člen z Microsoft Navision. Ta podatek določa, kateremu naročniku bomo zaračunali storitve. Ta podatek je vedno sedemmestni niz.

Vrstica z znakom 2 nam pove postavke računa

- Številka vira - šifra storitve
- Opis vira - opis storitve
- Začetni datum - začetni datum obdobja za katerega se naročniku obračunava storitev
- Končni datum - končni datum obdobja za katerega se naročniku obračunava storitev

Vrstica z začetkom 3 vsebuje datume

- Začetni datum – nepomembno (se pri uvozu spusti)
- Končni datum – nepomembno (se pri uvozu spusti)

- Datum izdelave računa – datum, do katerega je potrebno uvoziti, knjižiti in izdati račune. V vrstici, ki se začne z znakom 4, so dodatni podatki o postavkah računa
- Opis obdobja – opis, ki ga dodamo zraven postavk računa
- Cena brez DDV – znesek posamezne storitve, na primer pri telefonskih klicih je to cena minutne postavke
- Količina – količina storitve, na primer pri telefonskih klicih je to število minut, ki jih je uporabnik porabil s klicem na posamezno destinacijo
- Merska enota – enota za katero obračunavamo storitev. Vrednosti niza so lahko:
 - »mes« - mesec, kar se uporablja pri naročinah,
 - »min« - minute, ki jih merimo pri telefonski in internetni porabi,
 - »kos«, ki se uporablja za vse priključnine

Vrstica z znakom 5 nam pove podatke o artiklu

- Šifra artikla – vsi artikli imajo svoje enolične šifre.
- Lokacija – pove, kje se nahaja določen artikel. Zavzame lahko le dve vrednosti:
 - »naše«, ki pove, da se artikel nahaja v Voljatelovem skladišču ali
 - »tuje«, ki pove, da se artikel nahaja v drugem skladišču.
- Opis – opis artikla, ki bo izpisan na postavkah računa.
- Cena brez DDV – znesek cene artikla.
- Količina – količina artikla. Ta je vedno celo število, saj se artikli vedno prodajajo kot celota in ni možno kupiti dela artikla.
- Merska enota – je vedno »kos«.
- Popust(%) – popust na artikel, ki je celo število med 0 in 100 in predstavlja % popusta.

Ob prehodu na nov program je seveda prišlo do problema, ker so bili podatki pripravljeni za prejšnjo obliko. Ta pa je vsebovala druge podatke kot so: ime, priimek, naslov, pošta, mesto, davčna številka in ostale podatke. Zato je bilo potrebno podatke, oblikovane, kot jih je zahteval program Mit, preoblikovati v obliko, kot jih pričakuje Microsoft Navision.

1.5. Sistem IMS

Sistem IMS je namenjen vzdrževanju baze Voljatelovih naročnikov. V sistemu IMS za vsakega naročnika vodimo podatke o tem, na katere storitve je naročen.

V sistemu IMS imamo torej podatke o naročnikih, paketih storitev, ki jih uporabljajo in podobno. Prav tako se s pomočjo pomožnih programov v ta sistem stekajo vsi pomembni podatki o tem, koliko je določen naročnik uporabljal posamezne storitve. Tako se v sistem IMS za vsakega naročnika stekajo podatki o prenesenih internetnih paketih, izvedenih telefonskih klicih in podobno. Te storitve se obračunavajo sproti.

Poleg tega v sistemu IMS administriramo in osvežujemo podatke o cenikih, ki se uporabljajo pri zaračunavanju telefonskih klicev in dodajamo nove pakete storitev z ustrezno funkcionalnostjo.

IMS je sistem, katerega glavna naloga je naročnikom zagotavljati funkcionalnost storitev. Če na primer želimo naročniku zagotoviti dostop do povezave ADSL, mora administrator naročnika obvezno vpisati v sistem IMS in mu dodati paket ADSL storitev. Ko administrator doda paket storitev, se v ozadju sistema aktivirajo pomožni programi. Ti aktivirajo dostop do storitve z ustreznim uporabniškim imenom, rezervira se ustrezeni prostor za poštna sporočila, prostor za objavo spletne strani in podobno.

V sistemu IMS vsakega drugega dne v mesecu izvedemo obračun, ki ustvari podatke v obliki datotek XML. Pri tem obračunu se v datoteke poleg podatkov o porabi določene storitve zapišejo tudi mesečne naročnine za storitve. Žal je v teh datotekah premalo podatkov za ustrezno specificirane račune. To se zaenkrat odraža samo pri telefonskih klicih, ker so v datotekah XML le podatki o stroških klicev na različne destinacije, ni pa zavedeno število porabljenih minut in minutna postavka.

Te podatke potem obdelamo in ustvarimo uvozne datoteke, v katerih so računi. Te datoteke s pomočjo uvoznega modula, ki je bil razvit posebej v ta namen, uvozimo v program Microsoft Navision. Pri tem ta potrebne podatke o naročniku (ime, priimek, naslov in ostalo) pridobi iz svoje baze. V program Microsoft Navision uvozimo neknjižene račune, ki jih pred izdajo računov pregledata, uredita in dopolnita ustrezni osebi, ki skrbita za izdajo računov.

Pri programu Microsoft Navision gre predvsem za računovodstvo. Tu se izvaja knjiženje računov ter zavajanje opomb, reklamacij in drugih podatkov o naročnikih.

Trije poglavitni sklopi podatkov, vsebovani v sistemu IMS, so:

- Naslovi
- Osnovni podatki o naročniku
- Storitve

Oglejmo si, kakšne podatke vodimo v posameznem sklopu.

Naslovi

V sistemu IMS o naročnikih vodimo štiri vrste naslovov:

- Osnovni naslov
- Plačniški naslov
- Najemnikov naslov
- Začasni naslov

Vsaka posamezna vrsta naslovov vsebuje dva sklopa podatkov:

- Podatki o naročniku:
 - Podjetje
 - Pozdrav
 - Ime
 - Srednje (osebno) ime
 - Priimek
 - Dve dodatni polji za različne podatke. Taki podatki so na primer za naročniške številke podjetja Diners. Njim izdajamo račune po točno določeni specifikaciji vsakega prvega dne v tekočem mesecu.
- Podatki o naslovu:
 - Dve polji za dodatne opombe
 - Naslov ulice
 - Hišna številka
 - Sobna številka

- Koda države
- Številka poštnega nabiralnika
- Poštna številka
- Kraj
- Pokrajina
- Kontaktna telefonska številka

The screenshot shows the 'IMS Web Customer Care' interface in Microsoft Internet Explorer. The main content area displays 'Contact information' for 'Germovsek, Anze'. The form includes fields for 'Company', 'Salutation' (set to 'Gospod'), 'First name' (Anze), 'Surname' (Germovsek), 'Additional 1' (315874), 'Middle name', 'Additional 2', 'Address information' (Street name: Žigarnja vas, Apartment no., Post box, Zip code: 4294, City: Križevci, County, State, Country: Slovenia), 'Electronic addresses' (reach by, Identifier), and 'reach by' (Domalč telefon 031 431331). The left sidebar shows a navigation menu with options like 'contacts', 'default naslov', '2005.12.20 - 2005.12.20', '0004.1.18 - 2005.11', 'Najemnikov naslov', 'Začasni naslov', 'Methods of payment', 'financial transactions', 'financial receipts', 'discounts', 'subscriptions', 'history', 'customer', 'document messages', 'movements', and 'usage details'. The bottom left shows a 'Hierarchy' section with 'Germovsek, Anze' selected.

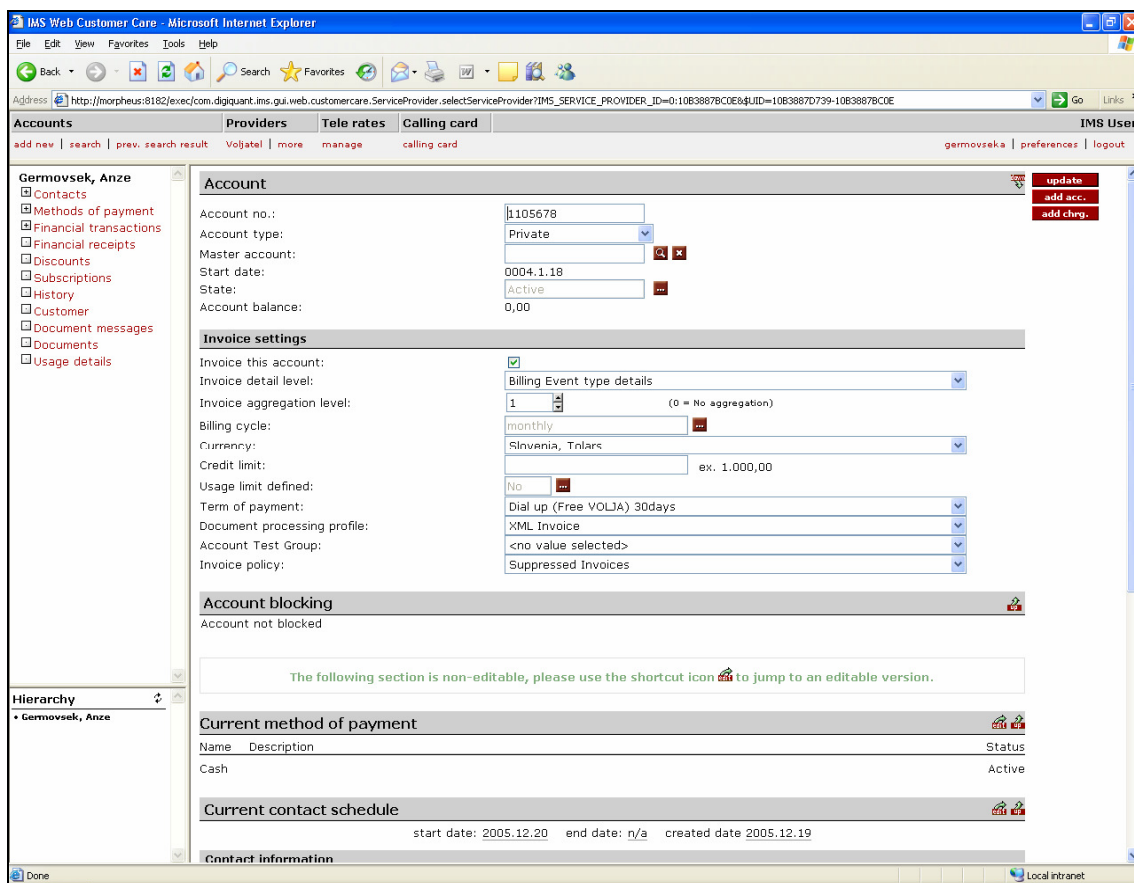
Slika 1: Podatki o naslovu v sistemu IMS

Osnovni podatki o naročniku

V tem sklopu so definirani osnovni podatki o naročniku. Posamezna polja pomenijo:

- Account no - naročniška številka, ki je enolično določena.
- Account type - tip naročnika, kjer določimo, ali je to poslovni naročnik, fizični naročnik ali naročnik zaposlen v podjetju Voljatelj.
- State - stanje naročnika. Ta je lahko blokiran, zaprt ali aktiven.
- Invoice this account – označimo, ali program izdela račun pri obračunu in ga zapiše v datoteko XML
- Invoice detail level – označimo, kako podrobne podatke o posameznem računu izdela sistem IMS .
- Billing cycle – izberemo, katere vrste obračun bomo izvajali za posameznega naročnika. Zaenkrat izvajamo samo mesečni obračun.
- Currency – označimo valuto, v kateri bomo izvajali obračun. Trenutno se vsi obračuni izvajajo v slovenskih tolarjih.
- Document processing profile – izberemo, kakšen tip dokumenta dobimo iz sistema IMS. V našem primeru dobimo račune v formatu XML.

Poleg tega lahko vodimo še nekaj ostalih podatkov, ki pa jih zaenkrat ne uporabljamo.



Slika 2: Glavno okno sistema IMS

Storitve

Voljatelj ponuja različne storitve:

- Internetne storitve
 - ADSL paketi z različnimi hitrostmi
 - ADSL fit paket z minutno porabo
 - Kabelski paketi (L, XL, XXL, MAXI)
 - Internet brez naročnine
- Telefonija
 - IP telefonija
 - Klasična telefonija z različnimi ceniki za poslovne in rezidenčne naročnike
 - 080 storitve
 - 090 storitve
- Dodatne storitve
 - Digitalna potrdila
 - C panel
 - Registracija SI in ostalih domen
 - Internetna varnost
 - Spletno gostovanje
 - Virtualni strežnik
 - Gostovanje strežnikov
 - Spletno oglaševanje

Ko se naročnik naroči na vsaj eno storitev, v sistem IMS dodamo novega naročnika in storitev, opisano s potrebnimi parametri. Ti so različni od storitve do storitve. Ko nekdo v

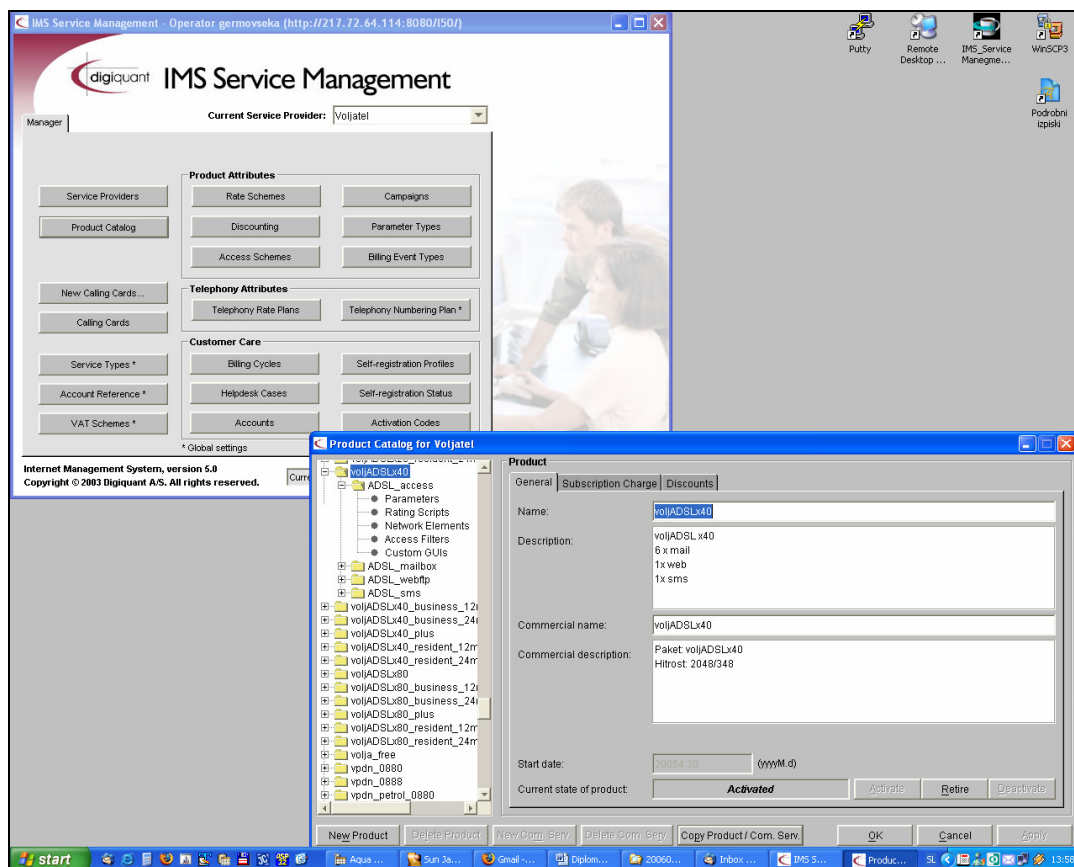
sistem IMS naročniku doda storitev, se potrebni parametri avtomatsko razpošljejo na različne lokacije. Tam se aktivirajo zahtevane stvari.

Kot primer bom opisal, kaj se zgodi, če nekdo naroči storitev ADSL. Opisal bom tudi primer aktivacije telefonske storitve.

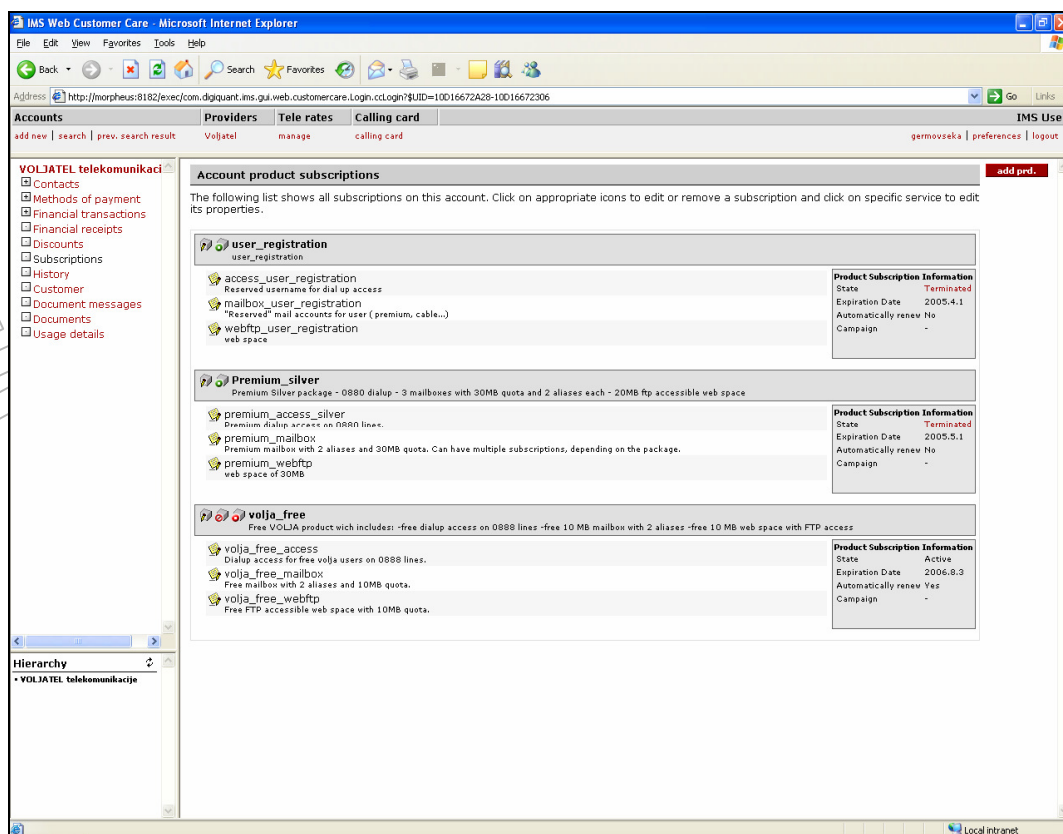
Naročnik najprej pošlje naročilo za storitev ADSL. Odgovorna oseba sprejme naročilo in preveri, če je na tem območju možna izvedba priključka ADSL, bodisi na lastnem omrežju podjetja Voljatelj, bodisi na omrežju podjetja Telekom. Če je izgradnja možna, se naročniku pošlje pogodbo. Ko naročnik pogodbo vrne, se preko spletne aplikacije Telekomu Slovenije posreduje podatke za vključitev priključka ADSL. Naročniku se pošlje s strani Voljatelja podpisana pogodba z aktivacijsko kodo, kodo puk in navodilo za uporabo, v primeru lastne namestitve pa tudi modem. Naročnik na spletni strani na podlagi svoje aktivacijske kode in kode puk izvede samoaktivacijo. Takrat se iz baze podatkov PostgreSQL, kjer vodimo podatke o storitvah ADSL, podatki o naročniku prenesejo na spletno stran samoaktivacije, kjer jih naročnik vidi in potrdi veljavnost podatkov. S tem, ko naročnik potrdi podatke, se v sistemu IMS doda nov naročnik in ustreza storitev. Obstaja več vrst paketov storitev ADSL. Sistem IMS s pomočjo aktivacijske in kode puk prepozna storitev. Ko se pri samoaktivaciji doda nov naročnik in ustrezen paket storitev, se ob uspešni aktivaciji paketa zapiše v bazo podatkov PostgreSQL podatek o ključu naročnika iz sistema IMS in datum aktivacije. V ozadju sistema IMS se sprožijo dodatne zahteve. Tako se na ustrezna mesta pošlje zahteva za izdelovanje novih poštnih predalov z ustreznimi podatki, za rezervacijo dodatnega prostora za postavitev spletne strani, za ustvarjanje dostopa za uporabnika in podobno.

Podobno poteka aktivacija telefonskih storitev. Tu sistem IMS po aktivaciji pošlje telefonski centrali ustrezne podatke. Centrala zabeleži novo telefonsko številko, ki jo lahko uporablja naročnik.

Na strežniku, kjer je telefonska centrala, se v 15 sekundnem intervalu ustvari datoteka v formatu AMA, kjer so zabeleženi vsi klici v tem času. To datoteko posredujemo programu ICE TRAFFIC, ki služi za različne statistike. Vsak klic (tisti, pri katerem je bila zveza vzpostavljena in tisti, pri katerih ni bila) se zapiše v bazo MSSQL. V tej bazi so poleg klicev naših naročnikov zavedeni še vsi klici, ki so nastali pri naročnikih, ki kličejo naše naročnike, klici, ki jih preusmerjamo v druga slovenska in tuja omrežja. Tako s programom izberemo le uspešne klice, torej tiste, pri katerih se je zveza vzpostavila. Ti se vsakodnevno uvozijo v sistem IMS. Za to je zadolžen pomožni program, ki klice iz ustreznih datotek XML pravilno vrednoti po destinacijah. Pri tem upošteva, kakšen cenik velja za naročnika in upošteva različne popuste, če so zavedeni na naročniški storitvi. V bazi sistema IMS so torej zavedeni vsi telefonski klici in so pravilno vrednoteni. Tako pri obračunu sistem IMS združi naročnine, ki jih zaračuna in porabe različnih storitev iz baze podatkov Oracle in izdela datoteke XML.



Slika 3: Dodajanje nove storitve v sistem IMS



Slika 4: Storitve, na katere je naročen naročnik

1.6. Izvajanje obračuna v sistemu IMS

Obračun posameznih telefonskih klicev se izvaja sproti in zapisuje v bazo podatkov Oracle. Prav tako se sproti obračunavajo vse porabe pri storitvi ADSL in drugih internetnih storitvah, ki temeljijo na principu zaračunavanja porabe po polminutnem intervalu. Obračun naročnin v sistemu IMS se izvaja vedno zadnjega dne v mesecu, ker so tako ustvarjeni paketi storitev in je tako določena politika podjetja.

Obračun se izvaja vsakega drugega dne v mesecu. Razlog temu je dejstvo, da prvega v mesecu še enkrat preverimo, če so uvoženi vsi podatki o telefonskih klicih.

Obračun zaženemo na spletni strani Bill Cycle Run

Date	Total accounts	Selection	Calculation	Commit	Doc. gen.	Close
7/1/06 - 8/1/06	346,583	346,583	346,570	346,570	9,200	
		0	0	0	0	
		0	13	0	337,370	
6/1/06 - 7/1/06	345,139	345,139	345,126	345,126	8,941	8,941
		0	0	0	0	
		0	13	0	336,185	
5/1/06 - 6/1/06	343,531	343,531	343,518	343,518	8,708	8,708
		0	0	0	0	
		0	8	0	334,810	
4/1/06 - 5/1/06	341,554	341,554	341,541	341,541	8,083	8,083
		0	0	0	0	
		0	13	0	333,458	
3/1/06 - 4/1/06	339,656	339,656	339,643	339,643	7,405	7,405
		0	0	0	0	
		0	13	0	332,238	
2/1/06 - 3/1/06	336,592	336,592	336,582	336,582	6,612	6,612
		0	0	0	0	
		0	10	0	329,970	
1/1/06 - 2/1/06	333,295	333,295	333,283	333,283	1,485	
		0	0	0	0	
		0	12	0	331,798	
12/1/05 - 1/1/06	330,132	330,132	330,120	330,120	5,428	5,428
		0	0	0	0	
		0	7	0	324,692	
11/1/05 - 12/1/05	326,932	326,932	326,919	326,919	4,972	4,972
		0	0	0	0	
		0	6	0	321,947	
10/1/05 - 11/1/05	323,207	323,207	323,194	323,194	4,630	4,630
		0	0	0	0	
		0	10	0	318,564	
		0	3	-1		

Slika 5: Spletna stran obračuna

Na tej spletni strani imamo podatke o vseh do sedaj izvedenih obračunih.

Obračun se izvaja v štirih fazah in sicer:

1. Selection – izbira naročnikov, ki so v tem obračunskem obdobju koristili storitve, ki se zaračunavajo.
2. Calculation – izbira storitev in cen, ki se zaračunajo naročnikom.
3. Commit – kontrola ustreznosti podatkov.
4. Document generator – izdelovanje datotek XML.

Vsako fazo zaženemo posebej. Faze potekajo zaporedno, tako da moramo imeti najprej izvedeno prvo fazo, šele nato pa lahko zaženemo fazo 2 in tako naprej do faze 4.

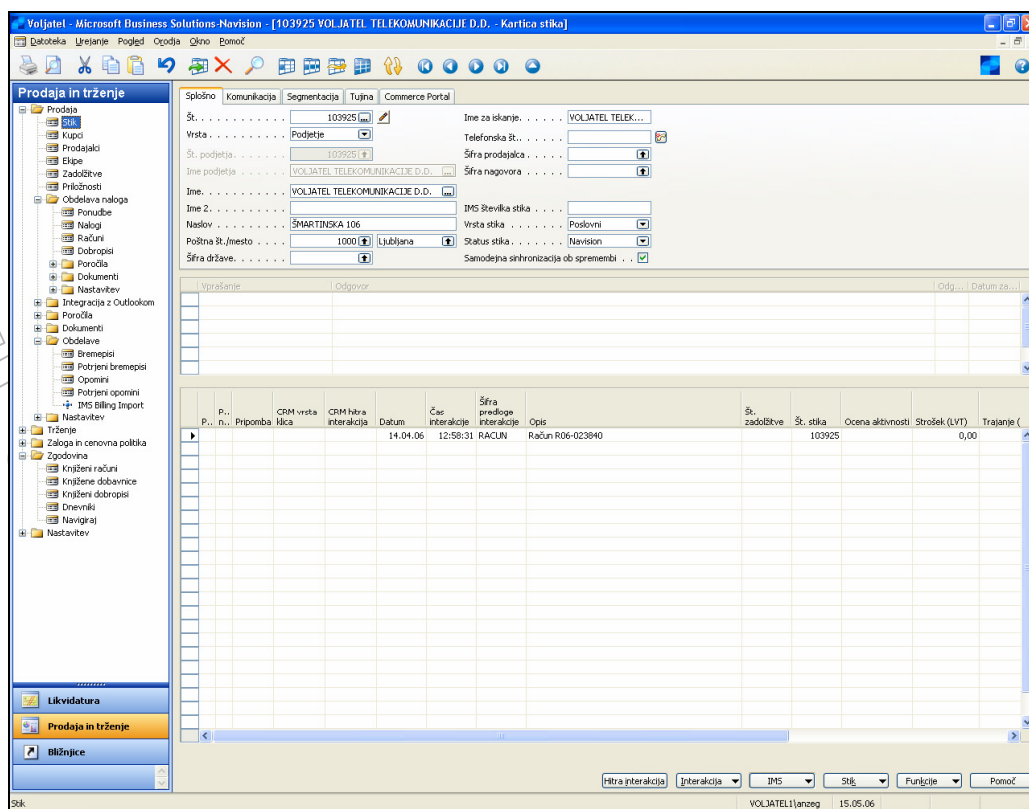
S končan. Po končanem obra
XML na svoj računalnik in p
ns – Navision
rešitev za poslovanje srednje

1.7. Aplikacija Microsoft Business Solutions – Navision

Microsoft Business Solutions – Navision je programska rešitev za poslovanje srednje velikih podjetij. Glede na podatke podjetja Microsoft jo vsak dan uporablja več kot 30.000 podjetij v več kot 50 državah. Omogoča upravljanje podatkov o dobaviteljih in kupcih, knjiženje računov, pregledovanje različnih statistik, omogoča zapisovanje različnih opomb glede stikov z naročnikom in drugih opomb. Tako se za vsakega naročnika točno vidi kdaj in kdo je izdal račun, vse opombe pri reševanju njegovih reklamacij, opomini, ki so mu bili izdani in še mnogo stvari. V našem primeru je aplikacija povezana s sistemom IMS, od koder v Navision uvažamo podatke o naročnikih in pregledujemo storitve, na katere je naročen naročnik in ki so bile opravljene zanj v določenem obdobju.

S poslovno rešitvijo Microsoft Navision smo nadomestili svoj obstoječi sistem Mit. S tem smo dobili bolj uporaben in lažje nadzorovan sistem. Ob vpeljavi so ga pooblaščen Microsoftovi partnerji še dokončno izoblikovali po naših potrebah. Na voljo so aplikacijska področja za vodenje financ, proizvodnjo, distribucijo, upravljanje odnosov s kupci in za elektronsko poslovanje.

Lokalni pooblaščen Microsoftov partner za poslovne rešitve je programsko opremo natančneje prilagodil našim potrebam. Tako so bili pripravljene moduli, ki omogočajo prenos naročnikov iz sistema IMS, uvoz izdelanih računov, uvoz stikov, uvoz interakcij za stike iz tekstovnih datotek in podobno. Še vedno pa je bilo potrebno datoteke za uvoz podatkov pripraviti v obliki, kot jo pričakuje program.



Slike 6: *Glavno okno programa Microsoft Navision*

2. IMPLEMENTACIJA PROGRAMA

2.1. Načrtovanje programa

Odkar sem prišel v podjetje Voljatel Telekomunikacije d.d., se ukvarjam z urejanjem računov, ki jih generira sistem IMS. Kot sem omenil v razdelku 1.5., iz sistema IMS dobimo podatke v obliki datotek v formatu XML.

Pred pol leta smo še uporabljali obračunski program Mit. V program Mit smo uvažali neknjižene račune, ki jih je bilo potrebno pripraviti po določeni specifikaciji v datotekah. Tu so morali biti poleg podatkov o storitvah, ki jih ima naročene naročnik, navedeni tudi podatki o naročnikih samih. Moja naloga je bila na osnovi datotek XML, pridobljenih iz sistema IMS, pripraviti datoteke s podatki o računih za program Mit. Takrat še nisem poznal razreda v programskem jeziku Java, ki vsebuje metode, s katerim lahko prebiramo (»parsamo«) dokumente, napisane v formatu XML. Zato sem te datoteke bral kakor tekstovne datoteke vrstico za vrstico. Pri tem sem imel veliko težav, saj sem moral predvideti veliko različnih situacij. Tako so se ponekod značke jezika XML zaključile v isti vrstici, drugod spet v naslednji vrsti.

Pri izdelovanju uvoznih datotek je nastal problem, ker administratorji pogosto niso poskrbeli za ažurne podatke. Tako so pogosto manjkali obvezni podatki o davčnih številkah poslovnih uporabnikov, zavedeni so bili napačni naslovi, imena, velikokrat so manjkali šumniki in še bi lahko našteval. Velikokrat so manjkali pomembni podatki o naslovu ali kakšni drugi podatki in zaradi tega uvoz ni bil mogoč.

Nato smo zamenjali sistem z novim. Pri novem programu posamezna postavka v datoteki za uvoz ne vsebuje več vseh podatkov o naročniku. Potreben je le še ključ – IMS številka, ki je vezni člen med naročnikom in storitvami, na katere je naročnik naročen. S tem se je moje delo s pripravo datoteke za uvoz precej poenostavilo, saj podatkov o naročniku ni bilo več potrebno vstavljati v datoteko.

Pri prehodu na nov sistem sem moral zato bodisi popraviti stari program bodisi izdelati novega. V vsakem primeru so bile spremembe precejšnje, zato je projekt priprave programa zahteval skrbno načrtovanje.

Prvi problem, ki sem ga želel rešiti z novim programom, je bilo branje datotek v formatu XML. Sodelavec Dalibor Šabič me je opozoril na razred DOMParser iz paketa org.apache.xerces.parsers. Ta omogoča zelo elegantno prebiranje datotek XML.

Drug problem, ki sem ga imel, je bila neurejena in za spreminjanje zelo težavna koda. Celoten program je bil napisan v enem razredu, ki je bral in s pomočjo tridimenzionalnih tabel urejal račune. Ker je bil program tako kompleksen in nepregleden, je bilo zelo težavno karkoli spreminjati. Zato sem rajši program napisal na novo in uporabil več manjših razredov.

Osnovni razred je sedaj razred Racun. Ta vsebuje metode, s katerimi lahko uredimo posamezen račun in ga zapišemo v ustrezno datoteko. Glavni program je v razredu BeriXML, kjer iz določene mape preberemo datoteke XML. Polnimo objekte tipa Racun, jih uredimo in zapišemo v ustrezno datoteko. Vsi ostali razredi so pomožni.

Navedimo sedaj razrede, ki so vsebovani v naši aplikaciji. Pri vsakem razredu so navedene glavne metode in njihova osnovna funkcionalnost. Podrobnosti o posameznih metodah bomo navedli ob sami implementaciji, v razdelku 2.2.

Racun	Osnovni razred, kjer hranimo podatke o ključu naročnika in storitvah, ki jih ima naročene naročnik (ključi storitev, opisi storitev, cene storitev, količnik, datum izdelave računa,...). V tem razredu imamo metodo, s katero uredimo in zapišemo posamezen račun v ustrezno datoteko	
Javne metode	<code>void Racun()</code>	Osnovni konstruktor, ki ustvari prazen račun
	<code>void zapisiVDatoteko (IMS povezava)</code>	Metoda, ki poskrbi za to, da se račun v zahtevani obliki zapiše v ustrezno datoteko
	<code>void dodajStoritev (String[] s)</code>	Metoda, ki polni vektor s storitvami. Parameter, ki ga metoda sprejme, je tabela nizov. Ta vsebuje ključ paketa storitve, šifro storitve, tip storitve in zaračunano ceno storitve. Primer naročnine pri paketu storitev voljADSLx10: Ključ paketa storitve je 1000805, šifra storitve je 1001096, tip storitve je Narocnina20, cena brez DDV pa je 3325,00.
	<code>void dodajPosebneStoritve (String opisni_kljuc_storitve, String znesek)</code>	Metoda, ki polni vektor s posebnimi storitvami. Parameterna, ki jih metoda sprejme sta dva in sicer opisni ključ storitve in cena storitve. Primer posebnih storitev so pogodbeni kazni, namestitve ADSL, ponovni priklop storitev, ...
Privatne metode	<code>String zaokrozi (double znesek)</code>	Metoda, ki dano vrednost znesek zaokroži na dve decimalni mesti in jo vrne v obliki niza.
	<code>void urediSifreStoritev ()</code>	Metoda, ki uredi šifre storitev računa, tako da se sprehodi čez vse zaračunane šifre storitev posameznega računa in za vsako šifro storitve preveri ali se mora vezati na katero drugo. Če se mora vezati na drugo šifro storitve, zamenja obstoječo. To je pomembno zato, da se zaradi naknadnih statistik iz Navisiona določene nove storitve vežejo pod isto storitev, ki že obstaja.

	<pre>String spremeniCenoVStaroCeno (String cs_id, String cena_iz_IMS, String account_id)</pre>	Metoda je namenjena desetim kabelskim storitvam, kjer so administratorji prepozno spremenili cene storitev v IMS in sicer konec zadnjega dne v mesecu. Tako so se cene teh storitev v datotekah XML zaračunale po starih cenah. V datoteki, kjer imamo celomesečne naročnine, pa so že nove cene. Tako komponento <code>cena_iz_IMS</code> s pomočjo te metode zamenjam v staro. S tem se pravilno obračunava delež naročnine. Če šifra storitve ne obstaja v datoteki, metoda vrne komponento <code>cena_iz_IMS</code>. Primer: cena storitve v XML je 5241.67, <code>cena_iz_IMS</code> je 3750.0, šifra storitve <code>cs_id</code> je 1000729, Ker se ta šifra nahaja v datoteki »kabel produkti - nove cene - anze.csv« vrne metoda staro ceno 5241.67, tako da se pravilno obračunajo storitve.
	<pre>String spremeniCenoVNovoCeno (String cs_id, String cena_iz_IMS, String account_id)</pre>	Tudi ta metoda je namenjena kabelskim storitvam, kjer so administratorji prepozno spremenili cene storitev v IMS. Tako so se cene teh storitev v datotekah XML zaračunale po starih cenah. V datoteki, kjer imamo celomesečne naročnine, so že nove cene. Tako komponento <code>cena_iz_IMS</code> s pomočjo te metode zamenjam v novo, ko so že preračunani deleži naročnin. Če šifra storitve ne obstaja v datoteki, metoda vrne komponento <code>cena_iz_IMS</code>. Primer: cena storitve v XML je 5241.67, <code>cena_iz_IMS</code> je 5241.67, šifra storitve <code>cs_id</code> je 1000729, Ker se ta šifra nahaja v datoteki »kabel produkti - nove cene - anze.csv« vrne metoda novo ceno 3750.00, tako da se na račun izpiše pravilna nova cena.

	<pre>String zamenjajSifroStoritve (String cs_id)</pre>	V datoteki "branje\Zamenjava_cs_id.txt" so zavedene le tiste nove šifre storitev, ki se morajo zamenjati. Tako je v tej datoteki v vsaki vrstici zavedena nova in stara šifra storitve. To se zgodi, kadar administrator v sistem IMS doda novo storitev, ki pa se mora zaračunati pod isto šifro storitve, ker ta nova storitev predstavlja le posodobljena različico stare. Tako vsi stari naročniki obdržijo staro različico storitve, vsem novim naročnikom pa se doda nova storitev, ki pa se pri posameznem računu veže na staro šifro storitve. Primer, ko v datoteki obstaja šifra storitve: šifra storitve cs_id je 1003081, ki je zavedena v tej datoteki. Metoda v tem primeru vrne staro šifro storitve 1001941, ki je zavedena poleg nove. Primer, ko v datoteki ne obstaja šifra storitve. Šifra storitve cs_id je 1001941. Metoda vrne isto šifro storitve 1001941.
	<pre>String najdiImeStoritve (String cs_id, String account_id)</pre>	Metoda v datoteki "branje\cs_id_name.txt" poišče ime za storitev, ki pripada dani šifri storitve cs_id. Če šifre storitve v datoteki metoda ne najde, izpiše na zaslon šifro storitve, naročniško številko account_id in delne podatke o tej storitvi iz baze Oracle, obenem pa se podatki o tej storitvi zapišejo v zgoraj navedeno datoteko.
	<pre>String najdiCenoIzIMS (String account_id, String cs_id)</pre>	Prvega dne v mesecu se požene samostojen program, ki zapiše vse naročniške številke, šifre storitev in cene le teh v točno določeno datoteko z imenom, ki vsebuje datum pridobljenih podatkov. Metoda, kjer za podano šifro storitve cs_id in naročniško številko account_id poišče ceno celomesečne storitve, ki je zavedena v zgoraj omenjeni datoteki. Če šifre storitve zaradi kakšnega razloga metoda ne najde v tej datoteki, s pomočjo dodatne metode doda v datoteko vse šifre storitev, na katere je naročen ta naročnik.

	<pre>void oblikujPodatkeVRacun (IMS povezava)</pre>	Metoda, ki uredi in obdela storitve ter določi datume izdelave, datume knjiženja storitev računa. Metoda prebere vektorja <i>v</i>, kjer so shranjene vse storitve in posebne_storitve. Iz teh dveh vektorjev prebere vse storitve in jih ustrezno obdela. Zapiše jih v tri nove vektorje <i>v2</i>, <i>v4</i> in <i>v5</i>, kjer pomeni <i>v4</i> vektor četrtih vrstic, <i>v5</i> vektor petih vrstic itd. Zapiše se tudi niz tretje vrstice. Metoda preveri, ali se določena storitev zaračunava za prihajajoče mesečno obdobje ali za mesečno obdobje, ki je minilo. Ustrezno uredi in zapiše tudi naročnine. Tu metoda preveri, ali je to storitev, ki se zaračunava za preteklo ali tekoče obdobje. Storitve, ki se zaračunavajo za preteklo obdobje, imajo lahko ceno zaračunane storitve manjšo kot je cena za celomesečno storitev. To se zgodi takrat, ko se je naročnik prijavil v preteklem mesecu in mu moramo zaračunati le del meseca, ko je naročil storitev. Lahko pa ima ceno zaračunane storitve enako celomesečni. To pomeni, da se mora storitev naročniku zaračunati za celotno preteklo obdobje. Storitve, ki se zaračunavajo za tekoče obdobje, imajo lahko večjo naročnino od celomesečne. Do tega pride, ko je naročnik naročil novo storitev v preteklem mesecu. Takrat se naročniku zaračuna del cene storitve za preteklo obdobje in celotna naročnina za naprej. Tudi tu je cena zaračunane storitve lahko enaka celomesečni ceni. V tem primeru se naročniku zaračuna storitev le za tekoče obdobje. Metoda najprej obdela vse posebne storitve. Te so navedene v vektorju posebne_storitve. Potem obdela še ostale standardne storitve iz vektorja <i>v</i>. Standardne storitve nato loči po tipu storitve.
--	---	--

BeriXML	Osnovni razred z metodo main. Tu program prebere vse datoteke XML, ki so v določeni mapi in jim odstrani dve vrstici, ki se sklicujeta na DTD. Nato prebere in polni objekte tipa Racun ter vsakega po končanem branju zapiše v ustrezno datoteko.	
Javne metode	<pre>void main (String[] args)</pre>	Glavna metoda, s katero zaženemo projekt. V metodi vzpostavimo povezavo s sistemom IMS in z bazo podatkov Oracle. Metoda poskrbi tudi zato, da izbere ustrezne datoteke XML.

	<pre>void beriXML (String ime_xml_datoteke, IMS povezava)</pre>	Metoda, s katero iz datoteke XML s pomočjo razreda DOMParser razberemo podatke za vsakega naročnika. Za vsakega naročnika ustvari objekt Racun ter poda zahtevo za zapis tega računa v ustrezno datoteko. Metoda sprejme ime datoteke XML in parameter povezava, kjer je podana povezavo do baze Oracle in sistema IMS.
	<pre>Racun preberiPodatkeORacunuIzXML (Node racun, IMS povezava)</pre>	Metoda, ki v parametru racun sprejme objekt tipa Node. Ta razred izvira iz paketa org.w3c.dom. V njem so podatki o posameznem naročniku in njegovih storitvah iz datoteke XML. Za izvedbo obdelave pomembne podatke zapisuje v objekt Racun. V razredu IMS je podana povezava do baze Oracle in sistema IMS.
Privatne metode	<pre>void odstraniDTDVrstici (String ime_datoteke)</pre>	Metoda, ki prebere datoteko XML in odstrani točno določeni vrstici, kjer je sklic na datoteko DTD.

SeznamProduktov	Pomožni razred, ki ustvari datoteke, ki so potrebne za razvrščanje računov po različnih storitvah. Poleg tega ustvari datoteko, ki je potrebna za dodajanje internetne porabe določenim naročnikom. Potreba po zadnji izvira iz dejstva, da za storitev VOLJADSL FIT v datotekah XML ne dobim pravih podatkov.	
Javne metode	<pre>void ustvariDatoteke (Connection povezava)</pre>	Metoda, ki ustvari datoteke za potrebe razvrščanja računov in datoteko, kjer je za storitev VOLJADSL FIT zavedena minutna poraba interneta. Podatke pridobi iz baze Oracle, na katero se povežemo preko parametra povezava.
	<pre>Vector nastaviDatume()</pre>	Metoda, ki v obliki vektorja vrne 5 datumov, ki so pomembni za obračunavanje storitev.

CeneIzIMS	Pomožni razred, ki za posameznega naročnika v ustrezno datoteko doda cene vseh storitev, na katere je ta naročnik naročen. Cene storitev pridobi iz sistema IMS.	
Javne metode	<pre>void zapisiCeneVsehStoritev (String datoteka, String datum, boolean zNicVrednostmi, String account_id)</pre>	Metoda v ustrezno datoteko (za avgust 2006 je to datoteka »Produkti_01.08.2006.csv«) zapiše cene vseh storitev, na katere je naročen naročnik z naročniško številko account_id in so zavedene v sistemu IMS. Parameter datum uporabimo za to, da se podatki zapišejo v ustrezno datoteko, ki je poimenovana z datumom. Če želimo, da se zapišejo samo cene storitev, ki imajo ceno večjo od nič, uporabimo parameter zNicVrednostmi z vrednostjo true, drugače pa zapišemo v vrednost false.

Privatne metode	<pre>Vector vrniVseStoritve (String account_id)</pre>	Metoda, ki za danega naročnika iz baze podatkov Oracle pridobi seznam vseh storitev, ki jih je imel ali jih ima naročene.
-----------------	---	---

TelefonskiCenik	Pomožni razred, ki ustvari cenik klicev v vse telefonske destinacije na določen dan in ga zapiše v datoteko. Podatke zajema iz podatkovne baze Oracle.	
Javne metode	<pre>TreeMap vrniTelefonskiCenik (String cenik, String datum)</pre>	Metoda sprejme ime željenega cenika in datum na katerega želimo, da je cenik veljaven. Metoda vrne preslikavo med destinacijami, opisanimi z identifikacijskimi oznakami in cenami.
	<pre>void seznamVDatoteko (TreeMap drevo, String datoteka, char delim)</pre>	Metoda, ki v datoteko z imenom datoteka zapiše podatke o cenah klicev v destinacije, shranjene v preslikavi drevo. Drevo vsebuje pare (ime destinacije, cena). S parametrom delim povemo znak, ki med sabo loči posamezne podatke pri zapisu v datoteko.
Privatne metode	<pre>void zgradiCenik (Connection povezava, TreeMap klici, TreeMap cenik, int koren_id, double cena)</pre>	Rekurzivna metoda, ki zgradi cenik klicev v vse možne destinacije. Parametri, ki jih sprejme metoda: ▪ povezava je povezava na podatkovno bazo Oracle, ▪ klici je preslikava med imeni destinacij in njihovo ceno ▪ cenik je preslikava med identifikacijskimi oznakami in cenami klicev v destinacije. ▪ koren_id je identifikacijska oznaka destinacije, ki jo trenutno obdelujemo. ▪ cena je cena, ki jo metoda uporabi, če za destinacijo z identifikacijsko oznako koren_id ni navedene cene v preslikavi cenik.

CenikModemov	Pomožni razred, ki iz sistema IMS vrača podatke o tipu modema in serijski številki modema. Ti sta navedeni zraven storitve, na katero je naročnik naročen.	
Javne metode	<pre>static String[] cenikModemov (String account_id)</pre>	Metoda, ki za podanega naročnika iz sistema IMS pridobi številko modema in njegovo serijsko številko.

TelefonskiCenikNajTrajaDlje	Pomožni razred namenjen temu, da ustvarimo cenik VLT_NajTrajaDlje z vsemi cenami na določen dan. Razred je podoben razredu TelefonskiCenik. Posebnost tega cenika je, da ima za določene destinacije cenik razdeljen na časovne intervale. Za posamezen klic se prve tri aktivne minute zaračunavajo po normalni ceni, druge tri minute po drugi tarifi in vse nadaljne minute po tretji tarifi.	
Javne metode	<pre>TreeMap vrniTelefonskiCenik (String cenik, String datum)</pre>	Metoda sprejme ime željenega cenika in datum za katerega želimo, da je cenik veljaven. Metoda vrne preslikavo med destinacijami, opisanimi z idji in nizi. Niz je sestavljen iz dveh nizov, ločenih s podpičjem. Prvi niz je čas za katerega velja cena, drug niz pa je cena. Čas, za katerega velja cena, je trenutno podan v obliki: 0 – ki velja za prve tri minute 180 – ki velja za druge tri minute 360 – ki velja za vse ostale minute
	<pre>void seznamVdatoteko (TreeMap drevo, String datoteka, char delim)</pre>	Metoda, ki v datoteko z podanim imenom datoteka zapiše podatke o cenah destinacij, shranjenih v preslikavi drevo. Preslikava vsebuje pare (ime destinacije, cena). Metoda sprejme tudi parameter delim, ki določa ločitveni podatek pri zapisu v datoteko. Primer zapisane vrstice z tremi različnimi cenami za destinacijo Španija: <pre>Calls from Slovenia to Spain;0;21.67;180;20.15;360;18.42</pre> Ta zapis pomeni, da je cena minute pogovora za prve tri minute 21.67 sit, cena minute pogovora za naslednje 3 minute 20.15 sit in cena minute pogovora za vse pogovore daljše od šestih minut 18.42 sit. Primer vrstice, ki vsebuje le eno ceno : <pre>Calls from Slovenia to Telekom Slovenije;0;4.42</pre> Ta vrstica pomeni, da je cena minute pogovora z nekom v omrežju Telekom Slovenije 4.42 sit ne glede na dolžino klica.

Privatne metode	<pre>void zgradiCenik (Connection povezava, TreeMap klici, TreeMap cenik, int koren_id, double cena)</pre>	Rekurzivna metoda, ki zgradi cenik vseh možnih destinacij in njihovih cen. Parametri, ki jih sprejme metoda, imajo enak pomen, kot parametri metode zgradiCenik iz razreda TelefonskiCenik.
-----------------	--	---

DatumiInProcenti	Pomožni razred, ki vsebuje dve metodi. Prva vrne vse datume, ki so potrebni pri izdelavi računov, druga pa vrne dvojno tabelo velikosti 100.	
Javne metode	<pre>Vector vrniDatume ()</pre>	Metoda, ki vrne 7 podatkov o datumih in sicer: prvi dan prejšnjega meseca, zadnji dan prejšnjega meseca, prvi dan tekočega meseca, zadnji dan tekočega meseca, mesec in leto prejšnjega meseca, mesec in leto tekočega meseca in deseti dan v tekočem mesecu

```
String[][] vrniProcente
(
    int st_dni_v_tekocem_mesecu,
    int st_dni_v_prejsnem_mesecu
)
```

Metoda vrne tabelo velikosti 100 x 2 z nizi vrednosti, ki predstavljajo dneve v mesecu za pravilno zapisovanje obdobja naročnin storitev. Te vrednosti nam pomagajo določiti začetni dan v primeru, ko je potrebno zaračunavanje delnih naročnin in sicer v primeru novih priklopov storitev. Kaj pomenijo podatki v tabeli, si oglejmo s primerom.

Primer storitve, ki se zaračunava za tekoče obdobje in je bil opravljen priklop v prejšnem mesecu in sicer v septembru 2006. Tekoča naročnina se zaračunava za mesec oktober 2006.

Storitev: ADSL FIT – naročnina

Celomesečni znesek iz IMS: 2491,67

Zaračunan znesek v XML: 2990,004

Znesek v XML je večji od zneska iz IMS. To nam pove, da bo potrebno na račun zapisati dve postavki in sicer naročnino za tekoče obdobje in naročnino, ki se mora zaračunati za preteklo obdobje.

Preprost izračun

$(\text{znesekXML} - \text{znesekIMS}) / \text{znesekIMS}$

$(2990,004 - 2491,67) / 2491,67 = 0,20$ (20%)

Ker računamo tekočo naročnino za mesec oktober 2006, ki ima 31 dni, priklope novih storitev pa je potrebno zaračunati še za obdobje september 2006, ki ima 30 dni. pokličemo to metodo tako

```
String[][] procenti =
vrniProcente(31, 30);
```

Potem pogledamo 20 vrstico (ker je 20%) tabele procenti

```
procenti [20][0] = 26
```

```
procenti [20][1] = 25
```

Tako iz tabele procenti[20][1] preberemo vrednost 25, ki nam pove, da bomo naročniku na račun zapisali dve postavki. Prva postavka bo za obdobje od 25.09.2006 do 30.09.2006, količino 0,20 po ceni 2491,67 kar skupaj nanese 498,33. Druga postavka na računu pa za tekoče obdobje od 01.10.2006 do 31.10.2006, količina 1,00 po ceni 2491,67 kar skupaj nanese 2491,67. Prvi stolpec v tabeli procenti[x][0] se zaenkrat še ne uporablja. Podatki v tabeli so pripravljeni za zaračunavanje odklopov storitev, ki se opravijo sredi meseca in se zaračunavajo za preteklo obdobje. Zaenkrat so vse pogodbe sklenjene na način, da veljajo do konca meseca in ta postopek ni potreben.

VrniPodatkeIzIMS	Pomožni razred, ki iz sistema IMS vrne podatke o naročniku in njegovih storitvah.	
Javne metode	String vrniPodatkeIzIMS (String account_id)	Metoda, ki s pomočjo naročniške številke iz sistema IMS pridobi podatke o tem naročniku. To so ime podjetja, ime, priimek ter storitve, na katere je naročen.

DodajanjeImenNovihStoritev	Pomožni razred namenjen dodajanju imen novih storitev v datoteko. Uporabljamo ga za poimenovanje storitev na računih.	
Javne metode	String vrniInDodajImeNoveStoritve (String cs_id)	Metoda, ki za podano šifro storitve iz baze Oracle prebere ime storitve, ki pa se dostikrat razlikuje od tiste, ki se mora zapisati na računu. To ime vrne kot rezultat. Poleg tega šifro in tako ime storitve zapiše tudi v datoteko, ki jo prebiramo pri določevanju imen storitev na računih.

UrejanjeKlicev	Pomožni razred namenjen urejanju telefonskih klicev v iste destinacije.	
Javne metode	<pre> TreeMap urediKlicePoDestinacijah (String anglesko_ime, double cas_sekund, TreeMap preslikava) </pre>	Metoda je namenjena urejanju posameznega telefonskega klica po destinaciji. Ta metoda je namenjena vsem cenikom, ki imajo na posamezno destinacijo podano le eno ceno čez cel dan. Metoda sprejme tri parametre in sicer angleško ime destinacije, število sekund klica za to destinacijo in preslikavo z vsemi destinacijami, ki jih je naročnik že klical za ta obračunski interval. V tej preslikavi je ključ anglesko_ime. Metoda preveri, če v podani preslikavi ta ključ že obstaja. Če ključ obstaja, sešteje cas_sekund in vrednost, ki je že zavedena v preslikavi. Če pa ključ ne obstaja, doda novo vrednost preslikave, torej ključ anglesko_ime in vrednost cas_sekund.

	<pre> TreeMap razbijInUrediKlicePoDestinacijah (String anglesko_ime, double cas_sekund, TreeMap preslikava, String datoteka) </pre>	Metoda je namenjena urejanju posameznega telefonskega klica po destinaciji. Ta metoda je namenjena posebnemu obliki cenika in ima poseben način zaračunavanja. V ceniku je cena za posamezno destinacijo definirana po dveh principih. Prvi je ta, da je cena minute pogovora definirana za vse minute enako. Drugi je, da je cena za klic v določeno destinacijo razdeljena na tri dele. Za prve tri minute je cena posameznega klica osnovna cena, za druge tri minute prva znižana cena in za vse klice, ki presežejo šest minut druga znižana cena. Metoda sprejme štiri parametre in sicer angleško ime destinacije, čas sekund posameznega klica, preslikavo z vsemi že zaračunanimi klici v tem obračunskem obdobju in datoteko, kjer je zapisan ta cenik. Za vsako destinacijo pred urejanjem metoda preveri, ali ima ta destinacija ceno razdeljeno na tri dele ali pa je le ena cena. V primeru treh delov metoda v preslikavo vpiše prvih 180 sekund (3 minute) z osnovno ceno, drugih 180 sekund (če klic traja več kot 180 sekund) v drugo preslikavo, vse ostale sekunde pa v tretjo preslikavo. Seveda metoda vedno preveri, če ključ preslikave že obstaja. Takrat prišteje tistemu ključu vrednost <code>cas_sekund</code>, drugače pa doda novo vrednost. Če ima destinacija le eno ceno, je postopek enak kot pri metodi <code>urediKlicePoDestinacijah</code>.
--	---	---

InternetnaPorabaADSLFit	Pomožni razred namenjen dodajanju internetne porabe za storitev VOLJADSL FIT. Pri tej storitvi je potrebno plačati vse minute, ki presežejo 600 minut.	
javne metode	<pre> String porabaFit (String account_id) </pre>	Metoda v obliki niza vrne število minut, ki jih je naročnik z naročniško številko <code>account_id</code> porabil v sklopu storitve VOLJADSL FIT in presegajo že vključenih 600 brezplačnih minut. Če podatka ne najde ali pa je bilo število porabljenih minut manjše kot 600, vrne niz 0.0

IMS	Pomožni razred namenjen vzpostavljanju povezave do sistema IMS-a in baze Oracle, za zapiranje obeh povezav in za pridobitev podatka o imenu cenika iz IMS-a za določeno telefonsko storitev.	
Javne metode	<pre>String vrniImeCenika (String account_id, String product_id)</pre>	Metoda, ki za podan ključ naročnika <code>account_id</code> in ustrezno storitev, vrne ime cenika iz sistema IMS, ki se za to storitev uporablja.
	<pre>double vrniCenoStoritve (String account_id, String product_id, String cs_id)</pre>	Metoda, ki vrne ceno storitve za določenega naročnika. Metoda sprejme tri parametre in sicer šifro storitve, ključ paketa storitev in ključ naročnika.
	<pre>void zgradiDrevoTelefonskihCenikov (Connection povezava)</pre>	Metoda, ki v globalno spremenljivko shrani vse ključe in imena cenikov.
	<pre>void odpriPovezavoIMS()</pre>	Metoda, ki odpre povezavo do sistema IMS.
	<pre>Connection odpriPovezavoOracle ()</pre>	Metoda, ki vrne povezavo do baze podatkov Oracle.
	<pre>void zapriPovezavoIMS()</pre>	Metoda, ki zapre povezavo do sistema IMS.
	<pre>void zapriPovezavoOracle()</pre>	Metoda, ki zapre povezavo do baze podatkov Oracle.

BeripodatkeOKlicih	Pomožni razred namenjen branju telefonskih zapisov iz baze Oracle	
Javne metode	<pre> TreeMap vrniZdruženePodatkeOTelefonskihKlicih (String account_id, String product_id, String cs_id, String datum, boolean vlt_naj_traja_dlje, Connection conn, String dat_price_list_name) </pre>	Metoda namenjena branju telefonskih klicev iz baze Oracle, ki jih je naročnik opravil v sklopu določene storitve. Kot parameter prejme tudi podatek vlt_naj_traja_dlje, ki pove, ali naj se cene klicev obravnavajo po principu te akcije ali kot navadni klici. Sprejme tudi povezavo na bazo Oracle in ime datoteke, v kateri je zapisan ustrezen cenik

Telefonija	Pomožni razred, namenjen pregledovanju telefonske porabe pri telefonskih storitvah.	
Javne metode	<pre> TreeMap vrniZdruženeKliceIzBaze (String account_id, String product_id, String cs_id, IMS povezava, String datum, String ime_cenika, String ime_datoteke) </pre>	Metoda, ki vrne seznam klicev za določenega naročnika in njegovo storitev urejen po destinacijah in minutah. Za klic metode potrebujemo naročniško številko account_id, ključ paketa storitve product_id, šifro storitve cs_id. Prav tako potrebujemo parameter za izvedbo povezave na bazo Oracle, ki ga podamo v parametru povezava. Dano mora biti tudi ime cenika, za katerega zaračunavamo storitev in ime datoteke, v kateri je zapisan cenik, ki se upošteva pri obračunu teh klicev. Parameter datum je namenjen izključno zapisovanju na račune.
	<pre> String[] vrniInUstvariCenik (String account_id, String product_id, String cs_id, IMS povezava, String datum) </pre>	Metoda, ki vrne ime cenika in datoteke, kjer je cenik zapisan. Če cenik za dani datum ne obstaja, ga izdelamo. Metodi podamo naročniško številko, ključ osnovne storitve, šifro storitve, povezavo na bazo Oracle in datum uporabe cenika.

Pripomoki	V tem razredu imamo različne priročne metode. Tako je tu metoda, s katero v datoteki za dano angleško ime destinacije poiščemo ustrezno slovensko ime. Razred vsebuje tudi metodo za iskanje cen za podan cenik in podano destinacijo. Z eno od metod poiščemo ime datoteke, kamor bo izpisan račun. Zadnja metoda pa ugotovi, če ima naročnik odobren popust pri telefonskih klicih na določeno destinacijo.	
Javne metode	<pre>String najdiSlovenskoImeDestinacije(String anglesko_ime)</pre>	Metoda, ki za podano angleško ime destinacije vrne slovensko ime.
	<pre>double vrniCenoEneMinuteKlica(String anglesko_ime, String datoteka_cenika)</pre>	Metoda, ki za podan cenik in destinacijo vrne ceno ene minute klica.
	<pre>double vrniCenoEneMinuteKlica(String anglesko_ime, String datoteka_cenika, int kolicnik)</pre>	Metoda je namenjena akcijskemu ceniku z imenom »Naj traja dlje«, ki za podano pot do cenika datoteka_cenika, angleško ime destinacije in kolicnik vrne ceno za to destinacijo. Z vrednostjo parametra kolicnik povemo, ali nas zanima cena za prve tri minute (vrednost parametra je takrat 0), druge tri minute (parameter naj ima vrednost 180) ali ostale minute (vrednost parametra je 360).
	<pre>String vrniImeDatotekeSPomocjoPrveStoritve(Vector v, String account_id)</pre>	Metoda, ki preveri prvo storitev, ki je bila zavedena v datoteki XML in vrne ime datoteke, kamor naj se zapiše račun.
	<pre>boolean popust(String anglesko_ime, String account_id)</pre>	Metoda, ki za podanega naročnika in telefonsko destinacijo vrne podatek, ali naročniku pripada popust za klic na to destinacijo ali ne.

2.2. Razred TreeMap

Ker v naših metodah zelo veliko uporabljamo preslikave (razred TreeMap) na kratko opišimo, zakaj gre pri tem razredu in kako deluje.

Razred TreeMap je podatkovna struktura, ki ji pravimo »preslikava«. Uporabljamo jo takrat, ko želimo poiskati določen podatek po njegovem ključu. TreeMap je razširitev razreda AbstractMap in implementira razrede SortedMap, Cloneable in Serializable. Notranji razredi so dedovani iz razreda java.util.Map. V razredu je zagotovljeno, da bodo vse preslikave

urejene v naraščajočem zaporedju glede na ključ. Vse operacije (containsKey, put, get, remove) se v razredu izvedejo v času $O(\log(n))$. Razred TreeMap je implementacija vmesnika SortedMap na osnovi rdeče-črnega drevesa. Rdeče-črno drevo je uravnoteženo iskalno dvojiško drevo, ki omogoča, da se operacije, kot so vstavi podatek v drevo, briši podatek iz drevesa in poišči podatek v drevesu izvedejo v času $O(\log(n))$.

Konstruktorji in metode, ki jih uporabljamo v naših metodah

Konstruktorji	Opis
TreeMap()	Ustvari novo prazno drevo, ki bo urejeno na »naravni« način

Metode	Opis
boolean containsKey(Object key)	Vrne true, če ključ key obstaja v drevesu, sicer vrne false.
Object get(Object key)	Vrne vrednost za ključ key
Set keySet()	Vrne drevo (vrednosti) prikazano v obliki razreda Set.
Object put(Object key, Object value)	Vstavi ključ in njegovo vrednost v drevo.

Primer uporabe dodajanja elementov v preslikavo, ki se uporablja v razredu UrejanjeKlicev

```
/**
 * Metoda je namenjena urejanju posameznega telefonskega klica po
 * destinaciji. Ta metoda je namenjena vsem cenikom, ki imajo
 * na posamezno destinacijo podano le eno ceno čez cel dan.
 * Metoda sprejme tri parametre in sicer angleško ime destinacije,
 * število sekund klica za to destinacijo in preslikavo z vsemi
 * destinacijami, ki jih je naročnik že klical za ta obračunski
 * interval. V tej preslikavi je ključ anglesko_ime.
 * Metoda preveri, če v podani preslikavi ta ključ že obstaja.
 * Če ključ obstaja, sešteje cas_sekund in vrednost,
 * ki je že zavedena v preslikavi. Če pa ključ ne obstaja,
 * doda novo vrednost preslikave, torej ključ anglesko_ime
 * in vrednost cas_sekund.
 */
public static TreeMap urediKlicePoDestinacijah(String anglesko_ime,
        double cas_sekund, TreeMap preslikava) {
    //ali preslikava že vsebuje podan ključ
    if (preslikava.containsKey(anglesko_ime)) {
        //preberemo vrednost preslikave
        double stara = Double.parseDouble(
            preslikava.get(anglesko_ime).toString());
        //stari vrednosti dodamo novo vrednost
        double nova = stara + cas_sekund;
        //in prepišemo vrednost z novo
        preslikava.put(anglesko_ime, new Double(nova));
    }
    //če preslikava še ne obstaja
    else {
        //dodamo novo preslikavo
        preslikava.put(anglesko_ime, new Double(cas_sekund));
    }
    //vrnemo novo preslikavo
    return preslikava;
}
```

Za pridobivanje podatkov iz objektov lahko uporabljamo različne postopke. Če uporabljamo tabele, se z zanko for sprehodimo čez vse elemente tabele:

```
for (int i = 0; i < tab.length; i++) {  
    tu nekaj naredimo s tab[i]  
}
```

Isti postopek lahko uporabimo pri pridobivanju podatkov iz razreda Vector. Če imamo vektor v, preberemo vrednosti na ta način:

```
for (int i = 0; i < v.size(); i++) {  
    tu nekaj naredimo z v.elementAt(i)  
}
```

Vendar to ni najboljša rešitev, ker prav v ta namen obstaja objekt, ki se imenuje iterator. To je objekt s pomočjo katerega se sprehodimo čez vse elemente vektorja, preslikave in drugih objektov. Priporočen način je torej takšen:

```
Iterator it = v.iterator();  
while (it.hasNext()) {  
    Object x = it.next();  
    //tu nekaj naredimo z x  
}
```

Metoda hasNext() vrne vrednost true, če ima iterator na voljo še kak objekt. Metoda next() vrne naslednji element.

Primer pridobivanja podatkov iz preslikave, ki je del kode razreda Racun

```
// zberemo vse ključe iz preslikave tm  
Iterator iterator = tm.keySet().iterator();  
// se sprehodimo po vseh ključih  
while(iterator.hasNext()) {  
    //preberemo trenutni ključ preslikave  
    Object kljuc = iterator.next();  
    //poiščemo njegovo vrednost  
    Object st_sekund = tm.get(kljuc);  
    //ključ spremenimo v niz  
    String anglesko_ime = "" + kljuc;  
    //vrednost pretvorimo v tip double in sekunde  
    //pretvorimo v minute  
    double st_minut = Double.parseDouble("" + st_sekund)/60.0;  
    //označimo, da bomo na račun izpisali vsaj eno destinacijo  
    vsaj_en_zapis = true;  
    //spremimo angleško ime destinacije v slovensko  
    String slovensko_ime =  
        Pripomočki.najdiSlovenskoImeDestinacije(anglesko_ime);  
    //v ceniku poiščemo še primerno ceno  
    double cena_minute =  
        Pripomočki.vrniCenoEneMinuteKlica(anglesko_ime,  
            ime_datoteke);  
    //preverimo ali ima naročnik za to destinacijo popust  
    boolean popust = Pripomočki.popust(anglesko_ime, account_id);  
    if(popust)  
        cena_minute = cena_minute*0.9;
```

```

//shranimo vse vrednosti, ki se pojavijo, da jih
//pozneje zapišemo v datoteko
cena_vseh_destinacij += (st_minut * cena_minute);
//zapišemo vsako destinacijo na račun (postavki 2 in 4)
String dvojka = "2" + LOCILO +
                cs_id + LOCILO +
                slovensko_ime + LOCILO +
                zacasni_datum_evj + LOCILO +
                koncni_datum_evj + LOCILO;
v2.add(dvojka);
String stirka = "4" + LOCILO +
                "za obdobje od " + zacasni_datum_evj +
                " do " + koncni_datum_evj + LOCILO +
                zaokrozi(cena_minute) + LOCILO +
                zaokrozi(st_minut) + LOCILO +
                "min" + LOCILO;
v4.add(stirka);
}

```

2.3. Branje datotek XML

Prvi korak pri pretvorbi dokumentov iz enega formata v drugi je ta, da uspešno preberemo osnovno datoteko XML. Pri branju XML datotek si pomagamo z že izdelanimi razredi v Javi. Kot osnovni objekt uporabljamo razred `DOMParser` iz paketa `org.apache.xerces.parsers`.

Pri tem sem se srečal z večimi problemi. V nadaljevanju bom opisal nekatere.

Skoraj pri vseh metodah zapisujemo napake (neobstoječe ključe, napačne vrednosti, ...) in sprotne pomembne opombe na zaslon. Na zaslon napake izpisujemo zaradi lažjega sprotnega iskanja le-teh. Projekt namreč izvajam vedno med delovnim časom ob svoji prisotnosti in sproti pregledujem, če obdelava poteka brez napak.

2.3.1. Manjkajoča datoteka DTD

Prva težava, na katero sem naletel pri preoblikovanju datotek XML, je, da je v datotekah XML naveden sklic na datoteko DTD, ki pa je ni. Zato metode razreda `DOMParser` pri branju javijo napako, da datoteka DTD ne obstaja. Ker datoteke DTD sploh ne potrebujemo, je bila rešitev enostavna. Iz datoteke XML je potrebno odstraniti sklic na to datoteko. Napisal sem metodo, ki jo pokličem, še preden začnem s prebiranjem (parsanjem) datoteke XML. V metodi preberem datoteko kot tekstovno datoteko. Če naletim na vrstici, ki se sklicujeta na datoteko DTD, ju spustim, ostale pa zapišem v začasno datoteko. Na koncu originalno datoteko izbrišem, začasno pa poimenujem enako, kot je bilo ime originalne datoteke. Šele, ko je ta postopek narejen, začnem z resničnim preoblikovanjem datoteke XML.

V programu je ta metoda v osnovnem razredu `BeriXML`, kjer imamo tudi glavno metodo `main`.

Poglejmo si kodo.

Metoda, kjer odstranimo vrstici DTD v datoteki XML

```

private static void odstraniDTDVrstici(String ime_datoteke) {
    try {
        // zapomnimo si objekt razreda File, kjer imamo originalno
        // datoteko in začasen objekt poimenovan temp
        File orig_dat = new File(ime_datoteke);
    }
}

```

```

File temp = new File("temp");

// odpremo pomožno datoteko za pisanje
PrintWriter pw = new PrintWriter(new FileWriter(temp));

//odpremo originalno datoteko za branje
BufferedReader br = new BufferedReader(new FileReader(orig_dat));

//vrstici, ki ju iščemo
String vrstica1 = "<?xml-stylesheet type=\"text/xml\"";
vrstica1 += "href=\"/dtd/BillingEventTypeDetails.xml\"?>";
String vrstica2 = ("<!DOCTYPE RecurringInvoice SYSTEM";
vrstica2 += "\"/dtd/BillingEventTypeDetails.dtd\">";

//prebiramo originalno datototeko vrstico po vrstico
while(br.ready()) {
    // preberemo tekočo vrstico
    String vrstica = br.readLine();
    // če smo naleteli na vrstico, ki je enaka
    // eni od dveh vrstic, ki povzročata težavo, jo spustimo
    // in ne zapišemo v temp datoteko
    if ((!vrstica.equals(vrstica1) && (!vrstica.equals(vrstica2)) {
        // vse druge vrstice zapišemo v datoteko temp
        pw.println(vrstica);
    }
}
//zapremo obe datoteki
br.close();
pw.close();
// originalno datoteko izbrišemo
orig_dat.delete();
// začasno datoteko preimenujemo v originalno datoteko
temp.renameTo(new File(ime_datoteke));

} catch (Exception ex) {
    // v primeru izjem izpišemo stanje sklada in
    // zaključimo
    ex.printStackTrace();
    System.exit(1);
}
}

```

2.3.2. Nezaključene končne značke

Drugo težavo pri prebiranju datotek XML so predstavljale nezaključene končne značke. Izvorna datoteka, kot jo je izdelal sistem IMS, torej ni vedno predstavljala pravilnega dokumenta v XML. Analiza teh napak je pokazala, da je datoteka "skoraj" pravilna. Namreč le tu in tam manjka ustrezna zaključna značka (</INVOICES>). Zaradi točno določene strukture je bilo enostavno razbrati, na katerem mestu značka manjka. Zato sem ta problem rešil z uporabo bloka try catch, ki je ulovil določeno napako. Takrat je datoteki enostavno dodal končno značko ter po uspešnem dodajanju poklical isto metodo s popravljeno datoteko.

Shematsko je ta del programa videti takole

```
try {
    ... // prebiramo datoteko XML
} catch(org.xml.sax.SAXParseException ex) {
    // Popravimo zadnjo značko
    ...
    // ponovno začnemo od začetka
    beriXML(name_xml, povezava);
} catch (Exception ex) {
    // gre za neko drugo napako pri branju, ki
    // ni povezana s strukturo XML
    ...
}
```

Če smo torej pri pregledovanju prišli do izjeme tipa `SAXParseException`, gre za napako povezano s strukturo datoteko XML. Zaradi opravljene analize tovrstnih napak, vemo, da takrat manjka značka `</INVOICES>` in jo zato enostavno dodamo. Na naslednjem prehodu bo prebiranje te datoteke XML uspešno.

Več o tem, za kaj gre pri izjemah `org.xml.sax.SAXParseException` pa bom opisal v razdelku 2.3.4, kjer bo govora o prebiranju datoteke XML.

2.3.3. Obdelava večjega števila datotek

V mapi programa z imenom `xml` imamo podmapo, v njej pa večje število datotek XML. Vsako posebej moramo obdelati s pomočjo metode `beriXML`. Oglejmo si, kako poskrbimo, da se obdelajo vse datoteke v določeni mapi.

V mapi `xml` je poleg ene same podmape tudi nekaj datotek tipa `zip`, ker imam kot arhiv shranjene datoteke XML starih obračunov. Teh ne smem zajemati v obračunu. Torej moram poskrbeti, da resnično preberemo le podmapo, kjer so dejanske datoteke XML. V tej podmapi ni nikoli drugih datotek, le tiste v formatu XML. Zato lahko privzamem, da ni potrebno pregledovati, ali so datoteke v tej mapi resnično datoteke formata XML.

Ideja postopka je v tem, da se postavimo v osnovno mapo `xml`. V tej mapi s pomočjo metode `listFiles()` dobimo seznam vseh datotek, ki so v mapi. Za vsako datoteko z metodo `isDirectory()` preverimo, če je mapa. V tem primeru preberemo seznam datotek še v tej mapi in obdelamo vsako datoteko, ki je običajna datoteka (to preverimo z metodo `isFile()`). Ker vemo, da mape niso gnezdene globlje, ni potrebno preverjati še, če je znotraj podmape še kakšna podmapa.

Ko smo prišli do datoteke XML, ki jo moramo obdelati z metodo `beriXML`, potrebujemo še točno (popolno) pot do te datoteke. V ta namen je v javi na voljo metoda `getAbsolutePath()`. Vsakič, ko naletim na posamezno datoteko, na zaslon še izpišemo njeno ime. Ker ne potrebujemo polnega imena (skupaj s potjo), izpišemo tisto, kar nam da metoda `getName()`, ki vrne osnovno ime datoteke.

Pri obdelavi datotek pregledujemo tudi ali imamo za posamezno datoteko ali mapo dovoljenje za branje. To storimo z metodo `canRead()`, ki preveri ali datoteko (ki je bodisi mapa, bodisi običajna datoteka) sploh lahko beremo.

```

try {
    String xml = "xml";

    // mapaXML povežemo z datoteko z imenom xml v tekoči mapi.
    File mapaXML = new File(xml);
    // gledamo ali je v tej mapi res podmapa xml
    // in če je lahko beremo
    if (mapaXML.exists() && // ali mapa obstaja
        mapaXML.isDirectory() && // ali ime xml res označuje imenik
        mapaXML.canRead()) { // ali se mapo da brati
        // iz mape xml izberemo seznam datotek in map
        File[] mapa = mapaXML.listFiles();
        for (int i = 0; i < mapa.length; i++) {
            // za vsako datoteko pregledamo ali je
            // mapa in če jo lahko beremo
            if (mapa[i].isDirectory() &&
                mapa[i].canRead()) {
                // če gre za podmapo preberemo seznam datotek in map
                // znotraj posamezne podmape
                File[] datoteke = mapa[i].listFiles();
                // se sprehodimo čez ta seznam
                for (int j = 0; j < datoteke.length; j++) {
                    if (datoteke[j].isFile()) { // če je v mapi datoteka
                        // zaradi kontrole izpišemo zaporedno številko datoteke
                        // in ime vsake datoteke XML, ki jo obdelujemo
                        System.out.println(j+1 + "\t" +
                            datoteke[j].getName() + " ----- ");
                        // pokličemo metodo za odstranjevanje vrstic DTD
                        // v datoteki xml
                        odstraniDTDVrstici(datoteke[j].getAbsolutePath());
                        // obdelamo vsako posamezno datoteko
                        beriXML(datoteke[j].getAbsolutePath(), povezava);
                    } else {
                        System.out.println("Ni datoteka xml, ampak mapa: " +
                            datoteke[j].getName());
                    }
                }
            } else {
                System.out.println("Znotraj mape xml ni podmape");
            }
        }
    } else {
        System.out.println("Ne najdemo mape xml, " +
            "ali pa ni mogoče za branje");
    }
} catch (Exception ex) {
    System.out.println("Napaka pri Branju mape z XMLji");
    ex.printStackTrace();
    System.exit(1);
}

```

2.3.4. Prebiranje datoteke XML

Eden od problemov, ki sem ga rešil z novim programom, je bil učinkovito prebiranje datoteke XML. Tukaj želimo iz posamezne datoteke XML izluščiti ustrezne podatke za posamezni račun. Datoteke XML, ki jih dobim iz sistema IMS, oziroma potem, ko sem odstranil sklice na DTD datoteko, imajo naslednjo strukturo (prikazanih je le prvih nekaj nivojev)

```

- <INVOICES>
- <INVOICE>
  - <INVOICE_HEADER>
    <INVOICE_NUMBER>93541503</INVOICE_NUMBER>
    <INVOICE_DATE>2006-10-04</INVOICE_DATE>
    <DOCUMENT_TYPE>Recurring Invoice</DOCUMENT_TYPE>
    <DOCUMENT_SUBTYPE>Billing Event Type Details</DOCUMENT_SUBTYPE>
    <ACCOUNT_NUMBER>1340416</ACCOUNT_NUMBER>
    <PRICE_AMOUNT>5666.67</PRICE_AMOUNT>
  </INVOICE_HEADER>
  + <CONTACT></CONTACT>
  + <CONTACT></CONTACT>
  <ACCOUNT_PRICE_AMOUNT>5666.67</ACCOUNT_PRICE_AMOUNT>
  - <PRODUCT>
    <PRODUCT_ID>1000806</PRODUCT_ID>
    <PRODUCT_NAME>voljADSLx20</PRODUCT_NAME>
    <PRODUCT_PRICE_AMOUNT>5666.67</PRODUCT_PRICE_AMOUNT>
  - <COMMERCIAL_SERVICE>
    <CS_ID>1001857</CS_ID>
    <CS_PRICE_AMOUNT>5666.67</CS_PRICE_AMOUNT>
    <CS_VAT_AMOUNT>1133.334</CS_VAT_AMOUNT>
  - <BILLING_EVENT_TYPE>
    <BET_ID>1000010</BET_ID>
    <BET_NAME>Naročnina20</BET_NAME>
    <BET_PRICE_AMOUNT>5666.67</BET_PRICE_AMOUNT>
    <BET_VAT_AMOUNT>1133.334</BET_VAT_AMOUNT>
  </BILLING_EVENT_TYPE>
  </COMMERCIAL_SERVICE>
</PRODUCT>
</INVOICE>
</INVOICES>

```

Uporabil sem razred DOMParser iz paketa org.apache.xerces.parsers. Z enostavnim klicem parser.parse(ime_datoteke_XML) razčlenimo tekstovno datoteko z imenom ime_datoteke_XML na osnovne sestavne dele, ki predstavljajo podatkovno strukturo drevesa xml vozlišč. Tako dobimo objekt Document, ki ga nato obdelujemo. Pri tem uporabljamo dva objekta in sicer:

- Node, ki predstavlja posamezno vozlišče xml dokumenta in
- NodeList, ki predstavlja seznam vozlišč z istim staršem.

V sami kodi uporabimo tudi postopek, s pomočjo katerega rešimo problem nezaključene glavne značke. Ta postopek smo opisali v razdelku 2.3.2.

Poglejmo si kodo

Prebiranje datoteke XML s pomočjo objekta DOMParser

```
public static void beriXML(String ime_datoteke_XML, IMS povezava) {
    try {
        //naredimo nov objekt razreda DOMParser
        DOMParser parser = new DOMParser();
        //sparsamo datoteke xml
        parser.parse(ime_datoteke_XML);
        //dobimo objekt Dokument iz datoteke xml
        Document doc = parser.getDocument();
        //dobimo osnovni glavni dokument
        //(v našem primeru INVOICES)
        NodeList seznam_glavnih_elementov = doc.getChildNodes();
        //glavni element vozlišča (spisek vseh računov)
        Node glavni_element = seznam_glavnih_elementov.item(0);
        // iz osnovnega vozlišča dobimo seznam vseh njegovih potomcev
        NodeList invoices_list= glavni_element.getChildNodes();
        // gremo čez seznam vseh vozlišč, v našem primeru pomeni vsako
        // vozlišče svoj račun.
        for (int i = 0; i<invoices_list.getLength(); i++){
            // zapomnimo si vsako vozlišče (račun)
            Node invoice = invoices_list.item(i);
            // če je ime vozlišča INVOICE, smo našli račun
            if(invoice.getNodeName().equals("INVOICE")){
                //začnemo obdelovati storitve za vsak račun
                Racun r = obdelajRacun(invoice, povezava);
                //račun, ki ga preberemo iz xml, zapišemo v datoteko
                r.zapisiVDatoteko(povezava);
            }
        }
    }
    catch(org.xml.sax.SAXParseException ex) {
        // Popravimo zadnjo značko
        System.out.println("Popravimo zadnjo značko");
        try {
            PrintWriter datoteka =
                new PrintWriter(new FileWriter(ime_datoteke_XML, true));
            datoteka.println("</INVOICES>");
            datoteka.close();
        } catch (Exception e) {
            System.out.println("Napaka pri pisanju v "+
                                ime_datoteke_XML + "");
            e.printStackTrace();
            System.exit(1);
        }
        // ponovno začnemo od začetka
        beriXML(ime_datoteke_XML, povezava);
    } catch (Exception ex) {
        // gre za neko drugo napako pri branju, ki ni povezana s
        // strukturo XML
        System.out.println("Napaka pri parsanju dokumenta");
        ex.printStackTrace();
        System.exit(1);
    }
}
```

2.4. Razred Racun

Osnovni objekt, ki ga uporabljamo v našem programu, je **Racun**.

V njem hranimo naslednje podatke:

- Ključ, ki je enolična naročniška številka.
- Podatke o storitvah, na katere je naročnik naročen. Za vsako storitev so ti podatki cena storitve, ime storitve, ki naj se pojavi na računu, datum začetka zaračunavanja storitve in datum konca zaračunavanja storitve.

Podatke pridobimo iz večih virov:

- iz datotek XML,
- iz baze Oracle, ki je podatkovna baza sistema IMS,
- iz sistema IMS s pomočjo že napisanih metod v programskem jeziku java, saj je sistem IMS sprogramiran večinoma v programskem jeziku java,
- iz pomožnih datotek, ki se nahajajo v podmapi aplikacije. Ime te podmape je »Branje« in vsebuje tudi podmape:
 - V datoteki »cs_id_name.txt« so podatki o slovenskih imenih storitev.
 - Ko zaženemo aplikacijo, se v podmapi »Produkti« ustvarijo datoteke s podatki o različnih storitvah. Te so namenjene zapisovanju računov v različne datoteke. To olajša uvoz podatkov v program Microsoft Navision. Vseh teh datotek je 14.
 - Prav tako se ob zagonu aplikacije v podmapi »Ims« ustvari datoteka »Poraba_fit2006-08-01.txt" (seveda z ustreznim datumom). V njej so zapisani podatki o porabi storitve "VOLJADSL FIT" za mesec obračuna. Iz datotek XML namreč ne dobimo pravilne porabe.
 - V podmapi »Ims« se ob začetku izvajanja ustvari tudi datoteka »popusti_telefonija.txt«. V njej so vpisane naročniške številke in destinacije tistih naročnikov, ki imajo pri klicih na te destinacije 10% popust. Datoteka se uporablja pri obračunavanju klicev pri telefonskih storitvah.
 - v podmapi »Ceniki« se ustvarijo vsi ceniki, ki so potrebni pri zaračunavanju klicev. Cenik se ustvari, ko se aplikacija izvaja. Na začetku obračuna ni nobenega cenika, Ti se ustvarijo sproti, ko aplikacija naleti na naročnika s cenikom, ki še ni ustvarjen. Imena cenikov, ki so se uporabljali v mesecu juliju 2006, so:
 - VLT_080 31.07.2006.csv
 - VLT_IP_Pricelist 31.07.2006.csv
 - VLT_Pricelist 31.07.2006.csv
 - VLT_Pricelist_NajTrajaDlje 31.07.2006.csv
 - VLT_Telekom_Austria_AG 31.07.2006.csv

Posamezen cenik se ustvari (če ne obstaja), ko naletimo na prvega naročnika, ki mu moramo zaračunati telefonske klice po tem ceniku. Pravilo pri uvajanju novih cen v cenik v podjetju Voljatel je tako, da se cene spreminjajo vedno prvega dne v mesecu. Tako pri pisanju cenika vedno vzamem cenik na zadnji dan meseca, za katerega izvajamo obračun klicev. Namen teh cenikov je obračun klicev po destinacijah.

- Včasih iz uprave sporočijo, da so se določenim storitvam cene spremenile. V sistemu IMS se teh cen iz različnih vzrokov ni dalo popraviti. Zato so v datotekah »kabel produkti - nove cene - anze.csv« in »Drugacne_cene.txt« zavedene tiste cene, ki jih je pri obračunu potrebno upoštevati.
- Določenim storitvam so se spremenile šifre. V sistemu IMS so za storitve uporabljene le nove šifre. Zaradi potreb določenih statistik, ki jih izvaja finančni oddelček po uvozu in knjiženju računov v programu Navision, pa je potrebno v določenih datotekah, ki jih pripravljamo za uvoz v Navision uporabiti stare šifre storitev. V datoteki »Zamenjava_cs_id.txt« se poleg nove šifre storitve nahaja tudi stara šifra storitve.
- V datoteki so zavedene cene vseh storitev za posameznega naročnika na prvega dne v mesecu. Primer imena datoteke za mesec avgust 2006 je »Produkti_01.08.2006 z 0sit.txt«. Ta datoteka nam služi za celomesečno ceno njegove storitve. V datoteki so v vsaki vrstici zavedeni naslednji podatki: naročniška številka, šifra storitve, celomesečna cena storitve in dve dodatni polji z angleškim imenom paketa storitve in angleškim imenom storitve. S pomočjo te datoteke lahko preračunamo deleže naročnin. Za storitve, ki se zaračunavajo za preteklo obdobje, sme biti v datoteki XML zavedena cena enaka ali pa manjša od te celomesečne cene. Če je v XML manjša cena, se zaračuna ustrezeni delež celomesečne naročnine. Ta primer se pojavi pri vseh priklopih storitev v preteklem mesecu. Za storitve, ki se zaračunavajo za obdobje, ki prihaja, pa je cena iz XML lahko enaka kot celomesečna cena ali pa do enkrat večja v primeru priklopa. Tako lahko razbijemo večjo ceno iz XML datoteke na dva dela in sicer na naročnino, ki se zaračuna za prihajajoče obdobje in ustrezeni delež naročnine za preteklo obdobje.
- V podmapi »Navision« je datoteka »blokade.txt«. Vanjo se zapišejo tisti naročniki, ki so ob zagonu aplikacije v programu Navision blokirani. Računov naročnikov, ki so blokirani, namreč v program Navision ne moremo uvažati. Zato jih pri preurejanju računov damo v posebno datoteko.
- Druga datoteka, ki se ustvari v podmapi »Navision«, je poimenovana »vse_ims_stevilke.txt«, kjer so zavedene naročniške številke, ki obstajajo v programu Navision. Ta datoteka nam prav tako pomaga pri razvrščanju storitev v datoteke. Če namreč v program Navision želimo uvoziti račune naročnikov, katerih naročniških številke v programu ni, program pri uvozu javi napako, da naročniška številka ne obstaja. S pomočjo omenjene datoteke ločimo račune tistih naročnikov, ki so v Navision in tistih, katere številke v programu Navision ne obstajajo.
- V datoteki »IMS_slo_imena_drzav.txt« so zapisane kombinacije angleških in prevedenih slovenskih imen destinacij za možne telefonske klice, ki se lahko pojavijo pri obračunavanju telefonskih klicev. Posamezen zapis angleškega imena in prevedenega slovenskega imena je ločen s tabulatorjem. V tej datoteki je v vsaki vrstici zapisana po ena destinacija.

2.4.1. Komponente razreda Racun

Oglejmo si, katere komponente vsebuje razred, čemu služijo in njihove tipe

Komponente razreda Racun

```
// ali naj se pri obdelavi upoštevajo tudi telefonske storitve
// false : v objektih Racun so tudi telefonske storitve
// true : obračun bo spustil vse telefonske storitve
private boolean spusti = false;

// znak, ki loči podatke na računih
private final static char LOCILO = '|';

// ključ oziroma enolična naročniška številka
String account_id = null;

// pomembno za pravilno nastavitvev obdobja knjiženja na računih
private boolean datum_nazaj = false;

// skupna cena vseh storitev na računu za posameznega naročnika
double znesek_celotnega_racuna = 0.0;

// vektor podatkov o storitvah, ki jih dobimo iz datoteke XML
private Vector v = new Vector();

// vektor vseh posebnih storitev, ki jih polnimo iz datoteke XML
private Vector posebne_storitve = new Vector();

// v3 pomeni tretjo vrstico v računu
private String v3 = null;

//vektorji, posameznih vrstic, ki jih polnimo pri urejanju storitev

// v2 pomeni druge vrstice v računu
private Vector v2 = new Vector();

// v4 pomeni četrte vrstice v računu
private Vector v4 = new Vector();

// v5 pomeni pete vrstice v računu
private Vector v5 = new Vector();
```

S pomočjo komponente `spusti` nadziramo, ali naj se pri obračunu upoštevajo telefonske storitve ali ne. Ker moramo za posameznega naročnika za vsako telefonsko storitev pregledati porabo njegovih telefonskih klicev v bazi Oracle, to stane precej dragocenega časa. Zato vedno prvič poženem obračun brez telefonskih storitev. Obračun namreč vedno poganjam dvakrat. Prvi prehod je namenjen testiranju vseh storitev razen telefonskih in iskanju novih storitev. Pri prvem prehodu se obračun sprehodi po vseh datotekah XML. Če naletimo na novo storitev, ki nima imena, na zaslon izpišemo, za katero naročniško številko in za katero šifro storitve manjka opis. Obenem se zažene metoda, ki v datoteko z imenom »`cs_id_name.txt`« doda šifro storitve in angleški opis storitve ter podatke o opisu tudi izpiše na zaslon. Po končanem pregledu vseh datotek XML v programu Textpad odprem to datoteko in uredim imena novih storitev za pravilni prikaz na računih. Ko to naredim,

nastavim komponento `spusti` na `false` in ponovno zaženem aplikacijo, kjer so sedaj zavedena vsa nova imena storitev. Pri tem prehodu pa se telefonske storitve upoštevajo.

V komponento posebne storitve tipa `Vektor` shranimo podatke o posebnih storitvah. Podatka, ki ju dodamo v ta vektor, sta ime posebne storitve in njena cena. Primer posebnih storitev so na primer ponovni priklop, pogodbeni kazni, namestitve `adsl`, ...

V komponenti `datum_nazaj` si zapomnimo, če smo pri pregledu računov naleteli na porabo ali pa na vsaj eno storitev, ki se obračunava za nazaj. Vrednost `false` pomeni, da take storitve nimamo. To komponento potrebujemo, da pravilno nastavimo datum v tretji vrstici.

Skupna cena vseh storitev `znesek_celotnega_racuna` je namenjena razporejanju računov v datoteke. V datoteko z imenom `»cena_pod_100_sit.txt«` so zapisani računi, ki imajo ceno manjšo od 100 sit. Vsak račun, ki ima znesek pod 100 sit, po obdelavi ročno pregleдам in ugotavljam razlog, zakaj je zaračunan takšen znesek.

V vektor `v` shranimo podatke o storitvah, ki jih dobimo iz datoteke `XML`, ter jih naknadno obdelamo. Podatki, ki jih dodajamo v vektor, so ključ paketa storitve, šifra storitve, tip storitve in cena za to storitev. Tip storitve nam pove, ali gre za naročnino, internetno ali telefonsko porabo ali pa za priključnino.

V komponento `v3` tipa `String` shranimo tretjo vrstico računa. Ta je sestavljena iz niza `»3«`, začetnega datuma, končnega datuma, datuma knjiženja računov. Vsa polja so ločena z komponento `LOCILO`. Ti datumi so zapisani v obliki nizov in so zapisani po naslednjem principu. Če obstaja vsaj ena storitev, ki se zaračunava za preteklo obdobje, torej če imamo kakršnokoli telefonsko ali internetno porabo, priključnino ali če se naročnina za katerokoli storitev zaračunava za prejšni mesec, se določijo tu vsi datumi za prejšni mesec. Po navodilih finančnega oddelka se kot datum knjiženja računov zapiše zadnji dan v preteklem mesecu. Če imamo samo storitve, ki se zaračunavajo za tekoče mesečno obdobje, se tu nastavijo datumi za tekoče obračunsko obdobje. Kot datum knjiženja obračuna se nastavi 10. dan v tekočem mesecu.

Poglejmo še opis vektorjev `v2`, `v4` in `v5`. Posamezni element vektorja predstavlja posamezno vrstico:

- v `v2` hranimo vrstice v posameznem računu, ki se začnejo z oznako 2 (glej stran 16). Teh vrstic je lahko več. Vsak zapis v vektorju je napolnjen s celotno vrstico, ki bo zapisana na računu in ima vrednosti: niz `»2«`, šifra storitve, ime storitve, začetni datum, in končni datum. Vsi podatki so ločeni s konstanto `LOCILO`.
- v `v4` hranimo vrstice v posameznem računu, ki se začnejo z oznako 4 (glej stran 16). Teh vrstic je lahko več, vedno pa mora biti število teh vrstic enako številu vrstic, ki se začnejo z dvojko. Torej dolžini obeh vektorjev `v2` in `v4` morata biti enaki. Vsak zapis v vektorju je napolnjen s celotno vrstico, ki bo zapisana na računu in ima vrednosti: niz `»4«`, opis datuma vrstice na računu, cena storitve, količina storitve, opis enote za storitev (min, mes, kos). Vsi podatki so ločeni s konstanto `LOCILO`.
- v `v5` hranimo vrstice v posameznem računu, ki se začnejo z oznako 5 (glej stran 16). Teh vrstic je lahko več. Vsak zapis v vektorju ustreza celotni vrstici, ki bo zapisana na računu in ima vrednostmi: niz `»5«`, `»A023«`, `»TUJE«`, ime artikla, cena artikla, število kosov, oznaka enote, popust izražen v procentih. V tem vektorju so zavedeni artikli, ki se morajo zaračunati naročniku. Ti morajo v programu `Microsoft Navision` skozi drugi postopek, ker mora artikel iti najprej iz zaloge, šele nato se ga lahko uvozi. Vsi podatki so ločeni s konstanto `LOCILO`.

2.4.2. Metode razreda Racun

Razred vsebuje precej metod. Glavne metode sem na kratko opisal v razdelku 2.1. Tu naštejmo vse, navedimo njihove parametre in njihov namen. Za določene ključne metode si bomo ogledali, kako smo jih sprogramirali in kakšne postopke smo ob tem uporabili.

Metode razreda Racun

```
// metoda, kjer iz datotek XML v objekt Racun dodajamo
// podatke o storitvah
public void dodajStoritev(String[] s)

// metoda, kjer iz datotek XML v objekt Racun dodajamo
// posebne storitve
public void dodajPosebneStoritve (String[] s)

// metoda, kjer obdelamo in oblikujemo vse podatke iz osnovnih dveh
// komponent in sicer komponente v in komponente posebne_storitve
private void oblikujPodatkeVRacun (IMS povezava)

// metoda, ki nam za podana dva parametra naročniško številko in šifro
// storitve vrne v obliki niza ceno storitve za celotni mesec iz
// sistema IMS.
private static String najdiCenoIzIMS (String account_id, String cs_id)

//metoda, ki za dano šifro storitve vrne niz z imenom storitve
private static String najdiImeStoritve (String cs_id, String account_id)

// metoda, ki za dano šifro storitve poišče, če
// imamo v posebni datoteki zavedeno drugo šifro
private static String zamenjajSifroStoritve (String cs_id)

// metoda, ki vrne novo ceno za vse storitve, ki so zavedene v
// datoteki »kabel produkti - nove cene - anze.csv«, drugače vrne
// podano ceno iz parametra znesek. Prej metoda še preveri ali
// je naročniška številka zavedena v datoteki »Drugacne_cene.txt«.
// V primeru, da je zavedena vrne znesek zaveden v tej datoteki.
private static String spremeniCenoVNovoCeno (String cs_id, String znesek,
String account_id)

// metoda, ki vrne staro ceno za vse storitve, ki so zavedene v
// datoteki »kabel produkti - nove cene - anze.csv«, drugače vrne
// podano ceno iz parametra znesek. Prej metoda še preveri, ali
// je naročniška številka zavedena v datoteki »Drugacne_cene.txt«.
// V primeru, da je zavedena vrne znesek zaveden v tej datoteki.
private static String spremeniCenoVStaroCeno (String cs_id, String
znesek, String account_id)

// metoda, ki za dano število v obliki niza vrne to število
// zaokroženo na dve decimalni mesti
private static String zaokrozi (double znesek)

// metoda, ki uredi vse šifre storitev
// kličemo jo šele potem, ko je končana obdelava vseh računov
private void urediSifreStoritev ()

// metoda, ki račun zapiše v datoteko
public void zapisiVDatoteko (IMS povezava)
```

Metode:

- `public void dodajStoritev (String[] s)`

Metoda, s pomočjo katere v objekt tipa `Racun` shranujemo podatke o storitvah. Te smo dobili iz datotek XML. Metoda sprejme en parameter in sicer tabelo nizov `s`. Tabela `s` je tabela 4 elementov. Prvi, `s[0]`, je ključ paketa storitev. Ta nam pomaga pri razporejanje računov, pridobivanju določenih podatkov o paketu storitev iz baze Oracle. Drugi element `s[1]` je šifra storitve, s katero točno opredelimo in pridobimo ustrezne podatke o imenu storitve. V `s[2]` je naveden tip storitve, ki nam pove ali je storitev naročnina, priključnina, telefonska ali internetna poraba. Zadnji element, `s[3]`, je cena storitve, ki je prebrana iz XML. Metoda ne vrača ničesar, le polni komponento `v` tipa `Vector`.

- `public void dodajPosebneStoritve (String opisni_kljuc_storitve, String znesek)`

Metoda, s katero prenesemo podatke v komponento posebne_storitve. Prvi parameter je `opisni_kljuc_storitve`, ki je kratko ime storitve, s katerim definiramo vse potrebne podatke. Drugi parameter `znesek` je cena za to storitev.

- `private void oblikujPodatkeVRacun (IMS povezava)`

Metoda, ki uredi in obdela storitve in določi datume izdelave in knjiženja računov. Metoda sprejme en parameter in sicer objekt razreda `IMS`, kjer imamo vzpostavljeno povezavo na bazo Oracle.

Metoda črpa osnovne podatke iz osnovnih komponent objekta, vektorja `posebne_storitve` in vektorja `v`, kjer so shranjene vse storitve. Iz teh dveh vektorjev prebere vse storitve in jih ustrezno obdela in zapiše v štiri nove komponente `v2`, `v3`, `v4` in `v5` (glej stran 51). Metoda ne vrača ničesar, le ustrezno uredi vse vrstice računov.

- `private static String najdiCenoIzIMS (String account_id, String cs_id)`

Metoda v obliki niza vrne ceno storitve za celotno mesečno obdobje iz sistema `IMS`. Sprejme dva parametra in sicer naročniško številko in šifro storitve. Metoda nam koristi za pridobitev podatka o ceni storitve za celotno mesečno obdobje. Na ta način preračunamo število dni v mesecu, za katere moramo izstaviti račun.

Če obravnavamo storitev, ki se mora zaračunati za preteklo mesečno obdobje, je cena iz datotek XML manjša ali kvečjemu enaka tej ceni. Če je cena iz datotek XML manjša, to je v primeru priklopa nove storitve, ustrezno izračunamo delež dni v mesecu za katerega moramo izstaviti račun. Postopek, kjer je razloženo zakaj je potrebna cena storitve za celotno mesečno obdobje, ki se izvaja v metodi `oblikujPodatkeVRacun`.

Če moramo storitev zaračunati za tekoče mesečno obdobje, je cena iz datotek XML enaka ali kvečjemu enkrat večja ceni storitve za celotno mesečno obdobje. V primeru, ko je cena v XML večja od cene storitve za celotno mesečno obdobje, zaračunamo naročniku dve postavki. Prva postavka je cena za prejšno mesečno obdobje z ustreznim deležem dni v mesecu, druga postavka na računu pa je za tekoče mesečno obdobje.

Vse cene iz sistema `IMS` so pobrane na prvi delovni dan v mesecu in se zapišejo v določeno datoteko. Iz te datoteke dobimo ceno storitve za podano naročniško številko in šifro storitve.

- `private static String najdiImeStoritve (String cs_id, String account_id)`

Metoda, ki za podano šifro storitve vrne niz z imenom storitve. Če podana šifra storitve v datoteki »cs_id_name.txt«, kjer so zavedene šifre in opisi storitev, ne obstaja, aplikacija preveri originalno ime storitve v bazi podatkov Oracle. Nato zapiše šifro in opis storitve v to datoteko, zapiše originalno ime storitve na račun in izpiše podatke o storitvi in naročniški številki na zaslon.

- `private static String zamenjajSifroStoritve (String cs_id)`

Metoda za podano šifro storitve v datoteki »Zamenjava_cs_id.txt« poišče šifro, v katero se mora spremeniti podana šifra. Če šifra v tej datoteki ne obstaja, metoda vrne podano šifro.

- `private static String spremeniCenoVNovoCeno (String cs_id, String price, String account_id)`

Metoda je namenjena desetim kabelskim storitvam, kjer so administratorji prepoznano spremenili cene storitev v IMS in sicer konec zadnjega dne v mesecu. Tako so se cene teh storitev zaračunale po starih cenah in so take cene v datotekah XML. V datoteki, kjer imamo cene storitev za celotno mesečno obdobje, pa so se že pojavile nove cene. Tako komponento `cena_iz_IMS` s pomočjo te metode zamenjam v novo, ko so že preračunani deleži naročnin. Če šifra storitve v datoteki ne obstaja, metoda vrne komponento `cena_iz_IMS`.

Primer:

Cena storitve v XML je 5241.67, `cena_iz_IMS` je 5241.67, šifra storitve `cs_id` je 1000729. Ker se ta šifra nahaja v datoteki »kabel produkti - nove cene - anze.csv« vrne metoda novo ceno 3750.00, tako da se na račun izpiše pravilna nova cena.

- `private static String spremeniCenoVStaroCeno (String cs_id, String price_iz_ims, String account_id)`

Tudi ta metoda je namenjena desetim kabelskim storitvam, kjer so administratorji prepoznano spremenili cene storitev v IMS. S pomočjo te metode komponento `cena_iz_IMS` zamenjam v staro, da se pravilno obračunava delež naročnine. Če šifra storitve v datoteki ne obstaja, metoda vrne komponento `cena_iz_IMS`.

Primer:

Cena storitve v XML je 5241.67, `cena_iz_IMS` je 3750.0, šifra storitve `cs_id` je 1000729.

Ker se ta šifra nahaja v datoteki »kabel produkti - nove cene - anze.csv« vrne metoda staro ceno 5241.67, tako da se pravilno obračunajo storitve.

- `private static String zaokrozi (double znesek)`

Metoda, ki za podan parameter `znesek` tipa `double`, kjer je zavedeno nezaokroženo število, v obliki niza vrne to število zaokroženo na dve decimalni mesti

- `private void urediSifreStoritev ()`

Metoda po končanem obdelovanju storitev, pred zapisom v datoteko, uredi vse šifre storitev tega računa. Ta metoda zamenja tiste šifre storitev, ki so nove in se morajo zaradi potreb naknadnih statistik iz programa Navision knjižiti pod starimi šiframi. Metoda ne vrača nobene vrednosti, samo ustrezno preoblikuje komponento `v2`. Spomnimo se, da so tam shranjeni podatki o storitvah, med njimi tudi šifra storitve. Dejansko se metoda

sprehodi po vseh elementih vektorja `v2`, izloči podatek o šifri in pokliče metodo `zamenjajSifroStoritve(String cs_id)`. Ta vrne šifro, ki se mora zapisati na račun. Nato se šifra v elementu vektorja `v2` zamenja.

- `public void zapisivDatoteko (IMS povezava)`
Metoda, ki sprejme en parameter in sicer objekt razreda `IMS`. Ta objekt je sklic na povezavo s podatkovno bazo `Oracle`. Metoda je namenjena zapisu računa v datoteko po postopku, opisanem na straneh 8 in 9.

Opišimo sedaj še nekatere zanimive postopke, ki smo jih uporabili pri pisanju metod. Pogosto v metodah razbijamo nize. Za to uporabljamo razred `StringTokenizer`. S pomočjo tega razreda razbijemo posamezen niz po podanem ločitvenem znaku.

Primer uporabe tega razreda si bomo pogledali v delu metode `najdiImeStoritve()` iz razreda `Racun`.

Metoda, kjer s pomočjo šifre storitve poišče opisno ime, ki naj bo izpisano na računu

```
/**
 * Metoda v datoteki "branje\cs_id_name.txt" poišče ime
 * za storitev, ki pripada dani šifri storitve cs_id.
 * Če šifre storitve v datoteki metoda ne najde,
 * izpiše na zaslon šifro storitve, naročniško številko account_id
 * in delne podatke o tej storitvi iz baze Oracle,
 * obenem pa se podatki o tej storitvi
 * zapišejo v zgoraj navedeno datoteko.
 * @param cs_id Šifra storitve
 * @param account_id Naročniška številka
 * @return Ime, ki naj bo izpisan na račun za to storitev
 */
private static String najdiImeStoritve(String cs_id, String account_id){
    try {
        String ime = null;
        //odpremo datoteko za branje
        BufferedReader cs = new BufferedReader(
            new FileReader("branje\cs_id_name.txt"));
        //se sprehodimo čez vse vrstice datoteke
        while(cs.ready()) {
            //preberemo tekočo vrstico
            String vrstica = cs.readLine();
            //razbijemo vrstico na posamezne dele
            //ločene z podpičjem
            StringTokenizer st =
                new StringTokenizer(vrstica, ";");
            //preberemo tekočo šifro storitve prvi del vrstice
            String sifra_storitve = "" + st.nextToken();
            //če je šifra storitve ista kot podana
            if(sifra_storitve.equals(cs_id)) {
                //sestavimo ime iz naslednjih dveh
                //podatkov vrstice
                ime = st.nextToken() + st.nextToken();
                //končamo zanko, ker smo rešitev našli
                break;
            }
        }
        //zapremo datoteko
    }
```

```

        cs.close();
        //če ime obstaja vrnemoc ime
        if(ime != null)
            return ime;
        //če ime ne obstaja
        else {
            System.out.print("Account id: " + account_id);
            System.out.println("\tNi imena za ta cs_id: " + cs_id);
            //vrnemo ime prebrano iz baze
            return DodajanjeImenNovihStoritev.
                vrniInDodajImeNoveStoritve(cs_id);
        }
    }
    //če je prišlo do kakršnekoli napake
    catch (Exception ex) {
        //izpišemo napako na zaslon
        System.out.println("Napaka pri Racun.najdi_cs_name" +
            "(cs_id: "+cs_id+" + acc: " + account_id + ")");
        ex.printStackTrace();
        //zaključimo program
        System.exit(1);
    }
    //moramo dodati v metodo, ker to zahteva java prevajalnik,
    //čeprav nikoli ne pride do tega stanja
    return null;
}

```

Pri urejanju zapisov na računih uporabljamo metodo `zaokrozi(double vrednost)`, ki sprejme vrednost, ki jo je potrebno zaokrožiti na dve decimalni mesti natančno in vrniti v obliki urejenega niza primerne za izpis na računu.

Metoda s pomočjo katere zaokrožimo podano vrednost na dve decimalni mesti natančno

```

/**
 * Metoda, ki dano vrednost znesek zaokroži na
 * dve decimalni mesti in jo vrne v obliki niza.
 * @param vrednost vrednost tipa double, ki ga želimo zaokrožiti
 * @return zaokroženo vrednost v obliki urejenega niza
 */
private static String zaokrozi(double vrednost) {
    //zaokrožimo vrednost na dva decimalna mesta natančno
    vrednost = Math.round(vrednost*100)/100.0;
    //razbijemo spremenjeno vrednost v tip String
    //z ločitvenim znakom pika
    StringTokenizer cen = new StringTokenizer(""+vrednost, ".");
    //prvi del niza, ki je pred znakom pika
    //preberemo in dodamo znak za vejco
    String ime_1 = cen.nextToken() + ",";
    //preberemo drugi niz, ki je za piko
    String decje = ""+cen.nextToken();
    //če je samo en znak za decimalno piko dodamo na koncu 0
    if(decje.length()==1){
        decje+="0";
    }
    //če ni znakov za decimalno piko dodamo na koncu dve ničli
    else if(decje.length()==0) {
        decje+="00";
    }
}

```

```
}  
//vrednosti dodamo decimalke  
ime_1 +=decje;  
//vrnemo vrednost  
return ime_1;  
}
```

Zaključek

Vsebina diplomske naloge obsega postopek projekta obračuna, ki ga dograjujem in izpopolnjujem z dodatnimi programi, ki analizirajo podatke, ki nastajajo pri izdelovanju.

Projekt se bo izpopolnjeval, ker se dodajajo vedno nove storitve, ki jih moram vgraditi v program, nova slovenska imena, drugačen način zaračunavanja storitev, dodatne spremembe cen, različni popusti.

Literatura in viri

- [1] **Vladimir Batagelj, Matjaž Zaveršnik**, *Označevanje in XML*, Delavnica MIRK 2004, (dostop 20.11.2006)
<http://www.educa.fmf.uni-lj.si/izodel/dela/XML/>
- [2] **Matjaž Prtenjak**, *Hitra šola XML*, prva izdaja, (dostop 20.11.2006)
http://www.matjazev.net/pnew/tutorials/XML_Tutorial/book.html
- [3] **Matjaž Prtenjak**, *Hitra šola DTD*, prva izdaja, (dostop 20.11.2006)
http://www.matjazev.net/pnew/tutorials/DTD_Tutorial/book.html
- [4] **Dokumentacija XMLParser**, (dostop 20.11.2006)
<http://xerces.apache.org/xerces-j/apiDocs/org/apache/xerces/framework/XMLParser.html>
- [5] **Digiquant**, *IMS 3.2.1 Internet Management System – User Manual*, 2000
- [6] **Digiquant**, *IMS Service Management*, 2003
- [7] **Digiquant**, *IMS SDK Training*, 22- 26. september 2003
- [8] **Interna dokumentacija aplikacije Microsoft Navision**, Uporabniški priročnik za Microsoft Business Solutions – Navision 3.60
- [9] **Tina Ivaniš**, *Seminarska naloga TreeMap*, (dostop 28. 11.2006)
http://haka.fmf.uni-lj.si/~robert/r2pra/seminarske/ivanis_tina/index.html
- [10] **Računalništvo1 (PRA)**, *Razred Vector*, (dostop 5.12.2006)
<http://haka.fmf.uni-lj.si/praracunalnistvo-1/lekcija13/index.html>