UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Rok Skupek

APLIKACIJA ZA AVTOMATSKO POSREDOVANJE ELEKTRONSKIH SPOROČIL

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: dr. Igor Rožanc

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

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

Št. naloge: 00512/2010
Datum: 05.04.2010

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

Kandidat: **ROK SKUPEK**

Naslov: **APLIKACIJA ZA AVTOMATSKO POSREDOVANJE ELEKTRONSKIH SPOROČIL**

APPLICATION FOR AUTOMATIC E-MAIL FORWARDING

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

Pri uporabi elektronske pošte včasih potrebujemo opravila, ki jih komercialni odjemalci elektronske pošte ne omogočajo. Diplomska naloga prikazuje, kako si tovrstno rešitev izdelamo sami.

V diplomski nalogi najprej predstavite način delovanja elektronske pošte z osnovnimi protokoli in standardi. Na osnovi teh zasnujte in izdelajte knjižnico za komunikacijo med odjemalcem in poštnim strežnikom preko protokola POP3 v .NET ogrodju. Z njeno uporabo načrtujete in izvedite aplikacijo za avtomatsko posredovanje elektronskih sporočil na več naslovov, ki deluje na podlagi nabora vnaprej določenih pravil. V zadnjem delu naloge predstavite uporabo aplikacije ter jo primerjajte s tipičnim odjemalcem elektronske pošte.

Mentor:
viš.pred. dr. Igor Rožanc

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Franc Solina

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Rok Skupek,
z vpisno številko 63040368,

sem avtor diplomskega dela z naslovom:
Aplikacija za avtomatsko posredovanje elektronskih sporočil

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom dr. Igorja Rožanca;
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela;
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 30.6.2010 Podpis avtorja/-ice:

Zahvala

Za pomoč in nasvete pri izdelavi diplomske naloge bi se rad zahvalil mentorju na Fakulteti za računalništvo in informatiko v Ljubljani, dr. Igorju Rožancu.

Zahvala gre tudi staršem, ki so mi študij omogočili.

KAZALO

POVZETEK 11

ABSTRACT 12

1 UVOD 13

2 OPISI UPORABLJENIH TEHNOLOGIJ 14

2.1 Splošno 14

2.2 POP3 protokol 14

2.3 POP3 ukazi 16

2.4 IMAP 17

2.5 Standard RFC822 - Internet Message Format 17

2.6 Trinivojska arhitektura 19

2.6.1 Predstavitveni nivo 20

2.6.2 Logični poslovni nivo 21

2.6.3 Podatkovni nivo 21

3 KNJIŽNICA ZA PRIDOBIVANJE ELEKTRONSKIH SPOROČIL IZ STREŽNIKA 22

3.1 Opis problema in zahtev 22

3.2 Komunikacija s strežnikom 22

3.2.1 Uporabljena rešitev 22

3.2.2 Razvoj razreda za komunikacijo z POP3 strežnikom 23

3.3 Pridobivanje podatkov iz elektronskega sporočila v izvorni obliki 26

4 PROGRAMSKI MODUL ZA AVTOMATSKO POSREDOVANJE ELEKTRONSKIH SPOROČIL 28

4.1 Opis zahtev 28

4.2 Razvoj 28

4.2.1 Podatkovni model 28

4.2.2 Razvoj grafičnega vmesnika 30

4.2.3 Arhitektura aplikacije 32

4.2.3.1 Razred za komunikacijo z podatkovno bazo 32
4.2.3.2 Prenos podatkov po nivojih 34
4.2.4 Nastavljanje lastnih pravil za obdelavo sporočil 36
4.3 Uporaba aplikacije ter primerjava z obstoječimi rešitvami 38
5 SKLEPNE UGOTOVITVE 42
Literatura in viri 43

Seznam slik

1 Primer tipične komunikacije odjemalec - strežnik preko protokola POP3.....14
2 Primer elektronskega sporočila v formatu RFC-882.....18
3 Primer zgradbe trinivojske aplikacije.....20
4 Diagram podatkovne baze.....29
5 Glavni obrazec aplikacije s prikazom neobdelanih sporočil.....30
6 Obrazec za pregled sporočila v uporabniku prijazni obliki.....31
7 Obrazec za pregled sporočila v izvorni obliki.....32
8 Obrazec za dodajanje novih pravil.....37
9 Obrazec za nastavljanje pravil v Outlook 2007.....39
10 Nastavljanje pravila za posredovanje sporočil v Microsoft Outlook.....40

Seznam uporabljenih kratic in simbolov

POP3	ang. Post Office Protocol 3 internetni protokol za pridobivanje elektronskih sporočil iz strežnika
IMAP	ang. Internet Message Access Protocol internetni protokol za pridobivanje elektronskih sporočil iz strežnika
TCP	ang. Transmission Control Protocol protokol za nadzor med prenosom podatkov
CLRF	ang. Carriage Return/Linefeed oznaka za skok na začetek nove vrstice
ASCII	ang. American Standard Code for Information Interchange ameriški standardni nabor za izmenjavo informacij
MIME	ang. Multipurpose Internet Mail Extensions večnamenska razširitev elektronske pošte
SMTP	ang. Simple Mail Transfer Protocol preprost protokol za prenos elektronskih sporočil
DLL	ang. Dynamic Link Library Microsoftova implementacija skupne knjižnice funkcij v operacijskih sistemih Windows in OS/2
SSL	ang. Secure Sockets Layer kriptografski protokol, ki omogoče varno komunikacijo po omrežju
SQL	ang. Structured Query Language strukturirani povpraševalni jezik za delo z podatkovnimi bazami
XML	ang. Extensible Markup Language razširljiv označevalni jezik

POVZETEK

Diplomska naloga ima dva cilja: prvi je izdelava knjižnice za pridobivanje sporočil elektronske pošte iz strežnika z uporabo protokola POP3. Drugi cilj obsega izdelavo aplikacije za obdelavo in razpošiljanje prejetih elektronskih sporočil.

V uvodnem delu je predstavljen praktičen primer uporabe aplikacije v realni situaciji. Opisani so predvideni postopki pri izdelavi knjižnice za komunikacijo s poštnim strežnikom z uporabo protokola POP3 ter primer vključitve knjižnice v dejansko uporabo.

Sledi opis protokola POP3, njegovega delovanja, vseh vrst ukazov ter primer dejanske komunikacije med odjemalcem in strežnikom. Odjemalec pošilja strežniku zahtevke s pomočjo dvanajstih preprostih ukazov, strežnik pa odgovori najprej z statusom in nato morebitnim odgovorom. Morebiten odgovor je lahko tudi sporočilo v formatu RFC-822, ki je sestavljeno iz glave sporočila (ang. header) ter morebitnih priponk. Pridobivanje uporabnih podatkov iz tega formata je tudi pomemben del naloge. Celotna aplikacija je narejena trinivojsko, kar omogoča lažje vzdrževanje, morebitne nadgradnje, uporabo druge podatkovne baze ali celo selitev aplikacije na splet.

Prva polovica jedra naloge je osredotočena na razvoj knjižnice za komunikacijo med odjemalcem in poštnim strežnikom preko protokola POP3. Knjižnica je razvita v ogrodju .NET v jeziku C#. Ključni del predstavlja metoda za prijavo odjemalca na strežnik, metoda, ki vrne vsa nova elektronska sporočila iz strežnika, ter del te metode, ki pridobi podatke in napolni objekt `MailDto` iz sporočila v formatu RFC-822.

Drugi del obsega razvoj aplikacije z uporabo trinivojskega pristopa v ogrodju .NET. Z diagramom je predstavljen podatkovni model za podatkovno bazo na strežniku SQL server 2008 Express, prenos podatkov po vseh treh nivojih aplikacije ter del uporabniškega vmesnika. Podrobno je predstavljen tudi sistem dodajanja pravil za pošiljanje elektronske pošte, metoda za tvorbo poizvedbe na podatkovni bazi, ter razred za komunikacijo z podatkovno bazo.

Na koncu so predstavljene ugotovitve, ter primer praktične uporabe aplikacije ter primerjava z odjemalcem elektronske pošte Microsoft Outlook.

1.1 Ključne besede

POP3 protokol, elektronska pošta, aplikacija za avtomatsko pošiljanje sporočil, ogrodje .NET

ABSTRACT

The diploma thesis has two main goals: the first objective is to develop a dynamic library, which will contain methods able to download e-mail messages from mail server using POP3 protocol. The second objective involves development of application for processing and distribution of e-mail messages.

Introduction cointains presentation of a practical application example in real time situation. Here are described procedures used for development of server - client communication library, which uses POP3 protocol and implicit inclusion of of such library in actual usage.

A description of the POP3 protocol, its operations, all types of commands and an example of effective communication between the client and the server follows. Client is sending commands to server through twelve simple commands and the server then replies first with the status and then with a possible answer. As a possible answer may also be a message in RFC-822 format, which consists of the message header and possible attachments. Parsing useful data from this format is also one of the tasks. The entire application is built on three levels, which makes it easier to maintain, upgrade, usage of other database types or even move application to the web.

The first half of the core diploma thesis task is focused on the development of library for communication between the client and the mail server using POP3 protocol. The library is developed in the .NET framework using C # programming language. It contains a method for connecting and loging client to the server, which can also be performed using one of two constructors. Essential is also a method that returns all the new e-mail messages from the server, and part of this method used to parse data from message format RFC-822 and fill in the `MailDto` object.

The second part describes the development of application using a three tier programming approach in the .NET framework. Data model diagram of the database is presented in SQL Server 2008 Express, the transfer of data across all three levels of applications and user interface is containing the system for adding of rules for sending e-mail. Also presented in detail ,the method for the generation of the search phrase in the database. This part has also description of a special class for communication with the database.

Finally, an example of practical use application and comparismet with application, which has simmilar functionality, is presented in conclusion.

1.2 Keywords

POP3 protocol, e- mail, automatic e-mail distribution application, .NET framework

1 UVOD

Elektronske komunikacije in v okviru teh elektronska pošta predstavljajo eno najpogostejših uporab spleta. To je glavna vrste komunikacije med podjetji in ustanovami, pa tudi med posamezniki. Začetki uporabe elektronske pošte segajo v začetek šestdesetih let prejšnjega stoletja, najprej v vojski in izobraževalnih ustanovah. Med poslovnimi uporabniki se je razširila kasneje, predvsem v devetdesetih letih prejšnjega stoletja. V vsakodnevnem življenju je elektronska pošta zelo pogosto prvi stik z internetom za večino uporabnikov, raziskave pa kažejo, da jo uporablja okrog tri četrtine rednih uporabnikov interneta.

Pri masovni komunikaciji z uporabo elektronske pošte lahko pride do težav, ki se kažejo kot težavno iskanje po pošti, nepreglednost ter otežen pregled elektronske pošte. Novejši odjemalci elektronske pošte sicer podpirajo več načinov urejanja, a so še vedno omejeni z vnaprej določenimi načini uporabe. To velja tudi za najbolj pogost odjemalec Microsoft Outlook, ki je del pisarniškega paketa Microsoft Office [6], oziroma odjemalec elektronske pošte Outlook Express v starejših operacijskih sistemih Windows ter odjemalec Windows Mail v operacijskem sistemu Windows Vista ter Windows 7 [7].

Za razvrščanje večjih količin elektronske pošte bi v odjemalcu Outlook in podobnih (Mozila Thunderbird, itd...) lahko uporabili sistem map (ang. folder), kjer bi si lahko z uporabo drevesne strukture uredili nekakšen sistem, potem pa v odjemalcu tako nastavili pravila, da elektronsko sporočilo, ki ustreza določenim pravilom, premakne v eno ali več ustreznih map. Vendar pa pri veliki količini elektronske pošte, ki jo prejme večje podjetje ali korporacija, pride do težav s tem, da en sam človek ne zmore uspešno odgovoriti na vsa elektronska sporočila. Še večja težava je na primer podjetje, ki zagotavlja uporabnikom svojih storitev podporo in informacije po elektronski pošti, na primer večja spletna trgovina, kjer en sam uslužbenec ne pokriva tako širokega nabora znanj, da bi lahko zagotavljal ustrezen hiter odziv.

Zato je smiselno razviti namensko aplikacijo, ki dela točno to, kar bi sicer moral delati uslužbenec, s tem da poleg tega obstaja še celotna baza elektronskih sporočil na enem mestu. Cilj je razviti knjižnico, ki zna komunicirati s poštnim strežnikom in kar je bistveno, sprejeti in posredovati elektronska sporočila. Za to je potrebno poznavanje poštnega protokola, preko katerega komunicirata odjemalec in poštni strežnik. Drugi cilj je izdelati aplikacijo, ki uporablja posebej razvito knjižnico za pridobivanje elektronske pošte iz strežnika, ter vsebuje funkcionalnosti za določanje pravil pri posredovanju elektronskih sporočil na določene naslove.

Nazadnje je potrebno razvito aplikacijo primerjati z podobnimi rešitvami, če obstajajo v okviru tipičnih poštnih odjemalcev ter predstaviti ugotovitve.

2 OPISI UPORABLJENIH TEHNOLOGIJ

2.1 Splošno

Aplikacijo smo zasnovali in načrtovali za izvedbo v Microsoftovem ogrodju .NET [1], v razvojnem okolju Visual Studio 2008 in z uporabo programskega jezika C# [4]. Pomembnejše kot razumevanje samega ogrodja .NET je predvsem poznavanje protokola POP3 ter njegovega delovanja in ukazov. Pri izbiri protokola za komunikacijo s poštnim strežnikom smo se zaradi večje razširjenosti in nasploh večje uporabe omejili na protokol POP3 [2], možnost bi pa bila uporaba protokola IMAP [8].

Prav tako se zdi pomemben moderen pristop h sami izvedbi načrtovane aplikacije, ki je izvedena trinivojsko, kar je tudi opisano v poglavju 2.6.

2.2 POP3 protokol

POP3 (ang. Post Office Protocol Version 3) je spletni protokol na aplikacijskem sloju, ki je namenjen prejemanju elektronske pošte iz strežnika. Na strežnik se poveže preko TCP/IP povezave [9] na vratih 110, kjer strežnik čaka na morebitne zahteve odjemalca.

Strežnik začne POP3 storitev tako, da posluša morebitne zahteve odjemalca na TCP vratih 110. Ko hoče odjemalec uporabiti POP3 storitev, vzpostavi TCP povezavo s strežnikom. Ko je povezava vzpostavljena, strežnik pošlje pozdravno sporočilo. Odjemalec in strežnik nato izmenjata zahteve in odgovore na zahteve (slika 1), dokler ni povezava zaključena ali prekinjena.

```
S: <wait for connection on TCP port 110>
C: <open connection>
S:    +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>
C:    APOP mrose c4c9334bac560ecc979e58001b3e22fb
S:    +OK mrose's maildrop has 2 messages (320 octets)
C:    STAT
S:    +OK 2 320
C:    LIST
S:    +OK 2 messages (320 octets)
S:    1 120
S:    2 200
S:    .
C:    RETR 1
S:    +OK 120 octets
S:    <the POP3 server sends message 1>
S:    .
C:    DELE 1
S:    +OK message 1 deleted
C:    RETR 2
S:    +OK 200 octets
S:    <the POP3 server sends message 2>
C:    QUIT
S:    +OK dewey POP3 server signing off (maildrop empty)
C:  <close connection>
S:  <wait for next connection>
```

Slika 1: Primer tipične komunikacije odjemalec - strežnik preko protokola POP3

POP3 ukazi so sestavljeni iz ukazne besede, ki mora biti vedno pisana z velikimi črkami ter z enim ali več argumenti. Vsak ukaz v izvajanju je lahko prekinjen z ukazom `CRLF`. Ukaz in argument (argumenti) morata biti ločena s presledkom.

Strežnikovi odgovori se začnejo z statusnim indikatorjem in možno ključno besedo, ki ji sledijo dodatne informacije. Statusna indikatorja pošlje strežnik odjemalcu v taki obliki:

- `+OK`
- `-ERR`

Odgovori na nekatere ukaze so večvrstični, v tem primeru strežnik pošilja vrstico po vrstico, vsaka vrstica je pa ločena z znakom `CRLF`. Zadnja vrstica je zaključena z decimalno kodo (`046`, ".") in `CRLF` parom. Seja s strežnikom je sestavljena iz več različnih stanj, v katerih lahko odjemalec strežniku pošilja različne ukaze.

V stanju prijave (ang. AUTHORIZATION state) se mora odjemalec predstaviti strežniku z uporabniškim imenom in geslom. Seja med odjemalcem in POP3 strežnikom se začne, ko odjemalec pošlje zahtevo za povezavo na strežnik preko povezave TCP. Povezava je vzpostavljena z uporabo standardnega tristranskega rokovanja (ang. three way handshake) in povezava se vzpostavi. Ko je povezava vzpostavljena, strežnik pošlje odjemalcu sporočilo v obliki `+OK server ready`, s katerim sporoča, da je na voljo za odjemalčeve zahteve. Odjemalec se mora sedaj prijaviti z uporabniškim imenom in geslom.

Sledi prehod v stanje prenosa (ang. TRANSACTION state). V tem stanju odjemalec pošilja strežniku različne ukaze - za pregled sporočil, prejem sporočila iz strežnika na odjemalca in brisanje sporočil. To stanje se zaključi tako, da odjemalec pošlje strežniku ukaz `QUIT`. Vrste ukazov so navedene v tabeli v poglavju 2.3.

Pridobivanje sporočil od POP3 strežnika običajno traja nekaj sekund, lahko pa tudi nekaj minut, odvisno od velikosti in števila sporočil v poštnem predalu. Trajanje povezave sicer ni omejeno in traja, dokler odjemalec pošilja ukaze strežniku. POP3 strežnik vsebuje nastavitev, koliko časa je lahko povezava neaktivna, preden jo sam prekine, kar se lahko tudi poljubno nastavlja, pri čemer vrednost ne sme biti manjša od desetih minut. Če je povezava neaktivna preko tega časa, strežnik razume to kot problem na strani odjemalca in s tem prekine povezavo, pri čemer pa ne izbriše sporočil, ki jih odjemalec ni uspešno sprejel.

Ukaz `QUIT` v stanju prenosa pomeni prehod v stanje posodobitve (ang. UPDATE state). Ta ukaz nima parametrov, s čimer sporoča strežniku, da želi zaključiti sejo. Ko strežnik sprejme ukaz `QUIT`, najprej pobriše vsa sporočila, ki so bila za to označena z uporabo ukaza `DELE` v stanju prenosa. Protokol sicer podpira brisanje sporočil v dveh fazah: najprej z ukazom `DELE`, nato pa s prehodom v stanje posodobitve, ko se sporočila dejansko pobrišejo. Z odlašanjem dejanskega izbrisa do prehoda v stanje prenosa lahko strežnik zagotovi obdelavo vseh ukazov, ki jih je prejel pred prehodom v stanje prenosa. Prav tako omogoča razveljavitev brisanja sporočil z uporabo ukaza `RSET`, ki mora biti podan v stanju prenosa. Brisanje se ne izvede tudi v primeru, da je povezava prekinjena pred prehodom v stanje posodobitve.

Ko so bila sporočila pobrisana, strežnik vrne odjemalcu potrditev `+OK`, če je prišlo vmes do napake pa `-ERR`. Ob predpostavki, da ni bilo nobenih težav, pomeni odgovor `+OK` tudi zaključek seje med strežnikom in odjemalcem, nakar se TCP povezava prekine.

2.3 POP3 ukazi

POP3 ukazi so naslednji:

Ukaz	Argumenti	Razlaga
USER	[uporabniško_ime]	Poslan je lahko samo v stanju prijave, ko odjemalec od strežnika dobi pozdravno sporočilo, ali pa po neuspešni prijavi. Če uporabnik s tem imenom obstaja, strežnik pošlje odgovor +OK
PASS	[geslo]	Poslan samo v stanju prijave, potem ko je strežnik poslal `+OK` na prejeto uporabniško ime. Možni odgovori strežnika so: • strežnik je pripravljen • napačno geslo • nekdo drug dostopa do tega računa, poizkusite kasneje
STAT	/	Poslan samo v stanju prenosa. Vrača odgovor tipa: `+OK nn mm` nn -> število sporočil mm-> skupna velikost sporočil
LIST	/	Prav tako kot STAT, je poslan samo v stanju prenosa. Vrne večvrstični odgovor, odvisno od števila sporočil na strežnik: [zaporedna številka sporočila] [velikost v bytih]
RETR	[številka_sporočila]	Poslan samo v stanju prenosa. Če strežnik odgovori z odgovorom `+OK`, sledi vsebina sporočila v več vrsticah. Možen je tudi negativen odgovor: `-ERR no such message`

DELE	[številka_sporočila]	Poslan samo v stanju prenosa. POP3 strežnik označi sporočilo kot izbrisano. Vsaka prihodnja referenca na sporočilo vrne napako. Sporočilo se zares izbriše, ko seja preide v stanje posodobitve.
		Možni odgovori:
		`+OK message deleted`
		`-ERR no such message`

NOOP	/	Strežnik samo vrne odgovor `+OK`.
RSET	/	Poslan samo v stanju prenosa. Če so bila kakšna sporočila izbrana za brisanje, jim to oznako odstrani.
UIDL	/	POP3 strežnik izbriše vsa sporočila, ki so za to označena, ter odgovori z statusom te operacije. V vsakem primeru pa strežnik sprosti povezavo in zapre TCP povezavo.

APOP	[naziv poštnega_računa, md5 hash]	Poslan samo v stanju prijave in je lahko uporabljen kot dodatna zaščita pred nezaželeno prijavo na strežnik.
		Možna odgovora strežnika:
		`+OK maildrop locked and ready`
		`-ERR permission denied`

TOP	[številka_sporočila]	Poslan v stanju prenosa. Če strežnik odgovori pozitivno, sledi večvrstični odgovor. Strežnik pošlje glavo sporočila.

QUIT	/	Poslan v stanju prenosa in sproži odjavo ter prekinitev TCP povezave med strežnikom in odjemalcem.

Vsa sporočila, ki se prenašajo med sejo z POP3 strežnikom, morajo ustrezati standardu RFC882 [5].

2.4 IMAP

IMAP je enako kot POP3 protokol na aplikacijskem sloju, ki je namenjen pridobivanju elektronskih sporočil iz strežnika [8]. Razvit je bil leta 1986 kot alternativa protokolu POP3. Bistvena razlika med protokoloma je ta, da IMAP pusti sporočilo na strežniku, dokler mu odjemalec eksplicitno ne ukaže brisanja. Pri POP3 je ravno obratno; če v odjemalcu ni nastavljeno, da se kopija sporočila hrani na strežniku, se po prejemu sporočila to sporočilo na strežniku pobriše. V praksi je uporaba IMAP protokola smiselna tedaj, ko hkrati dostopa več odjemalcev do poštnega strežnika. POP3 protokol podpirajo praktično vsi odjemalci, večina pa poleg tega tudi IMAP.

Slabosti protokola IMAP v primerjavi z POP3 so:

- zahtevnost protokola, ker omogoča več odjemalcem hkratno povezavo na isti poštni račun,
- večja poraba virov na strežniku, ker omogoča direktno iskanje tudi po sporočilih na strežniku, ki jih odjemalec še ni prenesel, ter
- zahteva po stalni povezavi strežnika in odjemalca, da lahko odjemalec stalno spremlja, ali so bila prejeta nova sporočila.

Kljub večji zahtevnosti ima IMAP nekaj pomembnih prednosti:

- stalna povezava med odjemalcem in strežnikom omogoča bolj ažurno prejemanje sporočil, saj je odjemalec obveščen takoj, ko se pojavi novo sporočilo na strežniku,
- hkratna povezava več odjemalcev na isti poštni račun,
- hranjenje trenutnega stanja sporočila, kjer lahko en odjemalec označi sporočilo kot prebrano, kar bodo nato videli tudi ostali odjemalci, ter
- iskanje po sporočilih direktno na strežniku, kjer si lahko odjemalec lahko nastavi pravila, kakšna sporočila bo prenesel.

2.5 Standard RFC822 - Internet Message Format

Elektronsko sporočilo (tipično elektronska pošta) je sestavljeno iz dveh delov:

- glava sporočila vsebuje vse ključne informacije glede prejemnika, pošiljatelja, naslova sporočila, datuma itd.
- telo sporočila, ki je dejanska vsebina sporočila.

Vsako sporočilo lahko vsebuje samo eno glavo sporočila. Vsaka vrstica v glavi se začne z imenom polja, ki mu sledi ločilni znak ":". Ločilu nato sledi vrednost polja, ki je lahko tudi večvrstična. Imena polj in vrednosti so omejene na 7-bitne ASCII znake [10]. Elektronsko sporočilo je bilo prvotno zasnovano za uporabo 7-bitne ASCII kode, standard MIME (ang. Multipurpose Internet Mail Extensions) pa je predstavil specifičen nabor znakov ter dva načina dekodiranja ne-ASCII znakov [11]. MIME služi kot standard za priponke in tekst sporočila, ki ne uporablja ASCII znakov. Čeprav protokola POP3 in SMTP (ang. Simple mail transfer protocol) [12] ne zahtevata sporočila v MIME obliki, se velika večina elektronske pošte pošilja v MIME formatu, zato morajo tudi odjemalci ta format podpirati.

V začetku so bila elektronska sporočila sestavljena samo iz tekstovne vsebine, sčasoma pa so se pojavile potrebe po pošiljanju različnih (tudi multimedijskih) vsebin, t.i. priponk v elektronskem sporočilu (slika 2). To je opredeljeno v standardih RFC 2045 [13] in RFC 2049 [14].

```
Return-Path: rok@ms3.si
Received: from RokSkupekPC ([86.61.66.63])
        by ms3web.ms3.si
        ; Sat, 6 Feb 2010 11:57:41 +0100
From: "Rok Skupek" <rok@ms3.si>
To: <pop3test@ms3.si>
Subject: Test subject
Date: Sat, 6 Feb 2010 11:57:52 +0100
Message-ID: <001d01caa71b$3b493710$b1dba530$@si>
MIME-Version: 1.0
Content-Type: multipart/alternative;
        boundary="----=_NextPart_000_001E_01CAA723.9D0D9F10"
X-Mailer: Microsoft Office Outlook 12.0
Content-Language: sl
Thread-Index: AcqnGzrN1x6YCvhIS42Yxvbk23srSw==

This is a multi-part message in MIME format.

------=_NextPart_000_001E_01CAA723.9D0D9F10
Content-Type: text/plain;
        charset="iso-8859-2"
Content-Transfer-Encoding: quoted-printable

Testno sporocilo.
```

Slika 2: Primer elektronskega sporočila v formatu RFC-882

Glava sporočila mora vsebovati naslednja polja:

- `From` - elektronski naslov in pogosto tudi naziv pošiljatelja sporočila.
- `To` - elektronski naslov prvega prejemnika sporočila, ostali naslovi so ponavadi našteti v poljih `Cc` oziroma `Bcc`.
- `Subject` - naziv sporočila, lahko tudi kratek povzetek sporočila
- `Date` - lokalni datum in čas, ko je bilo sporočilo ustvarjeno. Vrednost se doda avtomatsko pri pošiljanju sporočila, prikaz pri prejemniku pa je odvisen o njegovega lokalnega časa.

- `Message-ID` - enolična številka sporočila, ki preprečuje, da bi se sporočilo prevečkrat dostavilo. Uporablja se tudi za referenco v polju `In-Reply-To`.

Pogosto uporabljena polja, ki pa niso obvezna:

- `Bcc` (ang. Blind carbon copy) - naslov, na katerega se pošlje sporočilo, ki ostane ostalim prejemnikom neviden. Pogosto je imenovan tudi skrita kopija sporočila
- `Cc` (ang. Carbon copy) - sekundarni prejemnik sporočila
- `Content-Type` - vsebuje informacije, kako naj bo sporočilo prikazano (primer: text/plain, text/html...)
- `In-Reply-To` - vsebuje vrednost polja `Message-ID`, ki se uporablja za referenco, da se loči, katero sporočilo je odgovor na neko prvotno sporočilo
- `Precedence` - pogosto vsebuje vrednosti "junk", "bulk" ali "list", uporablja se lahko za avtomatski odgovor na sporočilo
- `Received` - vsebuje informacije o poštnih strežnikih, skozi katere je potovalo sporočilo. Strežniki so razvrščeni v obratnem vrstnem redu, tisti, ki je zadnji preposlal sporočilo, je zapisan najprej
- `References` - vsebuje vrednost polja `Message-ID`, ki se navezuje na sporočilo, odgovor katerega je to sporočilo
- `Reply-To` - elektronski naslov, na katerega naj se pošlje odgovor
- `Sender` - elektronski naslov dejanskega pošiljatelja, če je bilo sporočilo poslano v imenu nekoga drugega (v imenu avtorja, ki je naveden v polju `From`)

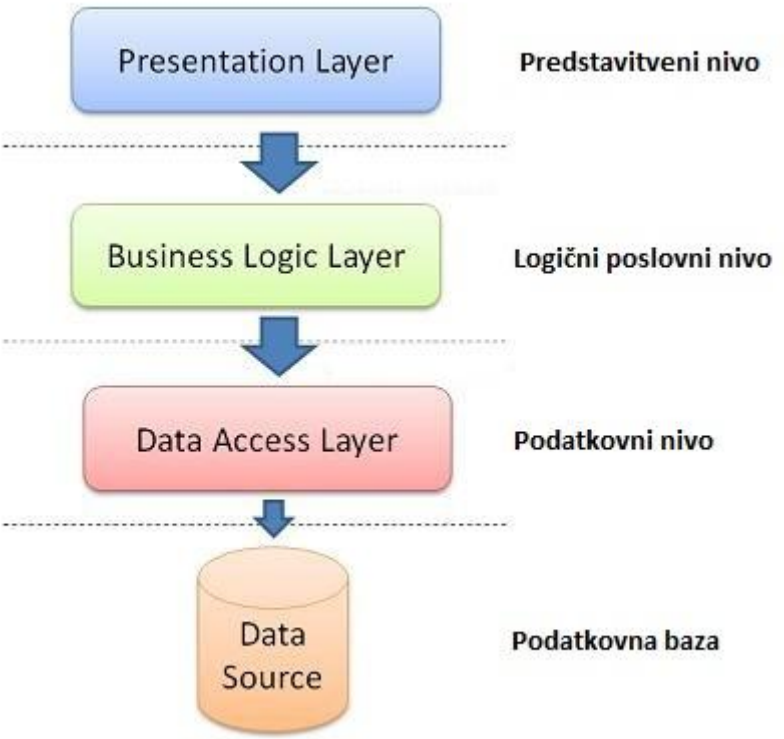
Telo sporočila je z uporabo sodobnih grafičnih e-poštnih odjemalcev lahko sestavljeno iz golega teksta (ang. plain text) ali iz HTML vsebin. V tem primeru jedro elektronskega sporočila vsebuje posebej samo besedilo, posebej pa stilsko oblikovanje za ta tekst, tako da se v odjemalcu elektronske pošte prikaže kot oblikovano besedilo (HTML). Prednosti HTML vsebin so v oblikovanju sporočil, dodajanju spletnih povezav, slik itd, slabost je vsekakor večje sporočilo, saj je potrebno poslati veliko več podatkov, kot v golem tekstovnem formatu [15]. Slabost je tudi možnost vdorov v zasebnost preko zlonamerne programske kode.

2.6 Trinivojska arhitektura

Tri nivojska arhitektura aplikacije običajno vsebuje (slika 3)[19]:

- Predstavitveni nivo (ang. Presentation Layer)
- Poslovni / logični nivo (ang. Business Logic Layer / Business Access Layer)
- Podatkovni nivo (ang. Data Access Layer)

Nivo je v ogrodju .NET največkrat definiran kot samostojni projekt v sklopu rešitve. Tako programiranje deluje časovno bolj potratno od drugih pristopov, dolgoročno gledano pa ima veliko prednosti. V primeru menjave podatkovne baze za neko aplikacijo točno vemo, kje se nahajajo klici baze: na podatkovnem nivoju. Zato ni potrebno spreminjati programske kode, ampak le prilagoditi podatkovni nivo, ki je običajno največkrat knjižnica funkcij (ang. DLL). Ko to zamenjamo, naša aplikacija enako dela s popolnoma drugo podatkovno bazo.



Slika 3: Primer zgradbe trinivojske aplikacije

V nekaterih pristopih se uporablja ločen razred za fizični dostop do baze. Metode, ki imajo vhodne parametre (recimo poizvedbe na bazi) vračajo neko podatkovno strukturo (odvisno od programskega jezika).

2.6.1 Predstavitveni nivo

Na tem nivoju se izvaja interakcija med uporabnikom in aplikacijo. Je najpomembnejši nivo zaradi tega, ker ga uporabniki vidijo kot en sam nivo, čeprav sta logični in podatkovni nivo dobro zasnovana, in to daje zelo slab vtis o aplikaciji. Konkretno v spletnem jeziku ASP.NET [16] je primer datoteka s končnico `.aspx`, pri .NET aplikaciji za Windows okolje, napisani v ogrodju .NET, pa je to na primer obrazec. Prikazani so podatki, ki jih uporabnik lahko vidi, vnaša ali spreminja.

2.6.2 Logični poslovni nivo

Logični nivo vsebuje poslovno logiko aplikacije, torej vse preračune in obdelave podatkov. Uporablja se kot most med podatkovnim in predstavitvenim nivojem. Vsi podatki se tako prenašajo iz podatkovnega nivoja preko logičnega do predstavitvenega nivoja, pri čemer poskrbimo za varen dostop ter pravilen prikaz podatkov. Ko se pojavi zahtevek po pridobivanju podatkov iz predstavitvenega nivoja, se pokliče ustrezno funkcijo v podatkovnem nivoju, ki izvede poizvedbo do baze. Po tem vrne podatke v neki določeni strukturi, kjer jih mora logični nivo obdelati in pretvoriti v lastne objekte, namesto da bi poslal naprej objekt, ki ga vrne podatkovni nivo (primer v ogrodju .NET je lahko `DataTable` ali `DataSet`). Prav tako se tukaj izvajajo vsi preračuni in ostala obdelava podatkov.

2.6.3 Podatkovni nivo

Na tem nivoju se dejansko izvajajo klici na podatkovno bazo, lahko tudi preko posebnega razreda, ki se lahko sporazumeva tudi z več različnimi podatkovnimi bazami. V tem primeru pravimo, da je dodan še en podnivo. V praksi to pomeni, da v primeru prehoda iz MSSQL na MySQL menjamo samo razred, ki se sporazumeva s podatkovno bazo, ali pa ga že na začetku tako prilagodimo, da tudi menjava kode ni potrebna. Celoten podatkovni nivo mora biti tako zasnovan, da v svojem bistvu niti ne ve, s katero podatkovno bazo se sporazumeva.

Glavne prednosti uporabe trinivojske arhitekture [18]:

- **Ločevanje** - logika je ločena od dostopa do podatkov, kar omogoča lažje vzdrževanje
- **Neodvisnost** - v kolikor se en nivo spremeni do neke mere (primer je menjava podatkovne baze), bodo ostali nivoji delovali nemoteno
- **Ponovna uporaba** - ker so plasti ločene, se jih lahko uporablja v ločenih aplikacijah (primer je uporaba logičnega nivoja, ki je bil razvit za Windows aplikacijo, v spletni aplikaciji)

3 KNJIŽNICA ZA PRIDOBIVANJE ELEKTRONSKIH SPOROČIL IZ STREŽNIKA

3.1 Opis problema in zahtev

Predstavljajmo si moderno podjetje, ki z uporabniki svojih storitev/izdelkov veliko komunicira po spletu. Tipičen primer je spletna trgovina, ki jo uporabljajo računalniško in internetno pismeni uporabniki. Ti uporabniki se v primeru uporabe tehnične pomoči ali drugih stikov s ponudnikom storitev/izdelkov, poslužujejo elektronskih komunikacij - bodisi neposredno preko elektronske pošte, bodisi preko kontaktnih obrazcev na spletni strani.

Če hoče večje podjetje zagotavljati kvalitetno in hitro podporo uporabnikom po elektronski pošti, potrebuje več ustrezno izobraženih ljudi, od katerih vsak dobro pozna svoje področje. Prav tako pa potrebuje nek avtomatiziran sistem dodeljevanja nalog oziroma pošiljanja zahtevkov določenemu zaposlenemu. Ob velikem številu zahtevkov preko elektronske pošte, je za to potreben en zaposlen, ki vse to razvršča, uredi po prioritetah ter razpošlje na točno določeno mesto tehnične podpore. Ker pa to vzame veliko časa in terja velike stroške, je cilj naloge vsaj delno avtomatizirati ta proces.

Rešitev problema je izdelava aplikacije, ki ta proces avtomatizira. Aplikacija je sestavljena iz dveh glavnih delov:

- iz knjižnice (razred), ki zna komunicirati z POP3 poštnim strežnikom, ter
- uporabniškega vmesnika, kjer lahko uporabnik nastavlja pravila za posredovanje prejete e-pošte.

3.2 Komunikacija s strežnikom

Odjemalec mora biti sposoben strežniku pošiljati ukaze in od strežnika prejemati ustrezne odgovore. Zato mora biti vzpostavljena določena stalna povezava med odjemalcem in strežnikom. Pri medsebojni komunikaciji je odjemalec podrejen strežniku v tej meri, da mora pošiljati strežniku ukaze v vnaprej definirani obliki, strežnik pa odgovori v ustrezni obliki nazaj. Strežnik pošlje elektronsko sporočilo odjemalcu v posebni obliki, ki je določena v standardu RFC-882, zato je potrebno pridobiti iz tega podatke, ki nas dejansko zanimajo ter izvesti prikaz v uporabniku prijazni obliki.

3.2.1 Uporabljena rešitev

Uporabljen je razred `System.Net.Sockets.TcpClient`, ki je že del ogrodja .NET [17]. Razred `TcpClient` zagotavlja enostaven način za povezovanje, pošiljanje in prejemanje toka podatkov preko omrežja v sinhronem načinu. Povezava na strežnik je možna na dva načina:

- Inicializacija razreda `TcpClient` in klic ene od treh `Connect` metod

- Inicializacija razreda `TcpClient` s podanimi parametri (naslov strežnika in številka vrat), kjer nato konstruktor razreda poizkusi vzpostaviti povezavo

V knjižnici je uporabljena prva možnost. Za pridobivanje toka podatkov je potrebno poklicati metodo `GetNetworkStream`, za sprostitev vseh virov pa metodo `Close`.

3.2.2 Razvoj razreda za komunikacijo z POP3 strežnikom

Razvoj knjižnice (ang. DLL) v Microsoftovem razvojnem okolju Visual Studio 2008 začnemo z novim Windows projektom knjižnica (ang. class library). V projekt lahko dodajamo razrede, ki so v tem razvojnem okolju datoteke s končnico `.cs`. V sam projekt sta bila dodana dva razreda: `Pop3.cs` in `MimeParser.cs`. Razred `Pop3` vsebuje vse potrebno za samo komunikacijo s POP3 strežnikom. Gre se za pošiljanje ukazov in sprejema podatkov, ki jih strežnik posreduje. Razred `MimeParser` vsebuje dve metodi za pridobivanje podatkov iz izvorne oblike elektronskega sporočila.

Razred `Pop3` vsebuje tri naštevne tipe (ang. enumerator):

```
public enum ConnectionType { PlainText, SSL }
public enum ConnectionState { Authenticating, Connected,
NotConnected, Error }
public enum CommandExecutionStatus { OK, Error }
```

Koda 1: Naštevni tipi razreda `Pop3`

Naštevni tip določa vse možne vrednosti, ki se jih lahko določi neki spremenljivki tistega tipa. Tip `ConnectionType` določa vrsto povezave na POP3 strežnik, ki je lahko navadna ali kriptirana povezava SSL. Tip `ConnectionState` določa stanje povezave med odjemalcem in poštnim strežnikom, tip `CommandExecutionStatus` pa status zadnjega prejetega sporočila, ki ga je strežnik posredoval (koda 1).

Deklariramo tudi spremenljivke, ki so potrebne za delovanje razreda. Vse so zasebne (ang. private), vidne samo znotraj tega razreda, medtem ko so metode javne (ang. public). Razred vsebuje tudi metodo `Dispose`, ki je namenjena uničenju objekta oziroma sprostitvi pomnilnika, ki ga je objekt porabljal. Vsebuje dva konstruktorja:

```
public Pop3()
        {
            _connectionState = ConnectionState.NotConnected;

            _username =
ConfigurationSettings.AppSettings["username"];
            _password =
ConfigurationSettings.AppSettings["password"];
            _server = ConfigurationSettings.AppSettings["server"];
            _serverPort =
Convert.ToInt32(ConfigurationSettings.AppSettings["port"]);
```

```
            _mailClient = new TcpClient();

            OpenConnection();
        }
```

Koda 2: Konstruktor, ki ne sprejme nobenega parametra

Prvi konstruktor (koda 2) ne potrebuje nobenega vhodnega parametra, ampak vse potrebne nastavitve pridobi iz konfiguracijske datoteke `App.config`, kjer lahko te vrednosti nastavimo. Najprej se nastavi vrednosti spremenljivke tipa `ConnectionState` na vrednost `NotConnected`, nato nastavimo vse potrebne lastnosti in na koncu pokličemo še metodo `OpenConnection`, ki dejansko odpre povezavo s strežnikom.

Drugi konstruktor (koda 3) sprejme štiri vhodne spremenljivke, in sicer uporabniško ime (`username`), geslo (`password`), naslov strežnika (`server`) in vrata na strežniku (`serverPort`). Te vrednosti se potem priredijo lokalnim spremenljivkam v razredu, kar je edina razlika s prvim konstruktorjem.

```
public Pop3(string username, string password, string server, int
serverPort)
        {
            _connectionState = ConnectionState.NotConnected;

            _username = username;
            _password = password;
            _server = server;
            _serverPort = serverPort;
            _mailClient = new TcpClient();

            OpenConnection();
        }
```

Koda 3: Konstruktor, ki sprejme 4 parametre

Metoda `GetInboxStatus` (koda 4) se uporablja na začetku povezave s strežnikom in vrača predefinirano strukturo tipa `MailStatus`. Struktura `MailStatus` (koda 6) vsebuje dve spremenljivki in konstruktor, ki prejme dva parametra in spremenljivkama nastavi vrednosti. Spremenljvka `InboxMails` tipa `integer` vsebuje podatek o številu sporočil na strežniku, spremenljivka `Size` tipa `uint` pa vsebuje velikost sporočil. Metoda najprej preveri, če je povezava s strežnikom vzpostavljena. Če slučajno ni, pokliče metodo `OpenConnection`, ki povezavo vzpostavi. Potem se izvede klic metode `SendCommandToServer` (koda 5), ki sprejme dva parametra. Metoda preko objekta tipa `StreamWriter` pošlje ukaz strežniku, ki ukaz obdela in vrne nek odgovor. Odgovor preberemo preko objekta tipa `StreamReader` in preverimo, če je odgovor tipa `+OK`, je bil ukaz uspešno izveden, v primeru `-ERR` je prišlo do napake. Metoda vrača odgovor tipa `CommandExecutionStatus`, v primeru uspešne izvedbe pa vrne tudi vrstico z odgovorom tipa `string`, ki nam jo je vrnil POP3 strežnik.

```
public MailStatus GetInboxStatus()
        {
            if (_connectionState != ConnectionState.Connected)
                OpenConnection();
            string output = String.Empty;
            if (SendCommandToServer("STAT", ref output) ==
CommandExecutionStatus.Error)
                throw new ApplicationException("Error getting inbox
status");
            string[] outputSplit = output.Split(' ');
            if (outputSplit.Length < 3)
                throw new ApplicationException("Response in wrongly
formed");
            return new MailStatus(Convert.ToInt32(outputSplit[1]),
Convert.ToUInt32(outputSplit[2]));
        }
```

Koda 4: Metoda `GetInboxStatus`

```
private CommandExecutionStatus SendCommandToServer(string command,
ref string response)
        {
            CommandExecutionStatus execStatus;
            string line;
            if (_mailStreamWritter == null)
                throw new SocketException();
            _mailStreamWritter.WriteLine(command);
            _mailStreamWritter.Flush();
            if ((line =
_mailStreamReader.ReadLine()).StartsWith("+OK"))
                execStatus = CommandExecutionStatus.OK;
            else
                execStatus = CommandExecutionStatus.Error;

            if (execStatus == CommandExecutionStatus.OK)
                response = line;
            return execStatus;
        }
```

Koda 5: Metoda `SendCommandToServer`

```
public struct MailStatus
    {
        public int InboxMails;
        public uint Size;

        public MailStatus(int InboxMails, uint Size)
        {
            this.InboxMails = InboxMails;
            this.Size = Size;
        }
    }
```

Koda 6: Definirana struktura `MailStatus`

3.3 Pridobivanje podatkov iz elektronskega sporočila v izvorni obliki

V poglavju 2.5 je predstavljen primer zgradbe elektronskega sporočila v izvorni obliki. Uporabnik, ki bo to sporočilo prejel, mora imeti prikazane informacije iz sporočila v jasni in enostavni obliki. Zato informacije iz te oblike tudi izluščimo. To je razdeljeno na dva dela:

- pridobivanje podatkov o glavi sporočila ter
- pridobivanje priponk.

V sporočilu je vse razen glave sporočila, priponka, četudi je to čisto običajen tekst. Pri pridobivanju podatkov iz izvorne oblike sporočila se upoštevajo vse priponke enako, pred shranjevanjem podatkov o sporočilu v podatkovno bazo pa se poišče priponko tipa `plain/text`, ki predstavlja vsebino (telo) sporočila.

```
private static void ParseHeader(List<string> lines, ref MailDto m)
        {
            foreach (string line in lines)
            {
                if (line.StartsWith("-----=_"))
                    return;
                else if (line.StartsWith("From:"))
                    m.Sender = line.Substring(line.IndexOf('<'),
(line.Length - line.IndexOf('<')));

                else if (line.StartsWith("To:"))
                    m.Receiver = line.Substring(line.IndexOf('<'),
(line.Length - line.IndexOf('<')));

                else if (line.StartsWith("Subject:"))
                    m.Subject = line.Substring(7, line.Length - 6);

                else if (line.StartsWith("Date:"))
                    m.Date = line.Substring(5, line.Length - 5);

                else if (line.StartsWith("Message-ID:"))
                    m.Id = line.Substring(line.IndexOf('<'),
(line.Length - line.IndexOf('<')));

            }
        }
```

Koda 7: Metoda `ParseHeader`

Metoda `ParseHeader` (koda 7) je namenjena pridobivanju podatkov iz glave sporočila. Sprejme dva parametra, in sicer `List<string> lines`, ki vsebuje seznam vseh vrstic enega sporočila v izvorni obliki, ter objekt tipa `MailDto`. To je razred, ki je uporabljen tudi v aplikaciji za avtomatsko obdelavo sporočil. V metodi se z zanko sprehodimo skozi vse vrstice sporočila, dokler se vrstica ne začne z naborom znakov "-----=_", kar označuje priponko. Takrat izvajanje metode zaključimo. Ker v metodi pošljemo objekt `MailDto` po referenci, lahko tam sproti nastavimo ključne vrednosti, metoda pa drugače ne vrača vrednosti.

Priponke v elektronskih sporočilih so v dveh oblikah: običajen tekst ali binaren zapis, če gre za sliko, PDF dokument itd.

```
private static void ParseAttachment(List<string> attachment, ref
MailDto m)
        {
            AttachmentDto a = new AttachmentDto();
            a.Type =
AttachmentTypeDto.GetAttachmentType(attachment[1].Substring(12,
(attachment[1].Length - 12)));

            for (int i = 4; i < attachment.Count; i++)
                a.Data = a.Data + attachment[i];

            m.Attachments.Add(a);
        }
```

Koda 8: Pridobivanje podatkov o priponki

V metodi `ParseAttachment` (koda 8) se pridobijo vsi podatki o priponki določenega elektronskega sporočila. Kot vhodna parametra sprejme `List<string>` (seznam vseh vrstic ene priponke, v izvorni obliki) ter objekt tipa `MailDto` po referenci, ki mu po izvajanju for zanke doda priponko. Najprej se določi tip priponke, ki se prebere iz druge vrstice izvornega sporočila. S klicem metode `GetAttachmentType(string type)`, se pokliče v podatkovni nivo metodo z istim imenom, ta pa izvede klic na bazo, kjer vrne `id` ter naziv tipa priponke (če le-ta že obstaja), v nasprotnem primeru pa tip priponke vstavi v tabelo `s_attachment_type` ter vrne `id` zadnjega vnešenega zapisa s klicem SQL funkcije `SCOPE_IDENTITY()`.

Po uspešni izvedbi obeh metod imamo za sporočilo vse podatke v obliki, ki smo jo določili: objekt `MailDto` ter objekta `AttachmentDto` in `AttachmentTypeDto`.

4 PROGRAMSKI MODUL ZA AVTOMATSKO POSREDOVANJE ELEKTRONSKIH SPOROČIL

4.1 Opis zahtev

Pri razvoju aplikacije za posredovanje elektronskih sporočil je potrebno izpolniti tri cilje:

- izdelati enostaven in zmogljiv sistem za urejanje pravil za posredovanje sporočil,
- razviti logiko za posredovanje sporočil, ter
- izdelati uporaben uporabniški vmesnik.

Kot je nakazano, bo aplikacija razvita v ogrodju .NET in z uporabo trinivojske arhitekture. Uporabila se bo knjižnica, ki smo jo razvili za komunikacijo in prejemanje sporočil iz poštnega strežnika, z uporabo protokola POP3. Hranjenje podatkov bo v podatkovni bazi Microsoft SQL Server Express 2008, čeprav bi bila možnost uporabiti kar datotečni sistem na trdem disku. Ta rešitev bi zahtevala večji vložek v razvoj.

4.2 Razvoj

4.2.1 Podatkovni model

Uporabljena podatkovna baza je Microsoft SQL Server Express 2008, ki je brezplačna. Ta verzija je omejena z velikostjo shranjenih podatkov do 4 GB, kar je pa za tako aplikacijo dovolj. V kolikor bi se pokazala potreba po večjem prostoru za podatke, bi bila opcija selitev na Microsoftov SQL Server 2008, ki je plačljiv, ali pa z manjšo predelavo procedur na MySQL podatkovni strežnik.

Za poizvedbe na podatkovno bazo so uporabljene procedure, in sicer iz dveh vidikov (v primerjavi z direktnim klicem poizvedbe):

- **Hitrost izvajanja.** Procedura je prevedena SQL koda, ki sprejme določene parametre, in lahko vrača zapise. Če bi hoteli isti SQL ukaz izvesti z direktno poizvedbo na bazo iz programske kode, se mora ukaz najprej prevesti, ter se šele nato izvede. Prav tako pri tem principu ne pride do sintaktičnih napak. Ko proceduro na strežniku ustvarimo, se prevede le, če ne vsebuje sintaktičnih napak.

- **Varnost.** Če uporabljamo za poizvedbe do baze direktne tekstovne SQL ukaze, obstaja možnost vdora v podatkovno bazo (ang. SQL injection). Do takega vdora pride, ko napadalec vnese namesto pričakovanega vnosa SQL stavek, recimo `DROP table table`. V takem primeru se celotna tabela izbriše in podatki so izgubljeni. Ob pravilnem programiranju procedure do takega problema ne more priti. Je pa potrebno tudi upoštevati pravilne varnostne nastavitve tako, da uporabniku SQL strežnika ne damo pravic za direktno izvajanje SQL ukazov, ampak samo za izvajanje procedur.

V nadaljevanju sledi opis uporabljenih tabel (slika 4).

[QUEUE] - Tabela, ki hrani vse zahtevke po obdelavi sporočil. Z relacijami je povezana s tabelami [MAIL], [RULE] in [STATUS].

[MAIL] - Tabela, ki hrani elektronska sporočila z vsemi pripadajočimi atributi. Relacijsko je povezana s tabelo [ATTACHMENT].

[ATTACHMENT] - Hrani vse podatke o priponkah določenega sporočila, ki je preko polja [MAIL_ID] vezana na tabelo [MAIL] in tudi relacijsko povezana s tabelo šifrantov tipov priponk [S_ATTACHMENT_TYPE].

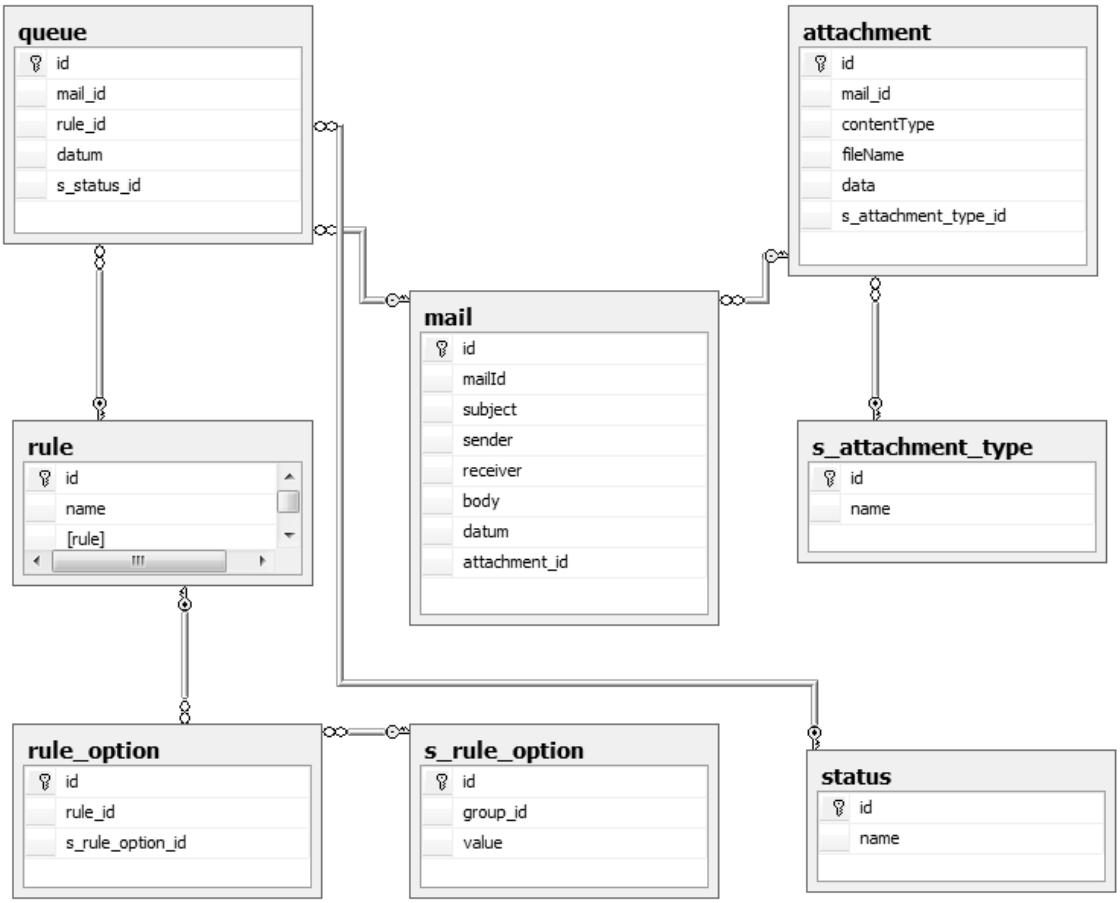
[S_ATTACHMENT_TYPE] - Tabela, ki vsebuje šifrant vseh tipov priponk.

[RULE] - Tabela, ki vsebuje podatke o pravilih, ki se uporabijo pri avtomatski obdelavi sporočil.

[RULE_OPTION] - Povezovalna tabela, ki hrani vse izbrane šifrante za neko pravilo.

[S_RULE_OPTION] - Tabela, ki hrani šifrant različnih možnosti pravila.

[STATUS] - Tabela, ki hrani šifrant vseh različnih statusov.



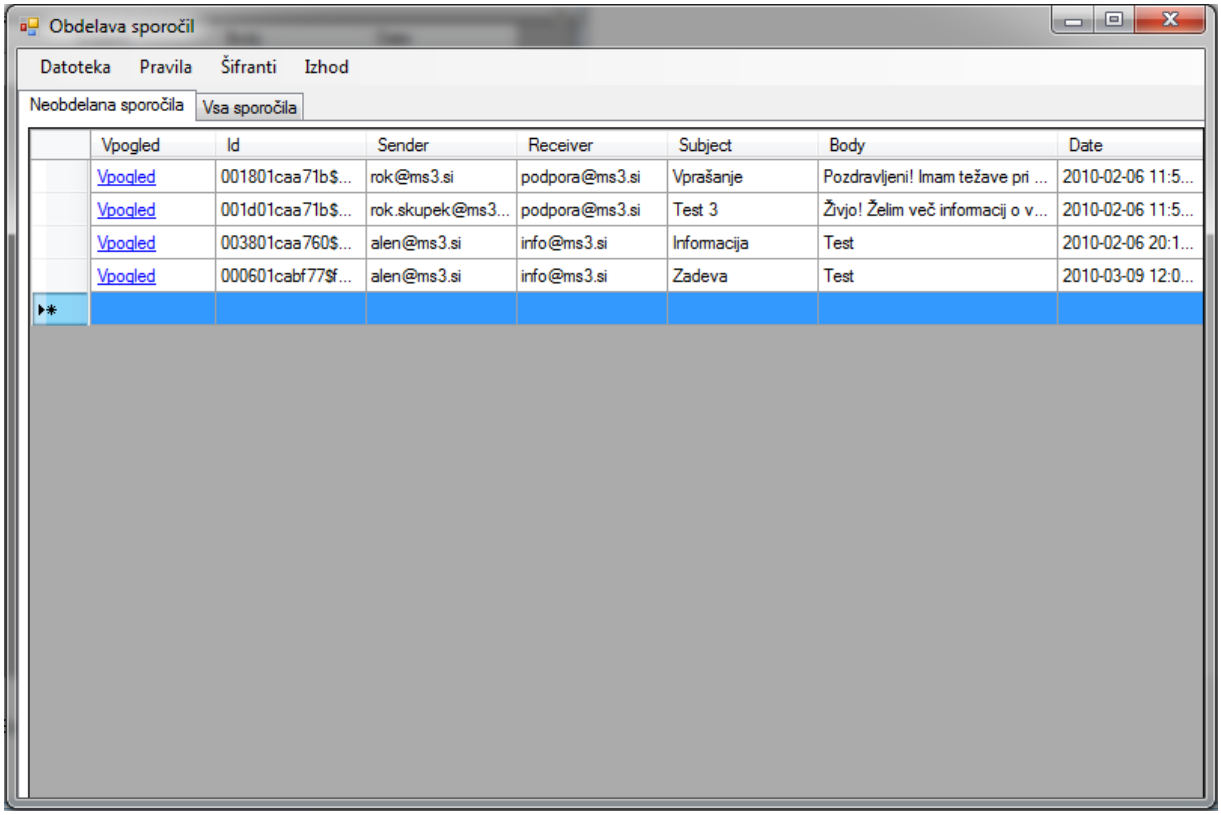
Slika 4: Diagram podatkovne baze

4.2.2 Razvoj grafičnega vmesnika

Pri razvoju uporabniškega vmesnika smo se osredotočili na čim bolj enostavno uporabo aplikacije, hiter dostop vseh pomembnih funkcij ter preglednost. V osnovnem obrazcu (slika 5) sta dva glavna prikaza podatkov, prikaz sporočil, ki se morajo še razposlati, ter arhiv vseh sporočil, kjer je možno tudi iskanje. Prikazani so bistveni podatki, klik na povezavo "vpogled" pa odpre vpogled v posamezno sporočilo. Ostali obrazci so dostopni iz menuja na vrhu glavnega obrazca.

Ker je grafični vmesnik vmesni člen med uporabnikom in aplikacijo, je treba pri oblikovanju vmesnika upoštevati nekaj osnovnih pravil:

- enostavnost, da uporabniki potrebujejo čim manj usposabljanja za delo z aplikacijo ter podpore v prihodnosti,
- grafični vmesnik mora upoštevati osnovni tok aplikacije, najvažnejše stvari se morajo prikazati najprej,
- ponovljivost čez celotni tok aplikacije, iste stvari so vedno na istih mestih. Gumb za shranjevanje podatkov v trenutnem obrazcu je vedno v spodnjem desnem kotu.

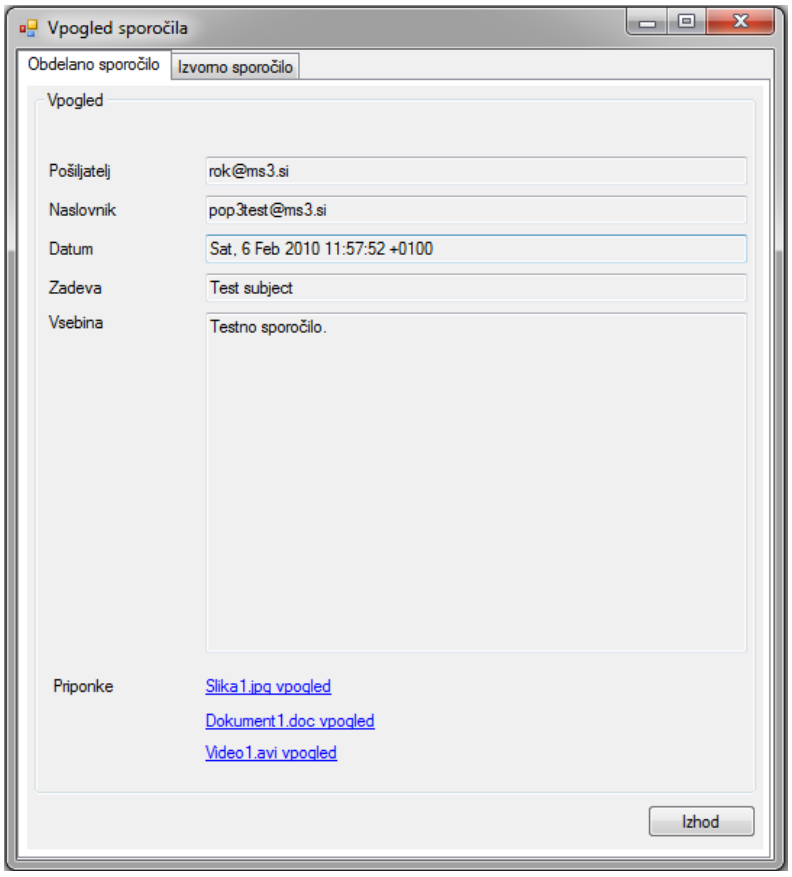


Slika 5: Glavni obrazec aplikacije s prikazom neobdelanih sporočil

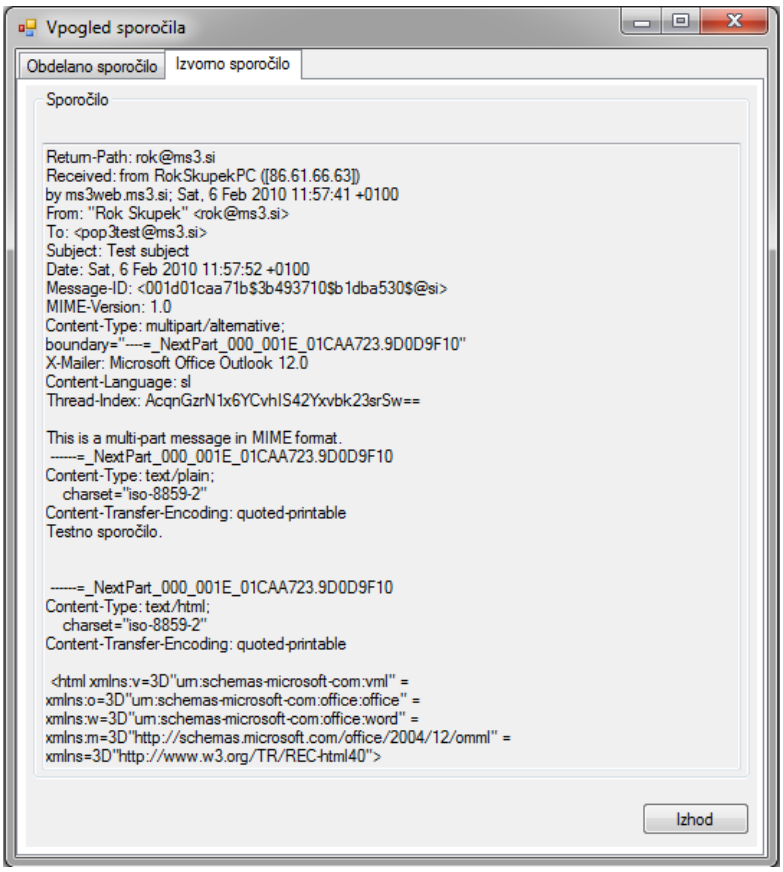
Pri oblikovanju obrazcev sicer ni potrebnega toliko znanja, kot je to potrebno pri vzpostavitvi podobnih kontrolnikov na spletni strani, je pa vseeno potrebno posvetiti obrazcu določeno pozornost. Kontrolnike se na obrazec lahko doda zelo hitro, je pa potrebno paziti, da:

- so kontrolniki razvrščeni smiselno, da logično sledijo poteku dela,
- se podobni kontrolniki z podobnimi funkcionalnostmi pojavljajo na istih mestih,
- se v primeru oženja oziroma širjenja obrazca kontrolniki nadzorovano širijo oziroma ožijo skupaj z obrazcem, ter
- so kontrolniki ustrezno grupirani: tisti z podobnimi funkcionalnostmi se nahajajo v isti skupini kontrolnikov, skupine pa so lahko ločene naprimer z barvami ali tabelami.

Kot je razvidno iz obrazcev za vpogled v sporočilo na dva načina (slika 6, slika 7), so ta pravila upoštevana. Glede na to, da obrazca ne vsebujeta velikega števila kontrolnikov, so vsi grupirani v en kontrolnik tipa `GroupBox`, kateremu se lahko tudi smiselno nastavi naziv. Gumb za zapiranje obrazca je vedno pozicioniran na istem mestu. Namesto, da bi bil obrazec večji ter vseboval prikaz obeh tipov vpogleda v sporočilo, smo smiselno ločili oba prikaza s pomočjo kontrolnika `TabControl`, ki je uporaben predvsem za prikazovanje različnih podatkov v enem samem obrazcu. Prikaz je ločen skoraj tako, kot bi imeli dva obrazca, prednost je pa v tem, da lahko ima obrazec skupno glavo in nogo, v tem primeru je skupen gumb za izhod.



Slika 6: Obrazec za pregled sporočila v uporabniku prijazni obliki



Slika 7: Obrazec za pregled sporočila v izvorni obliki

4.2.3 Arhitektura aplikacije

4.2.3.1 Razred za komunikacijo z podatkovno bazo

Kot je nakazano v poglavju 2.5 je aplikacija zasnovana trinivojsko. Uporabniški nivo vsebuje obrazce za vnos in prikaz podatkov ter interakcijo z njimi. V ozadju se ne izvaja nobena pomembnejša obdelava podatkov, ker je to vsebovano v drugem (logičnem) nivoju. Najnižji nivo vsebuje klice na podatkovno bazo, v našem primeru je na podatkovno bazo SQL Server 2008 Express. Komunikacija s podatkovno bazo poteka preko posebej razvitega razreda, ki je namenjen samo komunikaciji z bazo. Razred je bil dodan v svoj Visual Studio projekt, ki tvori knjižnico, hkrati pa vsebuje še XML datoteko, kjer se hranijo nastavitve za povezavo na bazo (ang. connection string). Prednost takega načina je predvsem to, da lahko tak razred sam skrbi za odpiranje in zapiranje povezav do baze, zato ne more priti do upočasnitve zaradi veliko odprtih povezav na bazo. Razred vsebuje vse potrebne metode za komunikacijo z bazo.

```
public Db()
        {

        }

public Db(string connString)
        {
            this._connString = connString;
            _sqlConn = new SqlConnection(connString);
```

```
            try
            {
                _sqlConn.Open();
            }
            catch (SqlException)
            {
                throw;
            }
        }
```

Koda 9: Konstruktorja za razred DB

Razred vsebuje dva konstruktorja (koda 9), prvi je brez parametrov, in služi enostavnejšemu upravljanju z povezavami do podatkovne baze, saj je treba posebej nastaviti podatke za povezavo. Pri drugem konstruktorju pa je podatek za povezavo argument, hkrati pa se v tem konstruktorju povezava do baze že odpre. Nadalje so v razredu tri metode za komunikacijo aplikacije s podatkovno bazo. V kolikor bi se pojavila potreba po bolj zahtevnih vpisih podatkov v bazo, bi bilo smotrno dodati še metodo, ki podpira transakcije. Druga možnost je uvedba transakcij na bazi sami z uporabo shranjenih procedur.

```
public DataTable GetDataTable(SqlCommand sqlCommand)
        {
            DataSet ds = new DataSet();
            this.HandleConnectionErrors();

            if (_sqlConn != null)
            {
                try
                {
                    sqlCommand.Connection = _sqlConn;
                    SqlDataAdapter sqlDataAdapter = new
SqlDataAdapter(sqlCommand);
                    sqlDataAdapter.Fill(ds);
                }
                catch (SqlException)
                {
                    throw;
                }
            }
            return ds.Tables[0];
        }
```

Koda 10: Metoda GetDataTable

Največkrat uporabljena metoda je `GetDataTable`(koda 10), ki sprejme parameter tipa `SqlCommand`. Ta sam po sebi že vsebuje že ime procedure ali neposredni klic do neke tabele s pripadajočimi parametri. Metoda najprej preveri, če je povezava na strežnik s podatkovno bazo vzpostavljena. Če ni, se ta najprej vzpostavi, sicer pa se preko objekta `SqlDataAdapter` izvede `SqlCommand`. `SqlDataAdapter` s klicem metode `Fill(DataSet ds)` napolni `DataSet`, ki je v bistvu seznam objektov tipa `DataTable`.

Ker je za prikaz večine podatkov `DataTable` bolj primeren, metoda vrne ta podatkovni tip `DataTable`, v konkretnem primeru z indeksom 0 iz objekta tipa `DataSet`.

Ko se pokaže potreba, da se izvede klic na podatkovno bazo brez pridobivanja podatkov iz podatkovne baze, je uporabna metoda `ExecuteSqlCommand` (koda 11), ki se od metode `GetDataTable()` razlikuje po tem, da SQL ukaz le izvede.

```
public void ExecuteSqlCommand(SqlCommand sqlComm)
        {
            this.HandleConnectionErrors();
            sqlComm.Connection = _sqlConn;
            try
            {
                sqlComm.ExecuteNonQuery();
            }
            catch (SqlException)
            {
                throw;
            }
        }
```

Koda 11: Metoda `ExecuteSqlCommand`

V tretjem primeru uporabe pride do potrebe po vračanju samo enega podatka iz podatkovne baze. Primer bi bil indeks iz tabele, v katero se je pravkar vstavil nov zapis, ali vračanje bitnih vrednosti iz podatkovne baze, pri preverjanju dostopnih pravic. Ker v tem primeru ni potrebe po tem, da bi pridobivali celoten objekt tipa `DataTable`, je enostavneje uporabiti metodo `ExecuteNonQuery` (koda 12). Metoda sprejme objekt tipa `SqlCommand`, nato pa vrne objekt tipa `Object`, ko izvede klic metode `ExecuteScalar()` iz objekta `SqlCommand`.

```
public Object ExecuteNonQuery(SqlCommand sqlComm)
        {
            this.HandleConnectionErrors();
            sqlComm.Connection = _sqlConn;
            try
            {
                return sqlComm.ExecuteScalar();
            }
            catch (SqlException)
            {
                throw;
            }
        }
```

Koda 12: Metoda `ExecuteNonQuery`

4.2.3.2 Prenos podatkov po nivojih

Poglejmo primer prenosa podatkov iz podatkovne baze do uporabniškega vmesnika, kjer uporabnik podatke dejansko vidi. Na podatkovnem nivoju imamo metode, ki večinoma vračajo podatkovni tip `DataTable`. Le metode, ki izvajajo zgolj vnos ali posodobitev

podatkov v bazi, pa vračajo podatkovni tip `int`. Metoda `GetAllRules` (koda 13) v razredu `RuleDao` določa ime bazne procedure, nastavi tip `SqlCommand` in potem vrne `DataTable` z direktnim klicem metode `GetDataTable` in parametrom `SqlCommand`. V metodi se uporablja tudi rezervirana beseda `using`. Virtualni stroj ogrodja .NET skrbi za izvajanje programov, upravljanje z napakami in spominom ter samodejno sprosti spomin, ki se je uporabil za shranjevanje objektov, ki niso več potrebni pri izvajanju programa. Sproščanje spomina je nedeterministično, kar pa pri povezavi na podatkovno bazo ni najboljše. Ob velikem številu odprtih povezav do podatkovne baze lahko privedejo do večjih obremenitev in zato zahtevki ne morejo biti več tako hitro izvedeni. Zato je najboljše, da izvajamo politiko čim manj odprtih povezav do podatkovne baze, ki jih odpremo le za čas pridobivanja podatkov in jih takoj nato zapremo. `Using` programerju omogoča, da sam določi, kdaj naj se sprostijo določeni viri (v tem primeru objekt). V konkretnem primeru se to izvede potem, ko metoda vrne vse potrebne podatke, tako da ni potrebno posebej zapirati povezave do podatkovne baze. Klic metode `db.Dispose()` ni potreben, ker se to opravi samodejno v tem razredu. Če želimo nek objekt uporabljati na tak način, moramo implementirati vmesnik (ang. interface) `IDisposable`, ki zahteva, da razred vsebuje metodo `Dispose()`. V tej metodi je lahko določeno vse tisto, kar naj se zgodi ob klicu te metode; pri povezavah na podatkovno bazo je to zapiranje povezave, v kolikor je le-ta še odprta.

```
public static DataTable GetAllRules()
        {
            using (Db db = new
Db(ConfigurationSettings.AppSettings["ms3db"].ToString()))
            {
                SqlCommand sql = new
SqlCommand("sp_RULE_GetAllRules");
                sql.CommandType = CommandType.StoredProcedure;

                return db.GetDataTable(sql);
            }
        }
```

Koda 13: Metoda `GetAllRules` iz razreda `RuleDao`

Logični nivo vsebuje objekt z določenimi metodami in spremenljivkami, ki so dostopne tudi izven razreda. Metoda je statična in vrača seznam objektov tipa `RuleDto` (koda 14).

```
public static List<RuleDto> GetAllRules()
        {
            List<RuleDto> rules = new List<RuleDto>();

            DataTable dt = RuleDao.GetAllRules();
            if (dt.Rows.Count < 1)
                return rules;

            foreach (DataRow dr in dt.Rows)
            {
                RuleDto rule = new RuleDto();
                rule.Write(dr);
```

```
                rules.Add(rule);
            }

            return rules;
        }

dgRules.DataSource = RuleDto.GetAllRules();
```

Koda 14: Metoda `GetAllRules()` iz razreda `RuleDto` ter klic metode

4.2.4 Nastavljanje lastnih pravil za obdelavo sporočil

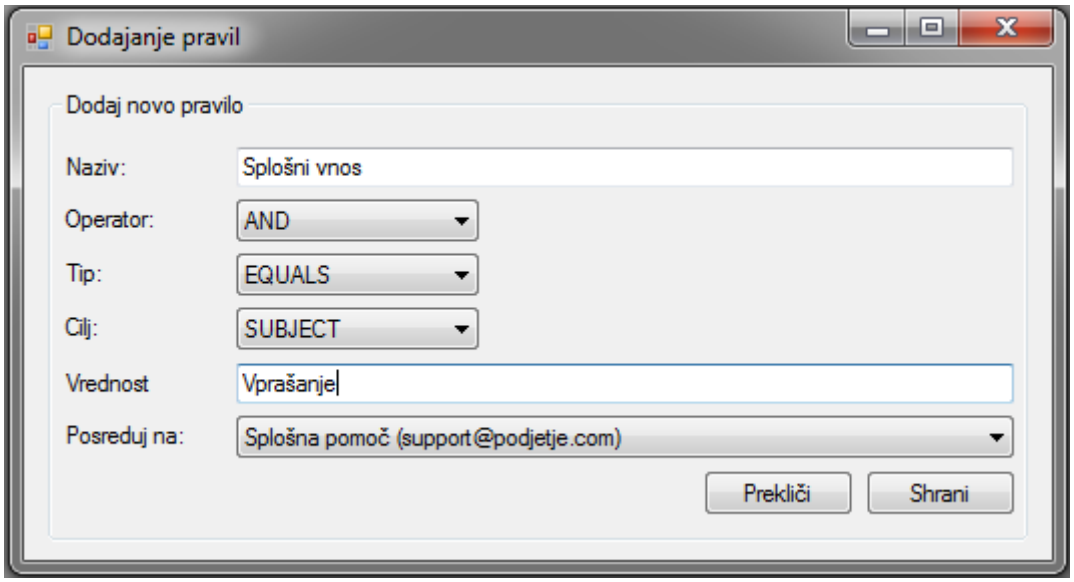
Bistvo aplikacije je avtomatsko posredovanje sporočil, zato se je pokazala izrazita potreba po čimbolj enostavnem, a zmogljivem nastavljanju pravil za posredovanje. Predpostavljamo, da prepoznava nikoli ne bo delovala 100%, se je pa potrebno temu odstotku čimbolj približati. Bolje bi se izrazili tudi tako, da bodo pravila tista, ki jih je težko prilagoditi, da bodo prepoznava 100% delovala. Problem nastane, ker je lahko neka iskana fraza v sporočilu zatipkana, ali pa je uporabljena beseda, ki se za tisti predmet zelo redko uporablja. V tem primeru prepoznava res ne bo delovala, cilj pa je, da se poizkusimo čimbolj približati 100% prepoznavi. Ta odstotek je odvisen tudi od časa, saj s časom pridobivamo izkušnje, kakšno vsebino vsebujejo sporočila, ter pravila ustrezno nastavimo za to.

Sama zasnova teh pravil je enostavna: v elementih elektronskega sporočila poiskati neka ujemanja ter nato glede na ujemanje postopati naprej in razposlati sporočilo na pravo mesto. Elektronsko sporočilo tukaj ne nastopa v izvorni obliki, ampak je že razdrobljeno na osnovne elemente: naslov, pošiljatelja, prejemnika, telo sporočila, datum prejema sporočila ter morebitne priponke. Obrazec za dodajanje (mimogrede, tudi za urejanje je zelo podoben, le naziv obrazca je drugačen, polja in izbirniki pa vsebujejo že definirane vrednosti, ki jih vsebuje določeno pravilo, ki ga urejamo) vsebuje šest različnih vnosov, ki jih moramo izpolniti (slika 8). Naziv je vključen zaradi ločevanja med pravili samimi, pri generaciji SQL poizvedbe nima tolikšnega pomena. Operator določa, ali naj se to pravilo jemlje kot sestavni del nekega drugega pravila, v tem primeru je izbrana opcija IN (ang. AND), kar pomeni, da se to pravilo doda k drugim s SQL stavkom AND. Prav tako sta določena tudi nasprotna operatorja, ki sta dejansko obratni vrednosti operatorjev `in` in `ali` (ang. AND NOT, OR NOT). Polje `tip` določa, po kakšnem vzorcu bomo v nekem sporočilu iskali zadetke. Na voljo so štiri opcije:

- EQUALS pomeni, da iščemo v nekem delu sporočila enako vrednost
- CONTAINS išče vzorec neke vrednosti v ciljnem delu sporočila.
- STARTS WITH na koncu nekega dela sporočila vrednost, ki se začne enako kot izbrana
- ENDS WITH na začetku nekega dela sporočila vrednost, ki se konča enako kot izbrana

Cilj pravila je možen del elektronskega sporočila, v katerem se išče nek določen vzorec. Na voljo imamo naslov sporočila, telo sporočila, datum, pošiljatelja ter prejemnika sporočila. Tukaj je potrebno tudi poudariti primerno sestavo pravila samega, kjer je zelo malo verjetno, da bomo zadeli z nekim pravilom, ki išče v vsebini sporočila neko točno določeno besedno frazo z izrazom EQUALS, kar pomeni neposredno ujemanje vsebine sporočila z našo vrednostjo. Za iskanje po vsebini sporočila je veliko bolj uporabna opcija CONTAINS, kjer se lahko določi pravilo, da ko nekdo vpiše besedo "monitor", se sporočilo posreduje na oddelek tehnične pomoči za strojno opremo, če je pa vpisana beseda enaka besedi "windows 7", se sporočilo pošlje na oddelek tehnične pomoči za programsko opremo.

Vrednost sporočila je tista ključna beseda, po kateri iščemo zadetke z SQL poizvedbo in je poleg naziva tudi obvezen podatek.



Slika 8: Obrazec za dodajanje novih pravil

Logika delovanja pri ustvarjanju novih pravil je enostavna, zato se lahko dodajanja pravil poslužujejo tudi uporabniki brez programerskega znanja. Končni rezultat enega pravila je pogoj, ki je sestavni del celotne poizvedbe na podatkovno bazo. Za to skrbi posebna funkcija `GetSQLString` ki uporablja dejstvo, da vsebuje objekt `RuleOptionDto` tudi lastnost `SQLString`, ki določa vrednost, ki direktno ustreza vrednosti v poizvedbi na podatkovnem strežniku. Ker je aplikacija narejena za podatkovni strežnik Sql Server 2008 Express, morajo ti izrazi ustrezati jeziku T-SQL. Konkreten primer za vrednost `EQUALS` je recimo potrebno prirediti izrazu vrednost "=" na podatkovnem strežniku, za vrednosti `CONTAINS` je pa ustrezen izraz `LIKE`.

Fukcija `GenerateSql` sprejme parameter `groupId` tipa `int`, ki pove, katera pravila iz podatkovne baze je potrebno združiti v ustrezen SQL stavek. Fukcija najprej pokliče fukcijo `GetRulesByGroupId` iz istega razreda, ki vrne seznam objektov tipa `RuleDto`. Fukcija vrne prazen objekt tipa `String`, v kolikor za trenutno skupino ne obstaja noben objekt tipa `RuleDto`. Jedro fukcije sta dve zanki. Zunanja se sprehodi čez objekte tipa `RuleDto`, notranja pa skozi vse objekte tipa `RuleOptionDto`, ki jih pridobi s klicem funkcije

GetRuleOptionsByRuleId. Ta sprejme parameter tipa int, ki je v tem primeru enak identifikatorju objekta RuleDto. V notranji zanki najprej sledi preverjanje spremenljivke isFirstRule tipa bool, ki je na začetku deklarirana z vrednostjo true, ko pa se notranja zanka prvič izvede, se nastavi na vrednost false. Namen te spremenljivke je pravilno generiranje SQL poizvedbe. Ker vemo, da se generira stavek, ki bo pripet na pogoj WHERE v SQL poizvedbi, je treba upoštevati, da prvi pogoj ne sme vsebovat določenega operatorja (v tem primeru AND, OR, OR NOT, AND NOT), saj v nasprotnem primeru SQL poizvedba vrne napako. Vsakič, ko je pogoju zadoščeno, se zgradi objekt tipa StringBuilder, kjer se pripne ustrezna vrednost za gradnjo poizvedbe.

```
private string GenerateSQL(int groupId)
        {
            List<RuleDto> rules =
RuleDto.GetRulesByGroupId(groupId);

            if(rules.Count < 1)
                return String.Empty;

            StringBuilder sb = new StringBuilder();
            bool isFirstRule = true;

            foreach(RuleDto rule in rules)
            {
                foreach (RuleOptionDto ruleOption in
rule.GetRuleOptionsByRuleId(rule.Id))
                {
                    if(!isFirstRule)
                        sb.Append(ruleOption.SqlValue + " ");

                    isFirstRule = false;
                }
            }

            return sb.ToString();
                            }
```

Koda 15: Funkcija GenerateSQLString

Primer dela zgenerirane SQL poizvedbe, ki jo zgenerira funkcija GenerateSQLString (koda 15):

```
mail.Subject = 'Vprašanje' AND NOT mail.Subject = 'Monitor' OR
mail.Body LIKE '%Vprašanje%'
```

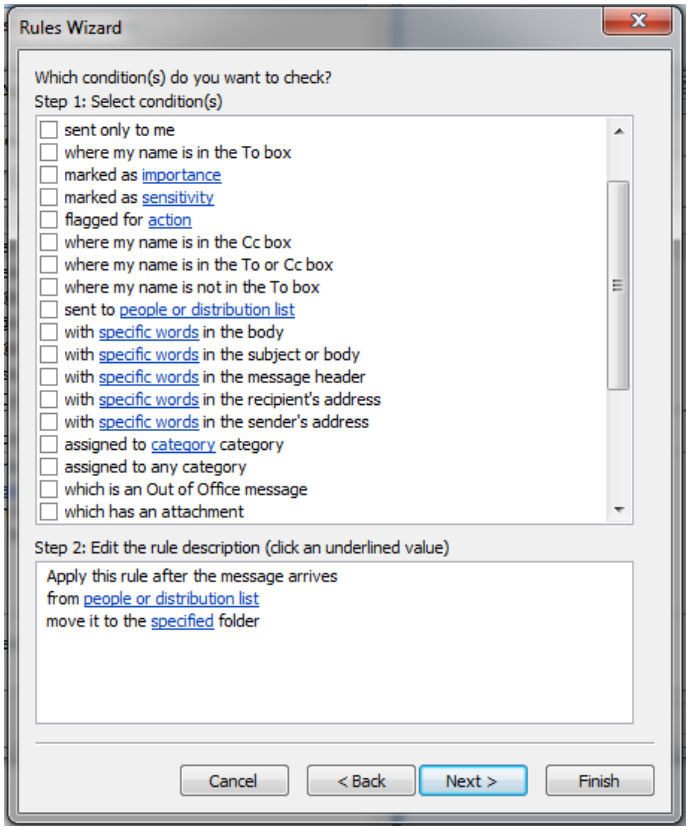
Ta del se doda zraven delno že napisane procedure na podatkovni bazi, skupaj pa je to ena poizvedba, ki vrne vsa elektronska sporočila, ki so shranjena v bazi in ustrezajo pogoju.

4.3 Uporaba aplikacije ter primerjava z obstoječimi rešitvami

Če gledamo masovno uporabo, aplikacija za posredovanje elektronske pošte nima tako širokega polja potencialnih uporabnikov kot bi ga imel običajen odjemalec elektronske pošte. To tudi ni cilj pri tej aplikaciji, ker je zastavljena za natančno definirano uporabo. V poštev za

uporabo pa pride POP3 knjižnica, seveda pod pogojem, da bi bil za prejemanje in pošiljanje elektronske pošte razvit še uporabniški vmesnik.

Podobno funkcionalnost, kot jo ima naša aplikacija, ima tudi Microsoftov Outlook 2007 (slika 9). Outlook 2007 je v bistvu namenjen prejemanju in pošiljanju elektronske pošte, ima pa tudi funkcijo osebnega organizatorja [18]. Podporo ima tudi za nadaljne poredovavnje elektronskih sporočil, kar se lahko določi z nekaj enostavnimi pravili. Kaj je torej v bistvu prednost naše aplikacije?



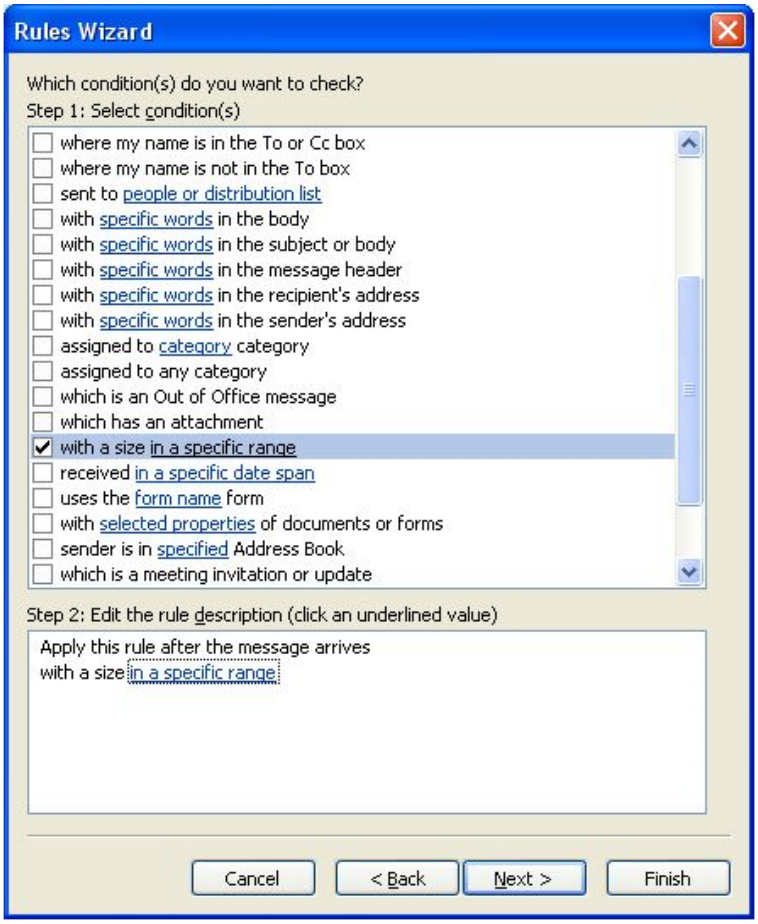
Slika 9: nastavljanje pravil v Outlook 2007

Prednost je predvsem njen izrazit namen za posredovanje elektronske pošte z določenimi pravili, s katerimi lahko uporabnik prosto upravlja. Aplikacija je v osnovi namenjena posredovanju elektronske pošte, Outlook je pa odjemalec in organizator. Namen aplikacije je preprosto prerazporediti pošto na nek pravi končen naslov, kjer lahko uporabnik kot svoj odjemalec še vedno uporablja Outlook. Dodajanje pravil je hitro dostopno, enostavno in se lahko uporablja brez posebnega znanja. Outlook podpira več različnih pravil, ki so precej fleksibilna, se je pa med njimi težje znajti.

Še ena možnost bi bila uporaba Microsoft Exchange poštnega strežnika, kjer bi več uporabnikov hkrati dostopalo do enega računa, večja količina elektronske pošte pa bi bila organizirana po sistemu map. Ko bi nekdo neko sporočilo predelal, bi to ustrezno označil, kar bi se nato posodobilo pri vseh uporabnikih poštnega računa.

Poglejmo konkreten primer, kako bi potekala obdelava sporočila v naši aplikaciji in kako bi šlo v Outlook 2007. Predpostavljamo, da pride sporočilo na določen naslov, kjer ga lahko z

Outlookom pridobimo iz strežnika, na strežniku pa sporočila na izbrišemo, tako da lahko v lastni aplikaciji uporabimo identično sporočilo. Predpostavimo tudi, da imamo v obeh aplikacijah že nastavljene vse nastavitve v zvezi z pridobivanjem sporočil iz strežnika.



Slika 10: Nastavljanje pravila za posredovanje sporočil v Microsoft Outlook

Če primerjamo obrazec za nastavljanje pravil za avtomatsko posredovanje sporočil v naši aplikaciji (slika 4) in v odjemalcu Outlook 2007 (slika 10), je hitro opazno večje število različnih možnosti v Outlooku, kar bi se v naši aplikaciji dalo še dodati. Vendar bi bilo pri tem potrebno upoštevati smiselnost, ker naprimer velikost sporočila nima ravno velike vloge s tem, komu naj se sporočilo posreduje.

V naši aplikaciji se nastavlja pravila za posredovanje sporočil v enem samem obrazcu, medtem ko je v Outlooku potrebno za celoten postopek odpreti kar sedem obrazcev, kar celoten postopek otežuje. Prav tako lahko boljše nastavimo pravila v naši aplikaciji, za kar ni potrebnega nič več znanja. Sama ustreznost pravil je v obeh aplikacijah podobna, bolj kot si je uporabnik sposoben pravila prilgoditi, boljši odstotek ujemanja sporočil bo dosegel.

Samo prejemanje in obdelava sporočil se ne razlikuje veliko, le da je naša aplikacija precej manjša in obsega le potrebne funkcionalnosti. Sporočila se prejemajo avtomatstko na določeno časovno obdobje, ki se ga lahko poljubno spreminja v nastavitvah, podobno kot Outlook. Prav tako je možno v naši aplikaciji sporočila samo prejeti, si jih ogledati v enem

izmed dveh možnih prikazov, ter nato še pred avtomatskih posredovanjem popraviti pravila, če bi se izkazalo, da je to potrebno.

5 SKLEPNE UGOTOVITVE

Cilj diplomske naloge je bil razvoj knjižnice za komuniciranje s POP3 poštnim strežnikom ter praktičen prikaz delovanja knjižnice z uporabo v namenski aplikaciji. Da bi dosegli te cilje, je bilo potrebno preučiti določene tehnologije in to znanje uporabiti v praksi. Največ poudarkov je bilo potrebno posvetiti POP3 protokolu in zgradbi elektronskega sporočila.

Knjižnica je razvita tako, da jo je mogoče zlahka uporabiti v poljubni aplikaciji, bodisi spletni, namizni aplikaciji ali windows servisu. Aplikacija za nastavljanje pravil in posredovanje elektronske pošte je razvita v trinivojskem načinu, kar pomeni enostavno vzdrževanje in dodajanje novih funkcionalnosti.

V praktični uporabi se aplikacija izkaže kot izvrstno orodje, ki služi svojemu specifičnemu namenu. V primerjavi z odjemalci elektronske pošte ima sicer manj funkcionalnosti, je pa veliko bolj enostavna za uporabo. Podana je tudi primerjava med obdelanim elektronskim sporočilom, ter sporočilom v izvornem formatu, kjer se takoj pokaže težavno branje informacij iz elektronskega sporočila, ki nima izluščenih podatkov.

Možna nadgradnja bi bila predvsem selitev aplikacije na splet, s čimer bi zelo veliko pridobili na fleksibilnosti in dostopnosti. To ne bi zahtevalo velikih vložkov v razvoja, ker bi bilo potrebno razviti samo eno obliko predstavitvenega nivoja, ostala dva nivoja bi bila pa ista. Za ta namen je bil razvit tudi spletni servis, v kolikor bi se pojavila potreba po klicih do podatkovne baze na oddaljenem strežniku.

Literatura in viri

[1] (2010) .NET Framework. Dostopno na: http://en.wikipedia.org/wiki/.NET Framework

[2] Johnson, K (2000). Internet Email Protocols: A Developer's Guide. Addison-Wesley Professional.

[3] (2010) RFC 1939. Dostopno: http://www.faqs.org/rfcs/rfc1939.html

[4] Randolph, Nick & Gardner, David (2008). Professional Visual Studio 2008.

[5] (2010) Format RFC-882. Dostopno na: http://www.faqs.org/rfcs/rfc822.html

[6] (2010) Windows Office. Dostopno na: http://en.wikipedia.org/wiki/Microsoft_Office

[7] (2010) Windows Mail. Dostopno na: http://en.wikipedia.org/wiki/Windows_Mail

[8] Mullet, K (2000). Managing IMAP. O'Reilly Media.

[9] (2010) Transmission control protocol. Dostopno na: http://en.wikipedia.org/wiki/Transmission_Control_Protocol

[10] (2010) ASCII. Dostopno na: http://en.wikipedia.org/wiki/ASCII

[11] (2010) MIME. Dostopno na: http://en.wikipedia.org/wiki/MIME

[12] Johnson, K (2000). Internet Email Protocols: A Developer's Guide. Addison-Wesley Professional.

[13] (2010) RFC 2045. Dostopno na: http://www.ietf.org/rfc/rfc2045.txt

[14] (2010) RFC 2049. Dostopno na: http://www.ietf.org/rfc/rfc2049.txt

[15] (2010) Hyper text markup language. Dostopno na: http://en.wikipedia.org/wiki/HTML

[16] Evjen (2009). Professional ASP.NET 3.5 SP1 Edition: In C# and VB. John Wiley & Sons.

[17] (2010) Razred System.Net.Sockets.TcpClient. Dostopno na: http://msdn.microsoft.com/en-us/library/system.net.sockets.tcpclient.aspx

[18] (2010) Microsoft Outlook 2007. Dostopno na: http://en.wikipedia.org/wiki/Microsoft_Outlook

[19] (2010) Three tier architecture. Dostopno na: http://www.dotnetfunda.com/articles/article71.aspx