Beter goed gejat dan slecht geschreven

De MAIL component

De Nederlandse Bridge Bond (NBB, zie [1]) heeft een ledenservice op Internet. Via deze service kan informatie worden uitgewisseld met de ledenadministratie van de NBB. De service biedt de volgende mogelijkheden: individuele leden kunnen hun actuele meesterpunten-saldo en rating-gegevens opvragen, en de secretarissen van verenigingen en districten kunnen rechtstreeks mutaties doorgeven en gegevens opvragen.

De ledenservice werkt via uitwisseling van e-mail. Aanvragen van internetgebruikers worden automatisch verwerkt door de NBB-ledenadministratie en informatie wordt verzonden via het e-mail adres dat wordt opgegeven bij de aanvraag. De NBB-ledenadministratie wordt al geruime tijd beheerd met Oracle. De mailverwerking werkte op basis van Microsoft Outlook versie '97 en een Visual Basic programma, dat berichten als bestand opsloeg. In Oracle werden deze bestanden ingelezen en verwerkt. De bridgebond had de volgende problemen met de mailverwerking:

- De mailverwerking stokte op verkeerde e-mail zoals spam.
- Berichten konden bij een fout niet nog een keer verwerkt worden.
- Het Visual Basic programma werkte niet met Outlook versie 2000 vanwege de strengere beveiligingsinstellingen van Outlook versie 2000. Hierdoor kon de NBB niet 'upgraden' naar Outlook versie 2000.
- De mailverwerking was niet robuust: operationele storingen veroorzaakten problemen met de mailverwerking.
- Een client-programma (Microsoft Outlook) moest op de mailserver van de NBB draaien om het Visual Basic programma te kunnen laten werken. Dit was een ongewenste situatie.

Transfer Solutions werd gevraagd de mailverwerking te vernieuwen op basis van de volgende eisen:

- Het moet mogelijk zijn om e-mail op te halen op basis van criteria zoals de velden 'To' of 'Subject'. De aanvragen via het internet hebben een vast adres en onderwerp en alleen die berichten moeten opgehaald worden.
- Bij het ophalen moet e-mail die niet aan de criteria voldoet, verwijderd worden uit de INBOX. De bridgebond kampt namelijk met veel spam en dit mag de mailverwerking niet hinderen. Verwijderen is dus een effectieve manier om het aantal berichten in de INBOX te beperken.
- Het moet mogelijk zijn om bijlagen te versturen. De bridgebond wil ook PDF-overzichten als bijlagen versturen. De oude mailverwerking voorzag hier niet in.
- De mailverwerking moet robuust zijn: er mogen geen berichten verloren gaan. Als een e-mail niet verwerkt of verstuurd kan worden, dan moet dat later nog een keer geprobeerd kunnen worden.
- Berichten moeten voor enige tijd bewaard kunnen worden om vragen van klanten te beantwoorden.
- De volgende mailprotocollen moeten ondersteund worden: SMTP, IMAP en POP3.
- Aanmelden via gebruikersnaam en wachtwoord bij de mailserver is gewenst.

Het web biedt ontwikkelaars grote voordelen bij het zoeken naarprogrammatuur en documentatie

Uitgaande van deze eisen waren er een aantal mogelijkheden. De eerste was om de oude mailverwerking aan te passen, maar dit viel af vanwege de te geringe robuustheid en de matige kwaliteit van de oude mailverwerking alsmede de afhankelijkheid van Microsoft Outlook en Visual Basic. Een tweede moge-

lijkheid, gebruik van Oracle Collaboration Suite, viel af vanwege kostenoverwegingen.
Wat overbleef was een derde mogelijkheid: nieuwbouw. Het is me in vijftien jaar IT opgevallen dat een deel van de ontwikkelaars lijdt aan het 'Bob de Bouwer' syndroom (zie Software Release Magazine 2002 nr. 5): effe een programmaatje schrijven, liefst zonder al te veel aandacht voor ontwerp, documentatie en testen. Ik wilde daar in ieder geval niet aan mee doen (zie hier ook de relatie tot de titel van dit artikel).

Mijn uitgangspunt was om zo min mogelijk zelf te maken. Dat heeft geresulteerd in het gebruik ban de standaardmogelijkheden van Oracle, en gebruik van andere standaardcomponenten, bij voorkeur Open Source. Oracle biedt standaard de volgende mogelijkheden:

- Er is een utl_smtp package voor het versturen van e-mail. Er is echter geen voorziening voor het versturen van bijlagen. Dat zou dus zelf gemaakt moeten worden. Een weinig aanlokkelijk idee gezien de complexiteit van bijlagen in een bericht.
- Het is mogelijk om een e-mail bericht te modelleren als een object. Bijlagen kunnen dan in het object zelf opgeslagen worden als een collectie.
- Oracle Advanced Queueing. Advanced Queueing kan objecten verwerken en is dan niet begrensd voor wat betreft de grootte van een object, dit in tegenstelling tot het gebruik van het RAW datatype. Verder kan met Advanced Queueing de eis van robuustheid geïmplementeerd worden: het is mogelijk om een bericht herhaald uit een queue te halen (dequeue) als een eerdere dequeue faalde (bijvoorbeeld als de mailserver plat ligt en de transactie daarom onderbroken wordt). Oracle Advanced Queueing maakt het ook mogelijk om objecten te bewaren.
- Oracle kan Java routines als stored procedures in de database uitvoeren.

Eén van de grote voordelen die het web voor ontwikkelaars biedt, is het zoeken naar programmatuur, voorbeelden, documentatie, enzovoorts. Vergelijk dat eens met vijftien jaar geleden. Tijdens mijn zoektocht op het web stuitte ik op een artikel dat beschreef hoe mail met bijlagen verzonden kan worden met PL/SQL (zie [2]). Hierbij wordt gebruik gemaakt van de JavaMail API (zie [3]). Verder had een collega van Transfer Solutions een Java-programma gemaakt dat e-mail met bijlagen kon ophalen via de JavaMail API en in database-tabellen kon stoppen. De zoektocht was klaar en mijn uitgangspunt om zo min mogelijk zelf maken kon ik realiseren. Het ontwerp zag er als volgt uit:

Mail objecttype

De volgende objecttypen definiëren het e-mail object:

```
create or replace type attachment_objtype as object (
  filename varchar2(4000),
  filetype varchar2(100), /· b.v. text/plain */
  content blob /· binary data */
)
/

/· nested table */
create or replace type attachment_tabtype as table of attachment_objty-
pe
/

create or replace type mail_msg_objtype as object (
  from_address varchar2(4000),
  to_address varchar2(4000),
  cc_address varchar2(4000),
  bcc_address varchar2(4000),
  subject varchar2(1000),
  content clob, /· character data */
  attachment_tab attachment_tabtype,
...
)
/
```

Mail manager package

Het mail_mgr package bevat de volgende categorieën van routines:

Queue management

- init: maakt met DBMS_AQADM packageroutines de queue table aan en de queues INBOX en OUTBOX.
- done: verwijdert de INBOX en OUTBOX queues en de queue table.

Mail in de queue zetten

- write: plaatst een bericht in de INBOX/OUTBOX queue.

Mail verzenden

- send: leest berichten uit de OUTBOX queue en verzendt deze naar de mailserver via de volgende routines die in Java zijn geïmplementeerd:
  - mail_open_transport: start een sessie met de mailserver via het smtp protocol.
  - mail_send: verzend één e-mail.
  - mail_close_transport: sluit de sessie.

Mail ontvangen

- receive: haalt berichten op van de mailserver en plaatst deze in de INBOX queue via de volgende routines die in Java zijn geïmplementeerd:
  - mail_open_folder: opent de INBOX van de mailserver via het imap of pop3 protocol.
  - mail_get_headers: haalt de header informatie op ('From', 'To', 'CC', 'Subject') van berichten uit de INBOX. Er kunnen zoekcriteria worden gegeven.
  - mail_get_bodies: haalt de inhoud van een e-mail op alsmede het aantal bijlagen van de berichten die zijn opgehaald via mail_get_headers.

- mail_get_attachments: haalt de bijlagen op van de berichten die zijn opgehaald via mail_get_headers.
- mail_delete: verwijdert de berichten uit de INBOX folder die reeds zijn opgehaald.
- mail_close_folder: sluit de INBOX folder.

Mail uit de queue halen

• read: haalt een bericht op uit de INBOX/OUTBOX queue.

Java programmatuur

Ik heb de Java programmatuur onderverdeeld in twee klassen:

• jmail.java: bevat functionaliteit om berichten naar een mailserver te sturen en van een mailserver op te halen.
• omail.java: is de interface met het mail_mgr package en roept routines aan in jmail.java.

Excepties die in Java optreden worden niet afgevangen en dus doorgegeven aan Oracle. Dit geeft de gebruiker van de MAIL component de mogelijkheid om zelf exception handling te implementeren. De Java routines die vanuit Oracle PL/SQL worden aangeroepen moeten statisch en publiek zijn. Hier volgt een voorbeeld van de definitie van de routine mail_mgr.mail_close_transport:

Java omail klasse:

```
public static void closeTransport(java.lang.Integer[] transportId)
        throws javax.mail.MessagingException
```

Oracle mail_mgr package:

```
procedure mail_close_transport(
  p_transport_id               in out number )
as
language java name 'my.omail.closeTransport( java.lang.Integer[] )';
```

Merk op dat output parameters (of input output) in Java als een array van 1 groot gedefinieerd worden.

Documentatie

Ik wilde graag een standaardmanier van documentatie vaststellen. Voor Java klassen is er Javadoc. De Open Source wereld heeft voor Oracle packages ook een vergelijkbare methode: het programma PLDoc (zie [4]). In een packagespecificatie moet dan Javadoc-achtige documentatie toegevoegd worden en het programma PLDoc maakt er HTML documentatie van. Het onderstaande voorbeeld levert professionele HTML documentatie op uit Figuur 2.

```
CREATE OR REPLACE package mail_mgr is
/**
--
-- This package is used for mail delivery (send and receive).
-- Messages can be received from a mail server and written into the
INBOX queue.
-- Messages can be read from the OUTBOX queue and sent to a mail ser-
ver.
-- The DBMS_AQ and DBMS_AQADM packages are used for implementing the
mail queues.
--
-- @headcom
*/

* rnps can not be added without compilation errors */
pragma restrict_references(mail_mgr, wnds, rnds, wnps);

/*
  The queue (table) names do have to include the schema name of the
owner.
*/

/· mailbox name */
subtype mail_box_type is varchar2(24);

subtype msgid_type is raw(16);

type ref_curtype is ref cursor;

**
--
-- Initialize the mail manager. Must be called once in a lifetime.
-- A mail queue table is created with payload type mail_msg_objtype.<br
/>
-- <br />
-- Example: mail_mgr.init(60, 'tablespace lob')<br />
-- <br />
-- @param p_retention_time      The queue retention time.
-- @param p_lob_storage_clause  The lob storage clause<br />
*/
procedure init(
  p_retention_time in pls_integer default 604800 /· one week */
```

Figuur 1. PLDoc HTML documentatie van package mail_mgr

```
, p_lob_storage_clause in varchar2 default null
);
...
end mail_mgr;
/
```

Aandachtspunten tijdens implementatie

Aanmaken van de queue table
De volgende syntax is nodig om een tabel x met kolom user_data van het type mail_msg_objtype aan te maken met daarin de nested table attachments_tab:

```
create table x ( user_data mail_msg_objtype )
nested table user_data.attachment_tab store as <table>
```

Merk op dat de mail queue table aangemaakt wordt door dbms_aqadm.create_queue_table. De mail queue table bevat onder andere de kolom user_data van het type mail_msg_objtype.
Verder is het mogelijk om de LOB kolommen van de mail objecten in een aparte tablespace op te slaan. De volgende code combineert de creatie van de mail queue tabel met een nested table kolom en een aparte tablespace voor de LOB kolommen. De parameter p_lob_storage_clause is leeg of het specificeert een tablespace zoals 'tablespace lob':

```
procedure init(
  p_retention_time in pls_integer default 604800 /· one week */
, p_lob_storage_clause in varchar2 default null
)
is
  l_storage_clause varchar2(32767) :=
    'nested table user_data.attachment_tab store as attachments';
begin
  if p_lob_storage_clause is not null
  then
    l_storage_clause :=
      l_storage_clause ||
      '(lob (content) store as (' || p_lob_storage_clause || ')) ' ||
      'lob (user_data.content) ' ||
      'store as (' || p_lob_storage_clause || ')';
  end if;

  dbms_aqadm.create_queue_table(
    queue_table => g_queue_table_cst
  , queue_payload_type => 'mail_msg_objtype'
  , message_grouping => dbms_aqadm.none
  , comment => 'Queue table created for storing e-mail messages'
  , storage_clause => l_storage_clause
  );
end;
/
```

Gebruik van Oracle arrays in Java klassen
De mail wordt opgehaald in batches. In de JavaMail API worden berichten geïdentificeerd met een nummer (vanaf 1). De berichtnummers zijn uniek binnen een sessie. Verder is de JavaMail API zo ontworpen dat het netwerkverkeer minimaal is. Van een bericht worden eerst de headers (bijvoorbeeld 'From', 'To' en 'Subject') opgehaald, daarna de tekst van een bericht en als laatste stap is het mogelijk om bijlagen op te halen. Elk berichtveld wordt in PL/SQL met behulp van een collectie opgehaald. Er is dus een array voor 'From', 'To', 'Subject', enzovoorts. Voor de MAIL component zijn de volgende collecties gedefinieerd:

- BLOB_TABTYPE: data van bijlagen.
- CLOB_TABTYPE: berichtteksten.
- INT_TABTYPE: voor berichtnummers en aantallen bijlagen per bericht.
- STR_TABTYPE: voor adressen, onderwerp en bijlagetype.

Een voorbeeld van het schrijven van een Java array msgNr naar een Oracle array msgNrArray van het type INT_TABTYPE:

```
Connection conn = defaultConnection();

ArrayDescriptor intDescriptor =
  ArrayDescriptor.createDescriptor(schemaPrefix + "INT_TABTYPE", conn);

msgNrArray[0] = new ARRAY(intDescriptor, conn, msgNr[0]);
```

Merk op dat het schema van de eigenaar van het arraytype (schemaPrefix) moet worden meegegeven, wanneer andere Oracle gebruikers dan de eigenaar van het type de code aanroepen.

Schrijven van CLOB en BLOB datatypes
Een CLOB of BLOB veld moet eerst worden geopend voordat het beschreven kan worden. Zowel in PL/SQL als in Java kan een Oracle CLOB/BLOB geopend worden. De logische plaats is in de Java code. Deze aanpak faalde echter met een Oracle foutmelding, dus de routine

```
dbms_lob.open(..., dbms_lob.lob_readwrite)
```

wordt aangeroepen in mail_mgr.receive alvorens mail_get_bodies en mail_get_attachments worden aangeroepen.

Sturen van mail zonder bijlagen
Bij een ander project - het tonen van vluchtinformatie voor het vliegveld Leipzig - wordt de MAIL component ingezet om

e-mail met teletekstinformatie op te sturen. Oorspronkelijk werden e-mails zonder bijlagen in MIME formaat opgestuurd, zoals in dit voorbeeld:

```
Date: Fri, 3 Dec 2004 16:23:19 +0100 (CET)
From: g.paulissen@chello.nl
To: g.paulissen@chello.nl
Subject: TFS page 4112
Mime-Version: 1.0
Content-Type: multipart/mixed;
          boundary="----=_Part_10_1259067560.1102087399897"

------=_Part_10_1259067560.1102087399897
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
Content-Disposition: inline

START|411200|20041203162307|
...
END|0000|01118246|

------=_Part_10_1259067560.1102087399897--
```

De Duitse teletekst kon met dit nieuwere formaat niet overweg. Daarom worden de e-mails zonder bijlagen volgens RFC 1521 (zie [5]) verstuurd. Deze worden nu wel goed verwerkt door de Duitse teletekst:

```
Date: Fri, 3 Dec 2004 16:18:05 +0100 (CET)
From: g.paulissen@chello.nl
To: g.paulissen@chello.nl
Subject: TFS page 4112
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 8bit

START|411200|20041203161722|
...
END|0000|01511970|
```

Gebruik van diakritische tekens

Het gebruik van diakritische tekens als é en ü gaf problemen. Deze tekens werden niet goed doorgegeven aan de mailserver en ook niet goed ontvangen door de mailserver.

Mijn eerste gedachte was dat het een Outlook-probleem was, maar daar kwam ik niet mee weg. Zeker niet toen het woord 'München' niet goed aankwam bij de Duitse teletekst, hoewel daar geen Outlook gebruikt wordt. Het probleem ligt in de conversie van Oracle naar Java karaktersets en vice versa. Java werkt zoveel mogelijk met UTF-16 en bij Oracle is het variabel. De Java-documentatie gaf de oplossing voor de conversie van Oracle naar Java. De Java-klasse InputStreamReader() zorgt ervoor dat diakritische tekens goed worden verzonden:

```
InputStream is = body.getInputStream();
BufferedReader in = new BufferedReader(new InputStreamReader(is));
StringBuffer buf = new StringBuffer();
int c;
int bytes;

for ( bytes = 0; ((c = in.read()) != -1); bytes++ )
    {
        buf.append((char)c);
    }

msg.setText(buf.toString());
```

De andere kant op:

```
InputStream is = bodyDataHandler[0][mNr].getInputStream();

if (!(is instanceof BufferedInputStream))
  {
    is = new BufferedInputStream(is);
  }

Writer os = clob[mNr].getCharacterOutputStream();

int curr;

while ((curr = is.read()) != -1)
  {
    os.write(prev);
  }
```

Testen

De Java klassen heb ik niet direct getest, wel het Oracle package mail_mgr. Hierdoor zijn indirect ook de Java klassen getest. Er is voor Oracle een goede Open Source testmethode beschikbaar: utPLSQL (zie [6]). Met deze methode is de MAIL component grondig getest.

Voor debugging is het package PLSDBUG gebruikt (zie [7]). Het PLSDBUG package is gemaakt met de EPC toolkit (zie [8]). PLSDBUG maakt het mogelijk om debugging naar een bestand te sturen, zonder te hoeven wachten op het einde zoals bij DBMS_OUTPUT. Verder geeft PLSDBUG veel informatie zoals hoe vaak een routine is aangeroepen, de minimumtijd, gemiddelde tijd en maximumtijd van alle aanroepen. Alle PLSDBUG code is toegevoegd als commentaar. Een voorbeeld:

```
/*DBUG
  dbug.enter('MAIL_MGR.INIT');
  dbug.print('input', 'p_retention_time: %s', p_retention_time);
/*DBUG*/
```

Standaard is de code inactief vanwege de commentaartekens /* en */. Als je /*DBUG globaal vervangt door --/*DBUG, dan wordt de code actief. Dit is een handige truc voor programmeertalen die niet over precompilers beschikken.

Beschikbaarheid

De MAIL component wordt door Transfer Solutions als Open Source beschikbaar gesteld onder de GNU LGPL licentie. Ga naar http://sourceforge.net/projects/transferware om de MAIL component op te halen.

Conclusie

Met het gebruik van standaardsoftware (Oracle Advanced Queueing, Oracle Object Types, Oracle Java-integratie, JavaMail) is een MAIL component gemaakt, die zijn waarde heeft bewezen in meerdere projecten. Het gebruik van Open Source software zoals PLDoc voor documentatie, utPLSQL voor testen en PLSDBUG voor debugging, zorgt ervoor dat de kwaliteit van de MAIL component hoog is en hoog kan blijven. De hoeveelheid code is beperkt gebleven (ongeveer 4600 regels programmacode). Het web is zeer nuttig gebleken tijdens de zoektocht naar software, documentatie en voorbeelden, ofwel: beter goed gejat dan slecht geschreven.

Gert-Jan Paulissen is consultant bij Transfer Solutions BV (Consulting). E-mail: gpaulissen@transfer-solutions.com).

Referenties

Nr	Titel	Bron
[1]	Nederlandse Bridge Bond	http://www.bridge.nl
[2]	"How to send personalized email to clients registered in my portal www.intrainternet.com using the information stored in our Database Oracle 8i automatically?"	http://asktom.oracle.com/pls/ask/f?p=4950:8:::NO::F4950_P8_DISPLAYID:255615160805
[3]	JavaMail API	http://java.sun.com/products/javamail
[4]	PLDoc	http://sourceforge.net/projects/pldoc
[5]	RFC 1521	http://www.freesoft.org/CIE/RFC/1521/index.htm
[6]	utPLSQL	http://sourceforge.net/projects/utplsql
[7]	TransferWare	http://sourceforge.net/projects/transferware
[8]	De EPC toolkit	Optimize, April 1999, Huub van der Wouden en Gert-Jan Paulissen

NIEUWS

Artikelen met praktische informatie, geschreven door en bestemd voor Oracle-professionals vindt u in het Online Archief van Array Publications. Vaktijdschriften als Database Magazine, Software Release en Java Magazine hebben hun artikelenarchief online gezet. Met een heldere zoekstructuur vindt u snel wat u zoekt op www.optimize.nl.

Oracle presenteert Lite 10g Database

Oracle presenteert Oracle Database Lite 10g, de eerste database die de voordelen van grid computing uitbreidt naar de mobiele medewerkers. Oracle Database Lite 10g is een complete en geïntegreerde oplossing voor het ontwikkelen en inzetten van vitale database-applicaties in mobiele omgevingen. De database biedt alle mogelijkheden om bedrijfskritische systemen via mobiele en ingebouwde apparaten toegankelijk te maken. 10g biedt zo constant toegang tot belangrijke informatie en applicaties zonder dat de gebruiker continu verbonden hoeft te zijn met de bedrijfssystemen aan de back-end. Oracle Database Lite 10g maakt de buitendienst een stuk efficiënter en productiever en zorgt dat de teams in het veld sneller kunnen reageren. Dit brengt kostenverlagingen en meer klanttevredenheid met zich mee. Mobiele technologieën zijn niet langer alleen het werkveld van de verkoopdienst. Ook binnen domeinen als defensie, nationale veiligheid, de zorg en openbare veiligheid zijn tegenwoordig buitendienstmedewerkers te vinden. Nu organisaties hun kritische data toevertrouwen aan mobiele technologie, worden gecentraliseerd management, beveiliging en de functionaliteit van applicaties essentieel. Oracle Database Lite 10g is gebaseerd op de Oracle grid computing-infrastructuur en biedt klanten daarom een veilige en betrouwbare technologische architectuur. "De mobiele markt in Nederland groeit op dit moment snel in vrijwel alle segmenten", aldus Aernoudt Bottemanne, Business Development Manager voor Mobiele oplossingen van Oracle Nederland. "Het gebruik van grid computing binnen mobiele omgevingen kan de toename van mobiele datatoegang bij bijvoorbeeld overheden, in de zorg en in de zakelijke dienstverlening gemakkelijker maken."