RESTful PL/SQL

Heiko Blau
Freiberuflicher Software-Entwicker

heiko.blau@blau-it.de
http://blau-it.de

Alexander Möckel
IBYKUS AG, Erfurt

alexander.moeckel@ibykus.de
http://www.ibykus.de

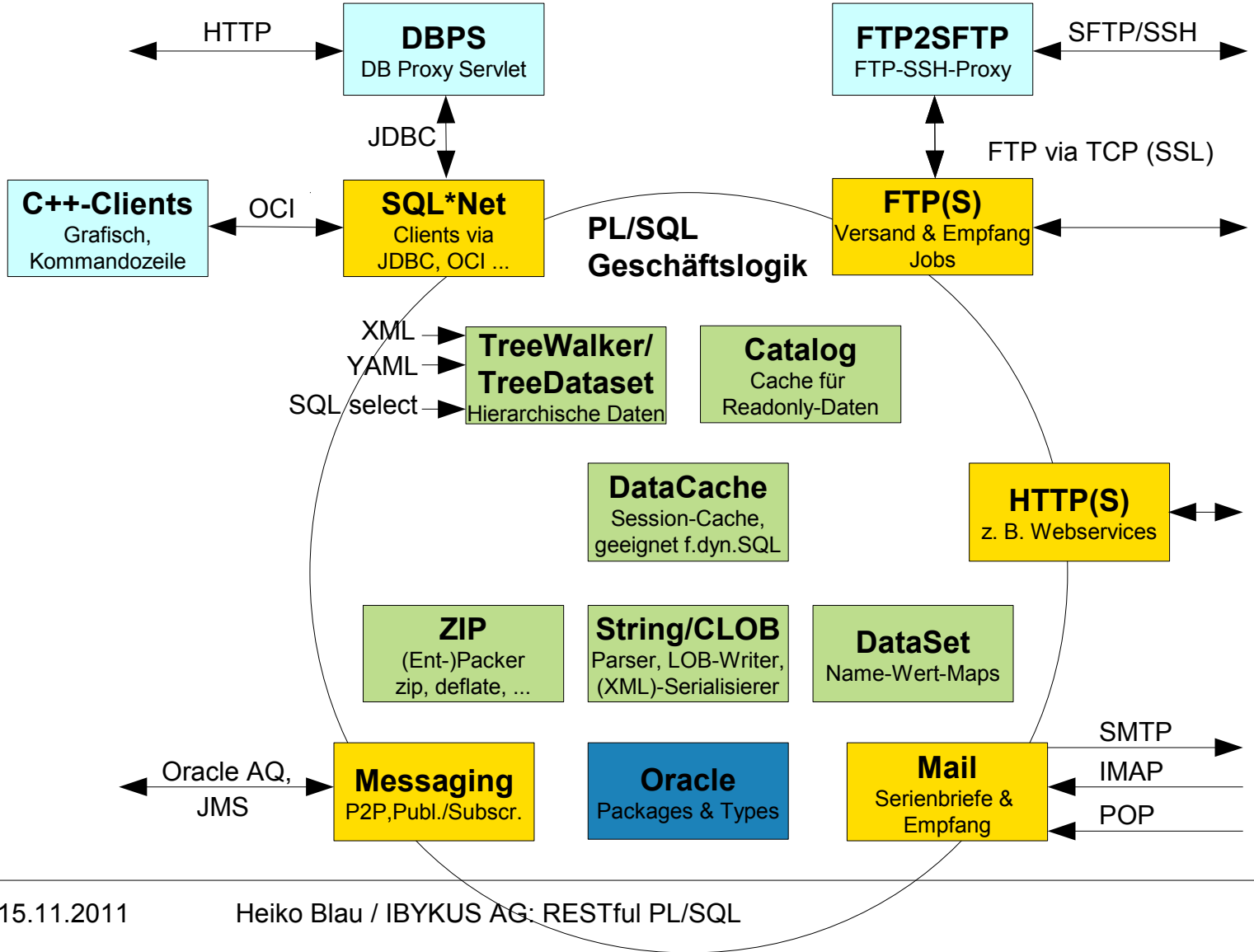
Agenda

- Pro und kontra PL/SQL – bzw wann ist PL/SQL eine gute Wahl.
- Komponenten einer gemeinsamen Code-Basis.
- Von CSV zu Daten-(De-)Serialisierung - String- und CLOB-Funktionalität.
- Universelle Hilfsmittel - Datasets, Datacache und Tree-Walker.
- Kommunikationskanäle jenseits von SQL*Net.
- Request/Response-Kommunikation mit der Datenbank
- Fazit

Wann ist PL/SQL eine gute Wahl?

- **Grundsätzlich:** Oracle-Datenbank ist gesetzt – Datenbankunabhängigkeit ist kein Thema.
- Heterogene Schnittstellen; z. B. über Web (JEE, .Net, Ruby, ...), Rich Clients (OCI), Massendaten-Import/Export (JDBC, OCI, ...).
- Existierende PL/SQL-Anwendungen/Architekturen/Artefakte.
- PL/SQL-erfahrene Teams bzw. Quereinsteiger in die Programmierung.
- Daten-zentrische Anwendungen.
- **Ansichtssache:** Einfachere Architektur, Zuverlässigkeit/Vorhersehbarkeit über viele Jahre, performanter als ...

Komponenten einer Code-Basis



String-Funktionalität I

- Kontinuierlich wachsende Package(s) durch Input aus den laufenden Projekten.
- Sehr gut (unit-)testbar.
- Implementierung austauschbar (`instr -> regex_instr`).

```
declare
  elemNo    integer;
  elems     dbms_sql.varchar2_table;
  csvList   long := 'red,yellow , orange,   green, blue,, black';
begin
  -- Listen komfortabel auseinandernehmen
  elemNo := IBK_String.Split(csvList, elems, ',',
              IBK_String.TRIM + IBK_String.IGNORE_EMPTY);

  -- das gleiche als table function
  select column_value bulk collect into elems
  from table(IBK_StringUtils.Split(csvList, ',',
              IBK_String.TRIM + IBK_String.IGNORE_EMPTY));
end;
```

String-Funktionalität II

- Flags machen flexibel: `TRIM`, `SET_OF_CHARS`, `IGNORE_EMPTY`, `KEEP_DELIM`, `C_STRING`, `SQL_STRING`, `IGNORE_CASE`, `NESTED_EXPR` ...
- Hoher Verbreitungsgrad rechtfertigt aufwändiges Tuning und ständige Weiterentwicklung.
- Automatisierbare Unit-Tests sind essentiell.

```
declare
  pairNo    integer;
  pairs     IBK_String.t_tabNameValuePairs;
  propList  long := 'c1=red, c2=yellow; c3 = "black, grey & white"';
begin
  AssertEqual(3, IBK_String.SplitPairs(propList, pairs, ',;',
                    IBK_String.SET_OF_CHARS + IBK_String.TRIM +
                    IBK_String.IGNORE_EMPTY + IBK_String.C_STRING));
  AssertEqual('c1', pairs(1).NAME);
  AssertEqual('red', pairs(1).VALUE);
  AssertEqual('c2', pairs(2).NAME);
  AssertEqual('yellow', pairs(2).VALUE);
  AssertEqual('c3', pairs(3).NAME);
  AssertEqual('"black, grey & white"', pairs(3).VALUE);
end;
```

String-Funktionalität III

Property-File? JSON oder YAML? DSL?

```
CONNECTIONS: {
  CONNECTION1 = {
    HOST: ftp.ibykus.de, PROTOCOL: ftp, WORKING_DIR="/data/DOAG 2011"
  }
  CONNECTION2 = { URL="https://www.ibykus.de/ws"; USER=scott }
}
DATA = '<?xml version="1.0" encoding="UTF-8"?>
  <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Body>
      <d:document xmlns:d="http://blau-it.de/doag">
        <d:DOC_IDENT>BLOG_2011-09-14_13:17</d:DOC_IDENT>
      </d:document>
    </soap:Body>
  </soap:Envelope>'
}
```

```
IBK_String.SplitPairs(text, pairs, ',;' || chr(10), ':=',
    IBK_String.SET_OF_CHARS + IBK_String.TRIM + IBK_String.IGNORE_EMPTY +
    IBK_String.NESTED_EXPR + IBK_String.SQL_STRING + IBK_String.C_STRING);
```

CLOB-Writer

Aus der Package **`DBMS_LOB`**, Funktion **`GetChunkSize`**:

„Performance is improved if you enter read/write requests using a multiple of this chunk size.“

(vgl. http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14258/d_lob.htm#i998451).

- Wrapper um `dbms_lob` ist notwendig.
- Unterschiedliche *chunk size* (ISO-8859-1 != UTF-8) „verstecken“.
- Ein CLOB-Writer kann
  - Daten aufsammeln, bis die *chunk size* oder ein Vielfaches erreicht sind,
  - in andere Zeichensätze konvertieren (vor allem UTF-8),
  - Mark/Rewind unterstützen (nachträgliches Schließen leerer XML-Elemente),
  - Formatierungen und Einrückungen ermöglichen (Begin/Continue/EndBlock)

XML-Serializer

- Wrapper um allg. CLOB-Writer
- Organisiert Einrückung, Abschluss von Elementen, Sonderzeichen-Ersetzung
- Größter Nachteil: Dokument muss seriell geschrieben werden

```
declare
  handle  integer := IBK_XMLUtil.OpenWriter(
                                    'PRETTY_PRINT, CHARSET=UTF-8');
  result  clob;
begin
  IBK_XMLUtil.WriteComment(handle, 'A very simple XML document');
  IBK_XMLUtil.OpenElement(handle, 'list', attr => 'count=???');
  for (select rownum id, name, text, type from content_table) loop
    if rec.type = 'SIMPLE' then
      IBK_XMLUtil.WriteElement(handle, rec.name, rec.text, 'id='||rec.id);
    else
      IBK_XMLUtil.OpenElement(handle, rec.name, 'id=' || rec.id);
      IBK_XMLUtil.WriteCData(handle, rec.text);
      IBK_XMLUtil.CloseElement(handle);
    end if;
  end loop;
  IBK_XMLUtil.CloseWriter(handle, result);
end;
```

Datasets

- Macht PL/SQL *associative arrays* benutzbar :-)
- Weg von den fixen Tabellen-*rowtype* zu variablen Attributen (Propertys, Parameter)
- Entscheidender Schritt zu Querschnittsfunktionalität und stabilen Schnittstellen

```
package IBK_DataSet is
  procedure Put (
    dataSet   in out nocopy t_recDataSet,
    name      in varchar2,
    value     in Type        -- varchar2, date, number etc.
  );

  function GetType (
    dataSet   in t_recDataSet,
    name      in varchar2,
    defValue  in Type default null
  ) return Type;

  function Exists (dataSet in t_recDataSet, name in varchar2)
    return boolean;
end;
```

Data Access Object (DAO) – Pattern

- Dataset als Transfer Object
- Fach- und allgemeine Packages/Routinen als DAO führen CRUD-OP's aus.
- Trennung von Read/Write (select/insert/update) von der Verarbeitungslogik
- Vermeidet verstreutes und Mehrfach-SQL (select, insert/update/delete)

```
-- Einlesen von Daten
IBK_DataSetIO.Read(dataset, table => 'IBK_DOC', ident => 'doag-2011');

-- Fachlogik
IBK_DataSet.PutIfNull(dataset, 'CREATED', sysdate);
IBK_DataSet.Put(dataset, 'VERSION',
                IBK_DataSet.GetNumber(dataset, 'VERSION', 0) + 1);

-- Aktualisieren der Daten
IBK_DataSetIO.Write(dataset);
```

Der Datacache

- Cacht Datasets für eine DB-Session.
- Ermöglicht dynamische Aufrufe.
- Geeignet für Client-Server-Konversation.
- Lookup/Release, Swapping, Datentransport über Database Links
- Stabile, auch dynamische Schnittstellen; Cache-Key ist immer in-Parameter

```
package IBK_DataCache is
  procedure PutDataSet (
    key in varchar2, dataset in IBK_DataSet.t_recDataSet
  );

  procedure RemoveDataSet (key in varchar2);

  procedure Put (
    key in varchar2, name in varchar2, value in Type
  );

  function GetType (
    key in varchar2, name in varchar2, defValue in Type
  ) return Type;
end;
```

TreeWalker – hierarchische Daten

DOM und XMLType sind XML-zentrisch.

➡ Umständlich für die Fachlogik.

➡ Effektiver: allgemeine Baumstruktur mit DOM- und Xpath-Navigationsmitteln.

- Baumstruktur wird über Handler ***bei Bedarf rudimentär oder vollständig*** gefüllt.
- Handler sind XML-, JSON-, YAML-, Property-Parser, Routinen mit SQL-selects u. a.
- Knoten werden im DataCache gehalten.
- Cache-Keys sind hierarchisch aufgebaut: `'/envelope/body/list/document'`
- Teilbäume können gebildet werden
- DOM-artige Navigation: `GetFirstChild, GetNextSibling, GetParent`
- Xpath-Abfragen: `Get(..., './document[@name="DOAG"]')`
- Kombination: `GetFirstChild(..., '*[@type="csv"]')`

TreeWalker-Beispiel

```
declare
  xmlData  long := '<?xml version="1.0" encoding="UTF-8"?>
                   <root>
                     <record id="1">
                       <ident>KERMIT</ident>
                     </record>
                     <record id="2" ident="FOZZY">
                       <name>Fozzy</name>
                     </record>
                   </root>';
  handle   long := IBK_TreeWalker.New(
                      xmlData, 'DIALECT=xml, LEAFS_AS_ATTR');
  key      long := IBK_TreeWalker.GetRoot(handle);
  children dbms_sql.varchar2_table;

begin
  IBK_TreeWalker.GetChildren(children, handle, key, '*[@id]');
  for ii in 1 .. children.COUNT loop
    ident := IBK_DataCache.GetString(children(ii), 'IDENT');
  end loop;
  IBK_TreeWalker.Close(handle);
end;
```

Tree-DataSets

- Speziell bei „kleinen“ Hierarchien ist auch eine TreeWalker-Navigation noch zu umständlich.
- Close muss sicher gerufen werden; Baum ist nicht besonders „transportabel“.
- Und s. o.: XML-Serializer ist XML-zentriert; außerdem stört die Serialität.

➡ Datasets mit hierarchischen Schlüsseln (analog zu den Cache-Keys des TreeWalker): `'/soap:Envelope/soap:Body/i:GetCustomers'`

```
declare
  ds            IBK_DataSet.t_recDataSet;
  resultAsXML   clob;
begin
  IBK_DataSet.Put(ds, '/s:Envelope/s:Body/i:GetCustomers/i:year', 2011);
  IBK_DataSet.Put(ds, '/s:Envelope/s:Body/i:GetCustomers/i:add.1', 'X');
  IBK_DataSet.Put(ds, '/s:Envelope/s:Body/i:GetCustomers/i:add.2', 'Y');
  IBK_DataSet.Put(ds, '/s:Envelope.xmlns:s',
                      'http://www.w3.org/2003/05/soap-envelope');
  IBK_DataSet.Put(ds, '/s:Envelope/s:Body/i:GetCustomers.xmlns:i',
                      'http://www.ibykus.de/ws');
  ...
```

Tree-DataSets II

```
  ...
  IBK_DataSetUtil.Serialize(resultAsXML, ds, 'FORMAT=xml, PRETTY_PRINT');
end;
```

```
<?xml version="1.0" encoding="UTF-8"?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
  <s:Body>
    <i:GetCustomers xmlns:i="http://www.ibykus.de/ws">
      <i:year>2011</i:year>;
      <i:add>X</i:add>
      <i:add>Y</i:add>
    </i:GetCustomers>
  </s:Body>
</s:Envelope>
```

Tree-DataSets III

- Zahlen- und Datumserkennung.
- Relevanter Subtree.
- Namespace-Eliminierung.

```
declare
  treeDS      IBK_DataSet.t_recDataSet;
  year        integer;
  clobXML     clob := :XML_DATA;
begin
  IBK_TreeWalker.GetTree(
    treeDS,
    'DIALECT=XML,
     TRIM_NAMESPACES,
     ROOT=/Envelope/Body,
     NUMBER_FORMAT=FM99990D999|FM99990,
     DATE_FORMAT=ISO',
    clobXML
  );
  year := IBK_DataSet.GetNumber(treeDS, '/GetCustomers/year');
end;
```

Kommunikationskanäle

- FTP, HTTP, Mail, Messaging Systeme.
- Sichere Varianten (HTTPS, SSL/TLS, SSH) ersetzen die „normalen“ Protokolle.
- Oracle-UTL-Packages und Java APIs:
  - `utl_tcp / java.net + javax.net,`
  - `utl_http / java.net.HttpURLConnection`
  - `utl_smtp / javax.mail`

```
declare
  connWithUtlTcp    integer;
  connWithJava      integer;
begin
  -- Funktionen geben eine Verbindungs-ID zurück.
  connWithUtlTcp := IBK_Messaging.Open(
                      'TYPE=FTP, USE_UTL_TCP, HOST=ftp.ibykus.de, ...');
  connWithJava   := IBK_Messaging.Open(
                      'TYPE=FTP, USE_UTL_TCP=false, USE_SSL, ...');
  ...
end;
```

Database Proxy Servlet

- Datenbank „als Webserver“.
- Alternative zu DBMS_EPG / mod_plsql.
- Servlet kann „beliebig viele“ Datenbanken bedienen.
- Mappings statisch oder zur Laufzeit.
- Request und Response via Cache-Key oder LOBs (XML, YAML).

```
<context-param>
  <param-name>/dbserver/ENTW1/WS/Customers</param-name>
  <param-value><![CDATA[
    driver   = oracle.jdbc.driver.OracleDriver
    scope    = request
    cache    = true
    url      = jdbc:oracle:thin:@dbserver:1521:ENTW1
    username = scott
    password = tiger
    call     = IBK_WS.GetCustomers
  ]]>
  </param-value>
</context-param>
```

Kommunikation über Cache-Key

- Robuste und relativ leicht testbare Schnittstelle.
- Programmatisch leicht nutzbar.
- Kein Serialisierungs- bzw. Deserialisierungs-/Parse-Aufwand.

```
declare
  dataset   IBK_DataSet.t_recDataSet;
  key       long := ?;  -- Cache-Key
begin
  IBK_DataSet.Put(dataset, 'REQUEST.CONTENT', ?);  -- z. B. SOAP-Message
  IBK_DataSet.Put(dataset, 'REQUEST.REQUEST_METHOD', ?);
  IBK_DataSet.Put(dataset, 'REQUEST.QUERY_STRING', ?);
  ... -- weitere CGI-Variablen

  IBK_DataCache.PutDataSet(key, dataset);
  execute immediate 'begin ' || ? /* handler */ || '(key); end;'
  using in key;

  ? := IBK_DataCache.GetNumber(key, 'RESPONSE.STATUS', 200);
  ? := IBK_DataCache.GetBlob(key, 'RESPONSE.CONTENT');
  ... -- weitere Ergebnisfelder
end;
```

Kommunikation über LOBs I

- Robuste, einfache und leicht testbare Schnittstelle.
- Hohe Unabhängigkeit vom Umfeld.
- Auch für den Aufrufer gut austauschbar / testbar.

```
import groovy.sql.Sql

def doRequest = { request ->
  def sql = Sql.newInstance('jdbc:oracle:thin:@db-server:1521:MYDB',
                            'me','secret', 'oracle.jdbc.OracleDriver')
  try {
    def result = ''
    sql.call('begin DoRequest(?, ?); end;', [ request, Sql.CLOB ])
      { response -> result = response.getAsciiStream().getText() }
    return result
  } finally {
    sql.close()
  }
}
```

Kommunikation über LOBs II

- (Ungefähr) das gleiche als HTTP-Request.
- Via DBPS
- PL/SQL-Handler wird transparent über zwei unterschiedliche Wege aufgerufen.

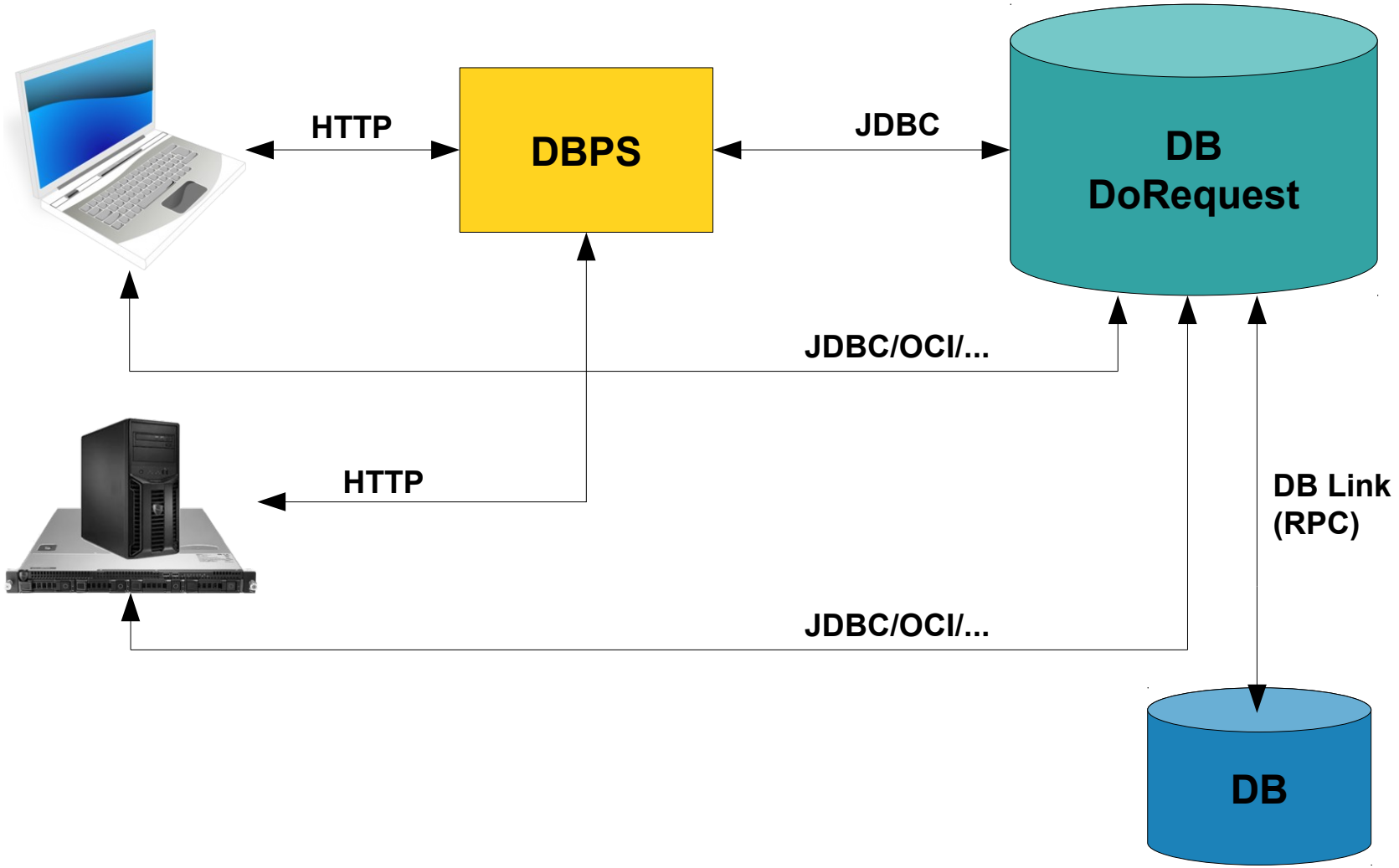
```
import groovyx.net.http.RESTClient
import static groovyx.net.http.ContentType.XML

def doRequest = { request ->
  def client = new RESTClient('http://ibykus.de/')
  def response = client.post(path: 'DBPS/MYDB/DoRequest',
                             contentType: XML
                             body: request)
  return response.data    // a GPathResult object
}
```

Kommunikation über LOBs III

```
-- Test-Mock als anonymer PL/SQL-Block
declare
  response   clob;
begin
  DoRequest(
    '<?xml version="1.0" encoding="UTF-8"?>
     <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
       <soap:Body>
         <d:document xmlns:d="http://blau-it.de/doag">
           <d:DOC_IDENT>BLOG_2011-09-14_13:17</d:DOC_IDENT>
           <d:DOC_TITLE>Blog entry from Sep. 14th, 2011</d:DOC_TITLE>
           <d:DOC_MIMETYPE>text/plain; charset="utf-8"</d:DOC_MIMETYPE>
           <d:DOC_TEXT_CONTENT>My first blog entry</d:DOC_TEXT_CONTENT>
         </d:document>
       </soap:Body>
     </soap:Envelope>',
     response);
  dbms_output.put_line(dbms_lob.substr(response));
end;
```

Request/Response mit PL/SQL



Fazit

- PL/SQL bewährt sich auch jenseits traditioneller SQL-Ergänzung.
- Einfach benutzbare, kontext-lose API's ergänzen sich und ermöglichen effektive Fachlogik.
- Entwicklung von Querschnittsfunktionalität parallel / just-in-time zu den laufenden Projekten resultiert in passgenauen Hilfsmitteln für die Fachentwickler.
- Gut entworfene APIs (→ Package Header) UND Testfälle erlauben Erweiterbarkeit, Alternativ-Implementationen und umfassendes Refaktoring.
- Maps und LOBs als Parameter eignen sich gut für Request/Response-Szenarien bzw. REST-orientierte Kommunikation

Fragen?

… Vielen Dank für Ihre Aufmerksamkeit.