Annexe

Annexe 1 : Technologische overview

1. Ontwikkelomgeving

De eerste versie van Pallas was volledig client-server georiënteerd. Het wijzigen en updaten van de gegevens gebeurde in een programma dat was geschreven in Oracle Basic, raadpleging (OPAC) met een programma geschreven in Microsoft Basic. Zolang het gebruik van het systeem intern bleef stelde zulks geen problemen, maar van zodra het er op aankwam de data ook via het internet beschikbaar te stellen van het publiek, bleek deze oplossing onvoldoende. Het eerste wat we dan ook ontwikkeld hebben in het project is een web-based OPAC, waarbij we gebruik hebben gemaakt van de HTP-module van Oracle, in essentie een extie van PL/SQL. De werking ervan is goed vergelijkbaar met de "Printwriter"- class van Java, waarbij een string (procedureel) wordt geconcateneerd een vervolgens door een gespecialiseerd programma naar een serverpoort wordt gesluisd. Hoewel de software daartoe eigenlijk op de (Oracle Apache) server draait, merkt de ontwikkelaar daar weinig van en kan hij gewoon PL/SQL-procedures en functies schrijven.

Hoewel deze manier van werken een zeer dynamische ontwikkelomgeving oplevert die alleen gelimiteerd wordt door de beperkingen van PL/SQL als programmeertaal (en die, het moet gezegd, de laatste jaren vrijwel volledig zijn opgelost), heeft deze methode toch ook nadelen, die te vergelijken zijn met het werken met Java-servlets. De code is niet altijd even gemakkelijk leesbaar, hetgeen het debuggen zeer tijdrovend kan maken, zeker als het software betreft die al een tijdje geleden werd ontwikkeld. Om toch op meerdere platforms te kunnen werken (hetgeen een vereiste was in het project) werd voor de catalografie aanvankelijk gekozen voor het ontwikkelen van een client pur sang, maar dan in Java. Dit bleek pas na een lange investering in tijd en energie niet de beste keuze, vooral ook omdat de ontwikkelaar ervoor had gekozen de zogenaamde "Business Component for Java" (BC4J) te implementeren, hetgeen een onaanvaardbaar zware overhead opleverde en de responstijd van de toepassing onaanvaardbaar slecht maakte.

Daarom werd op een bepaald moment in de ontwikkeling gekozen voor het afleveren van pure html en xml aan de client, waardoor we in één slag ook onafhankelijk werden van het toegepaste OS of de configuratie van de server. Ook hiervoor was er een keuze uit verschillende technologiën, en rekening houdend met hetgeen reeds gezegd werd over PL/SQL- en Java servlets, viel de eerste keuze op Oracle Java Server Pages. In deze technologie worden de dynamische gegevens van een door te sturen document, bv. in html, vervangen door JSP-tags, die herkend worden door de webserver en via geijkte Java-classes worden opgelost door de data af te halen op de databankserver.

Deze classes en de toepassing van de JSP-tags bleken echter dermate kryptisch gedocumenteerd, dat het uitbreiden van de te beperkte functionaliteit ervan een onmogelijke opgave was. Ook andere oplossingen, zoals PL/SQL-pages of Oracle XSQL, bleken te veel nadelen te hebben om bruikbaar te zijn. Daarom werd besloten een eigen set van tags te formuleren en daarvoor in PL/SQL een specifieke parser te ontwikkelen. Deze manier van werken heeft het voordeel dat de (relatieve) overzichtelijkheid van het JSP-principe behouden blijft, terwijl we toch onafhankelijk zijn van de Java-classes van Oracle en zelfs nooit rechtstreeks JDBC moeten gebruiken, terwijl we tegelijkertijd dezelfde, bijna pointilistische controle houden over de software omdat de parser van eigen maak is. Het resultaat is een praktisch stuk ontwikkelgereedschap, waarmee vrijwel alle invulformulieren van de webversie van de catalografie zijn ontwikkeld. De kans dat Oracle op termijn één van de twee of beide gebruikte technologiën zou laten vallen (PL/SQL en Java) is bijzonder gering, en voor het overige kunnen wij ons concentreren op de evolutie van de webstandaarden zoals DHTML en xml/xsl.

De parser werkt als volgt (syntax: annex 2): de browser van de client stuurt (in html) een pls-request naar de webserver, met als parameters de locatie en naam van het te parsen bestand en zaken zoals de taal en het ID (RECNUMM) van een beschrijving. De web server parseert de request en geeft het door aan de Oracle PL/SQL engine die deel uitmaakt van de Oracle Apache server (hetgeen allemaal Java classes zijn). Deze engine stuurt de request vervolgens naar de databank, die het ontvangt en verder ook behandelt als een call voor een procedure (in dit geval PL/SQL, maar dat kan ook Java zijn). In dit geval is die procedure de PL/SQL parser ("plsp").

De parser opent vervolgens het gevraagde bestand (alleen lezen), dat zich overigens obligaat in een directory moet bevinden die als opstartparameter van de Oracle databank is geformuleerd. Hierdoor is het bronbestand in elk geval ontoegankelijk via de webserver. De parser leest het bestand lijn per lijn in een PL/SQL-tabel in, terwijl de syntax wordt gecheckt en een aantal zaken worden geconcateneerd, en sluit vervolgens het bestand. Elke lijn in de tabel wordt vervolgens één voor één verwerkt, waarbij alle waarden en tags vervangen worden door de corresponderende waarden in de databank. De parser print tenslotte alle lijnen met het HTP-package van Oracle, tot aan het einde van de tabel. Dit package stuurt het nu gegenereerde document door naar de Apache server, die het over poort 80 terugstuurt naar de eindgebruiker.

Er is enige overhead omdat elke ssv-tag (zie de syntax in bijlage) op dezelfde pagina moet worden opgelost in een afzonderlijk SQL-statement (het ware beter het statement slechts één keer uit te voeren), maar dat is nou precies wat van een SQL-databank als Oracle mag verwacht worden, dat SQL-statements goed en snel worden uitgevoerd. De responstijd is in het algemeen meer dan bevredigend en de responstijd van de applicatie wordt momenteel meer bepaald door de snelheid waarmee de client HTML pagina's kan opbouwen dan door de verwerkingssnelheid van de data op de server. Het programmeerwerk wordt er in elk geval sterk door vereenvoudigd, omdat de ontwikkelaar niet meer te maken krijgt met locale cursors, of bepaalde stukken van een formulier die, in tegenstelling tot de rest, deel uit maken van een zeer specifieke SQL-select.

2. Virtuele databanken

Het systeem maakt nu ten volle gebruik van het principe van de Virtual Private Database, waarbij één fysische databank een in principe onbeperkt aantal "virtuele", van elkaar afgescheiden databanken bevat. Deze VPD is bovendien applicatie-onafhankelijk : het doet er niet toe vanuit welk stuk software verbinding wordt gelegd met de databank, eens de verbinding is gemaakt kan elke gebruiker, door die logon, in één en slechts één databank bewegen. Een uitzondering hierop zijn de systeemgebruikers, die sowieso extra afscherming nodig hebben, en de OPAC-gebruiker, omdat die in staat moet zijn meerdere databanken tegelijk te consulteren (in dat geval zijn de scheidingsmuren tussen de databanken wel applicatiegebonden).

3. Toegankelijkheidsniveau's

De toegankelijkheid van de gegevens wordt eveneens bepaald op het moment van de logon, en is geregeld door middel van specifieke hiërarchische queries (tot nader order en jammer genoeg specifiek voor Oracle). Daarbij kan elke gebruiker deel uitmaken van één of meerdere groepen, en elke groep van één of meerdere groepen, zodat de toegankelijkheidsniveau's, in combinatie met gebruikersprofielen, perfect zijn aan te passen aan om het even welke werkomgeving. De OPAC heeft in principe alleen toegang tot het publieke niveau. Hierdoor is het mogelijk gegevens in te voeren die niet onmiddellijk zichtbaar zijn voor een of meerdere groepen van gebruikers van de databank, en, als het moment gekomen is, deze gegevens te "publiceren" door het toegangsniveau ervan te veranderen.

4. Zoekmachine

Zowel de archiefboom als de zoekmodule zijn in de nieuwe versie identiek in de OPAC en de catalografie, hetgeen niet alleen een aanzienlijk besparing oplevert in het onderhoud en de ontwikkeling ervan, maar tevens extra scholing bij de gebruikers overbodig maakt. Uiteraard zitten de verschillen niet alleen in de opmaak, maar zijn de meeste van de catalografische functies in de OPAC-versie niet aanwezig.

5. Archiefbeschrijven

Het "publiceren" van archivalische gegevens is vooral interessant bij het opstellen en vervolgens "publiceren" van archiefinventarissen, maar ook de oude methode van publicatie is gehandhaafd. De gebruiker kan immers voor elke archiefbeschrijving die ook andere beschrijvingen bevat een EAD-inventaris genereren, en die vervolgens laten formatteren volgens een specifieke stylesheet. Momenteel zijn dat er 4, maar het aantal noch de vormgeving zijn beperkt. Hiermee proberen we tegemoet te komen aan een van de mogelijke bezwaren tegen EAD als catalografisch instrument, nl. de noodzaak voor de archivaris om zich te verdiepen in een systeem van tags en attributes dat de facto irrelevant is voor het beschrijven van archieven en alleen in technologischje zin interessant is. Op deze manier kan de archivaris zijn gegevens gewoon intoetsen in invulformulieren, en ze naderhand toch in EAD-formaat gieten.

Deze invulformulieren zijn, in vergelijking met versie 1, eveneens grondig aangepakt, waarbij een aantal beschrijvingselementen die voordien deel uitmaakten van een van de hoofdformulieren, naar de achtergrond zijn gebracht en (actief) door de gebruiker dienen te worden opgeroepen. De bedoeling daarbij was vooral het aantal invoervelden te beperken tot de meest gebruikte. De praktijk zal moeten uitwijzen of we in dat opzet geslaagd zijn.

In de nieuwe versie is de interface ook onderverdeeld in 2 "panes", waarbij links ofwel de zoekmodule, ofwel de archiefboomstructuur kan worden opgeroepen, en rechts de invoerformulieren. Op die manier kan de gebruiker een reeks beschrijvingen doorlopen behandelen op één bepaald kenmerk, bv. een trefwoord of een instellingsnaam. Alle archieffuncties zijn tevens in de twee andere modules op te roepen, zodat de context van documenten nooit hoeft te worden opgenomen, zelfs als ze fysisch uit het archiefbestand worden gelicht en beschreven met meer gespecialiseerde instrumenten (bv. boeken uit een archiefbestand die worden beschreven in cde bibliotheekmodule, en vervolgens terug gekoppeld aan het archiefbestand waaruit ze afkomstig zijn). Op die manier worden de instrumenten zo optimaal mogelijk benut.

Annexe 2 - Syntax plsp-parser

a. URL

http://server name/pls alias/dad/plsp.getplsdoc?rn=number&lan=letter&htdoc=name file

rn = RECNUMM of the record to display. Optional, default = 0. Don't use it if your app is not RECNUMM-based.
lan = display language (for catalography, only N or F). Optional, default = E. For any module of the Pallas-catalography, it should be indicated since there are only 2 languages (French and Dutch).
htdoc = name of pattern document. Should be a valid file. Include the extension in the URL, and indicate the folder containing the file separated from the file by a slash. Required.

For example,
http://hektor/pallas/catalo/plsp.getplsdoc?rn=10513&lan=N&htdoc=photo/phosub2.htm
would call the parsed version of pattern document "phosub2.htm" in Dutch for record 10513.
As I said before, these pattern documents should reside in a folder accessible by the database and declared as such in its startup parameters. On server Hektor, these are

D:\HTM\archives
D:\HTM\atools
D:\HTM\general
D:\HTM\library
D:\HTM\manage
D:\HTM\photo

Please keep the files pertaining to specific modules together.

All substitutions are straightforward and do not depend on their place in a tag hierarchy. That way, indefinite nesting is possible, although it does clutter up the readability of the pattern file.

If a tag is not ended on the same line, the parser wil concatenate it with the next line(s), until every tag is closed. If you were to omit a start or stop tag, the parser would concatenate all the lines of the pattern document and return an error.

b. Tags

1. Dummies

@$Value$@

Exist solely in the pattern document, the parser will remove them completely. It enables you to put some text where it should appear but is actually a tag value (e.g. labels of form fields or other dynamic text).

2. Substitutions

$Value

Straightforward substitutions, for now there are 3;
$lan
$rn
$burl
the first 2 of which correspond with the paramaters given in the URL.
"burl" stands for “base URL” and is a value in the database, for server Hektor obviously “http://hektor/”.
We'll probably need some more, obvious cadidates are
$tn (number keyword, “TRFWNUMM”)
$nn (name number, “NAAMNUMM”)
Let me know if they are needed.

3. Database tags

General syntax:

<!--pls:tag/pls◊

Do not leave out the stop tag, parsing depends on it.
Keep the tags in lowercase, and don't add any unneeded spaces (although the spaces won't kill the parser).

You don't have to mind the types of SELECTed variables, since everything gets converted to VARCHAR2 by the parser, unless your variables are part of a condition: then the usual SQL syntax should be applied.

Tags

selectsingleval or **ssv**

Independent tag, can be used anywhere, and nested in another ssv tag or srv tag. Selects one and only one value, make sure the query doesn't return more than 1!

<!–pls:ssv;table;value;[condition]/pls◊

table : Required. The name of the table. Make sure to include the full schema if there's no public synonym for the table.
value: Required. The name of the field.
condition: Optional. If you omit the condition, the parser assumes that the query is RECNUMM-based, so
<!–pls:ssv;pallas;m245a/pls◊
is equivalent to
<!–pls:ssv;pallas;m245a;recnumm=$rn/pls◊
and, assuming that rn equals 10513, both tags will be translated into the query
SELECT M245A FROM PALLAS WHERE RECNUMM=10513.

If you want to call a function that contains table specification, the table name should be “dual” and no condition is needed or added. Cite the function call literally! Tag
<!–pls:ssv;dual;pub.Get_Browse_Recnumm($rn,'FIRST')/pls–>
will be translated into
SELECT PUB.Get_Browse_Recnumm(10513,'FIRST') FROM DUAL

You can nest a ssv tag anywhere you want into another ssv (or srv) tag, and as many times as you want. For example, the tag
<!–pls:ssv;pallas;m245a/pls◊
is equivalent to
<!–pls:ssv;pallas;<!–pls:ssv;dual;'m245a'/pls◊/pls◊
though not very meaningfull. This option is however important if the values on display are translations of the values in the database. For example, the location of a document (as specification of the call number) is stored in character codes. For now, there are only 2 locations at the SOMA (strongroom and reading room) so, in order to display (in French) that record 10513 (a library description) is stored in the strongroom, the tag should look as follows:

<!–pls:ssv;HULPWAARDEN;TERM$lan;TYPE='M851E' AND WAARDESTR='<!–
pls:ssv;M851;M851E/pls–>'/pls–>

This is in fact a very tricky construction since the subtag could return several values and the whole tag should be part of a repeater construction whith the second tag of type srv, but in this case it will work. The subtag first gets translated into
SELECT M851E FROM M851 WHERE RECNUMM=10513
and returns the value “MZ” which, after substitution, turns the whole tag into
<!–pls:ssv;HULPWAARDEN;TERM$lan;TYPE='M851E' AND WAARDESTR='MZ'/pls–>
This tag gets translated into
SELECT TERMF FROM HULPWAARDEN WHERE TYPE='M851E' AND WAARDESTR='MZ'
which will translate into “Magasins”.

startrepeat

This is an indicative tag: it gets replaced by nothing (is removed by the parser) but indicates where a repetition of elements should start. It is of course most usefull in a one-to-many relation between tables. Repeat only the elements that need repeating, for example

```
<table>
  <!–pls:startrepeat;…/pls◊
  <tr>
     <td>…</td>
   </tr>
   <!–pls:stoprepeat/pls◊
</table>
```

This tag HAS to be placed on a separate line!
The values of the query are to be found inside the 2 tags in the srv tags, but ssv tags are allowed within the indicative tags.

<!–pls:startrepeat;tables;[condition;order]/pls◊

tables: required. Separate more than 1 table with comma's.
condition: optional if no order is specified, otherwise required (even when the query is RECNUMM based!). When no condition is given, the parser assumes the query is RECNUMM based.
order: optional. Only give the value(s) on which the query should be sorted.
If you want to sort the query but haven't got a condition (e.g. selecting all values of a given table), add a dummy condition. I suggest "1=1".

stoprepeat

Goes with startrepeat and indicates the end of the repetition. No parameters.
<!–pls:stoprepeat/pls◊
This tag HAS to be placed on a separate line!

selectrepeatedval or srv

Only allowed between a startrepeat and a stoprepeat tag : using this tag outside the indicative tags will always result in an error. The SELECT statement is constructed on the startrepeat tag combined with all the srv tags within the repeat indication.

<!–pls:srv;table;value/pls◊

table: Required. Name of the table to select from.
value: Required. Value to be selected.

Any srv tag can contain one or more ssv tags and vice versa. In fact, the example above should be rewritten as follows:

```
<!–pls:startrepeat;M851/pls–>
…
<!–pls:ssv;HULPWAARDEN;TERM$lan;TYPE='M851E' AND WAARDESTR='<!–
pls:srv;M851;M851E/pls–>'/pls–>
…
<!–pls:stoprepeat/pls–>
```

The SELECT statement of the repeated values will look like this:
SELECT M851.M851E FROM M851 WHERE RECNUMM=10513
and will translate the subtag in the ssv tag into the value “MZ”.

include

Use this tag to include another document into the main document. Usefull for repetitions of html code in several pattern files, Java scripts or other code, wich can be stored into separate files and repeated in any pattern file over and over. The included file gets parsed too, so it can contain any of the other tags.

<!–pls:include;name of document/pls◊

Quote the entire name, including the extension. The file should reside in the default directory on the server!

exec

In all cases where the tags won't do the trick, we'll have to write procedures line per line. This tag calls such a procedure, and simply executes it. Remember, this procedure has to produce html text through the Oracle htp package.

<!–pls:exec;name of procedure/pls◊

Quote the procedure exactly as you would in PL/SQL code. We should try to avoid using this tag whenever possible, because the final document will always display data not contained in the pattern document.

4. Traps

Since all dynamic values have to be selected (unless they only contain the straughtforward $-substitutions), all values have to be contained in the database. I've already created a new table, APPLABELS, which contains a whole bunch of expressions already used previously. The table has a trigger on INSERT, which automatically insert a unique value for NUM_KEY, but you can also use char-keys to find an expression.

Html syntax is not always very logical. A nice example is a dropdown box, called a “select” in html (as part of a form element). General syntax looks like this:
<select name=”name”>
<option value=”value”>displayed value
<option value=”value“ selected> displayed value
</select>
"selected" is an attribute of an option, which is silly of course, it should have been an attribute of the “select” element. But that's the way it is, so we have to adapt. I wrote a small function called “compare” which simply compares 2 values and returns a specified string if they are equal, otherwise it returns NULL:
compare(value1,value2,’returnstring’)

In order to construct such an option list, we need 4 tags:
1 with the value
1 with the displayed value
1 with the value actually contained in the database for a specific record
1 with the value to compare it with the latter.

It works (and doesn't degrade perfomance), but I have to admit it doesn't make the pattern text very readable. Let's take the following example :

All library descriptions are specified with a number flag indicating their general type (monography, serial, etc.). The flags get translated on display into their verbal expression in a given language through the table HULPWAARDEN (for now). A dropdown box displaying the verbal expressions and containing the flag values, selecting the appropriate value for this record, looks like this:

```
 <select name="select">
<!–pls:startrepeat;HULPWAARDEN;TYPE = 'BIBTYPESUB';TERM$lan/pls–>
<option value="<!–pls:srv;HULPWAARDEN;WAARDE/pls–>"
<!–pls:ssv;dual;compare('<!–pls:ssv;pallas;btypesub/pls–>','<!–pls:srv;HULPWAARDEN;WAARDE/pls–
>',' selected')/pls–>>
<!–pls:srv;HULPWAARDEN;TERM$lan/pls–>
<!–pls:stoprepeat/pls–>
</select>
```

Assuming that the call is in Dutch and the recnumm 10513, the SELECT of the repeater looks like this:
SELECT HULPWAARDEN.WAARDE,HULPWAARDEN.TERMN FROM HULPWAARDEN WHERE TYPE = 'BIBTYPESUB' ORDER BY TERMN
In the database, this will return the following cursor:
9;Koepel
1;Monografie
2;Monografie, meerdeling
3;Periodiek
4;Periodiek, clandestien
5;Persknipsel
6;Persknipselmap
7;Reeks
8;Verhandeling, onuitgegeven

The first subtag in the ssv tag will translate into
SELECT BTYPESUB FROM PALLAS WHERE RECNUMM=10513
and return the value "2" (for each repetition). If we take the second repetition and substitute the value, the block between the repeater tags looks like this :

```
<option value="<!–pls:srv;HULPWAARDEN;WAARDE/pls–>"
<!–pls:ssv;dual;compare('2','<!–pls:srv;HULPWAARDEN;WAARDE/pls–>',' selected')/pls–>>
<!–pls:srv;HULPWAARDEN;TERM$lan/pls–>
```

The value of the second subtag is taken from the repeater query and will return "2" for monographies (the second repeat). The block now looks like this:

```
<option value="<!–pls:srv;HULPWAARDEN;WAARDE/pls–>"
<!–pls:ssv;dual;compare('2','2',' selected')/pls–>>
<!–pls:srv;HULPWAARDEN;TERM$lan/pls–>
```

The compare function will return " selected" if both values are "2", in all other cases it returns NULL. After substitution, the block is

```
<option value="<!–pls:srv;HULPWAARDEN;WAARDE/pls–>"
 selected>
<!–pls:srv;HULPWAARDEN;TERM$lan/pls–>
```

and if we substitue the repeated values with those from the cursor, we end up with

```
<option value="1"
 selected>
Monografie
```

After all repetitions, the html will look like this:

```
<select name="select">
  <option value="9"
  >
  Koepel
  <option value="1"
   selected>
  Monografie
  <option value="2"
  >
  Monografie, meerdeling
  <option value="3"
  >
  Periodiek
  <option value="4"
  >
  Periodiek, clandestien
  <option value="5"
  >
  Persknipsel
  <option value="6"
  >
  Persknipselmap
  <option value="7"
  >
  Reeks
  <option value="8"
  >
  Verhandeling, onuitgegeven
</select>
```

Which is exactly what we need, but, again, the tagged html code is pretty cluttered.