

RIPPLL Project: technical report from NTU

Francis Lowry, March 2006

Overview

This report covers the technical work undertaken as Nottingham Trent's contribution to RIPPLL, the JISC-funded DeL Regional Pilot project led by the University of Nottingham. It describes the approach taken and the outcomes of the development work as part of the main project. A further document will report on the Shibboleth element of the project carried out by Nottingham Trent University (NTU), which is investigating use of Shibboleth as a means to accessing PDP-related data in a distributed environment.

The core project's technical aim was to investigate the use of the UK LeaP XML specification for learner data as a means to describe and model PDP data for use in transferring information from one educational institution to another, and on to an employer. This has been described as the 'pass the parcel' method, whereby all the learner's data is moved from institution to institution at each point of transition. (The NTU Shibboleth work has a slightly different perspective: it uses a distributed model where sections of the learner's data remain at originating institutions and are referenced dynamically on request.)

Background

NTU currently have an in-house developed Virtual Learning Portal (VLP) which all students can access. This has been developed using Microsoft technologies on top of MS Exchange. PDP data collection at NTU has evolved organically over several years, using the VLP as the main mechanism for accessing and storing this data. The main focus of the electronic PDP system at NTU has to been to provide mechanisms to allow the learner to record and manage their own data, not to manage it on their behalf.

There is a subtle distinction here; putting the onus on the learners to carry forward their own data from institution to institution has led to the creation of mechanisms to store the final data output in descriptive textual output in a set of Microsoft Word documents. Learners are then responsible for taking copies of this documentation with them when they leave.

Investigating the use of the UK LeaP standard and researching the possibilities of using Shibboleth as a mechanism to access historic/reference PDP data has encouraged a change in NTU's perspective. Rather than relying on the learner to copy and transfer his or her own PDP data, the automatic transfer or referencing of data on behalf of the learner – with the learner retaining control of the data – are now becoming a practical possibility.

Current PDP facilities at NTU

There are currently four distinct areas used for data collection/storage of PDP-related data within the VLP:

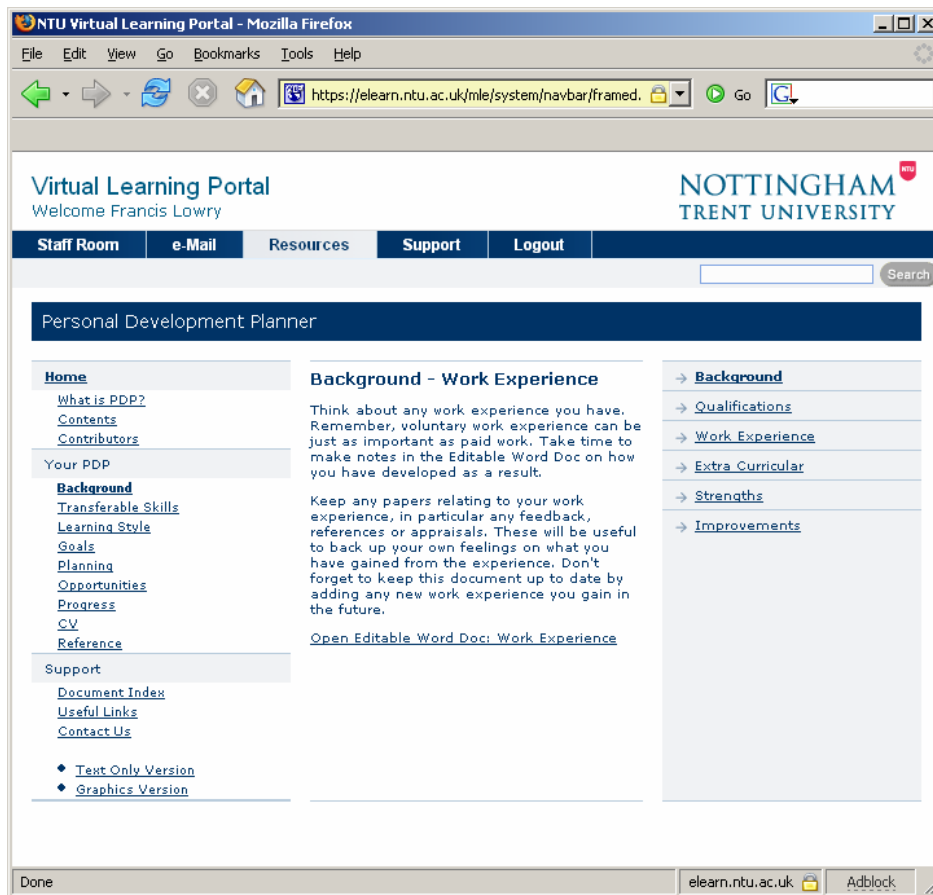
- Background data
- Skills
- CV Builder
- References

There are other sections which provide information and guidance to the learner, but do not provide facilities for data collection. These are:

- Learning Style
- Goals
- Planning
- Opportunities
- Progress

In addition to the facilities provided by the VLP, the NTU careers service also provide an on-line skills assessment web site.

Within the RIPPLL project, NTU have concentrated on using the UKLeaP specification to enable the transfer of a small section of PDP data. In addition, work has been done to replace some of the existing MS Word-based PDP record systems with more web form and database-driven storage. The aim is to provide NTU with a stronger foundation for data collection with will enable, in time, the generation of a UKLeaP-formatted XML document which can record all the data on behalf of a learner in addition to retaining the existing facilities for generating Word documents.



Transitions

Initial transition

The use of the UKLeaP specification and XML formatted data was completely new to NTU technical staff. Part of the initial development work for the project was to familiarise ourselves with the specification and technology. As part of this process we decided to perform an initial data transfer which would be a functional replication of work from an earlier project to transfer data from the Nottingham Passport to the University of Nottingham ePARs system. We used source code based on that used by the University of Nottingham to transfer basic identity data, generated from the Nottingham Passport and formatted into the UKLeaP format, into the NTU system.

Initial transition mapping

Our initial assumption was that the data could be translated from the UK LeaP format and inserted directly into the NTU PDP databases. As sections of the PDP areas within the NTU VLP were under review, the data was transferred into a set of parallel database tables which closely matched the structure of our student administration system (SCT

Banner 2000). The assumption was that we could then map the data directly to the internal systems once modelling had been completed.

The table below describes the data mapping from the UK LeaP data generated from the Nottingham Passport, through XSLT mapping to a more database normalised structure, to the final destination tables in the SQL Server database.

<i>UKLeap (source data)</i>	<i>Interim mapping</i>	<i>Destination</i>	<i>Destination field</i>
	<i>Generated</i>	person	person_key
/learnerinformation/identification/name/partname [typename/tyvalue = 'StudentID']/text"	person/line/person_key	person	person_id
/learnerinformation/identification/name/partname [typename/tyvalue = 'Given']/text"	person/line/forename	person	forename
/learnerinformation/identification/name/partname [typename/tyvalue = 'Surname']/text"	person/line/surname	person	surname
/learnerinformation/identification/demographics/date[typename/tyvalue = 'Birth']/datetime"	person/line/birth_date	person	date_of_birth
		address_details	person_key
	line/address_type	address_details	address_type
//identification/address/street/streetnumber	line/housenumber	address_details	housenumber
//identification/address/street/streetname	line/streetname	address_details	streetname
//identification/address/city	line/city	address_details	city
//identification/address/region	line/region	address_details	region
//identification/address/postcode	line/postcode	address_details	postcode
		email_address	person_key
	line/email_address_type	email_address	email_address_type
	line/email_seq_no	email_address	email_seq_no
	line/email_primary_ind	email_address	email_primary_ind
//identification/contactinfo/email	line/e_mail_address	email_address	email_address
		contact_numbers	
		contact_numbers	person_key
	line/contact_number_type	contact_numbers	contact_number_type
	line/contact_seq_no	contact_numbers	contact_seq_no
	line/contact_primary_ind	contact_numbers	contact_primary_ind
//identification/contactinfo/telephone/areacode	line/area_code	contact_numbers	area_code
//identification/contactinfo/telephone/indnumber	line/telephone_no	contact_numbers	telephone_no

The source data was transformed via a very basic XSL transformation, which effectively transformed the UK LeaP XML into a data mapping ready for SQL inserts into a normalised data structure.

```

<?xml version="1.0"?>
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" indent="yes" />

<xsl:variable name="apos">'</xsl:variable>

<xsl:template match="/">
  <data>
    <xsl:call-template name="person" />
    <xsl:call-template name="address_details" />
    <xsl:call-template name="email_address" />
    <xsl:call-template name="contact_numbers" />
  </data>
</xsl:template>

```

```

<xsl:template name="person">
  <person>
    <line>
      <person_key>
        <xsl:text>1001</xsl:text>
      </person_key>
      <forename>
        <xsl:call-template name="treatString">
          <xsl:with-param name="string">
            <xsl:value-of
select="/learnerinformation/identification/name/partname[typename/tyvalue =
'Given']/text" />
          </xsl:with-param>
        </xsl:call-template>
      </forename>
      <surname>
        <xsl:call-template name="treatString">
          <xsl:with-param name="string">
            <xsl:value-of
select="/learnerinformation/identification/name/partname[typename/tyvalue =
'Surname']/text" />
          </xsl:with-param>
        </xsl:call-template>
      </surname>
      <birth_date>
        <xsl:call-template name="treatString">
          <xsl:with-param name="string">
            <xsl:value-of
select="/learnerinformation/identification/demographics/date[typename/tyvalue =
'Birth']/datetime" />
          </xsl:with-param>
        </xsl:call-template>
      </birth_date>
    </line>
  </person>
</xsl:template>
<xsl:template name="address_details">
  <address_details>
    <line>
      <person_key>
        <xsl:text>1001</xsl:text>
      </person_key>
      <address_type>
        <xsl:text>HOME</xsl:text>
      </address_type>
      <houenumber>
        <xsl:call-template name="treatString">
          <xsl:with-param name="string">
            <xsl:value-of
select="/learnerinformation/identification/address/street/streetnumber" />
          </xsl:with-param>
        </xsl:call-template>
      </houenumber>
      <streetname>
        <xsl:call-template name="treatString">
          <xsl:with-param name="string">
            <xsl:value-of
select="/learnerinformation/identification/address/street/streetname" />
          </xsl:with-param>
        </xsl:call-template>
      </streetname>
      <city>
        <xsl:call-template name="treatString">
          <xsl:with-param name="string">
            <xsl:value-of
select="/learnerinformation/identification/address/city" />
          </xsl:with-param>
        </xsl:call-template>
      </city>
      <region>
        <xsl:call-template name="treatString">
          <xsl:with-param name="string">
            <xsl:value-of
select="/learnerinformation/identification/address/region" />
          </xsl:with-param>
        </xsl:call-template>
      </region>
      <postcode>

```

```

        <xsl:call-template name="treatString">
            <xsl:with-param name="string">
                <xsl:value-of
select="/learnerinformation/identification/address/postcode" />
                </xsl:with-param>
            </xsl:call-template>
        </postcode>
    </line>
</address_details>
</xsl:template>
<xsl:template name="email_address">
    <email_address>
        <line>
            <person_key>
                <xsl:text>1001</xsl:text>
            </person_key>
            <email_address_type>
                <xsl:text>HOME</xsl:text>
            </email_address_type>
            <email_seq_no>
                <xsl:text>1</xsl:text>
            </email_seq_no>
            <email_primary_ind>
                <xsl:text>Yes</xsl:text>
            </email_primary_ind>
            <e_mail_address>
                <xsl:call-template name="treatString">
                    <xsl:with-param name="string">
                        <xsl:value-of
select="/learnerinformation/identification/contactinfo/email" />
                        </xsl:with-param>
                    </xsl:call-template>
                </e_mail_address>
            </line>
        </email_address>
    </xsl:template>
<xsl:template name="contact_numbers">
    <contact_numbers>
        <line>
            <person_key>
                <xsl:text>1001</xsl:text>
            </person_key>
            <contact_number_type>
                <xsl:text>HOME</xsl:text>
            </contact_number_type>
            <contact_seq_no>
                <xsl:text>1</xsl:text>
            </contact_seq_no>
            <contact_primary_ind>
                <xsl:text>Yes</xsl:text>
            </contact_primary_ind>
            <area_code>
                <xsl:call-template name="treatString">
                    <xsl:with-param name="string">
                        <xsl:value-of
select="/learnerinformation/identification/contactinfo/telephone/areacode" />
                        </xsl:with-param>
                    </xsl:call-template>
                </area_code>
            <telephone_no>
                <xsl:call-template name="treatString">
                    <xsl:with-param name="string">
                        <xsl:value-of
select="/learnerinformation/identification/contactinfo/telephone/indnumber" />
                        </xsl:with-param>
                    </xsl:call-template>
                </telephone_no>
            </line>
        </contact_numbers>
    </xsl:template>
<xsl:template match="*" />
<xsl:template name="treatString">
    <xsl:param name="string" />
    <xsl:choose>
        <xsl:when test="contains($string, $apos)">
            <xsl:value-of select="substring-before($string, $apos)" />
            <xsl:call-template name="treatString">
                <xsl:with-param name="string">
                    <xsl:value-of select="substring-after($string, $apos)" />
                </xsl:with-param>
            </xsl:call-template>
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="$string" />
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>

```

```

        </xsl:with-param>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="$string" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
</xsl:transform>

```

Running this initial data transfer resulted in several critical findings:

1. We could not guarantee that the student ID (manually added through the mapping in this case) would be unique, i.e. that we could insert the data directly.
2. The vocabularies used to describe certain attributes would not match our local systems. We would need to ensure that there was an additional mapping which would need to be maintained.
3. Although the UK LeaP specification is broad enough to describe the majority of PDP-related information reasonably well, unless the local transforms/mapping have been designed to cater for all eventualities, there is still a requirement that the destination/recipient has details of the source's implementation of the UKLeaP, i.e. which aspects have been used in the mappings.

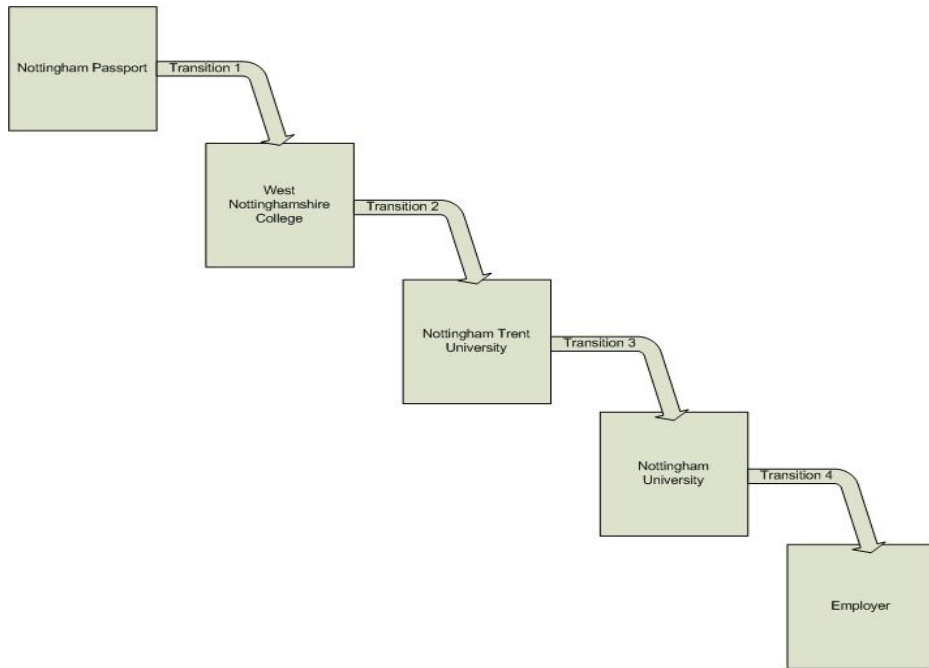
If we maintained the assumption that this data would be transferred directly into the destination system, these issues would be fairly problematic: the XSLT would have to be reasonably complex to allow for all possible data combinations, and would have to be constantly reviewed and maintained. We therefore made the decision that any PDP-related import data would always go to an interim set of relational tables, and this relation data would then be post-processed to allow a more flexible transfer of the data.

This decision allowed NTU to keep the XSLT fairly simple, and the subsequent manipulation of the PDP data to be done at the database level, where NTU's main development strengths lay.

For the third point above, the only solution was to ensure that a specific XSL transform is maintained for each separate source of PDP data and that a working relationship is maintained between the source and recipient to ensure that this mapping is maintained. The only other option would be to create a complete mapping of all possible UKLeaP attributes and store them all – given the breadth of the specification, this is not a practical option for NTU.

Final transfers

Following our initial test transfer, the project team realised that it could be possible to create a demonstration of the movement of a single learner's PDP data across several different transitions. The example here is a simple progressive one, with a natural progression through several levels of learning from KS4, through FE and HE into employment. (Although the steps outlined here are sequential, in a real example of lifelong learning they would be more cyclic with some transitions being repeated.)



- Transition 1: Learner leaves Secondary School to progress to an FE College
- Transition 2: Learner leaves FE College to progress to an Undergraduate degree at University.
- Transition 3: Learner leaves one University to progress to a postgraduate course at another
- Transition 4: Learner progresses to employment.

We decided that sections of the main data would change in each transition. Across each of these stages, the learner’s portfolio gets enhanced, old data is discarded and new data is recorded. In this example, the data physically moves with the learner (‘pass the parcel’ model).

Mappings

With the first set of data being transferred, the mapping was very simple. It assumed that each occurrence of the data elements would be single valued. (We appreciate that this may not be the case in real life.) Apart from some of the core identity data, the majority of elements in the UK LeaP specification will be repeatable values, and the transform/transfer needs to take this into consideration. A very simple example of this is addresses. A learner at a HE institution who is living away from home will have at least two addresses on file: home address and term address. Both the XSLT and the load mechanism therefore needed to be changed to cater for repeating sections of data. To test the transfer further, the data set was extended to include a Personal Statement.

<i>UKLeap (source data)</i>	<i>Interim mapping</i>	<i>Destination</i>	<i>Destination field</i>
		person	person_key
/learnerinformation/identification/name/partname [typename/tyvalue = 'StudentID']/text"	person/line/person_key	person	person_id
/learnerinformation/identification/name/partname [typename/tyvalue = 'Given']/text"	person/line/forename	person	forename
/learnerinformation/identification/name/partname [typename/tyvalue = 'Surname']/text"	person/line/surname	person	surname
/learnerinformation/identification/demographics/date[typename/tyvalue = 'Birth']/datetime"	person/line/birth_date	person	date_of_birth

		address_details	person_key
//identification/address/typename/tyvalue	line/address_type	address_details	address_type
//identification/address/street/streetnumber	line/housenumber	address_details	housenumber
//identification/address/street/streetname	line/streetname	address_details	streetname
//identification/address/city	line/city	address_details	city
//identification/address/region	line/region	address_details	region
//identification/address/postcode	line/postcode	address_details	postcode
		email_address	person_key
//identification/contactinfo/typename/tyvalue	line/email_address_type	email_address	email_address_type
	line/email_seq_no	email_address	email_seq_no
	line/email_primary_ind	email_address	email_primary_ind
//identification/contactinfo/email	line/e_mail_address	email_address	email_address
		contact_numbers	
		contact_numbers	person_key
//identification/contactinfo/typename/tyvalue	line/contact_number_type	contact_numbers	contact_number_type
	line/contact_seq_no	contact_numbers	contact_seq_no
	line/contact_primary_ind	contact_numbers	contact_primary_ind
//identification/contactinfo/telephone/areacode	line/area_code	contact_numbers	area_code
//identification/contactinfo/telephone/indnumber	line/telephone_no	contact_numbers	telephone_no
		personal_statement	person_key
//reflexion/typename/tyvalue	line/personal_statement_type	personal_statement	statement_type
//reflexion/date[typename/tyvalue = 'Create']/datetime"	line/statement_date	personal_statement	statement_date
//reflexion/description/short	line/statement_title	personal_statement	statement_title
//reflexion/description/long	line/statement	personal_statement	statement

With the transition from FE to NTU the structure of the data file did not change (i.e. the single valued approach taken for the initial transfer would have worked as there was no repeating blocks of data), however it would not work for all the transitions.

The development was changed so that it was more generic and could be applied to all transitions in the model scenario.

XSLT changes from the initial transfer

With the exception of the main identity block, all other sections of the PDP data were deemed to have the option to be repeating blocks. This implied the conversion of the initial XSLT from a single-valued approach to a repeating block model. For example, in the original block, the address details was coded:

```
<xsl:template name="address_details">
  <address_details>
    <line>
.....
    </line>
  </address_details>
</xsl:template>
```

In the new block this is coded:

```
<xsl:for-each select="//identification/address">
  <address_details>
    <line>
.....   Loop and process each address
    </line>
  </address_details>
</xsl:for-each>
</xsl:template>
```

A simple change, but crucial to allow subsequent conversion of the data within the ASP load pages.

Similarly, in the initial XSLT the email address and the telephone numbers were mapped as explicit sections of the //identification/contactinfo node. This worked as the data was not multi-valued. However, in real life, there will be multiple blocks of these sections.

These two are now treated as a conditional elements accessing the same data, e.g.

```
<xsl:for-each select="//identification/contactinfo">
<xsl:if test="email">
  <email_address>
    <line>
.....   Conditionally process an email address
    </e_mail_address>
    </line>
  </email_address>
</xsl:if>
</xsl:for-each>
</xsl:template>
<xsl:template name="contact_numbers">
<xsl:for-each select="//identification/contactinfo">
<xsl:if test="telephone">
  <contact_numbers>
    <line>
.....   Conditionally process a telephone number
    </line>
  </contact_numbers>
</xsl:if>
</xsl:for-each>
</xsl:template>
```

This now left the XSLT correctly parsing the repeating blocks and creating the data XML for loading. The load script needed to be enhanced to cater for these repeating blocks.

Load mechanism changes

The initial load script (.asp page) was designed along the same lines as the XSLT, i.e. there was only one record and the data did not repeat. This needed to change to allow the inserting of the repeating blocks of data.

In addition to this, with the decision that this data would only be loaded into an interim relational database load area, a mechanism was required to generate a unique identifying attribute for each learner record processed.

A very simple procedure was created to generate a new unique key for each learner load (GET_PERSON_KEY). This was called at the start of the .asp load page to generate the unique key for the person record.

```
<%
' --- Database Connection ---
Set conn = Server.CreateObject("ADODB.Connection")
conn.CursorLocation = 3
conn.open "xxxx", "xxxx", "xxxx"

'--- Load XML - Data from Nottingham Passport ---
set xml = Server.CreateObject("Msxml2.DOMDocument.4.0")
xml.load(Server.MapPath("Transitions/nottingham_passport.xml"))
```

```

'--- Load XSL stylesheet to transform data to UKLeaP format ---
set xsl = Server.CreateObject("Msxml2.DOMDocument.4.0")

xsl.load(Server.MapPath("PDPXMLTransform.xslt"))

' --- Parsing the XML ---

xml.loadXML(xml.transformNode(xsl))

'response.write xml.documentElement.xml

'set objRoot = xml.documentElement

set objRoot = xml.documentElement           'Set the root of the XML object

set objChannel = objRoot.selectSingleNode("//data")
set objAddress = objChannel.getElementsByTagName("address_details")
set objEmail = objChannel.getElementsByTagName("email_address")
set objContact = objChannel.getElementsByTagName("contact_numbers")
set objPStatement = objChannel.getElementsByTagName("personal_statement")

v_StudentKey      = objRoot.selectSingleNode("person/line/person_key").text
v_Forename        = objRoot.selectSingleNode("person/line/forename").text
v_Surname         = objRoot.selectSingleNode("person/line/surname").text
v_BirthDate       = objRoot.selectSingleNode("person/line/birth_date").text

if (isnull(v_StudentKey) or len(v_studentkey) = 0 ) then
    v_StudentKey = "NONE"
end if

' --- get Person Key field
sql = "exec get_person_key " & v_studentKey
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3

new_stud_key = rs("NewKey")

```

There are several other differences here as well. In our initial transfer each node and element were explicitly referenced at the top of the load, prior to the individual insert statements. Only the core identity block is specifically referenced. All the other repeatable blocks of code are now created as XMLObjects and are referenced by looping through their contents later.

```

' --- INSERT Record ---
sql = "INSERT INTO person(person_key,person_id, forename,surname,date_of_birth)"
sql = sql & " VALUES('" & new_stud_key & "', '" & v_StudentKey & "', '" & v_Forename &
"', '" & v_Surname & "', '" & v_BirthDate & "'"")"
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3

```

The insert for the core identity record is relatively unchanged: the only exception now is the pre-generated unique person_key.

The code to cause the blocks to do the repeat inserts is relatively straightforward and is replicated throughout the rest of the code:

Address Details

```

For Each objChild in objAddress

v_AddressType      = objChild.selectSingleNode("line/address_type").text
v_StreetNumber     = objChild.selectSingleNode("line/housenumber").text
v_StreetName       = objChild.selectSingleNode("line/streetname").text
v_City             = objChild.selectSingleNode("line/city").text
v_Region           = objChild.selectSingleNode("line/region").text
v_PostCode         = objChild.selectSingleNode("line/postcode").text

sql = "INSERT INTO address_details(person_key,address_type,
housenumber,streetname,city,region,postcode)"
sql = sql & " VALUES('" & new_stud_key & "', '" & v_AddressType & "', '" &
v_streetNumber & "', '" & v_streetname & "', '" & v_city & "', "
sql = sql & "'" & v_region & "', '" & v_postcode & "'"")"
Set rs = Server.CreateObject("ADODB.Recordset")

```

```
rs.Open sql, conn, 3, 3
next
```

For each of these blocks, the *new_stud_key* generated at the start of the script is inserted into each repeating block to ensure that the final records are matched.

Email

```
For Each objChild in objEmail

v_EmailType           = objChild.selectSingleNode("line/email_address_type").text
v_EmailSeqNo          = objChild.selectSingleNode("line/email_seq_no").text
v_EmailPrimaryInd     = objChild.selectSingleNode("line/email_primary_ind").text
v_Email               = objChild.selectSingleNode("line/e_mail_address").text

sql = "INSERT INTO email_address(person_key, email_address_type, email_seq_no,
email_primary_ind, email_address)"
sql = sql & " VALUES('" & new_stud_key & "', '" & v_EmailType & "', '" & v_EmailSeqNo
& "', '" & v_EmailPrimaryInd & "', '" & v_Email & "'"
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3

next
```

Telephone Numbers

```
For Each objChild in objContact

v_ContactNoType       =
objChild.selectSingleNode("line/contact_number_type").text
v_ContactSeq          = objChild.selectSingleNode("line/contact_seq_no").text
v_ContactPrimaryInd   = objChild.selectSingleNode("line/contact_primary_ind").text
v_ContactAreaCode     = objChild.selectSingleNode("line/area_code").text
v_ContactNumber       = objChild.selectSingleNode("line/telephone_no").text

sql = "INSERT INTO contact_numbers(person_key, contact_number_type, contact_seq_no,
contact_primary_ind, area_code, telephone_no)"
sql = sql & " VALUES('" & new_stud_key & "', '" & v_ContactNoType & "', '" &
v_ContactSeq & "', '" & v_ContactPrimaryInd & "', '" & v_ContactAreaCode
sql = sql & "', '" & v_ContactNumber & "'"
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3

next
```

Where in the source XML the email details and telephone numbers are grouped within the contact node, the XSLT conditional processing has separated these into their separate components – again, this is modelled on the NTU student system's internal representation of the data.

Personal Statement

```
For Each objChild in objPStatement

v_PStatementType      = objChild.selectSingleNode("line/personal_statement_type").text
v_PStatementDate      = objChild.selectSingleNode("line/statement_date").text
v_PStatementTitle     = objChild.selectSingleNode("line/statement_title").text
v_PStatementLong      = objChild.selectSingleNode("line/statement").text

sql = "INSERT INTO personal_statement(person_key, statement_type, statement_date,
statement_title, statement )"
sql = sql & " VALUES('" & new_stud_key & "', '" & v_PStatementType & "', '" &
v_PStatementDate & "', '" & v_PStatementTitle & "', '" & v_PStatementLong & "'"
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3

next
```

The final addition to the loads is to insert a basic personal statement. Again the logic here is very simple: it is just treated as another repeating block and inserted into the holding tables in the database.

Final transfer output

A final refinement to the load mechanism was to add some more enhanced feedback to the load script. Rather than query the database and screenshot the output, the data just inserted was re-queried using the *new_stud_key* and very simple tables were generated by the .asp web page. This enabled the quick testing and verification of the load process against all the transition XML,

e.g.

```

sql = "Select person_key, person_id, forename, surname, date_of_birth from person
where person_key = " & new_stud_key
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3
%>
<h3>Person Details</h3>
<table width="90%" border="1">
  <tr>
    <td>Unique ID</td>
    <td>Student ID</td>
    <td>Forename</td>
    <td>Surname</td>
    <td>Date of Birth</td>
  </tr>

  <%
if not rs.EOF then
DO WHILE NOT rs.EOF
%>
    <tr>
      <td><%=rs("person_key")%></td>
      <td><%=rs("person_id")%></td>
      <td><%=rs("forename")%></td>
      <td><%=rs("surname")%></td>
      <td><%=rs("date_of_birth")%></td>
    </tr>
  <%
rs.movenext
loop
end if
%>
</table>
<%
sql = "Select person_key, address_type, housenumber,streetname,city,region,postcode
from address_details where person_key = " & new_stud_key
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3
%>
<h3>Address Details</h3>
<table width="90%" border="1">
  <tr>
    <td>Unique ID</td>
    <td>Address Type</td>
    <td>House Number</td>
    <td>Street Name</td>
    <td>City</td>
    <td>Region</td>
    <td>Postcode</td>
  </tr>

  <%
if not rs.EOF then
DO WHILE NOT rs.EOF
%>
    <tr>
      <td><%=rs("person_key")%></td>
      <td><%=rs("address_type")%></td>
      <td><%=rs("housenumber")%></td>
      <td><%=rs("streetname")%></td>
      <td><%=rs("city")%></td>
      <td><%=rs("region")%></td>
      <td><%=rs("postcode")%></td>
    </tr>
  <%
rs.movenext
loop
end if

```

%>

</table>

The screenshot below shows completed output for the transition data from West Nottinghamshire College to NTU:

The screenshot shows a Mozilla Firefox browser window displaying a web page with the following sections:

Person Details

Unique ID	Student ID	Forename	Surname	Date of Birth
406	NONE	fred	blogs	09/10/1983

Address Details

Unique ID	Address Type	House Number	Street Name	City	Region	Postcode
406	Private	50	Somewhere St	Somewhere	Somewhereshire	SO12 3ME

Email Address

Unique ID	Email Type	Sequence No	Primary Ind	Email
406	Private	1	Yes	fredblogs1@gmail.com

Contact Numbers

Unique ID	Contact Type	Sequence No	Primary Ind	Area Code	Telephone
406	Private	1	Yes	1234	567890

Personal Statement

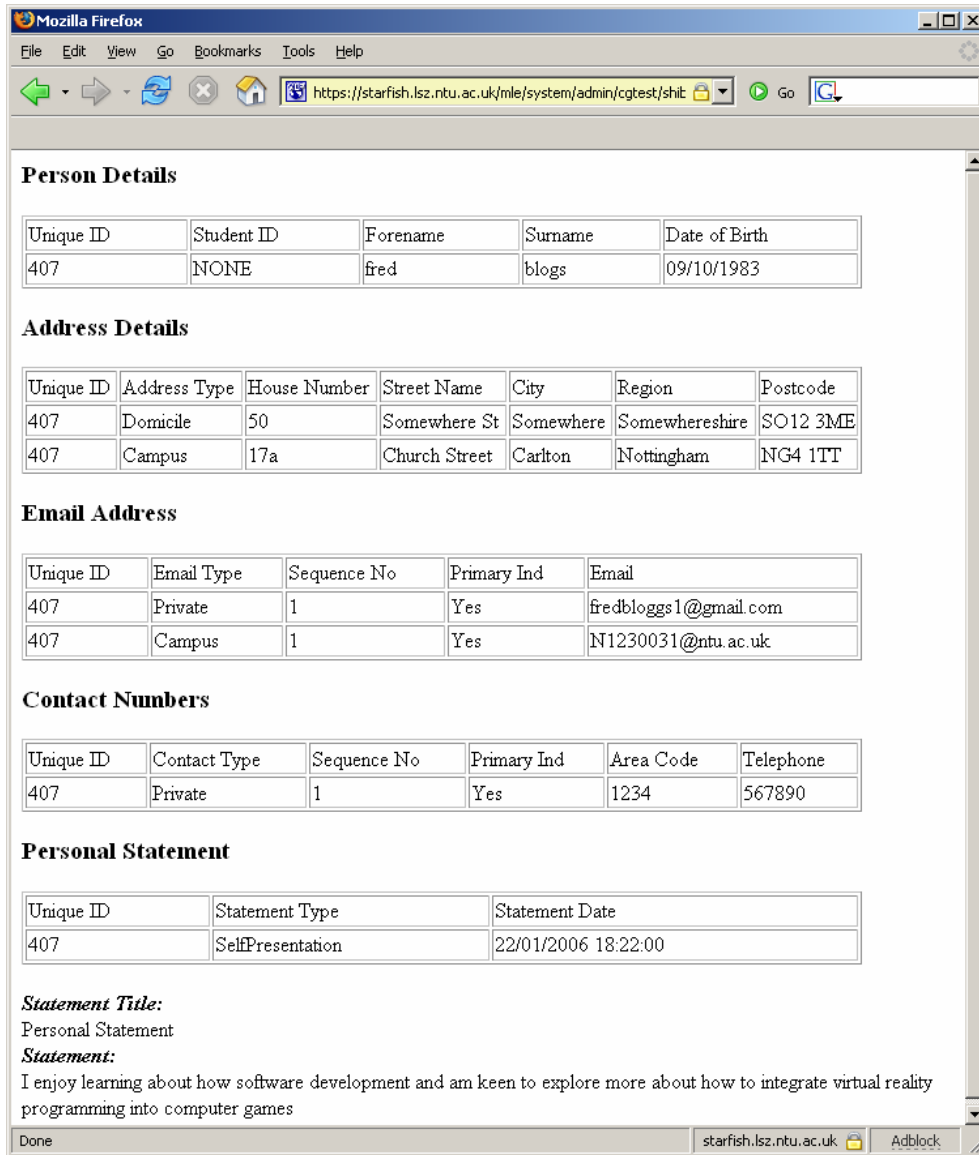
Unique ID	Statement Type	Statement Date
406	SelfPresentation	11/12/2005 09:15:00

Statement Title:
Personal Statement

Statement:
I am a great team player, but I am happy working on my own. I wish to extend my knowledge of operating systems and database development.

The browser status bar at the bottom shows "Done" and "starfish.lsz.ntu.ac.uk" with an "Adblock" extension icon.

The next transitions, i.e. output from NTU transferred to the University of Nottingham, demonstrate more fully the repeating blocks and are a better display example:



Here the learner has two different addresses and two contact email addresses.

The full listings of the XML, XSLT and the .ASP to generate the imports and web pages are included in appendices 1, 2 & 3.

NTU PDP mapping

Although we have now remodelled a substantial part of the NTU PDP system, a section remains as a set of MS Word documents to be completed by the learner. Our initial intention was to replicate the functionality that exists in the Word documents with only minimal changes in the structure. However, on starting to test imports and exports using the UK LeaP specification, it immediately became apparent that either the UK LeaP specification did not match our requirements (which was not the case) or we needed to take a few steps backwards to re-think how we wanted to record, store and manage this data on behalf of the learner. We will now be looking at this area again in a wider context with other component areas of the NTU PDP.

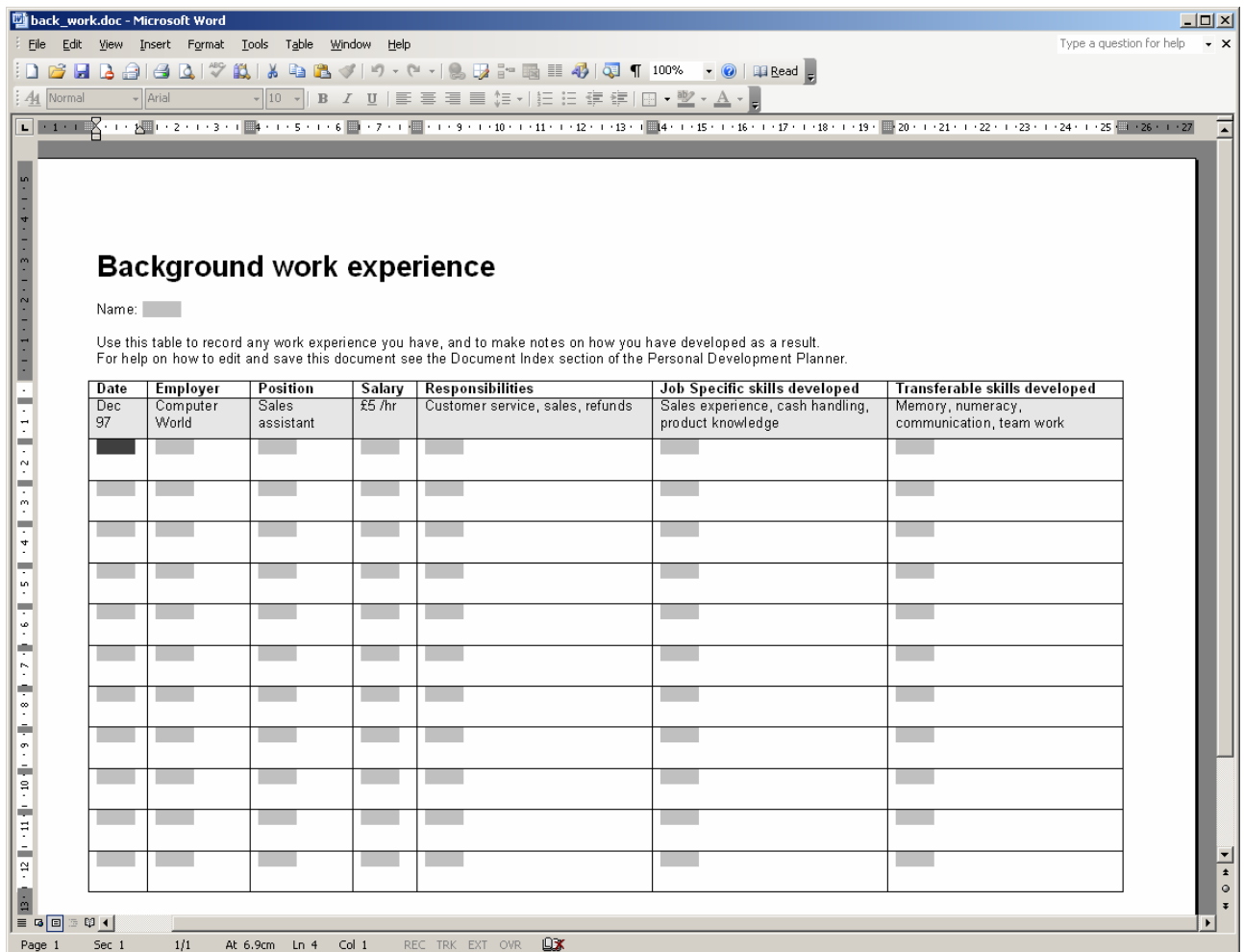
The areas which have been converted so far are:

- PDP | Background Qualifications
- PDP | Background Work Experience
- PDP | Background Extra Curricular Activities
- PDP | Background Strengths
- PDP | Background Areas of improvement

The areas which have caused the most problems are:

- PDP | Background Qualifications
- PDP | Background Work Experience
- PDP | Background Extra Curricular Activities

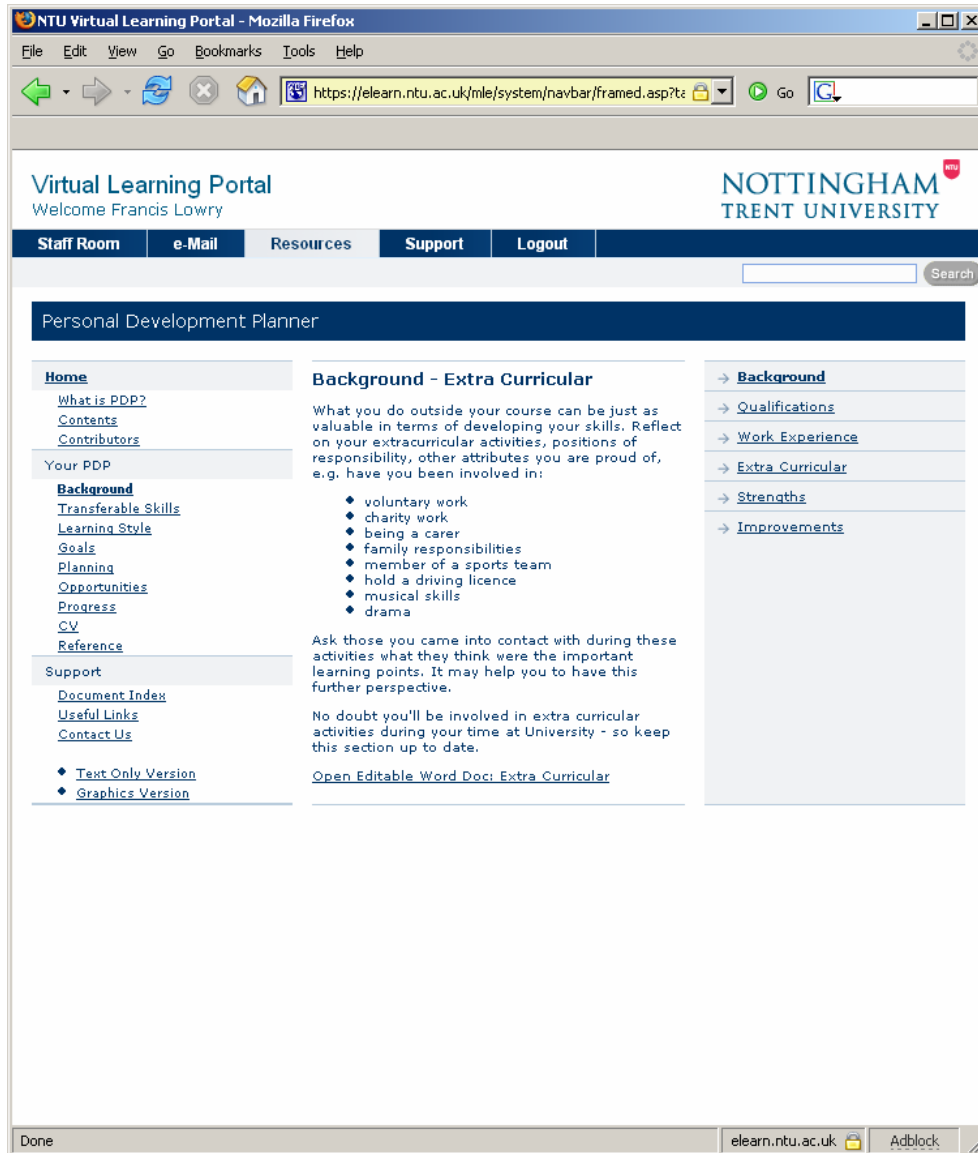
The main issue with these is that they contain a large chunk of textual data which would need to be split into more useful groupings to allow for more flexible mapping. The current Word documents look like this:



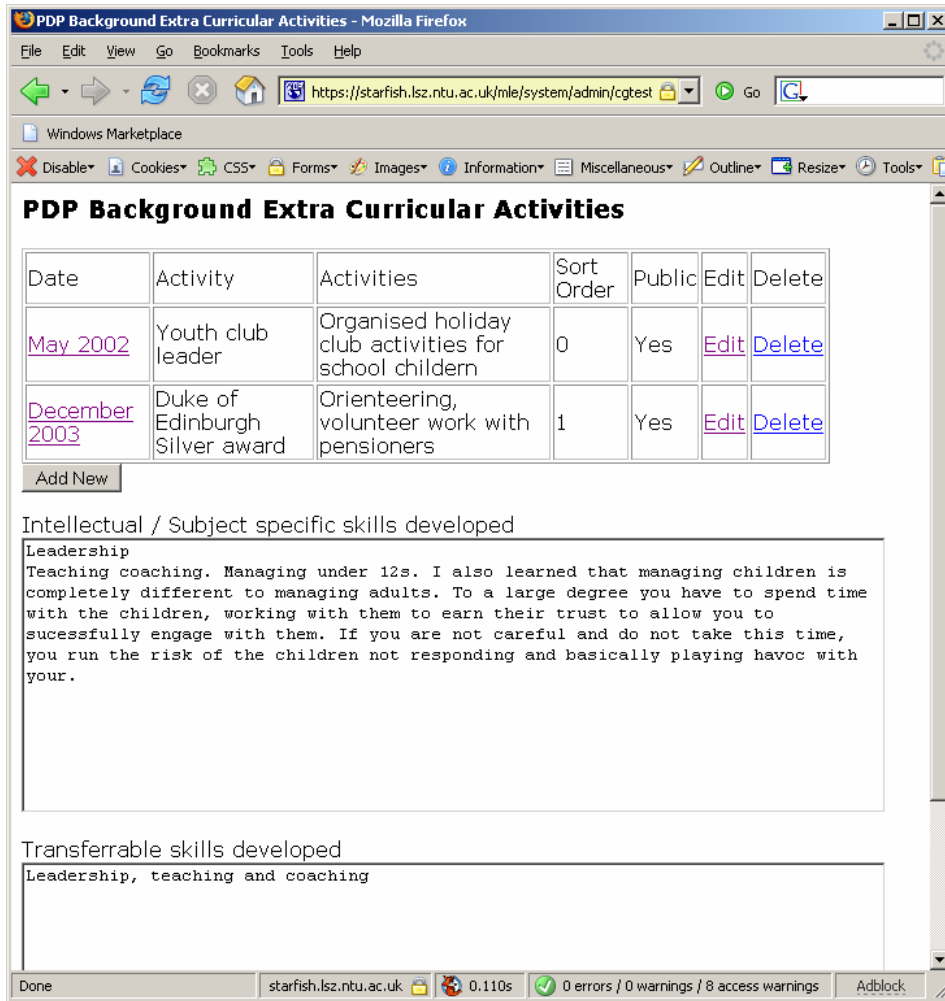
Learners edit these documents manually and store them in working folders on the network.

In all three problem areas, there are general text blocks where a learner can list their different skills developed in sentence format. Simply converting the Word document format into a database-driven one does not give us sufficient flexibility in how the data is structured.

The current Background Extra Curricular Activities section looks like this, with the downloadable Word document in a format similar to the previous image:



The initial conversion of this was to a web-based form which stored the equivalent sections in database fields. The screen shot below does not currently include the NTU styles and formatting as it is a prototype to enable us to re-model these areas more appropriately, not a completed development.



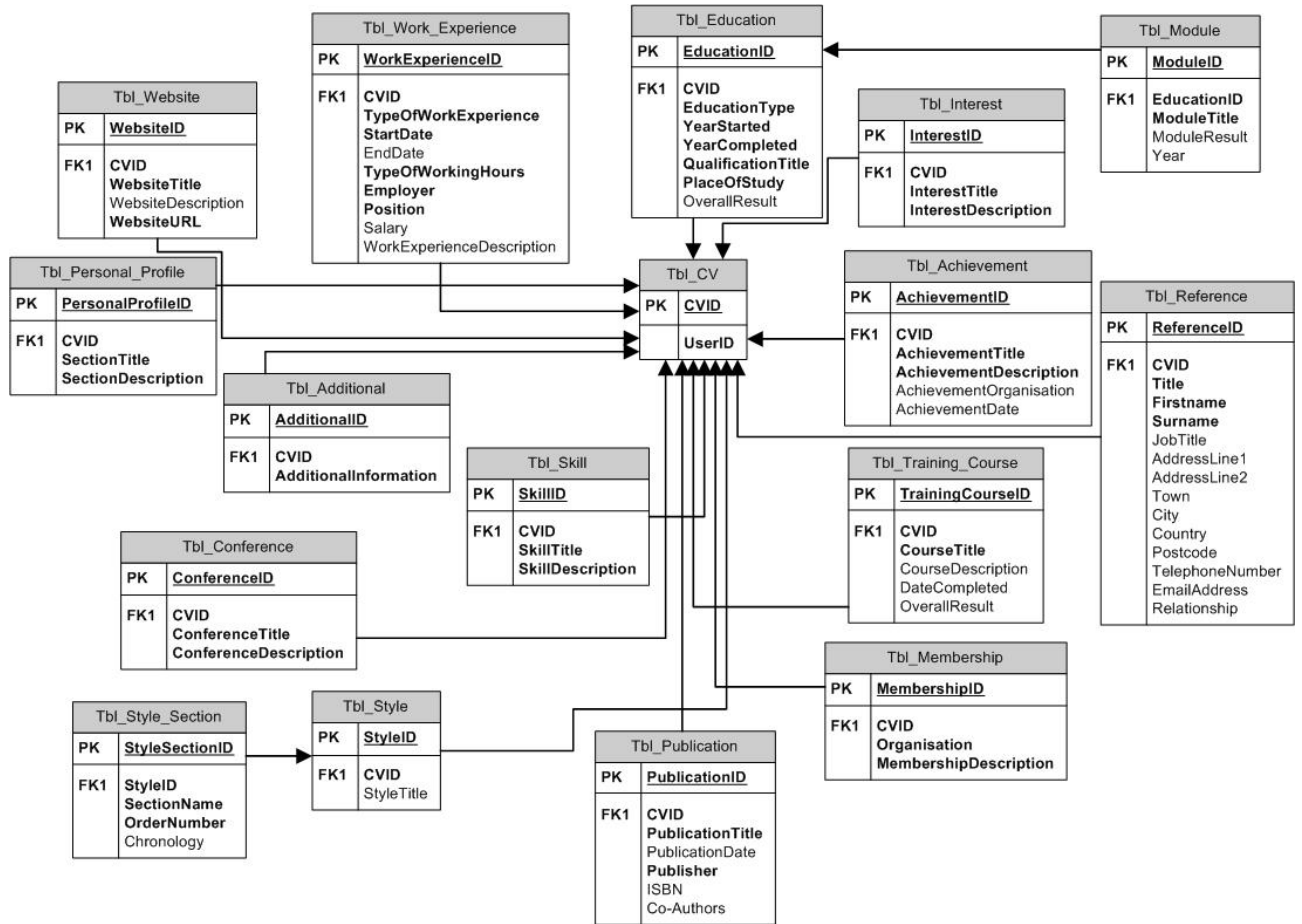
There is a lot of textual information here relating to skills, combining the skills themselves, evidence and reflection. While it is possible to simply transfer the text blocks, to utilise the UK LeaP model fully and allow a better transfer of data, we will need to split this out further into the component parts.

We are currently in the process of going back to the drawing board and reviewing how this data should be recorded.

CV Builder

Part of the work completed was to map the existing NTU CV Builder, which is already in a normalised relational format, to the UK LeaP specification to compare its fit. As the data was unpicked, it became clear that there were several areas of duplication within NTU PDP, with some areas holding more pertinent data. Again, the main problem area relates to Skills. Within the CV builder there is a very basic skills list, which does not correlate at all to any of the Background areas.

The current model (below) details all the tables within the current system.



There are some tables which are currently not used
 TBL_PERSONALPROFILE
 TBL_CONFERENCE
 TBL_PUBLICATION

And others that related solely to the look and feel of the CV.
 TBL_STYLE_SECTION
 TBL_STYLE

The UK LeaP mappings are listed in appendix 4.

Appendices

Appendix 1 – Final transfer XSLT

```

<?xml version="1.0"?>
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" indent="yes" />

<xsl:variable name="apos">'</xsl:variable>

<xsl:template match="/">
  <data>
    <xsl:call-template name="person" />
    <xsl:call-template name="address_details" />
    <xsl:call-template name="email_address" />
    <xsl:call-template name="contact_numbers" />
    <xsl:call-template name="personal_statement" />
  </data>
</xsl:template>

<xsl:template name="person">
  <person>
    <line>
      <person_key>
        <xsl:call-template name="treatString">
          <xsl:with-param name="string">
            <xsl:value-of
select="/learnerinformation/identification/name/partname[typename/tyvalue =
'StudentID']/text" />
          </xsl:with-param>
        </xsl:call-template>
      </person_key>
      <forename>
        <xsl:call-template name="treatString">
          <xsl:with-param name="string">
            <xsl:value-of
select="/learnerinformation/identification/name/partname[typename/tyvalue =
'Given']/text" />
          </xsl:with-param>
        </xsl:call-template>
      </forename>
      <surname>
        <xsl:call-template name="treatString">
          <xsl:with-param name="string">
            <xsl:value-of
select="/learnerinformation/identification/name/partname[typename/tyvalue =
'Surname']/text" />
          </xsl:with-param>
        </xsl:call-template>
      </surname>
      <birth_date>
        <xsl:call-template name="treatString">
          <xsl:with-param name="string">
            <xsl:value-of
select="/learnerinformation/identification/demographics/date[typename/tyvalue =
'Birth']/datetime" />
          </xsl:with-param>
        </xsl:call-template>
      </birth_date>
    </line>
  </person>
</xsl:template>
<xsl:template name="address_details">
<xsl:for-each select="//identification/address">
  <address_details>
    <line>
      <address_type>
        <xsl:call-template name="treatString">
          <xsl:with-param name="string">
            <xsl:value-of select="typename/tyvalue" />
          </xsl:with-param>
        </xsl:call-template>
      </address_type>
      <houenumber>

```

```

        <xsl:call-template name="treatString">
          <xsl:with-param name="string">
            <xsl:value-of select="street/streetnumber" />
          </xsl:with-param>
        </xsl:call-template>
      </housenumber>
      <streetname>
        <xsl:call-template name="treatString">
          <xsl:with-param name="string">
            <xsl:value-of select="street/streetname" />
          </xsl:with-param>
        </xsl:call-template>
      </streetname>
    </city>
    <xsl:call-template name="treatString">
      <xsl:with-param name="string">
        <xsl:value-of select="city" />
      </xsl:with-param>
    </xsl:call-template>
  </city>
  <region>
    <xsl:call-template name="treatString">
      <xsl:with-param name="string">
        <xsl:value-of select="region" />
      </xsl:with-param>
    </xsl:call-template>
  </region>
  <postcode>
    <xsl:call-template name="treatString">
      <xsl:with-param name="string">
        <xsl:value-of select="postcode" />
      </xsl:with-param>
    </xsl:call-template>
  </postcode>
</line>
</address_details>
</xsl:for-each>
</xsl:template>
<xsl:template name="email_address">
  <xsl:for-each select="//identification/contactinfo">
    <xsl:if test="email">
      <email_address>
        <line>
          <email_address_type>
            <xsl:call-template name="treatString">
              <xsl:with-param name="string">
                <xsl:value-of select="typename/tyvalue" />
              </xsl:with-param>
            </xsl:call-template>
          </email_address_type>
          <email_seq_no>
            <xsl:text>1</xsl:text>
          </email_seq_no>
          <email_primary_ind>
            <xsl:text>Yes</xsl:text>
          </email_primary_ind>
          <e_mail_address>
            <xsl:call-template name="treatString">
              <xsl:with-param name="string">
                <xsl:value-of select="email" />
              </xsl:with-param>
            </xsl:call-template>
          </e_mail_address>
        </line>
      </email_address>
    </xsl:if>
  </xsl:for-each>
</xsl:template>
<xsl:template name="contact_numbers">
  <xsl:for-each select="//identification/contactinfo">
    <xsl:if test="telephone">
      <contact_numbers>
        <line>
          <contact_number_type>
            <xsl:call-template name="treatString">
              <xsl:with-param name="string">
                <xsl:value-of select="typename/tyvalue" />
              </xsl:with-param>
            </xsl:call-template>
          </contact_number_type>
        </line>
      </contact_numbers>
    </xsl:if>
  </xsl:for-each>
</xsl:template>

```

```

        </contact_number_type>
        <contact_seq_no>
            <xsl:text>1</xsl:text>
        </contact_seq_no>
        <contact_primary_ind>
            <xsl:text>Yes</xsl:text>
        </contact_primary_ind>
        <area_code>
            <xsl:value-of select="telephone/areacode" />
        </area_code>
        <telephone_no>
            <xsl:value-of select="telephone/indnumber" />
        </telephone_no>
    </line>
</contact_numbers>
</xsl:if>
</xsl:for-each>
</xsl:template>
<xsl:template name="personal_statement">
<xsl:for-each select="//reflexion">
<xsl:if test="typename/tyvalue = 'SelfPresentation'">
    <personal_statement>
        <line>
            <personal_statement_type>
                <xsl:call-template name="treatString">
                    <xsl:with-param name="string">
                        <xsl:value-of select="typename/tyvalue" />
                    </xsl:with-param>
                </xsl:call-template>
            </personal_statement_type>
            <statement_date>
                <xsl:call-template name="treatString">
                    <xsl:with-param name="string">
                        <xsl:value-of select="date[typename/tyvalue =
'Create']/datetime" />
                    </xsl:with-param>
                </xsl:call-template>
            </statement_date>
            <statement_title>
                <xsl:value-of select="description/short" />
            </statement_title>
            <statement>
                <xsl:value-of select="description/long" />
            </statement>
        </line>
    </personal_statement>
</xsl:if>
</xsl:for-each>
</xsl:template>
<xsl:template match="*" />
<xsl:template name="treatString">
    <xsl:param name="string" />
    <xsl:choose>
        <xsl:when test="contains($string, $apos)">
            <xsl:value-of select="substring-before($string, $apos)" />
            <xsl:call-template name="treatString">
                <xsl:with-param name="string">
                    <xsl:value-of select="substring-after($string, $apos)" />
                </xsl:with-param>
            </xsl:call-template>
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="$string" />
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
</xsl:transform>

```

Appendix 2 – Transition 2 source XML

```

<?xml version="1.0" encoding="utf-8"?>
<learnerinformation>
  <identification>
    <name>
      <typename>
        <tysource sourcetype="standard">UKLeaP</tysource>
        <tyvalue>Full</tyvalue>
      </typename>
      <partname>
        <typename>
          <tysource sourcetype="standard">UKLeaP</tysource>
          <tyvalue>Given</tyvalue>
        </typename>
        <text>fred</text>
      </partname>
      <partname>
        <typename>
          <tysource sourcetype="imsdefault" />
          <tyvalue>Surname</tyvalue>
        </typename>
        <text>blogs</text>
      </partname>
    </name>
    <demographics>
      <date>
        <typename>
          <tysource sourcetype="standard">UKLeaP</tysource>
          <tyvalue>Birth</tyvalue>
        </typename>
        <datetime>1983-10-09T00:00:00</datetime>
      </date>
    </demographics>
    <address>
      <typename>
        <tysource sourcetype="standard">UKLeaP</tysource>
        <tyvalue>Private</tyvalue>
      </typename>
      <street>
        <streetnumber>50</streetnumber>
        <streetname>Somewhere St</streetname>
      </street>
      <city>Somewhere</city>
      <region>Somewhereshire</region>
      <postcode>S012 3ME</postcode>
    </address>
    <contactinfo>
      <typename>
        <tysource sourcetype="standard">UKLeaP</tysource>
        <tyvalue>Private</tyvalue>
      </typename>
      <telephone>
        <areacode>01234</areacode>
        <indnumber>567890</indnumber>
      </telephone>
    </contactinfo>
    <contactinfo>
      <typename>
        <tysource sourcetype="standard">UKLeaP</tysource>
        <tyvalue>Private</tyvalue>
      </typename>
      <email>fredbloggs1@gmail.com</email>
    </contactinfo>
  </identification>
  <reflexion>
    <typename>
      <tysource sourcetype="standard">UKLeaP</tysource>
      <tyvalue>SelfPresentation</tyvalue>
    </typename>
    <contenttype>
      <referential>
        <indexid>reflexion_1</indexid>
      </referential>
    </contenttype>
    <date>
      <typename>
        <tysource sourcetype="standard">UKLeaP</tysource>

```

```
<tyvalue>Create</tyvalue>
</typename>
<datetime>2005-12-11T09:15:00</datetime>
</date>
<description>
  <short>Personal Statement</short>
  <long>I am a great team player, but I am happy working on my own. I wish
to extend my knowledge of operating systemms and database development.</long>
</description>
</reflexion>
</learnerinformation>
```

Appendix 3 – Transition 3 Source XML

```

<?xml version="1.0" encoding="utf-8"?>
<learnerinformation>
  <identification>
    <name>
      <typename>
        <tysource sourcetype="standard">UKLeaP</tysource>
        <tyvalue>Full</tyvalue>
      </typename>
      <partname>
        <typename>
          <tysource sourcetype="standard">UKLeaP</tysource>
          <tyvalue>Given</tyvalue>
        </typename>
        <text>fred</text>
      </partname>
      <partname>
        <typename>
          <tysource sourcetype="imsdefault" />
          <tyvalue>Surname</tyvalue>
        </typename>
        <text>blogs</text>
      </partname>
    </name>
    <demographics>
      <date>
        <typename>
          <tysource sourcetype="standard">UKLeaP</tysource>
          <tyvalue>Birth</tyvalue>
        </typename>
        <datetime>1983-10-09T00:00:00</datetime>
      </date>
    </demographics>
    <address>
      <typename>
        <tysource sourcetype="standard">UKLeaP</tysource>
        <tyvalue>Domicile</tyvalue>
      </typename>
      <street>
        <streetnumber>50</streetnumber>
        <streetname>Somewhere St</streetname>
      </street>
      <city>Somewhere</city>
      <region>Somewhereshire</region>
      <postcode>S012 3ME</postcode>
    </address>
    <address>
      <typename>
        <tysource sourcetype="standard">UKLeaP</tysource>
        <tyvalue>Campus</tyvalue>
      </typename>
      <street>
        <streetnumber>17a</streetnumber>
        <streetname>Church Street</streetname>
      </street>
      <city>Carlton</city>
      <region>Nottingham</region>
      <postcode>NG4 1TT</postcode>
    </address>
    <contactinfo>
      <typename>
        <tysource sourcetype="standard">UKLeaP</tysource>
        <tyvalue>Private</tyvalue>
      </typename>
      <telephone>
        <areacode>01234</areacode>
        <indnumber>567890</indnumber>
      </telephone>
    </contactinfo>
    <contactinfo>
      <typename>
        <tysource sourcetype="standard">UKLeaP</tysource>
        <tyvalue>Private</tyvalue>
      </typename>
      <email>fredbloggs1@gmail.com</email>
    </contactinfo>
  </identification>

```

```

</contactinfo>
<contactinfo>
  <typename>
    <tysource sourcetype="standard">UKLeaP</tysource>
    <tyvalue>Campus</tyvalue>
  </typename>
  <email>N1230031@ntu.ac.uk</email>
</contactinfo>
</identification>
<reflexion>
  <typename>
    <tysource sourcetype="standard">UKLeaP</tysource>
    <tyvalue>SelfPresentation</tyvalue>
  </typename>
  <contenttype>
    <referential>
      <indexid>reflexion_1</indexid>
    </referential>
  </contenttype>
  <date>
    <typename>
      <tysource sourcetype="standard">UKLeaP</tysource>
      <tyvalue>Create</tyvalue>
    </typename>
    <datetime>2006-01-22T18:22:00</datetime>
  </date>
  <description>
    <short>Personal Statement</short>
    <long>I enjoy learning about how software development and am keen to
explore more about how to integrate virtual reality programming into computer
games</long>
  </description>
</reflexion>
</learnerinformation>

```

Appendix 4 – Source code for ASP import

Note: all the XML transitions can be loaded with this same source code.

```

<%
' --- Database Connection ---
Set conn = Server.CreateObject("ADODB.Connection")
conn.CursorLocation = 3
conn.open "xxxx", "xxxx", "xxxx"

'--- Load XML - Data from Nottingham Passport ---
set xml = Server.CreateObject("Msxml2.DOMDocument.4.0")
xml.load(Server.MapPath("Transitions/nottingham_passport.xml"))

'--- Load XSL stylesheet to transform data to UKLeaP format ---
set xsl = Server.CreateObject("Msxml2.DOMDocument.4.0")

xsl.load(Server.MapPath("PDPXMLTransform.xslt"))

' --- Parsing the XML ---

xml.loadXML(xml.transformNode(xsl))

'response.write xml.documentElement.xml

'set objRoot = xml.documentElement

set objRoot = xml.documentElement           'Set the root of the XML object

set objChannel = objRoot.selectSingleNode("//data")
set objAddress = objChannel.getElementsByTagName("address_details")
set objEmail = objChannel.getElementsByTagName("email_address")
set objContact = objChannel.getElementsByTagName("contact_numbers")
set objPStatement = objChannel.getElementsByTagName("personal_statement")

v_StudentKey           = objRoot.selectSingleNode("person/line/person_key").text
v_Forename             = objRoot.selectSingleNode("person/line/forename").text
v_Surname              = objRoot.selectSingleNode("person/line/surname").text
v_BirthDate            = objRoot.selectSingleNode("person/line/birth_date").text

if (isnull(v_StudentKey) or len(v_studentkey) = 0 ) then
    v_StudentKey = "NONE"
end if

' --- get Person Key field
sql = "exec get_person_key " & v_studentKey
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3

new_stud_key = rs("NewKey")

' --- INSERT Record ---
sql = "INSERT INTO person(person_key,person_id, forename,surname,date_of_birth)"
sql = sql & " VALUES(' & new_stud_key & ', ' & v_StudentKey & ', ' & v_Forename &
' & v_Surname & ', ' & v_BirthDate & ')"
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3

For Each objChild in objAddress

v_AddressType          = objChild.selectSingleNode("line/address_type").text
v_StreetNumber         = objChild.selectSingleNode("line/housenumber").text
v_StreetName           = objChild.selectSingleNode("line/streetname").text
v_City                 = objChild.selectSingleNode("line/city").text
v_Region               = objChild.selectSingleNode("line/region").text
v_PostCode             = objChild.selectSingleNode("line/postcode").text

sql = "INSERT INTO address_details(person_key,address_type,
housenumber,streetname,city,region,postcode)"
sql = sql & " VALUES(' & new_stud_key & ', ' & v_AddressType & ', ' &
v_streetNumber & ', ' & v_streetname & ', ' & v_city & ', ' &
sql = sql & "' & v_region & ', ' & v_postcode & ')"
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3

next

```

```

For Each objChild in objEmail

v_EmailType           = objChild.selectSingleNode("line/email_address_type").text
v_EmailSeqNo          = objChild.selectSingleNode("line/email_seq_no").text
v_EmailPrimaryInd     = objChild.selectSingleNode("line/email_primary_ind").text
v_Email               = objChild.selectSingleNode("line/e_mail_address").text

sql = "INSERT INTO email_address(person_key, email_address_type, email_seq_no,
email_primary_ind, email_address)"
sql = sql & " VALUES('" & new_stud_key & "', '" & v_EmailType & "', '" & v_EmailSeqNo
& "', '" & v_EmailPrimaryInd & "', '" & v_Email & "'"")"
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3

next

For Each objChild in objContact

v_ContactNoType       =
objChild.selectSingleNode("line/contact_number_type").text
v_ContactSeq          = objChild.selectSingleNode("line/contact_seq_no").text
v_ContactPrimaryInd   = objChild.selectSingleNode("line/contact_primary_ind").text
v_ContactAreaCode     = objChild.selectSingleNode("line/area_code").text
v_ContactNumber       = objChild.selectSingleNode("line/telephone_no").text

sql = "INSERT INTO contact_numbers(person_key, contact_number_type, contact_seq_no,
contact_primary_ind, area_code, telephone_no)"
sql = sql & " VALUES('" & new_stud_key & "', '" & v_ContactNoType & "', '" &
v_ContactSeq & "', '" & v_ContactPrimaryInd & "', '" & v_ContactAreaCode
sql = sql & "', '" & v_ContactNumber & "'"")"
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3

next

For Each objChild in objPStatement

v_PStatementType      = objChild.selectSingleNode("line/personal_statement_type").text
v_PStatementDate      = objChild.selectSingleNode("line/statement_date").text
v_PStatementTitle     = objChild.selectSingleNode("line/statement_title").text
v_PStatementLong      = objChild.selectSingleNode("line/statement").text

sql = "INSERT INTO personal_statement(person_key, statement_type, statement_date,
statement_title, statement )"
sql = sql & " VALUES('" & new_stud_key & "', '" & v_PStatementType & "', '" &
v_PStatementDate & "', '" & v_PStatementTitle & "', '" & v_PStatementLong & "'"")"
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3

next

sql = "Select person_key, person_id, forename, surname, date_of_birth from person
where person_key = " & new_stud_key
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3

%>

<h3>Person Details</h3>
<table width="90%" border="1">
  <tr>
    <td>Unique ID</td>
    <td>Student ID</td>
    <td>Forename</td>
    <td>Surname</td>
    <td>Date of Birth</td>
  </tr>

<%
if not rs.EOF then
DO WHILE NOT rs.EOF

%>

```

```

    <tr>
      <td><%=rs("person_key")%></td>
      <td><%=rs("person_id")%></td>
      <td><%=rs("forename")%></td>
      <td><%=rs("surname")%></td>
      <td><%=rs("date_of_birth")%></td>
    </tr>
<%
rs.movenext
loop
end if
%>

</table>

<%
sql = "Select person_key, address_type, housenumber,streetname,city,region,postcode
from address_details where person_key = " & new_stud_key
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3

%>

<h3>Address Details</h3>
<table width="90%" border="1">
  <tr>
    <td>Unique ID</td>
    <td>Address Type</td>
    <td>House Number</td>
    <td>Street Name</td>
    <td>City</td>
    <td>Region</td>
    <td>Postcode</td>
  </tr>

  <%
if not rs.EOF then
DO WHILE NOT rs.EOF

%>
    <tr>
      <td><%=rs("person_key")%></td>
      <td><%=rs("address_type")%></td>
      <td><%=rs("housenumber")%></td>
      <td><%=rs("streetname")%></td>
      <td><%=rs("city")%></td>
      <td><%=rs("region")%></td>
      <td><%=rs("postcode")%></td>
    </tr>
  <%
rs.movenext
loop
end if
%>
</table>

<%
sql = "Select person_key, email_address_type, email_seq_no, email_primary_ind,
email_address from email_address where person_key = " & new_stud_key

Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3

%>

<h3>Email Address</h3>
<table width="90%" border="1">
  <tr>
    <td>Unique ID</td>
    <td>Email Type</td>
    <td>Sequence No</td>
    <td>Primary Ind</td>
    <td>Email</td>
  </tr>
  <%
if not rs.EOF then
DO WHILE NOT rs.EOF

%>
    <tr>

```

```

        <td><%=rs("person_key")%></td>
        <td><%=rs("email_address_type")%></td>
        <td><%=rs("email_seq_no")%></td>
        <td><%=rs("email_primary_ind")%></td>
        <td><%=rs("email_address")%></td>
    </tr>
<%
rs.movenext
loop
end if
%>
</table>
<%
sql = "Select person_key, contact_number_type, contact_seq_no, contact_primary_ind,
area_code, telephone_no from contact_numbers where person_key = " & new_stud_key

Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3

%>

<h3>Contact Numbers</h3>
<table width="90%" border="1">
    <tr>
        <td>Unique ID</td>
        <td>Contact Type</td>
        <td>Sequence No</td>
        <td>Primary Ind</td>
        <td>Area Code</td>
        <td>Telephone</td>
    </tr>

<%
if not rs.EOF then
DO WHILE NOT rs.EOF

%>
    <tr>
        <td><%=rs("person_key")%></td>
        <td><%=rs("contact_number_type")%></td>
        <td><%=rs("contact_seq_no")%></td>
        <td><%=rs("contact_primary_ind")%></td>
        <td><%=rs("area_code")%></td>
        <td><%=rs("telephone_no")%></td>
    </tr>
<%
rs.movenext
loop
end if
%>

</table>

<%
sql = "Select person_key, statement_type, statement_id, statement_date,
statement_title, statement from personal_statement where person_key = " & new_stud_key

Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3

if not rs.EOF then
DO WHILE NOT rs.EOF

%>

<h3>Personal Statement</h3>
<table width="90%" border="1">
    <tr>
        <td>Unique ID</td>
        <td>Statement Type</td>
        <td>Statement Date</td>
    </tr>

<%

%>
    <tr>
        <td><%=rs("person_key")%></td>
        <td><%=rs("statement_type")%></td>

```

```
        <td><%=rs("statement_date")%></td>
    </tr>
</table>
<BR>
<I><B>Statement Title: </B></I>
<BR><%=rs("statement_title")%>
<BR>
<I><B>Statement:</B></I>
<BR><%=rs("statement")%>

<%

rs.movenext
loop
end if
%>
```

Appendix 5 – NTU CV Builder -> UK LeaP mappings

NTU CV Builder		UKDLIP Mappings	Notes
CVID			Used to identify a specific CV - maps to the NTU Username.
TBL_CV	USERID	affiliation/affiliationid	
TBL_REFERENCE	TITLE		
	FIRSTNAME	Identification/name/partname(Given)	
	SURNAME	Identification/name/partname(Surname)	
	JOBTITLE		
	ADDRESSLINE1	Identification/address/street/streetnumber	
	ADDRESSLINE2	Identification/address/street/streetname	
	TOWN	Identification/address	
	CITY	identification/address/city	
	COUNTRY	identification/address/country	
	POSTCODE	Identification/address/postcode	
	TELEPHONENUMBER	identification/contactinfo/telephone	
	EMAILADDRESS	identification/contactinfo/email	
	RELATIONSHIP		Not used
TBL_EDUCATION	EDUCATION_ID	qcl/contenttype/referential/indexid	This key joins the degree result to the component modules
		qcl[type='Degree:']	
	YEARSTARTED	qcl/date/[type='Start']	
	YEARCOMPLETED	qcl/date/[type='Award']	
	QUALIFICATIONTITLE	qcl/title	
	PLACEOFSTUDY	qcl/organization/short	
	OVERALLRESULT	qcl/level[text=NQF HE3]/level/text	
TBL_MODULE			Not currently implemented
	MODULEID	activity/learningactivityref/sourceid	
	MODULETITLE	activity/definition/description/short	
	MODULERESULT	activity/units/unitsfield[Type='Credits']	
	YEAR	activity/date[type='Score']	
TBL_WORK_EXPERIENCE	TYPEOFWORKEXPERIENCE	activity/[Type = (Placement / Development / Work)]	Mapping Work Experience Placement Relevant Work Development Professional Work Placement Placement Research Development Other Development
	STARTDATE	activity/date/[type = 'Start']	
	ENDDATE	activity/date/[type = 'Finish']	
	TYPEOFWORKINGHOURS	activity/text	Undefined - leave as a general text attribute
	EMPLOYER	affiliation/affiliationid	
	POSITION	activity/definition/description/short	
	SALARY	activity/text	Undefined - again leave as a general text attribute
	WORKEXPERIENCEDESCRIPTION	activity/definition/description/long	
TBL_INTEREST	INTERESTID		Referential - but not required for UKLEAP
		activity/[type = 'Personal']	
	INTERESTTITLE	activity/definition/description/short	
	INTERESTDESCRIPTION	activity/definition/description/long	
TBL_ACHIEVEMENT	ACHIEVEMENTID		Referential - but not required for UKLEAP

		QCL/[type = 'Noncertificated']	
	ACHIEVEMENTTITLE	qcl/description/short	
	ACHIEVEMENTDESCRIPTION	qcl/description/long	
	ACHIEVEMENTORGANISATION	qcl/organization	
	ACHIEVEMENTDATE	qcl/date/[type = 'Award']	
TBL_TRAINING_COURSE	TRAININGCOURSEID		Referential - but not required for UKLEAP
		activity/[type = 'Training']	
	COURSETITLE	activity/definition/description/short	
	COURSEDESCRIPTION	activity/definition/description/long	
	DATECOMPLETED	activity/date/[type = 'Finish']	
	OVERALLRESULT	activity/units/unitsfield[Type='Credits']	
TBL_MEMBERSHIP	MEMBERSHIPID	affiliation/affiliationid	
		affiliation(Official)	
	ORGANISATION	affiliation/organization/description/short	
	MEMBERSHIPDESCRIPTION	affiliation/role/description/short	
TBL_WEBSITE	WEBSITEID		not implemented but would possibly be Assertion/[Type='Selfpresentation']
	WEBSITETITLE		
	WEBSITEDESCRIPTION		
	WEBSITEURL		
TBL_SKILL	SKILLID	contenttype/referential	
	SKILLTITLE	competency/description/short	
	SKILLDESCRIPTION	competency/description/long	
TBL_ADDITIONAL	ADDITIONALID		Not required for UK leap
	ADDITIONALINFORMATION	assertion/[Type = 'Selfpresentation']	