

Table of contents

| | |
|--|----|
| 1. Introduction | 2 |
| 2. Deployment of mappings..... | 2 |
| Introduction..... | 2 |
| Making of mappings..... | 3 |
| About the exercises..... | 3 |
| Sample XML File | 4 |
| Exercise 1 | 5 |
| Exercise 2 | 7 |
| Exercise 3 | 9 |
| Exercise 4 | 11 |
| Exercise 5 | 14 |
| Installation of mappings | 15 |
| 3. Use of online platform | 15 |
| Sending of data..... | 15 |
| Receiving of data | 16 |
| Querying data | 16 |
| 4. The Application Programming Interface..... | 16 |
| Introduction..... | 16 |
| Scenarios..... | 16 |
| SendAPI | 16 |
| QueryAPI..... | 17 |
| xml_query file | 18 |
| XML for Exhibitions – Structure of Euromuse sent data | 19 |
| XML for Exhibitions – Structure of Euromuse result set | 20 |

1. Introduction

This document is a first version of the manual for the deployment of the Harmonise system in the context of the euromuse.net project. The prime intention is to use it alongside the trainings and it is not made for autodidactic trainings – although it could be used for autonomous training as well. The manual shall be updated constantly to incorporate feedback gained from users during the training sessions.

Harmonise provides a service for the transportation and transformation of data in the tourism domain. In order to exchange data between different information technology systems, it either has to follow the same format or it has to be transformed between different formats. The effort for these transformations rises significantly in a network of players between several partners, since it has to be made for each link between two partners. The absence of a broadly accepted standard, that would make the exchange of data easier, led to the development of the Harmonise technology, which reduces efforts for information exchange significantly.

The user of the Harmonise service only needs to implement transformation rules once to define the transformation of its own data format to the central reference model, the Harmonise ontology, or the other way round. Whenever sending data, it is first transformed to the Harmonise reference model and a second time right afterwards to the recipient's data format.

Thus, somebody using Harmonise only needs to define transformation rules to transform the data, called "mapping rules". These rules are stored in XSLT-files. The process itself is called "mapping" of data. After having these mappings done data can be sent as many times as wanted either by the use of a web-interface or by an application programming interface (API) – whatever is desired or supported by the organization using Harmonise.

This structure of this manual follows exactly the process to be followed in practice:

- First it describes how to make and install mappings.
- Second it describes the usage of the online platform to send and receive data.
- Third it gives the specification of the API the case that an automated connection with the Harmonise center is chosen (alternatively to the use of the online platform).

All of these steps will be shown live during the training sessions.

2. Deployment of mappings

Introduction

Harmonise mappings are translations of XML-files, more precisely the rules how to transform information from one XML-structure to another, without changing the information itself – only its representation. These mapping rules are stored in special files called XSLT files.

The process of generating these XSLT files is the process of making mappings. A partner joining the euromuse.net platform first always has to make mappings to send data. And mappings are also needed to receive data. One XSLT file can only work for one of the directions, not for both. Thus a user wanting to send and receive information via Harmonise has to prepare two XSLT files.

Euromuse.net – Harmonise manual

Special literature and online manuals are available on the market and in the internet on the generation of XSLT files. The intention of this document is not teach the making of XSLT files as such, but rather a cookbook to provide museums and tourism partners with information to be able to make mappings themselves. The trainings follow the pattern and material which was originally developed in the eTEN project “Harmo-TEN”, in which a business plan for the deployment of Harmonise was developed – and in this context also training material.

The mapping training starts with a general explanation of Harmonise, followed by some basic exercises about mappings. Depending on the composition of the trainees of a particular training session, the training might continue with the making of a concrete mapping-sample or with a more generic example covering the most common cases that could occur in the context of making mappings.

The only pre-requirements a participant has to bring to a training is some basic technical understanding. In addition it helps to have a description of the data schema of each partner (an XSD file) and sample data to test the mappings.

Making of mappings

About the exercises

These following comprise five exercises, starting always with a simple introduction that explains the objective of the exercise. This introduction is followed by the actual instructions (tasks).

The tasks can be completed in the pre-printed figures that follow the instructions:

The mapping figures represent the left, central, and right frames of the mapping tool where the requested bridges along with their path type arguments and their name are to be drawn. Here one can draw the necessary bridges and connect them to the adequate concepts, respectively attributes.

Below the mapping figures smaller figures are found representing the property frames of the “bridges” to be created. This section is called Bridge properties. Tables that represent Concept Bridges may contain execution conditions in certain exercises. In that case, the argument path has to be filled in appropriately. In the tables that represent Property Bridges first the Transformation Service has to be selected, followed by setting the right arguments for the selected service.

Please name the bridges drawn in the mapping figures as proposed in the Bridge properties section. As a convention, we name Concept Bridges with a name starting with “**cb**” and Property Bridges with a name starting with “**pb**”.

Sample XML File

The Figure 1 shows a sample XML file that complies with the source data model. The XML file contains two person records and gives a feeling how the actual data to be mapped will look like. For a mapping process such sample files are important to check whether the mapping conserves the semantics of the data. From the content of the XML file one can for example deduce the country and then further investigate whether this assumption is correct.

```
<Employees>
  <Employee>
    <FirstName>Max</FirstName>
    <SurName>Mustermann</SurName>
    <SocialSecurityNumber>1234010175</SocialSecurityNumber>
    <HomeAddress>
      <Street>Donaucity Straße, 1</Street>
      <City>Wien</City>
      <PostalCode>1220</PostalCode>
    </HomeAddress>
    <DrivingLicence>
      <canDriveBikes>true</canDriveBikes>
      <canDrivePersonalCars>true</canDrivePersonalCars>
    </DrivingLicence>
  </Employee>

  <Employee>
    <FirstName>Willi</FirstName>
    <SurName>Maier</SurName>
    <SocialSecurityNumber>5786191175</SocialSecurityNumber>
    <HomeAddress>
      <Street>Schwedenplatz, 5/12</Street>
      <City>Innsbruck</City>
      <PostalCode>6010</PostalCode>
    </HomeAddress>
    <DrivingLicence>
      <canDriveBikes>true</canDriveBikes>
    </DrivingLicence>
  </Employee>
</Employees>
```

Figure 1: Sample XML file

If you are not familiar with XML follow the instructions of the presenter.

Exercise 1

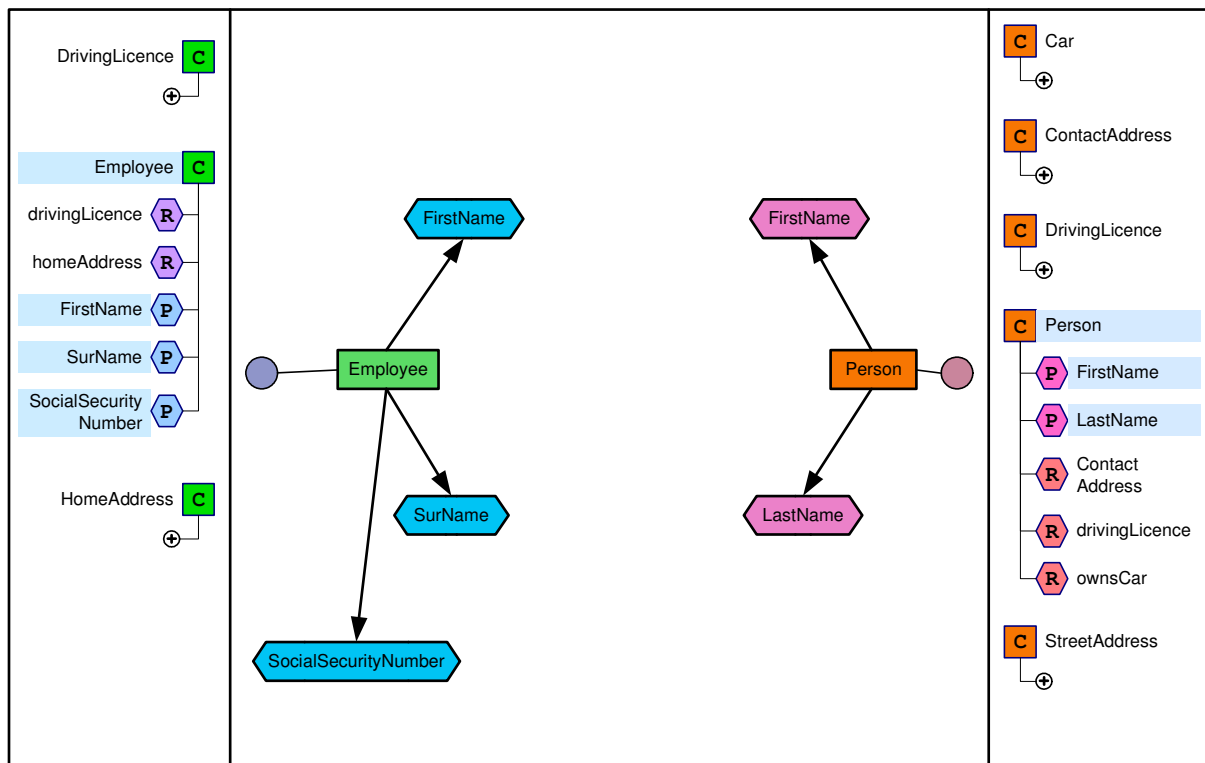
This exercise shows how to map one basic source concept to a target concept. Furthermore, it includes the task of mapping attributes (properties) of that source concept. It also includes peculiar concept attributes.

Tasks

1. Please create a bridge that maps the concept Employee to the concept Person (in the Mapping Figure and the adequate Bridge property figure).
2. Please map the attribute FirstName (in the Mapping Figure and the adequate Bridge property figure).
3. Do the same as in task 2 with the attribute SurName.
4. What about the attribute SocialSecurityNumber of Employee?
5. What about the attribute ownsCar of the concept Person?

Hint: You can find the necessary Bridge Property figures on the next page.

Mapping Figure



Bridge properties (Exercise 1)

| | | |
|--|--|--|
| <div style="border: 1px solid black; background-color: yellow; padding: 2px; display: inline-block;">cb Person</div> | <p>Extensional Specification (Conditions for Execution):</p> | <input type="radio"/> Exists <input type="radio"/> Equals Value: <input style="width: 100px;" type="text"/> |
|--|--|--|

| | | |
|---|--|---|
| <div style="border: 1px solid black; background-color: purple; padding: 2px; display: inline-block;">pb FirstName</div> | <input type="radio"/> Copy Attribute <input type="radio"/> Split <input type="radio"/> Constant Literal <input type="radio"/> Concatenate <input type="radio"/> Value Translation | <p>Literal Value or Split separator:</p> <input style="width: 100px;" type="text"/> |
|---|--|---|

| | | |
|--|--|---|
| <div style="border: 1px solid black; background-color: purple; padding: 2px; display: inline-block;">pb LastName</div> | <input type="radio"/> Copy Attribute <input type="radio"/> Split <input type="radio"/> Constant Literal <input type="radio"/> Concatenate <input type="radio"/> Value Translation | <p>Literal Value or Split separator:</p> <input style="width: 100px;" type="text"/> |
|--|--|---|

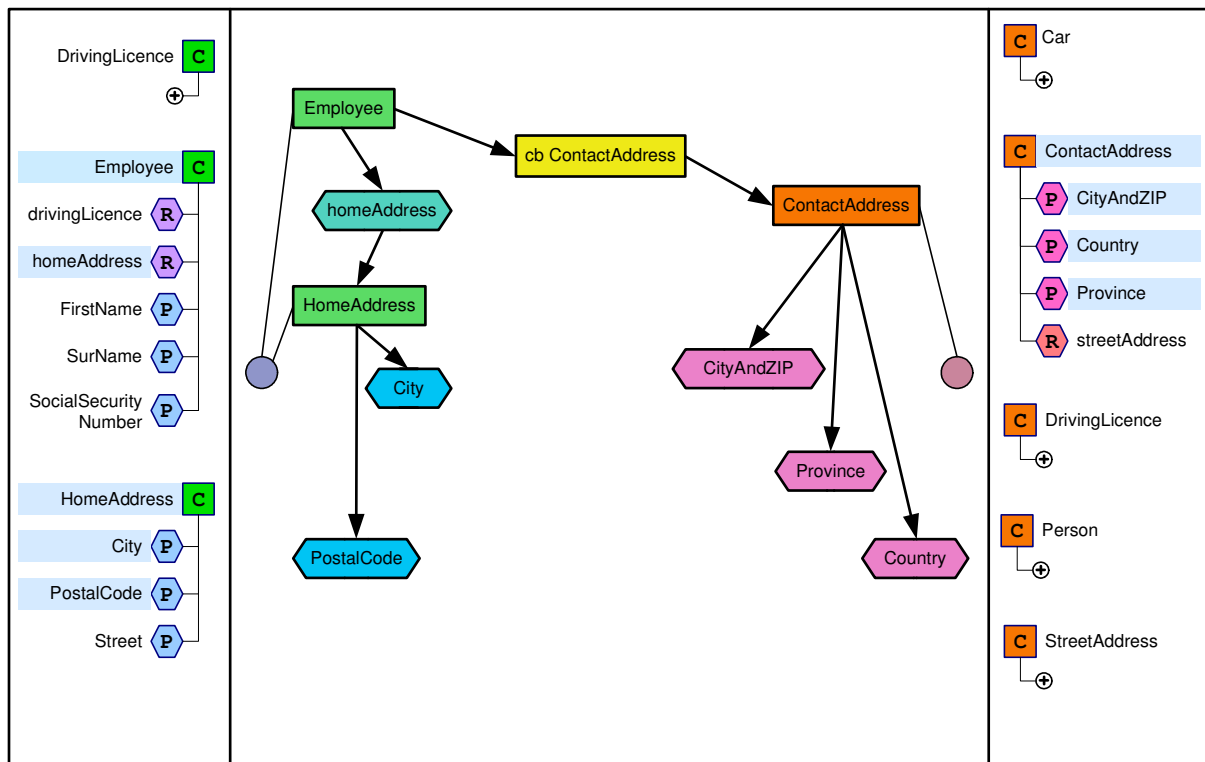
Exercise 2

In this exercise the Concept Bridge "cb Contact Address" is already given. It maps from the source Employee to the target concept ContactAddress. For sake of simplicity in the Mapping Figure, only the ContactAddress and its attributes are shown on the target side. This exercise teaches how to create an execution condition (Extensional specification) for a concept bridge and introduces new transformation services for the property bridges. It also shows how information that is not present in the data (model) can be mapped under certain circumstances.

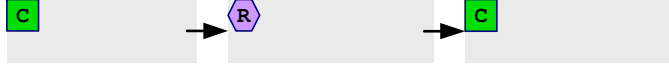
Tasks

1. Add a meaningful execution condition to "cb ContactAddress".
2. Map the HomeAddress's attribute City.
3. Map the HomeAddress's attribute Postal Code.
4. Find a mapping for the target attribute Country.
5. How can a mapping for the target attribute Province be realised?

Mapping Figure



Bridge properties (Exercise 2)

| | | |
|-------------------|---|--|
| cb ContactAddress | Extensional Specification (Conditions for Execution):  | <input type="radio"/> Exists <input type="radio"/> Equals Value: <input style="width: 100px;" type="text"/> |
|-------------------|---|--|

| | | |
|---------------|--|---|
| pb CityAndZIP | <input type="radio"/> Copy Attribute <input type="radio"/> Split <input type="radio"/> Constant Literal <input type="radio"/> Concatenate <input type="radio"/> Value Translation | Literal Value or Split separator: <input style="width: 100px;" type="text"/> |
|---------------|--|---|

| | | |
|------------|--|---|
| pb Country | <input type="radio"/> Copy Attribute <input type="radio"/> Split <input type="radio"/> Constant Literal <input type="radio"/> Concatenate <input type="radio"/> Value Translation | Literal Value or Split separator: <input style="width: 100px;" type="text"/> |
|------------|--|---|

| | | | | | | | | | | | | | | | | | | | | |
|---|--|---|---------------|----------------------|--|--------|--------|-------------------|--------|---------------------------------------|--------------------------------|--------|--------------------|-------------------------|--------|------------------|---------------------|--|--|------------------|
| pb Province | <input type="radio"/> Copy Attribute <input type="radio"/> Split <input type="radio"/> Constant Literal <input type="radio"/> Concatenate <input type="radio"/> Value Translation | Literal Value or Split separator: <input style="width: 100px;" type="text"/> | | | | | | | | | | | | | | | | | | |
| <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;"><i>Value:</i></td> <td style="width: 30%;"><i>Translate to:</i></td> <td style="width: 30%;"></td> </tr> <tr> <td>„1“xxx</td> <td>„Wien“</td> <td>„5“xxx „Salzburg“</td> </tr> <tr> <td>„2“xxx</td> <td>„Niederösterreich or Burgenland Nord“</td> <td>„6“xxx „Tirol oder Vorarlberg“</td> </tr> <tr> <td>„3“xxx</td> <td>„Niederösterreich“</td> <td>„7“xxx „Burgenland Süd“</td> </tr> <tr> <td>„4“xxx</td> <td>„Oberösterreich“</td> <td>„8“xxx „Steiermark“</td> </tr> <tr> <td></td> <td></td> <td>„9“xxx „Kärnten“</td> </tr> </table> | | | <i>Value:</i> | <i>Translate to:</i> | | „1“xxx | „Wien“ | „5“xxx „Salzburg“ | „2“xxx | „Niederösterreich or Burgenland Nord“ | „6“xxx „Tirol oder Vorarlberg“ | „3“xxx | „Niederösterreich“ | „7“xxx „Burgenland Süd“ | „4“xxx | „Oberösterreich“ | „8“xxx „Steiermark“ | | | „9“xxx „Kärnten“ |
| <i>Value:</i> | <i>Translate to:</i> | | | | | | | | | | | | | | | | | | | |
| „1“xxx | „Wien“ | „5“xxx „Salzburg“ | | | | | | | | | | | | | | | | | | |
| „2“xxx | „Niederösterreich or Burgenland Nord“ | „6“xxx „Tirol oder Vorarlberg“ | | | | | | | | | | | | | | | | | | |
| „3“xxx | „Niederösterreich“ | „7“xxx „Burgenland Süd“ | | | | | | | | | | | | | | | | | | |
| „4“xxx | „Oberösterreich“ | „8“xxx „Steiermark“ | | | | | | | | | | | | | | | | | | |
| | | „9“xxx „Kärnten“ | | | | | | | | | | | | | | | | | | |

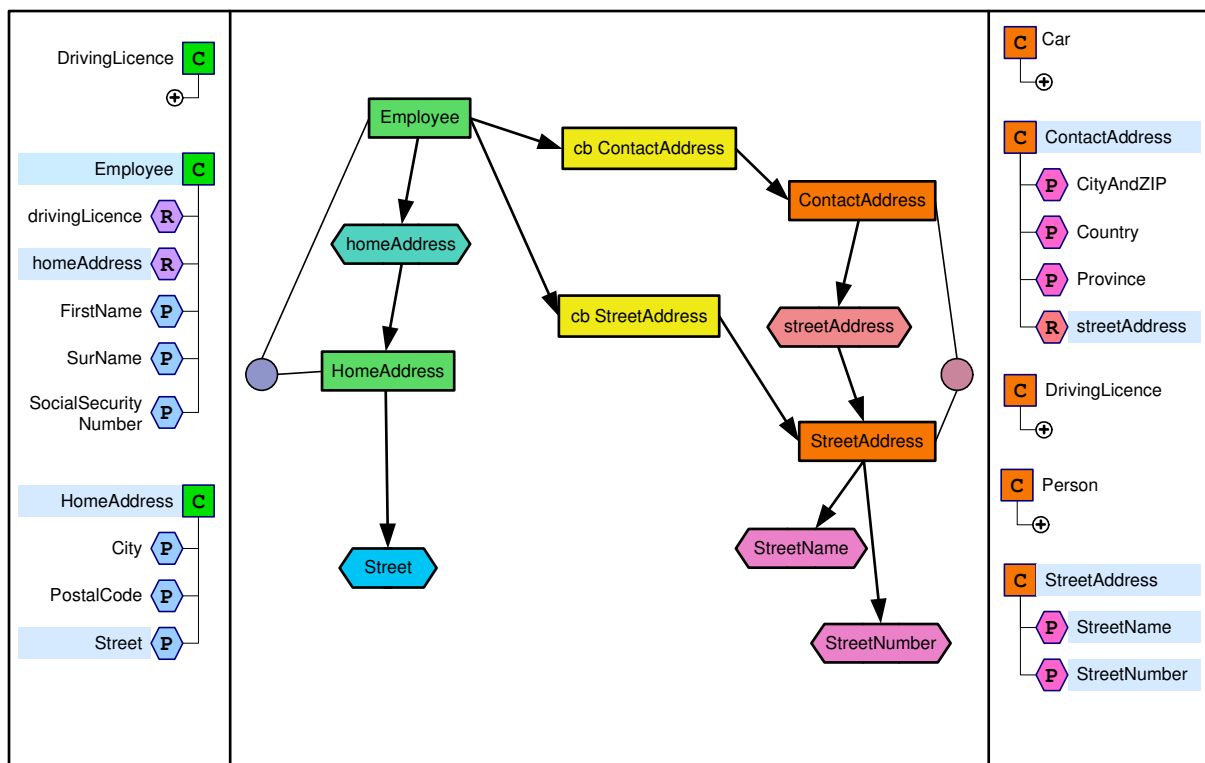
Exercise 3

In this exercise the Concept Bridges "cb ContactAddress" and "cb StreetAddress" are already given. There is also an Extensional Specification for the first one. This exercise will show how to map attributes at different data model levels. It also introduces long argument paths

Tasks

1. Find an execution condition (Extensional specification) for the Concept Bridge "cb StreetAddress".
2. Map the source attribute Street (attribute of HomeAddress).

Mapping Figure



Bridge properties

| | | |
|--|--|--|
| <div style="border: 1px solid black; background-color: yellow; padding: 2px; display: inline-block;">cb ContactAddress</div> | <p>Extensional Specification (Conditions for Execution):</p> <pre> graph LR Employee[C] --> homeAddress[R] homeAddress --> HomeAddress[C] </pre> | <input checked="" type="checkbox"/> Exists <input type="checkbox"/> Equals Value: <input style="width: 50px;" type="text"/> |
|--|--|--|

| | | |
|---|--|---|
| <div style="border: 1px solid black; background-color: yellow; padding: 2px; display: inline-block;">cb StreetAddress</div> | <p>Extensional Specification (Conditions for Execution):</p> <pre> graph LR Source[C] --> R[R] R --> P[P] P --> Destination[C] </pre> | <input type="checkbox"/> Exists <input type="checkbox"/> Equals Value: <input style="width: 50px;" type="text"/> |
|---|--|---|

| | | |
|--|---|--|
| <div style="border: 1px solid black; background-color: purple; padding: 2px; display: inline-block;">pb Street</div> | <input type="checkbox"/> Copy Attribute <input type="checkbox"/> Split <input type="checkbox"/> Constant Literal <input type="checkbox"/> Concatenate <input type="checkbox"/> Value Translation | Literal Value or Split separator: <input style="width: 50px;" type="text"/> |
|--|---|--|

Exercise 4

In this exercise the Concept Bridges "cb Person" and "cb DrivingLicense" are already given. "cb DrivingLicense" creates the target concept DrivingLicense.

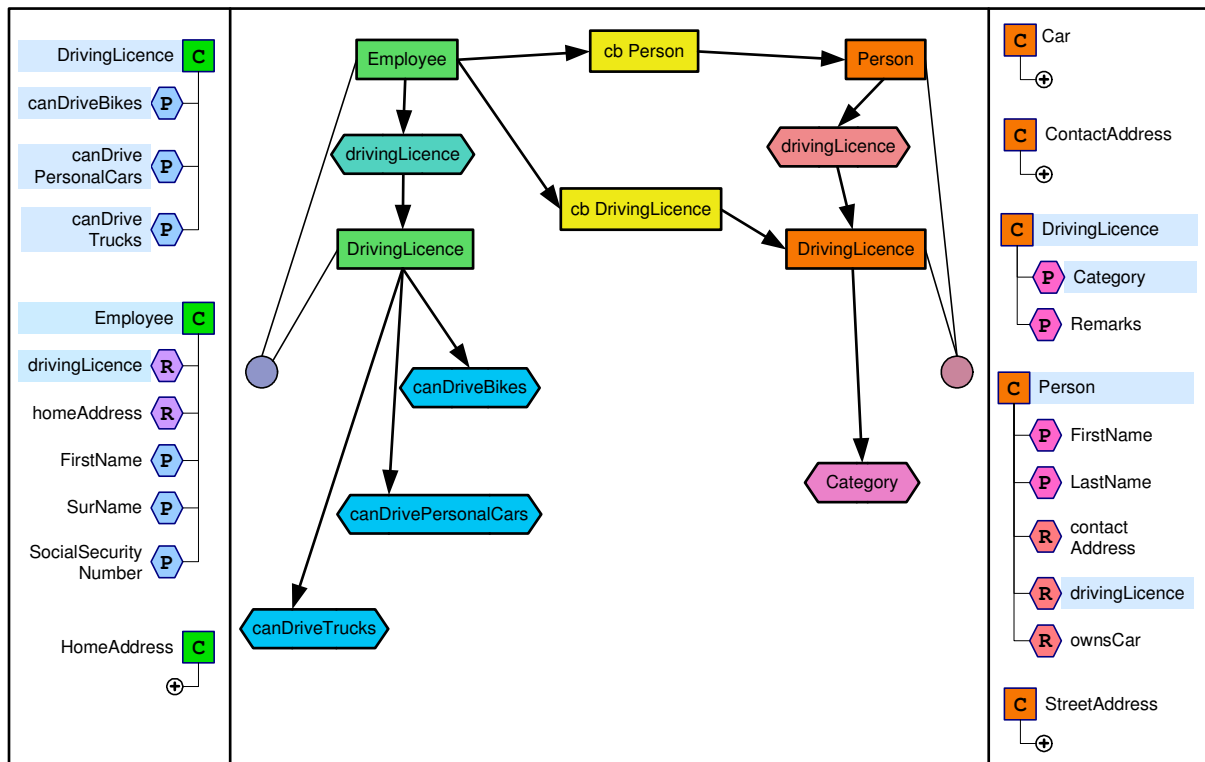
This exercise is a recap of the Extensional Specification topic and shows the transformation of boolean attributes into value attributes where the values must be specified separately.

Tasks

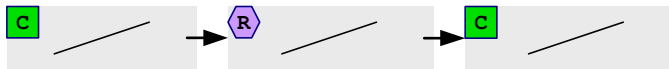
The attributes canDriveBikes, canDrivePersonalCars, and canDriveTrucks are special attributes: their absence in the XML file to map equals a value of "false".

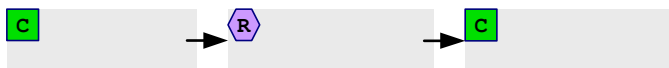
1. Define a meaningful Extensional Specification for the Concept Bridge "cb DrivingLicense".
2. Map the attributes canDriveBikes, canDrivePersonalCars, and canDriveTrucks! Don't forget about a meaningful condition and an adequate attribute value on the target side.

Mapping Figure



Bridge properties

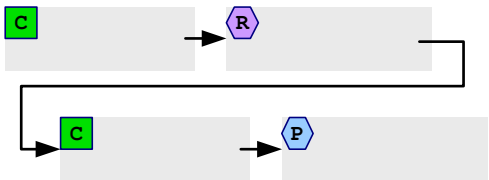
| | | |
|-----------|---|---|
| cb Person | Extensional Specification (Conditions for Execution):  | <input type="radio"/> Exists <input type="radio"/> Equals Value: |
|-----------|---|---|

| | | |
|-------------------|---|---|
| cb DrivingLicence | Extensional Specification (Conditions for Execution):  | <input type="radio"/> Exists <input type="radio"/> Equals Value: |
|-------------------|---|---|

| | |
|---------------|--|
| pb Category A | |
| pb Category B | |
| pb Category C | |

| | | |
|---------------|--|---|
| pb Category A | <input type="radio"/> Copy Attribute <input type="radio"/> Split <input type="radio"/> Constant Literal <input type="radio"/> Concatenate <input type="radio"/> Value Translation | Literal Value or Split separator: <input type="text"/> |
|---------------|--|---|

Note:
In Austria the driving licence category „A“ indicates that the driver may drive motorbikes.

| | |
|---|---|
| Execution Condition:  | <input type="radio"/> Exists <input type="radio"/> Equals Value: |
|---|---|

Euromuse.net – Harmonise manual

| | | |
|--|--|--|
| <div style="border: 1px solid black; padding: 2px; display: inline-block;">pb Category B</div> | <input type="radio"/> Copy Attribute <input type="radio"/> Split <input type="radio"/> Constant Literal <input type="radio"/> Concatenate <input type="radio"/> Value Translation | Literal Value or Split separator: <input style="width: 100%; height: 20px;" type="text"/> |
|--|--|--|

Note:
In Austria the driving licence category „B“ indicates that the driver may drive personal cars.

Execution Condition:

Exists
 Equals Value:

| | | |
|--|--|--|
| <div style="border: 1px solid black; padding: 2px; display: inline-block;">pb Category C</div> | <input type="radio"/> Copy Attribute <input type="radio"/> Split <input type="radio"/> Constant Literal <input type="radio"/> Concatenate <input type="radio"/> Value Translation | Literal Value or Split separator: <input style="width: 100%; height: 20px;" type="text"/> |
|--|--|--|

Note:
In Austria the driving licence category „C“ indicates that the driver may drive motor trucks.

Execution Condition:

Exists
 Equals Value:

Exercise 5

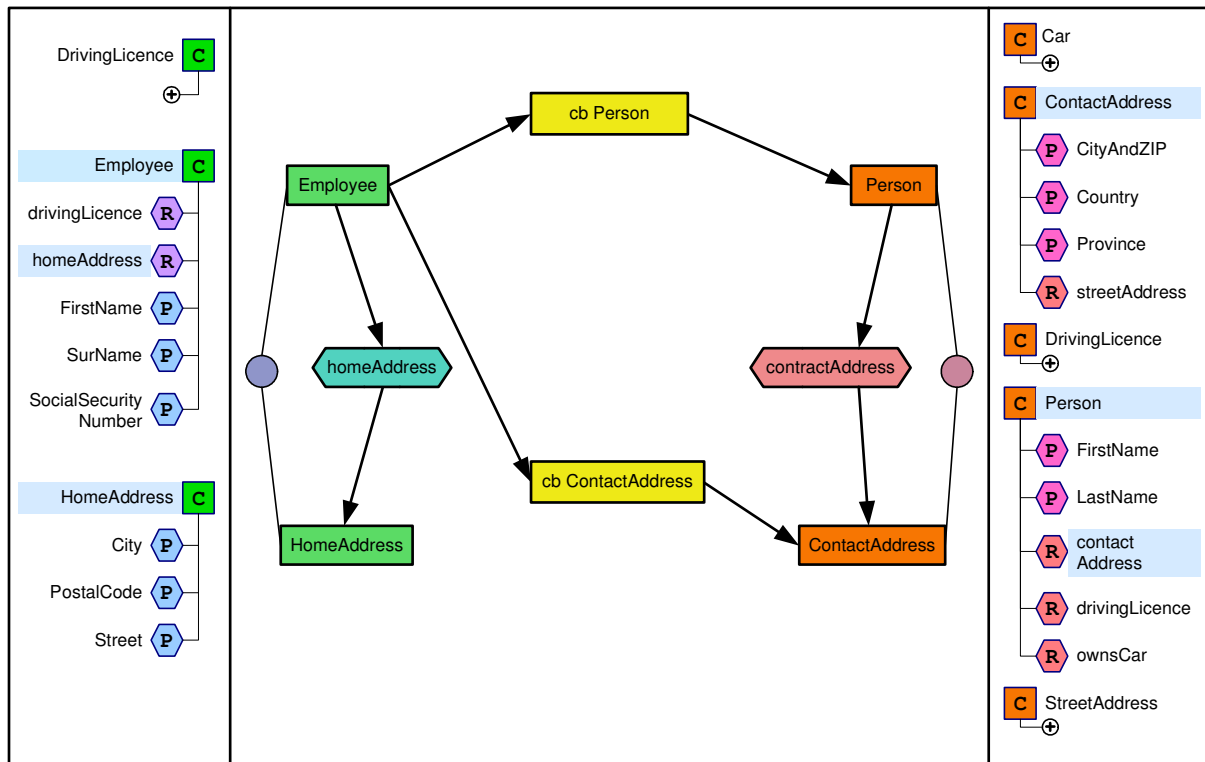
This exercise carrying out depends on whether the presenter is in time to also explain the concept of relations between concepts and the mapping of these relations. This exercise shows, as a result, how to create relation mappings.

Tasks

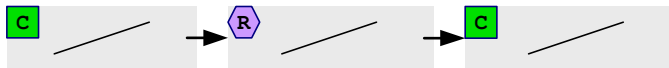
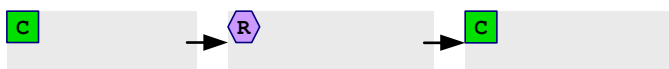
1. Create a relation mapping that reproduces the source side homeAddress relation on the target side.

Hint: A relation mapping needs two bridges: a Concept Bridge and a "related" Property Bridge with the correct Transformation Service.

Mapping Figure



Bridge properties (Exercise 5)

| | | |
|-------------------|--|--|
| cb Person | Extensional Specification (Conditions for Execution):  | <input checked="" type="radio"/> Exists <input type="radio"/> Equals Value: |
| cb ContactAddress | Extensional Specification (Conditions for Execution):  | <input type="radio"/> Exists <input type="radio"/> Equals Value: |
| pb CopyRelation | <input type="radio"/> Copy Attribute <input type="radio"/> Split <input type="radio"/> Constant Literal <input type="radio"/> Concatenate <input type="radio"/> Value Translation <input type="radio"/> Copy Relation | Literal Value or Split separator: <input type="text"/> Link to Concept Bridge: (for Copy Relation) <div style="background-color: yellow; text-align: center; padding: 2px;">cb</div> |

Installation of mappings

After mappings have been created they eventually have to be uploaded to the Harmonise platform, which can be managed there following this process:

1. Prepare a XSLT file according to the mapping rules and store it locally
2. Go to the Harmonise platform (currently <http://euromuse.ectrldev.com:8090>)
3. Login using your login name and password
4. Go the „Mappings“ section
5. Upload a new XSLT file (document with mapping rules)

The exact process will be shown in the trainings.

3. Use of online platform

Sending of data

1. Prepare the XML data file from your system and store it locally
2. Go to the Harmonise platform (currently <http://euromuse.ectrldev.com:8090>)
3. Login using your login name and password
4. Go the „Send data“ section
 - a. Upload the XML-file stored locally
 - b. Select a receiver
 - c. Click on send

Euromuse.net – Harmonise manual

5. The process is stored in the „Outbox“ section until it commenced successfully, afterwards it is stored in the „Sentbox“
6. The recipients gets informed automatically

The exact process will be shown in the trainings.

Receiving of data

1. You receive an email notification about a new file to be picked up
2. Go to the Harmonise platform (currently <http://euromuse.ectrldev.com:8090>)
3. Login using your login name and password
4. Go the „Inbox“ section
5. Select the file you wish to download

The exact process will be shown in the trainings.

Querying data

1. Go to the Harmonise platform (currently <http://euromuse.ectrldev.com:8090>)
2. Login using your login name and password
3. Go the „Query“ section
4. Select your query criteria
5. Click on „Query“ and you receive the list of results
6. Change the query criteria to refine the search

The exact process will be shown in the trainings.

4. The Application Programming Interface

Introduction

Goal of this chapter is to specify the application programming interface (API) to be implemented by a participant to the Euromuse network for:

- a) receiving data from the other participants through the Harmonise interface and
- b) supporting queries of Euromuse contents which can be posed by external actors through the Harmonise interface.

Scenarios

The API to be implemented are:

- Send API
- Query API

These are two HTTP post invocations which will be called by Harmonise to interact with Euromuse.

The **SendAPI** is called by Harmonise to send content to Euromuse. Euromuse should implement this HTTP call handler to get the XML file (passed as parameter) containing the sent content and store it in the CMS.

The **QueryAPI** is called by Harmonise to query the Euromuse content. This call should return the contents matching the query passed as XML.

SendAPI

The file is sent from Harmonise to Euromuse using a standard http multipart post request.

Euromuse.net – Harmonise manual

The parameters sent are:

- **senderid** : Harmonise id of the sender
- **sender** : name of the sender
- **semail** : email of the sender
- **filename** : name of the file to be uploaded
- **notes** : some annotation (optional)
- **xml_file** : the xml file to be uploaded (see Chapter 0)

The URL implemented by the receiver should be like:

<http://euromuse.ectrldev.com:8090/harmonise/sendfile>.

Harmonise will call the URL by passing the above mentioned parameters.

The invocation of Harmonise to Euromuse for sending content, from the Euromuse point of view, will look like a submit of the following form:

```
<form method="post"
  action="http://euromuse.ectrldev.com:8090/harmonise/sendfile"
  enctype="multipart/form-data">

  Sender id:      <input type="text" name="senderid" value="123">
  Sender name:    <input type="text" name="sender" value="Museum A">
  Sender email:   <input type="text" name="semail" value="museum@a.com">
  File name:      <input type="text" name="filename" value="File 1">
  Notes:          <input type="text" name="notes" value="My notes">
  XML File:       <input type="file" name="xml_file">

  <input type="submit" name="import" value="submit">
</form>
```

QueryAPI

For querying Euromuse, an xml file with an example of the “product“ will be sent along with other parameters to **Euromuse (Query By Example approach)**.

The file is sent from Harmonise to Euromuse using a standard http multipart post request.

The parameters sent are:

- **senderid** : id of the sender of the query
- **sender** : sender of the query
- **semail** : email of the sender
- **language** : language of the query
- **limit** : max number of results returned (an empty string means no restrictions)
- **xml_query** : the xml file containing a sample item used as „query by example“ (see Chapter 0)

The URL implemented by the receiver should be like:

<http://euromuse.ectrldev.com:8090/harmonise/query>.

Harmonise will call the URL by passing the above mentioned parameters.

Euromuse should return an XML file containing all items matching the query by example passed as parameter.

The invocation of Harmonise to Euromuse for querying content, from the Euromuse point of view, will look like a submit of the following form:

Euromuse.net – Harmonise manual

```
<form method="post"
  action="http://euromuse.ectrldev.com:8090/harmonise/query"
  enctype="multipart/form-data">

  Sender id:      <input type="text" name="senderid" value="123">
  Sender name:    <input type="text" name="sender" value="visiteurope.com">
  Sender email:   <input type="text" name="semail"
                  value="info@visiteurope.com ">
  Language:      <input type="text" name="language" value="en">
  Limit:         <input type="text" name="limit" value="10">
  XML File:      <input type="file" name="xml_query">

  <input type="submit" name="export" value="submit">
</form>
```

xml_query file

The format of the XML describing the query should be specified by the participant who receives the query, i.e. in this case by Euromuse, in order to easily map the XML query file with a real query to their CMS.

Harmonise will translate the query from a common query language to the local query language of Euromuse through the Reconciliation Engine.

Here below are the possible elements which can be queried:

- name of the exhibition
- description
- date range (from-to)
- location (country, city)
- category

On the basis of these search criteria the **xml_query** file parameter of the QueryAPI call may look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<webservice>
  <exhibitions>
    <exhibition>
      <name>name</name>
      <description>description</description>
      <date_start>YYYYMMDD</date_start>
      <date_end>YYYYMMDD</date_end>
      <location>
        <location_country>location_country</location_country>
        <location_town>location_town</location_town>
      </location>
      <categories>
        <category_id>category_id</category_id>
      </categories>
    </exhibition>
  </exhibitions>
</webservice>
```

Where:

1. the search values contained in the descriptive fields (name, description) should interpreted as keywords (i.e. a query should return all exhibitions whose name *contains* the value specified in the name parameter and not only the ones whose

Euromuse.net – Harmonise manual

- name *is exactly* that one) and should be considered together with the language parameter
2. date_start and date_end should be interpreted as a date range (i.e. respectively as date from and date to)
 3. location_country, location_town and category_id are identifiers taken from Euromuse data model

XML for Exhibitions – Structure of Euromuse sent data

The following data structure describes the data file sent to Euromuse through the Send API call.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<webservice>
  <exhibitions>
    <exhibition>
      <foreignId>uid</foreignId>
      <organiserid>organiserid</organiserid>
      <museumid>museumid</museumid>
      <locationid>locationid</locationid>
      <language>language</language>
      <name>titel</name>
      <name_en>titel_en</name_en>
      <subtitle>subtitle</subtitle>
      <subtitle_en>subtitle_en</subtitle_en>
      <shortdescription>shortdescription</shortdescription>
      <shortdescription_en>shortdescription_en</shortdescription_en>
      <description>description</description>
      <description_en>description_en</description_en>
      <date_start>YYYYMMDD</date_start>
      <date_end>YYYYMMDD</date_end>
      <permanent>permanent</permanent>
      <links>
        <link>link</link>
        <link_visitor>link_visitor</link_visitor>
      </links>
      <opening_time>
        <mon>
          <start>hh:mm</start>
          <end>hh:mm</end>
        </mon>
        <tue>
          <start>hh:mm</start>
          <end>hh:mm</end>
        </tue>
        <wed>
          <start>hh:mm</start>
          <end>hh:mm</end>
        </wed>
        <thu>
          <start>hh:mm</start>
          <end>hh:mm</end>
        </thu>
        <fri>
          <start>hh:mm</start>
          <end>hh:mm</end>
        </fri>
        <sat>
          <start>hh:mm</start>
          <end>hh:mm</end>
        </sat>
      </opening_time>
    </exhibition>
  </exhibitions>
</webservice>
```

```

    <sun>
      <start>hh:mm</start>
      <end>hh:mm</end>
    </sun>
    <information>information</information>
  </opening_time>
  <tickets>
    <full>full</full>
    <group_tarif>group_tarif</group_tarif>
    <group_info>group_info</group_info>
  </tickets>
  <categories>
    <category_id>category_id</category_id>
  </categories>
  <images>
    <image>
      <path>image_path</path>
      <type>logo|small|medium</type>
      <description>description</description>
      <copy>copy</copy>
    </image>
  </images>
</exhibition>
</exhibitions>
</webservice>

```

XML for Exhibitions – Structure of Euromuse result set

The following data structure describes the result set of the returned items matching the query sent to Euromuse through the Query API.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<webservice>
  <exhibitions>
    <exhibition>
      <foreignId>uid</foreignId>
      <language>language</language>
      <name>titel</name>
      <name_en>titel_en</name_en>
      <subtitle>subtitle</subtitle>
      <subtitle_en>subtitle_en</subtitle_en>
      <shortdescription>shortdescription</shortdescription>
      <shortdescription_en>shortdescription_en</shortdescription_en>
      <description>description</description>
      <description_en>description_en</description_en>
      <date_start>YYYYMMDD</date_start>
      <date_end>YYYYMMDD</date_end>
      <permanent>permanent</permanent>
      <links>
        <link>link</link>
        <link_visitor>link_visitor</link_visitor>
      </links>
      <museum>museum</museum>
      <location>
        <location_name>location_name</location_name>
        <location_name_en>location_name_en</location_name_en>
        <location_country>location_country</location_country>
        <location_plz>location_plz</location_plz>
        <location_town>location_town</location_town>
        <location_street>location_street</location_street>
        <location_info>location_info</location_info>
        <latitude>latitude</latitude>
      </location>
    </exhibition>
  </exhibitions>
</webservice>

```

```

    <longitude>longitude</longitude>
</location>
<opening_time>
  <mon>
    <start>hh:mm</start>
    <end>hh:mm</end>
  </mon>
  <tue>
    <start>hh:mm</start>
    <end>hh:mm</end>
  </tue>
  <wed>
    <start>hh:mm</start>
    <end>hh:mm</end>
  </wed>
  <thu>
    <start>hh:mm</start>
    <end>hh:mm</end>
  </thu>
  <fri>
    <start>hh:mm</start>
    <end>hh:mm</end>
  </fri>
  <sat>
    <start>hh:mm</start>
    <end>hh:mm</end>
  </sat>
  <sun>
    <start>hh:mm</start>
    <end>hh:mm</end>
  </sun>
  <information>information</information>
</opening_time>
<tickets>
  <full>full</full>
  <group_tarif>group_tarif</group_tarif>
  <group_info>group_info</group_info>
</tickets>
<categories>
  <category_id>category_id</category_id>
</categories>
<organiser>
  <name>organiser</name>
  <name_en>organiser_en</name_en>
  <link>link</link>
</organiser>
<images>
  <image>
    <path>image_path</path>
    <type>logo|small|medium</type>
    <description>description</description>
    <copy>copy</copy>
  </image>
</images>
<date_time_changed>YYYYMMDDhhmmss</date_time_changed>
</exhibition>
</exhibitions>
</webservice>

```