

# CSU22041 XML Report

---

## Table of Contents

<b>1. Introduction.....</b>	<b>1</b>
1.1. Background Research .....	1
1.2. Communication and Software.....	2
1.3. Work Distribution.....	2
<b>2. XML Design.....</b>	<b>4</b>
2.1. Changes when going from UML to XML.....	4
2.2 XML & DTD documents.....	5
2.3 Strengths and Weaknesses of the XML design.....	14
<b>3. XQuery Design.....</b>	<b>15</b>
3.1 XQueries, supporting UML Use-Cases & their purposes.....	15
3.2 Strengths and Weaknesses of the XQuery design.....	23
<b>4. Reflections and Sources.....</b>	<b>24</b>
4.1. Areas we could improve upon.....	24
4.2. Unavoidable Issues.....	24
4.3. Sources and References.....	24

---

## **1. Introduction**

### **1.1 Background Research**

Before we started the XML implementation for our domain, we were anxious about the intricacies of XML. We were confident with our UML project, and we wanted to make sure that we could properly transition from UML to XML.

For the first few group meetings, we talked about possible problems that we may encounter when we transition from UML to XML. We also discussed the possible changes that we may need to make when we transfer to XML. Luckily, we were given many examples of XML and were well prepared to make the transition.

Furthermore, there were many online resources that helped us for things like XQueries. We researched possible XQueries before beginning our implementation so that we could set a good foundation before we started.

## **1.2 Communication and Software**

**Group Communication:** Communication was pretty simple since we already had a foundation from our UML project. We continued to use Facebook Messenger to keep in contact between our main group meetings on Blackboard Collaborate.

**Cloud Platform:** We continued to use Google Docs and we frequently uploaded all of our work into our Google Drive Folder for this project. This worked seamlessly because we had already done this for our UML project.

**Software:** The software we used is “BaseX”, which is an excellent XML database engine, and allowed us to complete this project seamlessly.

- We were given information about basic BaseX Operations from Babatunji Omoniwa, which set us up nicely for learning about the software.
- Resources for using BaseX were very accessible online.
- After completing the XML implementation, we were glad we used BaseX.

## **1.3 Work Distribution**

A crucial thing that we wanted to have in place from the very beginning was a clear and fair delegation of tasks and responsibilities. To do this we did two things.

1. We choose not to elect a group leader so that all of our members inputs were equal and fair
2. That we meet up in a call to check each other’s progress each week and to decide next week’s goals and who should take on that responsibility.

We felt that by doing this in such a transparent way, we could eliminate any feelings of injustice and unfairness. Down below is the workload we distributed amongst ourselves across the weeks.

**Week 1:**

Member	Task(s)
Luke Seckerson	Writing XML documents
Brian Sharkey	Writing XML documents
April Sheeran	Researching possible xQueries
Prathamesh Sai	Researching possible xQueries
Tom Roberts	Research

**Week 2:**

Member	Task(s)
Luke Seckerson	Finishing XML documents and internal DTD's
Brian Sharkey	Finishing XML document and writing DTD's
April Sheeran	Developing two xQueries
Prathamesh Sai	Developing two xQueries
Tom Roberts	Research

**Week 3:**

Member	Task(s)
Luke Seckerson	Finishing DTD's
Brian Sharkey	Finishing DTD's
April Sheeran	Developing two more xQueries
Prathamesh Sai	Developing two more xQueries
Tom Roberts	Research

**Week 4:**

Member	Task(s)
Luke Seckerson	Final changes to XML documents
Brian Sharkey	Final changes to XML documents
April Sheeran	Finishing xQueries
Prathamesh Sai	Finishing xQueries
Tom Roberts	Research

**Week 5:**

Member	Task(s)
Luke Seckerson	Writing Report
Brian Sharkey	Writing Report
April Sheeran	Writing Report
Prathamesh Sai	Writing Report
Tom Roberts	Writing report

## 2. **XML Design**

### **2.1 What did we need to change when going from a UML design to an XML implementation?**

We only needed to **occasionally** make changes to our UML classes to suit the new XML format for this project. The changes we did need to make were **mostly minor** such as **adding new elements/attributes** for added detail or to meet the requirements for the XML documents.

Some attributes of the UML classes were not broken down into detail (**due to complexity**) but in the new XML format it was possible to express these in their intended form. Some of the changes we made when going from UML to XML is mentioned with each XML document description below.

## 2.2 XML and DTD documents

### Domestic Customer

This XML document includes the **customer** and **domesticCustomer** from our UML Class Diagram. Some of the changes going from UML to XML include the following:

- The addition of **IBAN as an attribute** to the customer class.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE DomesticCustomer_info
<!-- Can have zero or more Domestic customers -->
[<!ELEMENT DomesticCustomer_info (DomesticCustomer*)>

<!-- Each of the domestic customers must have one and only one of these
elements-->
  <!ELEMENT DomesticCustomer (Customer, paperlessBilling, unmissableRewards,
customerName, homeAddress)>

  <!-- Customer is used for every type of customer-->
<!-- each customer must have one of these elements-->
  <!ELEMENT Customer (Customer.accountNumber, Customer.email, Customer.mprn,
Customer.contactNumber)>
<!-- The customer's IBAN for billing purposes-->
  <!ATTLIST Customer IBAN CDATA #REQUIRED>
<!-- The customer's account number for identification purposes-->
  <!ELEMENT Customer.accountNumber (#PCDATA)>
<!-- The Customers email address for contacting them-->
  <!ELEMENT Customer.email (#PCDATA)>
<!-- The customer's mprn used to find the electricity meter at their property-->
  <!ELEMENT Customer.mprn (#PCDATA)>
<!-- The customers contact number for contacting them-->
  <!ELEMENT Customer.contactNumber (#PCDATA)>

<!-- If the customer has opted for paperless billing-->
  <!ELEMENT paperlessBilling (#PCDATA)>

<!-- if the customer has opted in for the unmissable rewards program-->
  <!ELEMENT unmissableRewards (#PCDATA)>

<!-- The customer may have zero or one title, one or more first names and one
and only one surname-->
  <!ELEMENT customerName (title?, firstname+, surname)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT firstName (#PCDATA)>
  <!ELEMENT surname (#PCDATA)>

<!-- The customers home address, it may have one or more street names and one
and only one city and county elements-->
  <!ELEMENT homeAddress (street+, city, county)>
  <!ELEMENT street (#PCDATA)>
  <!ELEMENT city (#PCDATA)>
  <!ELEMENT county (#PCDATA)>

]>
```

```

<!-- This document stores all the domestic customers -->
<DomesticCustomer_info>

<!-- Contains all the necessary information of a domestic customer-->
  <DomesticCustomer>

    <!-- A parent class that contains necessary information of a customer-->
    <Customer IBAN="IE1234AIB123432432">
      <Customer.accountNumber>123345567</Customer.accountNumber>
      <Customer.email>johnWill@email.com</Customer.email>
      <Customer.mprn>12345678911</Customer.mprn>
      <Customer.contactNumber>0018532123</Customer.contactNumber>
    </Customer>

    <paperlessBilling>true</paperlessBilling>
    <unmissableRewards>true</unmissableRewards>

    <customerName>
      <title>Mr</title>
      <firstName>John</firstName>
      <surname>Will</surname>
    </customerName>

    <homeAddress>
      <street>Kilsheelan</street>
      <city>Clonmel</city>
      <county>Tipperary</county>
    </homeAddress>
  </DomesticCustomer>
</DomesticCustomer_info>

```

## Business Customer

This XML document includes the **customer** and the **businessCustomer** from our UML Class Diagram. Some of the changes going from UML to XML include the following

- **The IBAN attribute** was added to the XML document.

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE BusinessCustomer_info
<!-- can have multiple businessCustomers in this document -->
[<!ELEMENT BusinessCustomer_info (BusinessCustomer*)>

<!-- Each businessCustomer must have one of these elements -->
  <!ELEMENT BusinessCustomer (Customer, sizeOfBusiness, meterTypeElectricity,
meterTypeGas, billFrequency, businessName, businessLocation)>

<!-- Customer is used for every type of customer-->
<!-- each customer must have one of these elements-->
  <!ELEMENT Customer (Customer.accountNumber, Customer.email, Customer.mprn,
Customer.contactNumber)>
<!-- The customer's IBAN for billing purposes-->

```

```

<!ATTLIST Customer IBAN CDATA #REQUIRED>
<!-- The customer's account number for identification purposes-->
<!ELEMENT Customer.accountNumber (#PCDATA)>
<!-- The Customers email address for contacting them-->
<!ELEMENT Customer.email (#PCDATA)>
<!-- The customer's mprn used to find the electricity meter at their
property-->
<!ELEMENT Customer.mprn (#PCDATA)>
<!-- The customers contact number for contacting them-->
<!ELEMENT Customer.contactNumber (#PCDATA)>

<!-- the size of the business as different sizes get different deals-->
<!ELEMENT sizeOfBusiness (#PCDATA)>
<!-- The type of electricity meter they have installed-->
<!ELEMENT meterTypeElectricity (#PCDATA)>
<!-- The type of gas meter they have installed-->
<!ELEMENT meterTypeGas (#PCDATA)>
<!-- How frequently their billed-->
<!ELEMENT billFrequency (#PCDATA)>
<!-- The business's name-->
<!ELEMENT businessName (#PCDATA)>
<!-- The business's location,can have one or more streets apart of their
address and one and only one city and county→

<!ELEMENT businessLocation (street+, city, county)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT county (#PCDATA)>
]>
<!-- This document stores all the business customers -->
<BusinessCustomer_info>
<!-- Contains all the necessary information of a business customer-->
<BusinessCustomer>
  <!-- A parent class that contains necessary information of a customer-->
  <Customer IBAN="IE9876AIB987654674">
    <Customer.accountNumber>1123456790</Customer.accountNumber>
    <Customer.email>Argos.Dalkey@email.com</Customer.email>
    <Customer.mprn>09439593201</Customer.mprn>
    <Customer.contactNumber>0334325876</Customer.contactNumber>
  </Customer>
  <sizeOfBusiness>medium</sizeOfBusiness>
  <meterTypeElectricity>standard</meterTypeElectricity>
  <meterTypeGas>standard</meterTypeGas>
  <billFrequency>monthly</billFrequency>
  <businessName>Argos</businessName>
  <businessLocation>
    <street>45 Castle st Dalkey</street>
    <city>Dalkey</city>
    <county>Dublin</county>
  </businessLocation>
</BusinessCustomer>
</BusinessCustomer_info>

```

## Repairs

This XML document is a representation of the **repairs class** from our UML class diagram. Some of the changes going from UML to XML include the following:

- Adding **more details for repairs** such as the name of the **repairman**, the **profession** and **contact information for the repairman**.
- Adding **more elements to the client's address** and allowing **flexibility for location**.

```
<?xml version='1.0' encoding='ISO-8859-1' standalone= "no" ?>
<!DOCTYPE repairs [
  <!-- can have multiple clients in this class -->
  <!ELEMENT repairs (client*)>
  <!ATTLIST client type CDATA #IMPLIED>

  <!-- name is not necessarily needed -->
  <!-- allows flexibility for details (0 or more) -->
  <!-- location is not known unless a repair has been made -->
  <!-- contact not known until a repair has been made -->
  <!ELEMENT client (name?, details*, location?, contact?)>

  <!-- can have a title or not -->
  <!-- clients have a first name and last name -->
  <!ELEMENT name (title?, first_name, surname)>

  <!-- some repairs take multiple days-->
  <!-- cost of repair is total -->
  <!-- can have more than one issue e.g. leaky pipes and broken boiler-->
  <!-- reported repair issue can be not accurate -> more tradesperson-->
  <!ELEMENT details (date_of_repair+, cost_of_repair
,repair_issue+, trade_person+) >

  <-- rare but some trade persons may have more than 1 trade -->
  <-- can have multiple numbers e.g. mobile and home -->
  <!ELEMENT trade_person (trade+, number*)>
  <!ATTLIST trade_person name CDATA #IMPLIED>

  <-- can have eircode or not -->
  <-- flexibility for address details -->
  <!ELEMENT location (Eircode?, street?, town?, city?)><!ATTLIST location
proximity CDATA #IMPLIED>

  <-- can have 0 or multiple emails depending on contact method -->
  <-- can have 0 or multiple numbers depending on contact method -->
  <!ELEMENT contact (email*, client_number*)>

  <!ELEMENT title (#PCDATA)>
  <!ELEMENT first_name (#PCDATA)>
  <!ELEMENT surname (#PCDATA)>

  <!ELEMENT date_of_repair (#PCDATA)>
  <!ELEMENT cost_of_repair (#PCDATA)>
  <!ELEMENT repair_issue (#PCDATA)>
  <!ELEMENT trade (#PCDATA)>
```



```
<!ELEMENT number (#PCDATA)>

<!ELEMENT Eircode (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT town (#PCDATA)>
<!ELEMENT city (#PCDATA)>

<!ELEMENT email (#PCDATA)>
<!ELEMENT client_number (#PCDATA)>
]>

<repairs>

<client type= "Domestic"> <!--type tells us if it's domestic or business -->

  <!-- more information on the name -->
  <name>
    <title> Mr. </title>
    <first_name> Luke </first_name>
    <surname> Seckerson </surname>
  </name>

  <!-- Main bulk of 'repairs' can have multiple repairs -->
  <details>
    <date_of_repair> 12/3/2019</date_of_repair>
    <cost_of_repair> 30.00 </cost_of_repair>
    <repair_issue> Broken Boiler </repair_issue>
    <trade_person name = "Peter">
      <trade> Plumber </trade>
      <number> 087-394-8789 </number>
    </trade_person>
  </details>

  <!-- location, different prices depending on location -->
  <location proximity = "close">
    <Eircode> A61C3FE </Eircode>
    <street> random Street </street>
    <town> Dundalk </town>
    <city> </city>
  </location>

  <!-- stores all the contact info available to contact client about repairs
  -->

  <contact>
    <!-- email is indented like that for correct nesting -->

    <email status = "subscribed">
      l.seckerson@gmail.com
    </email>
    <client_number> 086-111-2345 </client_number>
    <Eircode> A61C3FE </Eircode>
  </contact>
</client>
</repairs>
```

## Home Energy

This XML document is from the UML class diagram **'Home Energy Plan'**. We felt that this one was necessary as it is the primary function of Bord gais. Some changes that were made during the transition to UML to XML:

- **Specified details of contracts.** In the UML class it wasn't detailed regarding the dates of the contracts.

```
<?xml version='1.0' encoding='ISO-8859-1' standalone= "no" ?>

<!DOCTYPE home_energy [

<!-- can have multiple customers -->
<!ELEMENT home_energy (customer*)>

<!--can have only one payment plan, may have a discount or not-->
<!ELEMENT customer (name, payment_plan, discount?)>

<!-- may have a title or not -->
<!-- flexibility for first name -->
<!ELEMENT name (title?, first_name+, surname)>

<!--depending on type of contract there may be monthly bills or not-->
<!--depending on type of contract there may be an exit fee or not-->
<!ELEMENT payment_plan (length_of_contract, estimated_annual_bill,
estimated_monthly_bill?, contract_dates,
early_exit_fee?)>

<!ATTLIST payment_plan type CDATA #IMPLIED>
<!ATTLIST estimated_monthly_bill currency CDATA #IMPLIED>
<!ATTLIST estimated_annual_bill currency CDATA #IMPLIED>

<!ELEMENT contract_dates (contract_start_date, contract_end_date)>

<!--depending on type of contract can have 'marginal' discount-->
<!ELEMENT discount (discount_percentage*, length_of_discount*)>
<!ATTLIST discount status CDATA #IMPLIED>

<!ELEMENT title (#PCDATA)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT surname (#PCDATA)>

<!ELEMENT length_of_contract (#PCDATA)>
<!ELEMENT estimated_annual_bill (#PCDATA)>
<!ELEMENT estimated_monthly_bill (#PCDATA)>
<!ELEMENT contract_start_date (#PCDATA)>
<!ELEMENT contract_end_date (#PCDATA)>
<!ELEMENT early_exit_fee (#PCDATA)>

<!ELEMENT discount_percentage (#PCDATA)>
<!ELEMENT length_of_discount (#PCDATA)>

]>
```

```
<home_energy>

<customer>

<!-- need to link to customer class (association)== name -->
  <name>
    <title>Mr. </title>
    <first_name> Luke </first_name>
    <surname>Seckerson</surname>
  </name>

  <!-- attribute describes type of contract -->
  <payment_plan type = "Electricity Savings plan">
    <length_of_contract >
      1 Year
    </length_of_contract>
    <estimated_annual_bill currency = "euro">
      550.32
    </estimated_annual_bill>
    <estimated_monthly_bill currency = "euro">
      46.23
    </estimated_monthly_bill>
    <contract_dates>
      <contract_start_date>
        1/2/2019
      </contract_start_date>
      <contract_end_date>
        1/2/2020
      </contract_end_date>
    </contract_dates>
    <early_exit_fee>
      100.00
    </early_exit_fee>
  </payment_plan>

  <!-- depending on different factors discounts can be applied or not-->
  <discount status = "applied">
    <discount_percentage>
      2.2%
    </discount_percentage>
    <length_of_discount>
      2 months
    </length_of_discount>
  </discount>

</customer>

</home_energy>
```

## Account

This is from the UML class 'Account' from our UML class diagram. This particular class is also a pivotal class in our diagram so we felt it was necessary to include it.

```
<?xml version='1.0' encoding='ISO-8859-1' standalone= "no" ?>

<!DOCTYPE repairs [

<!ELEMENT Accounts (customer*)>

<!-- customer can have a joint account -> more owners -->
<!-- starts with 0 bills then multiple bills -->
<!ELEMENT customer (accounts_owner+, account,
Bill*, meter_readings)>

<!-- may or may not have a title -->
<!-- owner(s) multiple 'names' due to existence of joint accounts -->
<!-- can have multiple numbers e.g. home and mobile -->
<!ELEMENT accounts_owner (title?, first_name+, surname+, date_account_created,
number*)>
<!ATTLIST accounts_owner type CDATA #IMPLIED>
<!ATTLIST number type CDATA #IMPLIED>

<!-- at start, nothing in balance -->
<!ELEMENT account (account_id, payment_method,
balance?, bank)>

<!ELEMENT Bill (bill_status+, total, date_of_bill)>
<!ATTLIST Bill name CDATA #IMPLIED>
<!ATTLIST total type CDATA #IMPLIED>

<!-- depending on contract type can have none, one or both-->
<!ELEMENT meter_reading (electric_reading?,
gas_reading?)>
<!ATTLIST electric_reading measurement CDATA #IMPLIED><!ATTLIST gas_reading
measurement CDATA #IMPLIED>

<!ELEMENT title (#PCDATA)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT surname (#PCDATA)>
<!ELEMENT date_account_created (#PCDATA)>
<!ELEMENT number (#PCDATA)>

<!ELEMENT account_id (#PCDATA)>
<!ELEMENT payment_method (#PCDATA)>
<!ELEMENT balance (#PCDATA)>
<!ELEMENT bank (#PCDATA)>

<!ELEMENT bill_status (#PCDATA)>
<!ELEMENT total (#PCDATA)>
<!ELEMENT date_of_bill (#PCDATA)>

<!ELEMENT electric_reading (#PCDATA)>
<!ELEMENT gas_reading (#PCDATA)>
```

```
]>
```

```
<Accounts>
```

```
<customer>
```

```
<!-- needs to be linked to the customer.xml c because of  
associations -->
```

```
<accounts_owner type = "Home">
```

```
<title> Mr. </title>
```

```
<first_name> Luke </first_name>
```

```
<surname> Seckerson </surname>
```

```
<date_account_created>
```

```
1/7/2020
```

```
</date_account_created>
```

```
<number type = "mobile">
```

```
094-232-1232
```

```
</number>
```

```
</accounts_owner>
```

```
<Account type = "Checking Account">
```

```
<account_id> 1123456789 </account_id>
```

```
<payment_method> Fixed </payment_method>
```

```
<balance> 34.23 </balance>
```

```
<bank> PTSB </bank>
```

```
</Account>
```

```
<Bill name = "Electric Bill">
```

```
<bill_status>Outstanding</bill_status>
```

```
<total type = "due"> 45.67 </total>
```

```
<date_of_bill> 14/9/2020 </date_of_bill>
```

```
</Bill>
```

```
<Bill name = "Electric Bill">
```

```
<bill_status>Paid</bill_status>
```

```
<total type = "amount paid">
```

```
32.58
```

```
</total>
```

```
<date_of_bill> 12/7/2020 </date_of_bill>
```

```
</Bill>
```

```
<meter_readings>
```

```
<electric_reading>
```

```
244 16
```

```
</electric_reading>
```

```
<gas_reading> </gas_reading>
```

```
</meter_readings>
```

```
</customer>
```

```
</Accounts>
```

## 2.3 Strengths and Weaknesses of the XML design

### Strengths

For XML documents, it is possible to add comments in any capacity. We noticed when creating the UML class diagram that there's only so many comments explaining design choices we can add before it becomes **visually overwhelming**. This is not a problem for XML documents, so one of the strengths of a XML design is that **there is more communication between reader and author of the documents**.

### Weaknesses

Unlike the UML class diagram it is **not easy to visualise how the classes interact with each other**. In an XML format it is **difficult to get a general understanding** of the system without reading the documents several times. However we understand that this is a trade off for detail and practicality for X-Queries.

### 3. XQuery Design

#### 3.1 xQueries, their supporting UML Use-Cases & their purposes

##### AverageAnnualBill.xq

```
declare function local:average_annual_bill_domestic(){
let $doc := doc("C:\BordGais\HomeEnergy.xml")

let $average := avg($doc/home_energy/customer/payment_plan/estimated_annual_bill)

return
<average_annual_bill>
{$average}
</average_annual_bill>
};

local:average_annual_bill_domestic()
```

- **Supporting UML Use-Case:** Pay Online & Direct Debit
- **Purpose of Query:** Present an **average figure** of how much customers are paying per year. This may be useful for research and comparing to competitors.

#### Example Output



## BusinessBankDetails.xq

```
for $j in doc("C:\BordGais\account.xml")/Accounts/customer
for $y in
doc("C:\BordGais\BusinessCustomer.xml")/BusinessCustomer_info/BusinessCustomer
where $j/Account/account_id = $y/Customer/Customer.accountNumber

return
<bankdetails>
{$y/businessName}
{$j/Account/bank}
<IBAN>
{data($y/Customer/@IBAN)}
</IBAN>
</bankdetails>
```

- **Supporting UML Use-Case:** Direct Debit
- **Purpose of Query:** For business customers, it returns the name of the business, the **bank** they are with, and their **IBAN**. It shows how they are paying through direct debit.

### Example Output

 1 Result, 122 b

Result

```
<bankdetails>
  <businessName>Argos</businessName>
  <bank>AIB</bank>
  <IBAN>IE9876AIB987654674</IBAN>
</bankdetails>
```



## NegativeBalances.xq

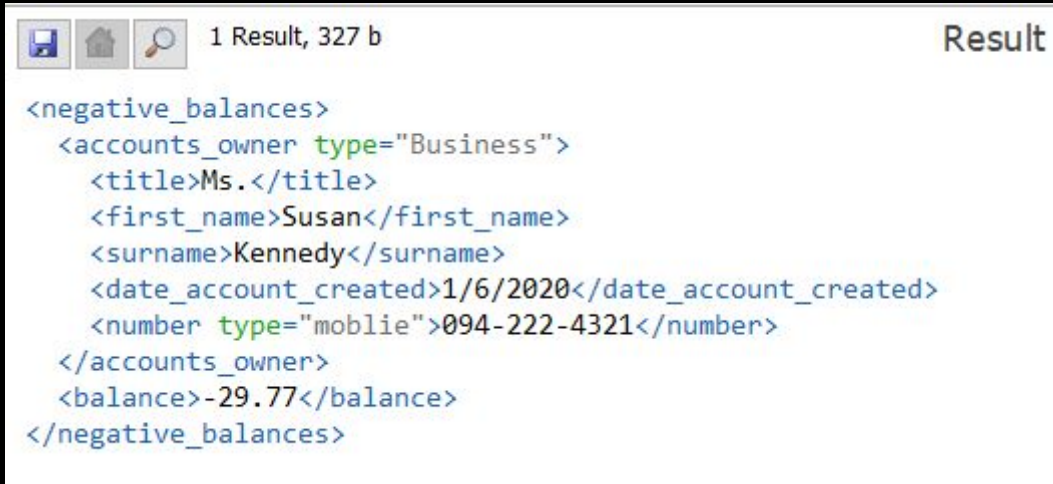
```
declare function local:find_negative_balances(){
  for $j in doc("C:\BordGais\account.xml")/Accounts/customer

  where $j/Account/balance < 0
  return
    <negative_balances>
      {$j/accounts_owner}
      {$j/Account/balance}
    </negative_balances>
};

local:find_negative_balances()
```

- **Supporting UML Use-Case:** Online Payment & Direct Debit
- **Purpose of Query:** Return the **details of the customers with negative balances** and their balance. This could be useful to Bord Gais to find the customers who are behind on payments.

### Example Output



The screenshot shows a web-based interface for displaying XQuery results. At the top, there are icons for file operations and a search icon, followed by the text "1 Result, 327 b". The word "Result" is displayed in the top right corner. The main area contains the XML output of the query, which is a single record for a customer with a negative balance.

```
<negative_balances>
  <accounts_owner type="Business">
    <title>Ms.</title>
    <first_name>Susan</first_name>
    <surname>Kennedy</surname>
    <date_account_created>1/6/2020</date_account_created>
    <number type="moblie">094-222-4321</number>
  </accounts_owner>
  <balance>-29.77</balance>
</negative_balances>
```

## PlanAndBillsOfCustomers.xq

```
for $j in doc("C:\BordGais\HomeEnergy.xml")/home_energy/customer
for $x in doc("C:\BordGais\account.xml")/Accounts/customer
where ($j/name/surname)=$x/accounts_owner/surname

return
<PlanAndBills>
{$j/name}
<plan_type>
{$j/payment_plan/@type/string(.)}
</plan_type>
{$x/Bill}
</PlanAndBills>
```

- **Supporting UML Use-Case:** Supply Gas, Supply Electricity, Online Payment, Direct Debit
- **Purpose of Query:** Show **who is receiving gas, electricity or both**. It returns the associated bill along with the customer details.

### Example Output

 1 Result, 529 b Result

```
<PlanAndBills>
  <name>
    <title>Mr.</title>
    <first_name>Luke</first_name>
    <surname>Seckerson</surname>
  </name>
  <plan_type>Electricity Saving's plan</plan_type>
  <Bill name="Electric Bill">
    <bill_status>Outstanding</bill_status>
    <total type="due">45.67</total>
    <date_of_bill>14/9/2020</date_of_bill>
  </Bill>
  <Bill name="Electric Bill">
    <bill_status>Paid</bill_status>
    <total type="amount paid">32.58</total>
    <date_of_bill>12/7/2020</date_of_bill>
  </Bill>
</PlanAndBills>
```

## MostExpensiveRepair.xq

```
let $xml := doc("C:\BordGais\Repairs.xml")

let $maxValue := max($xml/repairs/client/details/cost_of_repair)




for $j in
  doc("C:\BordGais\Repairs.xml")/repairs/client
where (
  $j/details/cost_of_repair = $maxValue
)

return
<most_expensive_repair>

<cost_of_repair>
{$maxValue}
</cost_of_repair>
{$j/name}
</most_expensive_repair>
```

- **Supporting UML Use-Case:** Do Repairs
- **Purpose of Query:** Find the **value of the most expensive repair**, and the name of the customer who requested that repair. This could be used by Bord Gáis to evaluate the most expensive repairs and record the customers who had to pay for them; hence allowing Bord Gáis to keep track of the most expensive repairs for future reference.

### Example Output

 1 Result, 205 b

Result

```
<most_expensive_repair>
  <cost_of_repair>210.23</cost_of_repair>
  <name>
    <title>Dr.</title>
    <first_name>Rachel</first_name>
    <surname>Powell</surname>
  </name>
</most_expensive_repair>
```

## SearchCustomersForInstallation.xq

```

for $j in
doc("C:\BordGais\DomestCustomer.xml")/DomesticCustomer_info/DomesticCustomer,
$x in
doc("C:\BordGais\BusinessCustomer.xml")/BusinessCustomer_info/BusinessCustomer
where (
  contains($j/customerName/firstName, "John")
  and
  contains($x/businessName, "Argos")
)

return
<Customers>
<Domestic_Customer>
{$j/customerName}
{$j/homeAddress}
{$j/Customer/Customer.accountNumber}
</Domestic_Customer>
<Business_Customer>
{$x/businessName}
{$x/businessLocation}
{$x/Customer/Customer.accountNumber}
</Business_Customer>
</Customers>

```

- **Supporting UML Use-Case:** Receive Installation
- **Purpose of Query:** Search domestic and business customers by their first name, and retrieve the county they reside in, their account number and their full name. With this information, Bord Gáis can send employees to their address to give them an installation.

## Example Output



1 Result, 706 b

Result

```

<Customers>
  <Domestic_Customer>
    <customerName>
      <title>Mr</title>
      <firstName>John</firstName>
      <surname>Will</surname>
    </customerName>
    <homeAddress>
      <street>Kilsheelan</street>
      <city>Clonmel</city>
      <county>Tipperary</county>
    </homeAddress>
    <Customer.accountNumber>123345567</Customer.accountNumber>
  </Domestic_Customer>
  <Business_Customer>
    <businessName>Argos</businessName>
    <businessLocation>
      <street>45 Castle st Dalkey</street>
      <city>Dalkey</city>
      <county>Dublin</county>
    </businessLocation>
    <Customer.accountNumber>123345500</Customer.accountNumber>
  </Business_Customer>
</Customers>

```

## RepairsOnACertainClient.xq

```
for $j in
doc("C:\BordGais\Repairs.xml")/repairs/client
where contains($j/name/first_name, "Rachel")
return
<customer_repair>

  {$j/name}

  {$j/details}

</customer_repair>
```

- **Supporting UML Use-Case:** Do Repairs
- **Purpose of Query:** Search customer repairs with their first name and retrieve all the details of their repair history such as their full name, date of repair(s), cost of repair(s), repair issue(s), name of the tradesperson who fixed their repair, the particular trade of that tradesperson, and the tradesperson's number. This can be used so that Bord Gáis can retrieve all the information about the repairs of a particular customer.

### Example Output

 1 Result, 699 b

Result

```
<customer_repair>
  <name>
    <title>Dr.</title>
    <first_name>Rachel</first_name>
    <surname>Powell</surname>
  </name>
  <details>
    <cost_of_repair>210.23</cost_of_repair>
    <date_of_repair>11/11/2020</date_of_repair>
    <repair_issue>Leaky pipes</repair_issue>
    <trade_person name="Bob">
      <number/>
      <trade>Plumber</trade>
    </trade_person>
  </details>
  <details>
    <cost_of_repair>140.03</cost_of_repair>
    <date_of_repair>29/11/2020</date_of_repair>
    <repair_issue>Faulty Electrics</repair_issue>
    <trade_person>
      <number>087-232-3234</number>
      <trade>Electrician</trade>
    </trade_person>
  </details>
</customer_repair>
```

## FindCustomerWithLengthOfContract.xq

```
for $j in
doc("C:\BordGais\HomeEnergy.xml")/home_energy/customer
where contains($j/payment_plan/length_of_contract, "3 Year")

return
<customer_with_changed_discount>
{$j/name}
{$j/payment_plan}
<discount status="applied">
<discount_percentage>8%</discount_percentage>
<length_of_discount>7 months</length_of_discount>
</discount>
</customer_with_changed_discount>
```

- **Supporting UML Use-Case:** Receive Rewards
- **Purpose of Query:** Search for a customer with a particular length of contract and return an output with a different discount percentage and discount length. This output can then be used as a foundation for another XML document.

### Example Output

 1 Result, 798 b

Result

```
<customer_with_changed_discount>
  <name>
    <title>Ms.</title>
    <first_name>Samus</first_name>
    <surname>Aran</surname>
  </name>
  <payment_plan type="Dual Fuel Plan">
    <length_of_contract>3 Year</length_of_contract>
    <estimated_annual_bill currency="euro">990.32</estimated_annual_bill>
    <estimated_monthly_bill currency="euro">88.90</estimated_monthly_bill>
    <contract_dates>
      <contract_start_date>23/5/2019</contract_start_date>
      <contract_end_date>24/5/2022</contract_end_date>
    </contract_dates>
    <early_exit_fee>500.00</early_exit_fee>
  </payment_plan>
  <discount status="applied">
    <discount_percentage>8%</discount_percentage>
    <length_of_discount>7 months</length_of_discount>
  </discount>
</customer_with_changed_discount>
```

### 3.2 Strengths and Weaknesses of our xQuery design

#### Strengths

- Our xQueries **are simple, yet effective**. During our discussions prior to developing our xQueries, we often brought up the point of keeping our xQueries simple yet useful. We wanted to make sure that each xQuery had a purpose, and that it fulfilled its purpose in the most straightforward way possible.
- Our xQueries are **based on real world scenarios**. Before making our xQueries, we decided to not only base our queries from our UML use-cases, but also base them off of real world scenarios. For example, **we thought that getting the customers with negative balances on their account would be a tangible query that would be required in the real world, hence we included it**; we decided to put more thought into our queries in this way.
- Our xQueries **use a variety of clauses and built-in xQuery functions** to make each query **efficient**. We met all the requirements for the clauses in our xQueries and decided to make use of many built-in xQuery functions that made our queries a lot easier and simpler to understand. Functions such as contains(), max(), and avg() made our queries extremely simple and easy to comprehend.

#### Weaknesses

- Although we wanted our xQueries to be informative, sometimes it was **not clear how much information we wanted to output for a particular query**. For example, in our xQuery files such as “RepairsOnACertainClient.xq”, we were unsure about whether our query outputted too much information which might not be needed; it was difficult to find the line between outputting too much information and outputting too little.
- **Sometimes we felt that some queries were repetitive at times**, but we tried to combat this by using a multitude of different functions (user-defined and in-built) so that we could bring some variety into our xQueries. We tried our best to make sure that each xQuery was unique, even if the clauses in each query were similar.



## 4.

**Reflections and Sources****4.1 Areas we could improve upon**

Nearing the end of the semester, we were stuck for time because of the workload from our assignments, therefore we wished that we could have spent more time discussing our XML implementation. Perhaps more group meetings could have fixed this.

Oftentimes we felt that our lack of experience in making an XML implementation was a disadvantage. We often had to discuss trivial things such as the layout of our XML documents and the amount of information in the output of an xQuery; these questions popped up as we designed our XML implementation. In the future we should discuss these topics beforehand, so we can save time when we are implementing our XML implementation so that we can advance our projects further.

We are happy with our XML implementation, however we found out that the learning curve was a bit hard at the beginning. However, with time and patience, we are confident that we will be able to carry forward the experiences and knowledge that we have gained from this project into our future projects.

**4.2 Unavoidable Issues**

As mentioned in 3.2, we felt that we could improve in some areas. However there are some challenges that we encountered that happened due to unavoidable circumstances or conditions.

- **Communication Issues:** Although we tried to stay in-contact with blackboard meetings and our Facebook Messenger group chat, communicating purely online didn't suit our style of work, as we benefit more from in-person meetings which couldn't happen due to Covid-19.
- **Stress:** As we reached the end of the semester, it was evident that most of our group members were stressed with many assignments. We agreed that it may have caused a decrease in quality of our XML implementation.
- **Circumstance:** One of our members had some important family commitments to take care of, which meant that we were left with less people to work on the project.

**4.3 Sources and References**

The following are the various links and references that helped us develop this project.

- [Our UML Group Report \(containing use-case diagrams\)](#)
- [Our XML & DTD documents, with xQueries](#)
- [An XML Example that helped our understanding of XML documents](#)
- [A resourceful XQuery website that helped our understanding of XQueries](#)