XML Report

A Comprehensive Exploration of XML, XPath, and XQuery

Liam Power

November 28, 2024

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

This document is submitted as part of the coursework for CSU22041 Information Management I

Team Members

Finn Clancy

Robin Schulz

Liam Power

Rachel

Nick

Roisin

Noah Scolard

1 Approach

1.1 Adjustments for Assignment Requirements

Some classes needed adjusting to meet the assignment requirements of having six elements in each document. We decided to add others for depth and understanding of the whole design. The events document was missing two elements, so we added `manager` and `short description`. Similarly, the artwork document was missing two elements, so we added `category` and `description`. This allowed us to create detailed XML documents without losing the general idea of our UML design.

1.2 XQuery Ideas

When coming up with ideas for interesting XQueries, we thought it would be beneficial to have a query to find out how many pieces of art the gallery currently has that fit in a specific category, such as "Post-Impressionism." This query will make it easier for curators to gauge if there are enough pieces of art available for their desired exhibition. This update forced us to adjust our artwork class to include a category variable to match the category elements in the XML DTD.

Another idea for an interesting XQuery was to have different ways of structuring important events. An event ID and a manager were mentioned. The ID existed in the use case diagram, but the manager did not. By adding this, we not only created the XML documents and their XQueries according to the assignment guidelines but also made our use cases more comprehensive. For instance, in this example, the relationship between external partners, events, and the administration of the Art Gallery was strengthened by adding a gallery manager overseeing the events.

1.3 Artist Class Adjustments

We added the following variables to the `Artist` class to ensure it matched the elements in the Artist XML DTD: `deathYear`, `specialty`, and `placeOfStudy`.

1.4 Curator Use Case

We also added a use case for the Curator called "Get Artwork For Each Artist," which will make the Curator's workflow more efficient. This allows them to gather information for potential exhibitions quickly or respond to inquiries from auctioneers, museums, and other galleries.

1.5 Group Processes

We held group meetings during the tutorial times to coordinate our efforts and ensure everyone was on the same page regarding project tasks. Meetings were scheduled during and after lecture times to maximize participation and accommodate different schedules. We brainstormed possible XML documents and XQueries to explore various approaches and solutions for our project.

The class diagram dimension was adjusted for the XML tasks to better align with the project requirements and improve design accuracy. Tasks were allocated among team members, and progress was discussed regularly to track our development and address any issues promptly. There was in-person collaboration on XML documents, which facilitated direct communication and quick problem-solving. Virtual collaboration occurred through a GitHub repository, allowing us to share updates efficiently and work asynchronously.

2 UML Design to XML Implementation

Some classes needed adjusting to meet the assignment requirements of having six elements in each document. We decided to add additional elements for depth and better understanding of the overall design. The events document was missing two elements, so we added "manager" and "short description." Similarly, the artwork document was

missing two elements, so we added "category" and "description." These additions allowed us to create detailed XML documents without losing the general idea of our UML design.

When brainstorming ideas for interesting XQueries, we thought it would be beneficial to have a query that determines how many pieces of art the gallery currently has in a specific category, such as "Post-Impressionism." This query will make it easier for curators to gauge if there are enough pieces of art available for their desired exhibition. This update required us to adjust our artwork class to include a category variable to match the category elements in the XML DTD.

Another idea for an interesting XQuery was to explore different ways of structuring important events. An event ID and a manager were mentioned. The ID existed in the use case diagram, but the manager did not. By adding this, we were able to create the XML documents and their XQueries according to the assignment guidelines while also making our use cases more comprehensive. For instance, in this example, the relationship between external partners, events, and the administration of the Art Gallery was strengthened by adding a gallery manager overseeing the events.

We added the following variables to the "Artist" class to ensure it matched the elements in the Artist XML DTD: deathYear, specialty, and placeOfStudy.

We also added a use case for the Curator called "Get Artwork For Each Artist," which will make the Curator's workflow more efficient. This allows them to gather information for potential exhibitions quickly or respond to inquiries from auctioneers, museums, and other galleries.

3 XQueries

4 XML Documents

4.1 Artist

- Finn

```
<?xml version="1.0" ?>

<!--
This XML document stores information about artists, including their biographical details,
artistic specialty, and place of study. It can be used in applications or queries to
retrieve artist-specific data.
-->

<!DOCTYPE artists [

  <!--
  The "artists" element contains one or more "artist" elements,
  hence the cardinality is "+" (at least one artist is required).
  -->
  <!ELEMENT artists (artist+)>

  <!--
  Each "artist" includes details about their name, nationality, birth and death years,
  artistic specialty, and place of study. These elements are all required,
  so no optional or zero cardinality is specified.
  -->
  <!ELEMENT artist (name, nationality, birthYear, deathYear, specialty, placeOfStudy)>

  <!ELEMENT name (#PCDATA)> <!-- The artist's name -->
```

```
  <!ELEMENT nationality (#PCDATA)> <!-- The artist's nationality -->
  <!ELEMENT birthYear (#PCDATA)> <!-- The year the artist was born -->
  <!ELEMENT deathYear (#PCDATA)> <!-- The year the artist passed away -->
  <!ELEMENT specialty (#PCDATA)> <!-- The artist's area of specialization -->
  <!ELEMENT placeOfStudy (#PCDATA)> <!-- The institution where the artist studied -->
]>

<artists>
  <!-- An example of an artist's information -->
  <artist>
    <name>Vincent van Gogh</name>
    <nationality>Dutch</nationality>
    <birthYear>1853</birthYear>
    <deathYear>1890</deathYear>
    <specialty>Post-Impressionist</specialty>
    <placeOfStudy>Académie Royale des Beaux-Arts</placeOfStudy>
  </artist>
  <!-- Additional artist records follow the same structure -->
</artists>
```

4.2 Events

- Robin

```
<?xml version="1.0" encoding="UTF-8"?>

<!--This is the DTD and the XML document for the events class and is used by two event
related XQueries-->

<!DOCTYPE events [

<!--The following is a description of the main elements, the cardinality and the general
outline of the DTD seen below:

Event is set to "*: zero-or-more" as no events are technically needed. There will not
always be an event planned to document. All of the event elements are set to one as we
at least need all the information in the document for each single event once. By using
an extra outside bracket for dates, we are not limited in the amount of dates for each
event, hence why it is "+: one-or-more". There is at least one location needed. Some
events might take multiple places/rooms but one is the minimum.

The 6 #PCData elements are all elements in the document helping to describe the events.

eventID ATTLIST is #REQUIRED for every new event for easier allocation and logistic
purposes-->

<!ELEMENT events (event*)>
<!ELEMENT event (eventName, dates, locations, host, manager, eventDescription)>
<!ELEMENT dates (date+)>
<!ELEMENT locations (location+)>

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

```
<!ELEMENT host (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT location (#PCDATA)>
<!ELEMENT manager (#PCDATA)>
<!ELEMENT eventDescription (#PCDATA)>

<!ATTLIST event eventID CDATA #REQUIRED>

]>

<!--The following XML document is used to give functionality to the events part of our
UML diagram.

The diagrams showed an external partner working with the admins to create and manage
events, so in this document the functionality of a system is created,that allows the
management to oversee certain events with all the necessary details. This could allow
them to properly prepare, to have a general overview, to structure the year, to
calculate funds and to make the necessary changes needed.

We have the events document, with the individual events all sorted via eventID. Within
these events, there are multiple other elements that had mostly been included in the
class diagram already. Some additional events were created. Every event has its eventName,
its dates, the location it will be occupying/using, the host/external partner, the manager
responsible for the event and a short event description. All of this information allows
everyone that has access, to get a quick overview of the most important facts and to
understand the dimensions of each event.

    There are 8 events with different data to be found below. -->

<events>
    <event eventID="230401001">
        <eventName>Historical Easter Getaway</eventName>
        <dates>
            <date>2023-04-01</date>
        </dates>
        <locations>
            <location>Front Court</location>
        </locations>
        <host>Trinity Food Society</host>
        <manager>Robin Schulz</manager>
        <eventDescription>
The Historical Easter Getaway is an event focused on exploring and celebrating the
cultural and historical traditions of Easter, featuring themed activities and
educational presentations.
        </eventDescription>
    </event>
    <!-- Additional events records follow the same structure -->
</events>
```

4.3 Membership

- Liam

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    This XML document represents a list of memberships for an organisation or institution.
    Each membership contains details e.g. purchase price, type, purchase data, expiration,
    date, visitor ID and membership type
-->

<!DOCTYPE Members [

<!ELEMENT Members (Membership+)> <!-- The '+' cardinality is used to indicate that there
must be at least one 'Membership' element in the document-->
<!ELEMENT Membership (price, expirationDate, visitorID, type)> <!-- Each 'Membership'
element contains child elements -->
<!ELEMENT purchasePrice (#PCDATA)>
<!ELEMENT purchaseType (#PCDATA)>
<!ELEMENT purchaseDate (#PCDATA)>
<!ELEMENT expirationDate (#PCDATA)>
<!ELEMENT visitorID (#PCDATA) >
<!ELEMENT type (#PCDATA)>

<!ATTLIST Membership membershipID CDATA #REQUIRED> <!-- The 'membershipID' attribute is
required for each 'Membership' element to uniquely identify it-->

]>


<Members>
    <!-- Membership record for Visitor ID 1234 with a Patron membership-->
    <Membership membershipID = "0001">
        <purchasePrice>500</purchasePrice>
        <purchaseType>Final</purchaseType>
        <purchaseDate>2023-11-14</purchaseDate>
        <expirationDate>2024-11-14</expirationDate>
        <visitorID>1234</visitorID>
        <type>Patron</type>
    </Membership>
    <!-- Additional membership records follow the same structure -->
</Members>
```

4.4 Artwork

- Rachel

```
<?xml version="1.0" ?>

<!-- This XML document represents a museum's collection of artworks.
        Each artwork entry contains detailed information about the piece,
```

```
        including its title, artist, submission date, status, category, and
        description. -->

<!DOCTYPE museum [

    <!-- The root element 'museum' contains multiple 'artwork' elements.
            Cardinality '*' is used to allow zero or more artworks. -->
    <!ELEMENT museum (artwork*)>

    <!-- The 'artwork' element represents an individual art piece.
            Each artwork must have exactly one of each of the following child
            elements: title, artistName, submissionDate, status, category, and
            description. -->
    <!ELEMENT artwork (title, artistName, submissionDate, status, category,
    description)>

    <!-- Simple text content (#PCDATA) is expected for each child element. -->
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT artistName (#PCDATA)>
    <!ELEMENT submissionDate (#PCDATA)>
    <!ELEMENT status (#PCDATA)>
    <!ELEMENT category (#PCDATA)>
    <!ELEMENT description (#PCDATA)>

    <!-- The attribute 'artworkID' is optional (IMPLIED) and is used as a
    unique identifier for each artwork. -->
    <!ATTLIST artwork artworkID CDATA #IMPLIED>
]>

<museum>
    <artwork artworkID="2001">
    <title>Starry Night</title>
    <artistName>Vincent van Gogh</artistName>
    <submissionDate>12/06/2024</submissionDate>
    <status>Displayed</status>
    <category>Post-Impressionism</category>
    <description>
        A depiction of a swirling night sky filled with stars and expressive
        brushwork that conveys intense emotion.
    </description>
    <!-- Additional artwork entries follow the same structure -->
</museum>
```

5 Strengths and Weaknesses

5.1 Strengths

- We used a standard to create our document, specifically XML, XPath, and XQuery, which ensured consistency across the project.
- We split up the individual XML documents among the team members and then distributed the planned XQueries in a similar way. This distribution had a bias towards those who had worked on certain XML

documents that are referenced in specific XQueries.

- The use of XQuery functions and modular XML designs allows for easier updates or modifications in the future.

5.2 Weaknesses

- We faced time constraints when designing, which limited our thoroughness.
- We gained a lot of knowledge through this project, but we have nowhere to put it, meaning not everyone will immediately understand our work.
- We had an enormous amount of classes to pick from in the Class Diagram, making it difficult to choose what to include in our XQueries.
- When designing XQueries, we had to reconfigure parts of our Class Diagram so that certain classes were properly connected after integrating them within XQueries.

6 Credits

6.1 Finn

- Created 1 XML document: Artist
- Added six elements
- Created 1 XQuery:

  artworkPerArtist: Outputs the amount of artwork the gallery has per artist.

- Established a GitHub to enable a collaborative workflow to store XML documents.

6.2 Robin

- Created 1 XML document: Events
- Added six elements
- Created 2 XQueries:

  eventsLengthInDays: Takes in an event title and prints out the number of days and which days of the month the event occurs.

  eventsByManager: Takes in a manager's first name and prints out all the events this manager is responsible for.

- Created a poll for task allocation
- Created the XML report template
- Contributed to:

  UML Design to XML Implementation

  XQueries

  XML Documents

6.3 Liam

- Created 1 XML document: Membership
- Added six elements
- Created 1 XQuery:

  getMemberRevenue: Calculates and displays total revenue generated from types of memberships.
- Compiled and formatted the final report

6.4 Rachel

- Created 1 XML document: Artwork
- Added two elements
- Created 1 XQuery:

  getArtworkForCategory: Implements user-defined function

6.5 Nick

6.6 Roisin

- Created 1 XML document: Donation
- Added six elements
- Created 1 XQuery:

  getSpecificDonor: Retrieves and displays donations by a specific donor.
- Identified strengths and weaknesses of the XML and XQuery design.

6.7 Noah

- Created 1 XML Document: Tickets
- Added six elements
- Created 1 XQuery:

  getRevenueFromDate: Implements user-defined function that retrieves all revenue across Tickets, Membership, and Donations on a certain date.
- Identified strengths and weaknesses of the XML and XQuery design.

Abstract

Note from the author: This was my first time writing in Latex - it took me 3 hours to write this document. Please ignore this in the marking, just felt like sharing.