Knowledge Graph Building

Final Report

CS4624 Multimedia, Hypertext, and Information Access

Client: Satvik Chekuri
Instructor: Dr. Edward A. Fox
Authors: Qianxiang Hao, Haoran Xing
Virginia Tech, Blacksburg VA 24061
May 9, 2024
# Table of Contents

Table of Figures.............................................................................................................................................. 2  
Table of Tables.................................................................................................................................................. 3  
1 Executive Summary..................................................................................................................................... 4  
2 Introduction.................................................................................................................................................... 5  
   2.1 Problem.................................................................................................................................................. 5  
   2.2 Motivation............................................................................................................................................... 5  
   2.3 General Approach.................................................................................................................................. 6  
3 Requirements.................................................................................................................................................. 7  
   3.1 Data Conversion...................................................................................................................................... 7  
   3.2 RDF Uploading....................................................................................................................................... 7  
   3.3 URI Resolution........................................................................................................................................ 8  
4 Design........................................................................................................................................................... 9  
5 Implementation.............................................................................................................................................. 13  
6 Assessment................................................................................................................................................... 16  
   6.1 Data Conversion...................................................................................................................................... 16  
   6.2 Data Uploading...................................................................................................................................... 17  
   6.3 GUI Data Expanding Application......................................................................................................... 17  
7 User’s Manual............................................................................................................................................. 19  
   7.1 Query ETD SPARQL interface.................................................................................................................... 19  
   7.2 URI Resolution Service............................................................................................................................ 24  
   7.3 Knowledge Graph Expansion GUI Application....................................................................................... 25  
8 Developer’s Manual...................................................................................................................................... 33  
   8.1 Object ID Insertion.................................................................................................................................. 34  
   8.2 Data Conversion...................................................................................................................................... 34  
   8.3 Data Uploading....................................................................................................................................... 37  
   8.4 GUI Application for Expanding RDF.................................................................................................... 38  
   8.5 Endeavour Ingress Configuration......................................................................................................... 39  
   8.6 Data...................................................................................................................................................... 42  
9 Lessons Learned......................................................................................................................................... 45  
   9.1 Timeline.................................................................................................................................................. 45  
   9.2 Future Work............................................................................................................................................ 46  
10 Acknowledgements.................................................................................................................................... 47  
11 References................................................................................................................................................... 48
Table of Figures

Figure 1: Design for Knowledge Graph Data Expansion....................................................................................... 10
Figure 2: Design for URI Resolution.......................................................................................................................... 11
Figure 3: SPARQL interface......................................................................................................................................... 19
Figure 4: SPARQL interface for searching all data in a graph.................................................................................... 21
Figure 5: Search Result from Figure 4....................................................................................................................... 22
Figure 6: Querying predicates containing string “abstractOf”.................................................................................. 23
Figure 7: Querying predicates containing “abstractOf”, subject and object with certain ETD ID24
Figure 8: Querying results from Figure 7.................................................................................................................. 24
Figure 9: Webpage shows metadata of ETD................................................................................................................. 25
Figure 10: App window when start............................................................................................................................. 26
Figure 11: Data Conversion interface.......................................................................................................................... 27
Figure 12: RDF data conversion interface after entering necessary parameters......................................................... 28
Figure 13: Notification window to indicate the success or failure.............................................................................. 29
Figure 14: RDF triples converted................................................................................................................................... 29
Figure 15: Upload RDF Data interface........................................................................................................................ 30
Figure 16: Data uploading interface after entering necessary parameters................................................................. 31
Figure 17: Notification window to indicate the success or failure.............................................................................. 32
Figure 18: KG Building project Gitlab repository....................................................................................................... 33
Figure 19: Initialization of IDGenerator class................................................................................................................ 34
Figure 20: Element title in XML file with Object ID “6452748”................................................................................... 35
Figure 21: ETD in XML Format...................................................................................................................................... 35
Figure 22: ETD in JSON Format...................................................................................................................................... 36
Figure 23: ETD RDF triple............................................................................................................................................ 36
Figure 24: errorlog.txt for data uploading.................................................................................................................. 37
Figure 25: HTTP log for Virtuoso database.................................................................................................................. 38
Figure 26: Endeavour Resources under Namespace “etd”............................................................................................. 41
Figure 27: Detail Setting of Endeavour Ingress for Virtuoso Database........................................................................ 42
Figure 28: Endeavour Ingress for Virtuoso Database.................................................................................................... 42
Figure 29: “bbox” attribute in an XML element.......................................................................................................... 43
Table of Tables

Table 1: Timeline.......................................................................................................................... 45
1 Executive Summary

Our team’s main objective was to expand the Virtuoso database by integrating a comprehensive dataset of 500,000 enriched Electronic Theses and Dissertations (ETDs). We built upon the preliminary framework of 200 XML records used for initial testing. This database expansion would enable the developers to deploy more robust testing and analysis of the current Knowledge Graph database. Additionally, our team focused on standardizing the data expansion process, ensuring that future developers have a consistent and reliable foundation for their work.

The current Knowledge Graph was established with the Virtuoso graph database system. We primarily worked on four steps to expand the KG database, including inserting Object IDs into each element in XML files, converting XML files to RDF triples, uploading RDF triples to the Virtuoso database, and URI resolution. We leveraged the power of Python, along with its robust libraries (rdflib, sparqlwarpper, requests, xmltodict, Node.js, NPM, tkinter) and tools (REST API, Docker) to execute these steps. Initially, our team successfully tested the data expansion process on a local Virtuoso instance to ensure the functionality and correctness of the expanding procedure. We prepared to deploy the process on the Virtuoso database within the Endeavour cluster upon confirmation. Although we successfully expanded the database by 333 ETDs, we were unable to reach our target of 500,000 ETDs due to a shortage of XML data. This limitation made us refocus our efforts on refining the data expansion process for better standardization and future scalability. We streamlined the data expansion process by integrating the Object ID insertion, data conversion, and data uploading processes into a single GUI application, creating a more straightforward and compact workflow. This visual interface would enhance usability for future developers and teams.
2 Introduction

2.1 Problem

An Electronic Thesis or Dissertation (ETD) is a digital version of a thesis or dissertation, typically stored in a university’s repository [2]. In recent years, the growth of ETDs in academic repositories has been continuous. With a vast amount of scholarly content available, the ability to efficiently navigate, access, and extract relevant data from ETD databases has become increasingly important. However, the current Knowledge Graph (KG) database, located on the Endeavour cluster [4], is limited by its small-scale collection of 200 XML records, which constrains the scope of testing and analysis that can be conducted. This limitation hinders developers from leveraging the full potential of the KG for advanced research and application development.

2.2 Motivation

The integration of 500,000 Electronic Theses and Dissertations (ETDs) represents a substantial enhancement, promising to amplify the robustness and analytical depth of the Knowledge Graph, thus driving forward the field of academic research and its applications. We want to enlarge the Knowledge Graph ETD database and improve how future developers interact with these resources by standardizing and simplifying the data expansion process.
2.3 General Approach

Our project utilizes a systematic and efficient approach to expand the current Knowledge Graph database. Our approach consists of four steps, including Object ID insertion, data collection, RDF triple generation, data uploading, and URI resolution. By employing a suite of powerful Python libraries, Node.js [7], NPM [11], and tools like REST API [12] and Docker [8], we ensure the seamless execution of these steps. Additionally, our initiative to encapsulate the entire process within a single GUI application, built using the Tkinter library [10], streamlined operations and enhanced the usability of the system for developers now and in the future.
3 Requirements

Overall requirements for our project include:

- Integrating 500,000 ETDs into the existing Knowledge Graph
- Ensuring accurate conversion of ETD elements into RDF triples for the graph
- URI resolution

These basic requirements ensure the successful expansion of the current Knowledge Graph database. Specific details and steps are shown in the following sections.

3.1 Data Conversion

The data conversion process is fundamental to our project. It involves transforming XML files into RDF triples, a crucial step for populating the Knowledge Graph. Each XML file must encapsulate an “Object ID” for each object inside of the XML file to unequivocally identify distinct elements such as authors, titles, and other metadata entries. This “Object ID” is used to identify each entity with its respective metadata in the RDF triples, laying the groundwork for subsequent URI resolution. Additionally, the conversion will output RDF triples in N3 format [13], enhancing their readability for developers to review stored RDFs. This format choice not only aids in human readability but also streamlines data integration into the Knowledge Graph.

3.2 RDF Uploading

The RDF uploading process is critical and must adhere to accuracy and completeness standards. Therefore, it’s important to ensure that the quantity of RDF triples uploaded is the same as the converted data without omissions or duplications. This requires verifying the correctness of the
RDF triples to maintain data integrity and to maintain the structure of the Knowledge Graph, as well as directing the RDF data to the appropriate endpoint within the Virtuoso database [9], ensuring it populates the correct graph. This step is vital for maintaining the database’s organizational schema and facilitating effective data retrieval.

3.3 URI Resolution

Upon the successful uploading of RDF triples to the Virtuoso database, URI resolution [3] becomes the focal point. This process is essential for Access and Interaction with RDF data. Clicking on any RDF element within the SPARQL interface or sending requests then, will redirect to a dedicated webpage. This page will display comprehensive details about the selected entry, providing an in-depth view of the data.
4 Design

The design architecture of our project is crafted to ensure a smooth and systematic expansion of the Knowledge Graph within the Virtuoso database, alongside standardizing the associated processes and workflows. See Figure 1.
Figure 1: Design for Knowledge Graph Data Expansion
The initial phase involves converting ETDs (Electronic Theses and Dissertations) from PDF to XML format, to make documents more accessible to computer programs for future processing. Once desired ETDs are in XML format, we’re able to parse them to JSON format. This intermediate step is important for facilitating the process and to convert these JSON data to RDF triples in N3 Format. The final step for data expansion is to upload all RDF triples to the Virtuoso database using a SPARQL [6] end point. This step will allow newly added data to be searchable and interlinked with an existing knowledge graph.

![Diagram of URI Resolution](image)

**Figure 2: Design for URI Resolution**

In addition, following our client’s requirements, an important step of our design includes applying URI resolution to each piece of metadata newly added to the database. This resolution process is a significant aspect of our project's second phase, as seen in Figure 2 [4][5], which illustrates the metadata retrieval mechanism once a user interacts with the Knowledge Graph.
Upon a user selects and clicks on the anchor of a URI of a particular RDF entry, the Express.js [5] application, which was developed by Team 5 in the previous semester [4], initiates a request to the PostgreSQL [15] database. This action triggers the retrieval of corresponding metadata, which depends on both the object ID and its associated ETD ID. Subsequently, the retrieved metadata is displayed on a webpage dynamically generated by the Express.js application, offering the user detailed information and content about the selected entry.

Design of the Knowledge Graph Expansion GUI application is the core part of our team’s effort to standardize the workflow of ETD data expansion. This interface is engineered to consolidate the various stages of data handling, from XML data conversion to RDF uploading, into a cohesive application, thus simplifying the workflow for future developers and researchers. This application will act as the gateway for the project's stakeholders to interact with the Knowledge Graph expansion process.
5 Implementation

The implementation of our project mainly focused on RDF triple generation, uploading and standardizing the whole process into an GUI app. For the RDF triple generation process, the basic workflow has already been set by Team 1 from CS5604 [3]; our group mostly focused on implementation of data uploading and on the GUI tool application.

Before the data conversion step, it is essential to ensure that each XML element that represents a significant part of the ETD has a unique Object ID. This identifier is critical for maintaining the traceability and integrity of data throughout the transformation process. We utilized a Python script leveraging the lxml [14] library to scan each XML file. The script automatically inserts a unique Object ID into an XML element. The project’s next phase involves converting ETD XML files to RDF triples in N3 format. This transformation is crucial for standardizing the data format, enabling automated processing, and utilizing the “xmltodict” library, using a Python script to convert the XML data into JSON format. This step makes the data more accessible and easier to manipulate and process in future steps. The conversion process is aimed at preserving the integrity and structure of the original ETD content, ensuring that all valuable information is accurately transformed. Then, the “rdflib” library was used, defining custom namespaces and predicates that precisely reflect the data and structural sections of the ETD papers. This library facilitates the definition of custom namespaces and predicates, allowing us to make the RDF triples to accurately represent the relationships and structures within ETD documents. Key elements such as chapters, figures, tables, and authors are semantically linked, creating a rich, interconnected web of information.
Our data uploading component is designed for efficiency and performance. Leveraging the SPARQLWrapper library, we interface with the Virtuoso SPARQL endpoint, ensuring that the RDF data we upload is immediately searchable and interactable via SPARQL queries. In the current implementation, RDF triples are aggregated into a single file, and we have implemented code to segment the data into manageable chunks for efficient uploading. Each chunk, containing a set number of RDF statements, is then individually sent to the Virtuoso database through a POST request authenticated with HTTP Digest access authentication, ensuring a smooth and secure data uploading process.

The Graphical User Interface (GUI) of our project has been developed using the Python Tkinter library to provide a better experience for users to convert XML documents to RDF triples and upload them to the Virtuoso database. Upon initialization, the main window is set up. The interface is divided into two main sections, each dedicated to a specific function: one for converting XML to RDF and another for uploading RDF data. The transition between these sections is facilitated by buttons, which are “Convert XML to RDF” and “Upload RDF Data”, to enhance navigation ease and make the application intuitive. For conversion, the user can select the directory containing XML files, specify the desired output file name for JSON, and set the directory for the output JSON files. A separate entry allows the user to input the RDF output file name. The “Start Conversion” button triggers the conversion process, which reads each XML file, parses it to JSON using xmltodict, and then generates RDF. The converted data is saved in N3 format. The upload section contains entries for endpoint URL, graph URI, username, and password, which are essential for authenticating with the Virtuoso database. A “Select RDF File”
button lets users pick the file containing RDF triples, which can be uploaded using the “Upload RDF” button.
6 Assessment

6.1 Data Conversion

The data conversion process was designed to accurately transform ETDs from XML to RDF in N3 format. We have designed and implemented a series of checks and assessments to ensure the conversion meets the required quality standards.

Validation:
   a. Each RDF triple generated should be validated against the RDF schema to ensure structural correctness.
   b. A subset of the original XML and the converted RDF data was manually inspected to confirm that the information was preserved accurately during conversion.

Accuracy:
   a. We checked that the custom predicates correctly represent the relationships expressed in the original XML.
   b. The RDF triples were verified with the XML data to ensure no data loss or corruption occurred during the conversion.

Robustness Test:
   a. Conducting scalability tests with larger datasets to ensure the conversion process remains efficient as data grows.
6.2 Data Uploading

The data uploading process aims to facilitate the efficient uploading of RDF triples into the Virtuoso database. We evaluate this process based on its ability to successfully upload data while ensuring the integrity and completeness of the uploaded data. We ensured all RDF triples generated during the conversion process were successfully uploaded to the specified “Graph URI” within the Virtuoso Database.

**Verification and Robustness Evolution:**

a. After the uploading process, SPARQL queries were run to verify that the uploaded data could be retrieved and was in the correct graph.
b. The upload script was tested against network interruptions to evaluate its error recovery capabilities.
c. Multiple uploads were performed to ensure the process’s stability and the consistency of the uploaded data across different tests.

**Performance:**

We measured the time taken to upload chunks of RDF data, seeking optimization to prevent timeouts and so as not to exceed the maximum uploading amount of 10,000 triples for a single uploading limited by the SPARQL endpoint.

In all, the data uploading process to the Virtuoso database performs well under the tested conditions, still with room for optimization.

6.3 GUI Data Expanding Application
Our GUI application was designed to provide a visual interface for converting XML to RDF, and the subsequent uploading of RDF triples. The evolution of the app was based on ease of use, clarity of instructions, and overall user experience.

To enhance the user experience, our application incorporates message boxes that provide immediate feedback, allowing users to check the outcome of their actions. We tested both the conversion and upload functionalities to guarantee their reliability and to ensure that they execute flawlessly under expected conditions. We completed our error handling assessments by conducting tests, including scenarios with incorrect input types and process interruptions, to ensure the robustness of the software. Moreover, we prioritized responsiveness, ensuring that users experience smooth transitions between tasks, thereby minimizing delays and streamlining the workflow.

**Feedback and Error Handling:**

a. **Success Messages:** After a successful conversion or upload, a message box confirms the operation's success.

b. **Error Messages:** If an error occurs due to incorrect inputs or connection issues, the message box provides clear, accurate messages or hints about the issue.

**Performance:**

a. **Benchmark Tests:** conducted benchmark tests to measure the response times for each core function under different conditions.
7 User’s Manual

7.1 Query ETD SPARQL interface

To access and utilize the SPARQL interface of the Knowledge Graph, users should follow these steps:

1. Navigating to SPARQL interface: Open a web browser and visit:
   https://virtuoso.endeavour.cs.vt.edu/sparql
2. The correct URL will navigate you to the query editor interface (Figure 3).

![Figure 3: SPARQL interface](image)

3. Querying Data in a Graph: Upon the interface loading, you will be presented with a screen where you can input SPARQL queries. To target a specific graph, enter its name in the “Default Graph URI” field. Write or paste your SPARQL query into the “Query Text”
area to retrieve RDF data from the designated graph. The example in Figure 4 shows retrieving all RDF data from a graph.
Figure 4: SPARQL interface for searching all data in a graph
4. Applying Filters: For more precise results, you can filter RDF triples using specific criteria. As shown in Figure 6, you may want to query for predicates containing the string “abstractOf”. To narrow down the results further, introduce additional FILTER expressions to the query, as illustrated in Figure 7. This example demonstrates how to filter predicate, subject, and object, possibly using a particular ETD ID.
Figure 6: Querying predicates containing string “abstractOf”
Figure 7: Querying predicates containing “abstractOf”, subject and object with certain ETD ID

![SPARQL Query Editor](image)

Figure 8: Querying results from Figure 7.

7.2 URI Resolution Service

1. Navigate to SPARQL’s data table; directly click the anchor of a URI (Figure 8).
2. A WebView with metadata of the selected entry will display (Figure 9).
7.3 Knowledge Graph Expansion GUI Application

This application facilitates the conversion of XML files to RDF format and their subsequent upload to a Virtuoso database. To use the KG Expansion Tool, ensure that Python is installed on your system. Place the application files into your local machine and run “RDFexpandApp.py” to start the application. The application interface is shown in Figure 10, reflecting when initially started.
Click the “Convert XML to RDF” button to show the data conversion interface as shown in Figure 11.
Instruction Steps for Data Conversion:

1. Click the “Browse” button to open a dialog and select the directory containing your XML files.
2. Enter the base name for the output JSON files in the corresponding entry field.
3. Click the “Browse” button to select the directory where the converted JSON files will be saved.
4. Specify the name for the output RDF file.
5. Click the “Start Conversion” button to begin converting XML files to RDF; see Figure 12.
6. A success message will appear upon completion as shown in Figure 13. Figure 14 shows the results, i.e., converted RDF triples.

![RDF Conversion and Upload Tool](image)

Figure 12: RDF data conversion interface after entering necessary parameters.
Instruction Steps for Data Uploading:

1. Enter the Virtuoso endpoint URL in the provided entry field.
2. Specify the URI of the graph in the Virtuoso where the RDF data will be uploaded.
3. Input the username and password required to access the Virtuoso database.
4. Click the “Browse” button to select the RDF file you wish to upload.
5. After filling in all necessary fields, click the “Upload RDF” button.
6. A notification window will show, and a message will indicate the success or failure of the upload, as shown in Figure 17.

![RDF Conversion and Upload Tool](image)

Figure 15: Upload RDF Data interface.
Figure 16: Data uploading interface after entering necessary parameters.
RDF data has been successfully uploaded.

OK

Figure 17: Notification window to indicate the success or failure
8 Developer’s Manual

The KG Building project repository is organized to accommodate different components of our system, stored in specific folders, as shown in Figure 18:

- **GUI_App**: Contains the front-end user interface application for easier interaction with the data expansion process.
- **XMLInsertedObj_ID**: Archives all XML files with inserted Object IDs that our team has processed this semester.
- **kg-triple-generation-main**: Includes scripts and resources for generating triples from the XML files, including scripts for Object ID insertion and conversion of XML files to RDF format.
- **kg-uri-resolution-master**: Handles the URI resolution service.

Figure 18: KG Building project Gitlab repository
8.1 Object ID Insertion

The following part of the manual introduces the process of Object ID insertion, which is aimed to automatically insert unique object IDs into XML files only when IDs in the XML files are missing.

Environment Setup:

- Python3
- Python libraries:
  - lxml

Data requirements: XML files that do not contain an “obj_id” element.

Running:

To run the object ID insertion script, navigate to the “kg-triple-generation-main” folder, and run command “python/python3 objectIDInjecter.py”

Script Configuration:

The IDGenerator class is designed to generate unique, sequential IDs for each XML element requiring an obj_id. It begins with a user-defined starting ID and increments with each new ID. The starter ID is the first ID that the script will begin incrementing for your XML file elements. To set the starter ID, simply call initialization of the IDGenerator within the process_directory function as shown in Figure 19; here it shows the starter ID is “7000000” so the first ID for your XML file element will be “7000001”.

```
id_generator = IDGenerator(start_id=7000000)  # Initialize the Start ID
```

Figure 19: Initialization of IDGenerator class.

8.2 Data Conversion

This part of the manual provides a guide for developers to convert ETDs from XML format to RDF triples in N3 notation, which is a compact and readable serialization format for RDF.
Environment Setup:

- Python3
- Python libraries:
  - rdflib (6.0.2)
  - sparqlwrapper (1.8.5)
  - requests (2.26.0)
  - xmltodict (0.12.0)
  - csv

Data requirements: XML file needs object ID for metadata retrieval as shown in Figure 20.

```
<title bbox="[337.92, 174.19, 1450.04, 305.56]" conf="0.89" pg_no="0" obj_id="6452748"">
```

Figure 20: Element title in XML file with Object ID “6452748”.

Running:

To run the data conversion script, navigate to the “kg-triple-generation-main” folder, and run command “python/python3 main.py”.

Conversion Process:

```
<xml version="1.0" encoding="utf-8">
<etd>

```

Figure 21: ETD in XML Format
1. After running the script, the program will first parse XML data (see Figure 21) into JSON format, as shown in Figure 22.

![Figure 22: ETD in JSON Format](image)

2. The program will convert ETD JSON to RDF triples in N3 format, as shown in Figure 23.

![Figure 23: ETD RDF triple](image)

3. Find the generated N3 format file in the current directory.
8.3 Data Uploading

This section outlines the procedure to upload RDF triples into the Virtuoso database. The RDF triples are assumed to be serialized in the N3 format.

**Configuration:**

a. Set the endpoint URL, which is the address of the Virtuoso SPARQL endpoint.

b. Specify the Graph URI, where data will be stored.

c. Define the namespace.

d. Enter Username and Password for the Virtuoso database.

**Running:**

Navigate to the navigate to the “kg-triple-generation-main” directory; run the command “python/python3 uploadRDF.py”

**Verification:**

a. Navigate to the SPARQL search interface, then query data to verify.

b. Check the error log, shown in Figure 24.

```
Data uploaded successfully.
Data uploaded successfully.
Data uploaded successfully.
Data uploaded successfully.
Data uploaded successfully.
Data uploaded successfully.
Data uploaded successfully.
Data uploaded successfully.
Data uploaded successfully.
Data uploaded successfully.
Data uploaded successfully.
Data uploaded successfully.
Data uploaded successfully.
Data uploaded successfully.
```

Figure 24: errorlog.txt for data uploading.

c. Checking HTTP log (Figure 25) for upload verification:
This is the developer guide for a Tkinter-based GUI application designed to integrate and visualize the process of converting XML files to RDF, and for uploading them to a Virtuoso database.
Environment Setup:

1. Python libraries:
   a. rdflib (6.0.2)
   b. sparqlwrapper (1.8.5)
   c. requests (2.26.0)
   d. xmltodict (0.12.0)
   e. Tkinter

Running:

Navigate to the “GUI_App” directory; run the command “python/python3 RDFexpandApp.py”

8.5 Endeavour Ingress Configuration

The Virtuoso database is integrated into the CS Endeavour cluster [4], requiring a proper configuration of the Ingress controller to manage external access to the database services. This part of the manual describes setting up the Ingress to route traffic to the Virtuoso related services set by Team 1 [3], which is crucial for enabling external access to the Knowledge Graph.

Requirements:

- Ensure you have the necessary permissions and network access to the CS Endeavour cluster.

Configuration Steps:

a. Navigate under the Endeavour Namespace “etd”, as shown in Figure 26.
b. Create the Ingress routes traffic for the virtuoso.endeavour.cs.vt.edu domain as shown in Figure 27, set paths /conductor, /sparql, and /sparql-auth which are directed to the Virtuoso service running in the container “team-1-container-virtuoso” [3] on port 8890.

c. After the configuration, the Ingress will be identical to what is shown in Figure 28.

Testing:

Ensure that the Ingress resource routes traffic correctly by accessing the Virtuoso SPARQL endpoint through your web browser or using curl.

a. Navigate to the Virtuoso Conductor interface to confirm that the Ingress is correctly routing traffic by accessing: https://virtuoso.endeavour.cs.vt.edu/conductor/.

b. Verify the SPARQL endpoint by visiting https://virtuoso.endeavour.cs.vt.edu/sparql for access to Virtuoso SPARQL endpoint.

Figure 26: Endeavour Resources under Namespace “etd”.
8.6 Data

In the development of Knowledge Building, we utilized a dataset of 333 XML files. Each file adhered to the specifications mentioned below, ensuring a consistent and robust dataset for testing and deployment purposes.

**XML Structure Requirements:**
Our application requires specific XML data structures to support efficient processing and integration into our system. Each XML file must include the following parameters:

- **etd**: Represents an Electronic Thesis or Dissertation.
- **front**: Encapsulates front matter of the document, which may include titles, abstracts, and author details.
- **title**: The title of the document.
- **obj_id**: A unique object identifier for each XML element. If an XML file does not contain an obj_id, our injector script could be used to insert one.

**XML File Encoding Requirement:**

Each XML file must be encoded in UTF-8 to ensure compatibility and proper processing.

**Handling Missing obj_id:**

If an XML file lacks an obj_id, it is suggested to use the provided injector script to insert a unique identifier. This ensures each element in the XML can be uniquely identified during processing and when converted into RDF triples.

1. **Bounding Box Requirement:**

   Ensure that each significant element within the XML file includes this “bbox” (bounding box) attribute as shown in Figure 29, to identify whether an obj_id needs to be added.

```
bbox="[198.95, 182.01, 1492.18, 294.74]" conf="0.89" pg_no="0" obj_id="7006491""
```

Figure 29: “bbox” attribute in an XML element.

2. **Object ID Injector Script Usage:**

   To insert obj_id in XML files without one, execute the injector script as follows:
a. Navigate to the “kg-triple-generation-main” folder.

b. Run command “python/python3 objectIDInjecter.py”.

This script scans each XML file in the input directory, inserts a unique obj_id if missing, and saves the modified XML to the specified output directory.
9 Lessons Learned

9.1 Timeline

Table 1 outlines the timeline for the Knowledge Graph Building project. It lists various tasks, team members assigned, completion status, and due dates. This schedule was essential for tracking progress and ensuring key milestones of our project were met on time during the project development phase.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Assignee</th>
<th>Status</th>
<th>Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic understanding of the knowledge for the KG Building description</td>
<td>Whole Team</td>
<td>Done</td>
<td>Feb 19</td>
</tr>
<tr>
<td>Understand the core concepts of Knowledge Graphs, ETDs, and tools. Review the previous team's work.</td>
<td>Whole Team</td>
<td>Done</td>
<td>Feb 27</td>
</tr>
<tr>
<td>Set up and test the data conversion and uploading processes on a local Virtuoso database.</td>
<td>Qianxiang Hao</td>
<td>Done</td>
<td>Mar 11</td>
</tr>
<tr>
<td>Start testing the uploading of RDF triples to the Virtuoso database on Endeavour. Resolve issues.</td>
<td>Whole Team</td>
<td>Done</td>
<td>Mar 23</td>
</tr>
<tr>
<td>Insert object IDs into XML files and convert these files into RDF triples.</td>
<td>Qianxiang Hao</td>
<td>Done</td>
<td>Mar 31</td>
</tr>
<tr>
<td>Complete the uploading process and perform testing including fuzzy search and exact math.</td>
<td>Haoran Xing</td>
<td>Done</td>
<td>Apr 5</td>
</tr>
<tr>
<td>Implement URI resolution. Perform comprehensive system testing.</td>
<td>Whole Team</td>
<td>Called Off</td>
<td>Apr 17</td>
</tr>
<tr>
<td>Finalize documentation, prepare for project presentation report, and perform final adjustments.</td>
<td>Whole Team</td>
<td>Done</td>
<td>May 1</td>
</tr>
</tbody>
</table>

Table 1: Timeline
9.2 Future Work

**Data Expansion**: Complete the expansion of the ETDs to reach the 500,000 mark, thereby providing a more comprehensive resource for researchers and developers. Our team was able to expand 333 ETDs based on the previous 200 ETDs in the Knowledge Graph this semester. The details and instructions about the Data Expansion process are shown throughout this report.

**URI Resolution**: Develop methods to handle XML files lacking object IDs, ensuring the system's robustness and ability to manage diverse data formats.

a. Current Problem: Unable to access PostgreSQL Database set up by the CS5604 Fall 2023 Team 5 [4].

b. Proposed Development:
   
   i. Database:
      1. Access Existing PostgreSQL Database: Gain access to the PostgreSQL database set up by the CS5604 Fall 2023 Team 5.
      2. Understand Database Schema: Analyze how Team 5 has arranged Object IDs and metadata within the PostgreSQL database.

   ii. Metadata Management:
      1. Upload XML Metadata: Extract metadata from XML files and upload this data into the PostgreSQL database.
      2. Retrieve meta using the URI resolution service under the current project directory “kg-uri-resolution-master".
10 Acknowledgements

We want to acknowledge Dr. Edward A. Fox for his invaluable guidance and academic support and our client Satvik Chekuri for his technical expertise and client-side insights, which significantly advanced our project. We are also grateful to Amr Aboelnaga and Bipasha Banerjee for their crucial role in supplying the XML files that formed the foundation of our work.

Contact Information:

Edward A. Fox
fox@vt.edu
Department of Computer Science

Satvik Chekuri
satvikchekuri@vt.edu
Department of Computer Science

Amr Aboelnaga
amraboelnaga@vt.edu
Department of Computer Science

Bipasha Banerjee
bipashabanerjee@vt.edu
Department of Computer Science
11 References


