Google
Summer of Code

Project Proposal

Ontology Time Machine / Package Manager using DBpedia Archivo for DBpedia

- Jenifer Tabita Ciuciu-Kiss

Contact information

- Name: Jenifer Tabita Ciuciu-Kiss
- GitHub User ID:
- Linkedin:
- Email:
- Phone:
- Location:

Project

- Project name:
  Ontology Time Machine / Package Manager using DBpedia Archivo

- Project description
  The project focuses on improving DBpedia Archivo, an ontology archive containing over 1800 versioned snapshots of ontologies. It addresses the challenges of ontology evolution and unavailability, which can impact the development of data-driven applications that rely on interconnected web ontologies. The project's primary goal is to improve the FAIRness (Findability, Accessibility,

Interoperability and Reusability) of ontologies for linked data and RDF knowledge graphs through DBpedia Archivo.

The key initiatives include:

- Creation of a time machine proxy: This proxy will enable access to specific historical versions of ontologies, ensuring that they can be retrieved even if no longer available at their original source.
- FAIRness evaluation: The project will assess and represent the FAIRness of an ontology and its dependencies, aiding developers in making informed decisions about which ontologies to adopt.
- Archivo extension: Enhancing Archivo to support the new features, including the time machine proxy and FAIRness evaluation.

The steps involve developing a time machine proxy that serves ontology versions from specific points in time or based on dependency locks, integrating a FAIR assessment tool, and creating a dependency package manager. This package manager will track ontology usage to recommend potential inclusions in Archivo.

- Basic ideology

  I followed the recommended steps for the project and created a plan to complete them all. Below, you can see my approach to each step and the potential challenges that might arise.

  1. Realize ontology wayback machine (transparent proxy):
     - Implement a transparent proxy that can intercept HTTP requests and respond with historical data in case the ontology is not available. The proxy will manage requests differently based on operational modes: The time-based mode is used for serving ontology versions from a specific point in time, catering to requests for historical data (e.g. using the Memento protocol). In the dependent-lock based mode, it serves specific ontology versions defined in a manifest or lock files. In

    the failover mode, it automatically redirects to the most recent version when the requested ontologies or their specified versions are not available or return a failure specifying the reason.
        - Challenges: Ensure that the proxy correctly interprets and respects HTTP headers, and handles requests without introducing significant latency. Handling ontologies over HTTPS-based IRIs, particularly those using w3id or similar vocabularies, add an extra challenge. These vocabularies often utilize secure connections, which may require handling of SSL/TLS certificates.
    - <u>Time-based mode:</u> Extend the proxy to parse datetime accept headers and retrieve ontology versions corresponding to the requested time.
    - <u>Dependent-lock based mode:</u> Develop functionality within the proxy to understand manifest files or existing lock files (e.g. poetry.lock, package-lock.json, etc.) specifying ontology version dependencies. When a request is made, the proxy should parse the manifest and serve the correct versions of dependent ontologies.
        - Challenges: Manifest files and lock files must be structured and parsed accurately to prevent serving incorrect ontology versions.
    - <u>Failover mode:</u> Implement logic to detect when a requested version is unavailable and handle the situation accordingly. In some cases, the request should be automatically redirected to the latest available version, but in others, the failure might be preferred.
2. Realize dependency analysis:
    - Use ontology processing tools like the OWL API to analyze *owl:imports, rdfs:subClassOf and rdfs:subPropertyOf* within each ontology version recursively, building a dependency graph including the

    specific versions used (potentially based on the previous time machine).
        - Challenges: Dependency analysis should be thorough and accurate, accounting for all potential dependencies and their versions.

3. FAIR assessment tool integration:
    - Integrate or develop a tool to assess and assign FAIRness scores to each ontology version, possibly using automated metrics based on Findability, Accessibility, Interoperability, and Reusability.
        - Findable: The first step is ensuring that the ontology versions are easy to locate for both humans and computers. Unique identifiers and a detailed metadata description enhance the findability of ontologies.
        - Accessible: Once found, ontologies should be readily retrievable using standard protocols, with clear and accessible metadata and data usage licenses, even when the data itself is no longer available.
        - Interoperable: Ontologies should be structured in a way that allows their integration with other datasets and tools, which typically involves the use of standard vocabularies and clear data semantics.
        - Reusable: Ontologies should be well-described so they can be used in different contexts, which involves detailed metadata, clear usage licenses, and provenance information.
    - Based on the above, I plan to develop a weighted scoring system where each FAIR principle is assigned a specific weight based on its importance to the project's objectives. This way, the final score is a weighted sum of the individual scores, providing a view of an ontology's FAIRness.
    - For the additional level of scoring that integrates the transitive dependencies of the ontology, I will adopt a

hierarchical scoring model. The goal is that the dependency's scores would influence the main ontology's score to a certain degree. This could involve assigning a smaller weight to the dependency scores to ensure they don't disproportionately affect the main score.

4. Extend ontology version numbers with overlays:
    - Modify the ontology metadata to include additional versioning information, such as owl:versionIRI, using tools like the OWL API.
        - Challenges: Extended version information should be clear, consistent, and not conflict with existing versioning schemes.
5. Realize dependency “package” manager and lockfile option:
    - Develop a system to manage ontology dependencies, similar to package managers in software, where dependencies are defined in a lockfile.
        - Challenges: The package manager should resolve dependencies accurately and handle conflicts or circular dependencies.
6. Track ontology usage via the proxy:
    - Implement logging within the proxy to record ontology requests, using this data to recommend inclusions to Archivo.
        - Challenges: Logging should be comprehensive and privacy-preserving, providing valuable insights without compromising user data.

- Evaluation

  Each of the above steps (1-6) will be evaluated based on different strategies, but in general, unittest will be used to evaluate each of them.

    1. Realize ontology wayback machine (transparent proxy): Check if the proxy correctly intercepts and responds to HTTP requests with the appropriate historical data. Testing could involve sending requests for specific ontology versions and verifying the response matches the expected version. This

step corresponds of 3 parts, each of them will be evaluated separately:

   a. <u>Time-based mode:</u> Confirm that the proxy correctly interprets datetime accept headers and returns the ontology version corresponding to the requested time. This can be done by requesting ontology versions from specific dates and verifying that the returned version matches the archive records for those dates.
   b. <u>Dependent-lock based mode:</u> Verify that the proxy serves the correct versions of ontologies based on a given manifest file. This can be assessed by creating test scenarios with predefined manifest files and checking if the served ontologies align with the manifest specifications.
   c. <u>Failover mode:</u> Test the proxy's ability to redirect to the latest available version when a requested version is not available. This can be evaluated by requesting a non-existent version of an ontology and verifying that the proxy responds with the latest available version.
2. Realize dependency analysis:
   Evaluate the accuracy of the dependency analysis by comparing the proxy's identified dependencies for a given ontology version against a manually verified dependency list. This would involve checking if all dependencies and their versions are correctly identified.
3. FAIR assessment tool integration:
   a. <u>Findable:</u> Measure if the ontologies have unique identifiers and rich metadata. Success can be gauged by the ease with which ontologies can be located using these identifiers and metadata.
   b. <u>Accessible:</u> Confirm that ontologies are retrievable through standard protocols and that metadata remains accessible even when the data is not. This could involve automated checks for link functionality and metadata accessibility.
   c. <u>Interoperable:</u> Evaluate whether ontologies use standard vocabularies and can be integrated with other

datasets. Success can be assessed through testing ontology integration with other data sets or tools.

   d. Reusable: Check if the ontologies have clear metadata regarding their context, usage licenses, and provenance. This could be evaluated by reviewing the metadata for completeness and clarity.

4. Extend ontology version numbers with overlays:
   Verify that the extended version numbers and overlays are correctly applied and retrievable. This can involve checking that the overlays are consistent, correctly formatted, and provide the intended additional context.
5. Realize dependency “package” manager and lockfile option:
   Test the package manager by creating scenarios where ontologies with complex dependencies are requested. The manager should resolve and serve the correct versions based on the lockfile. The evaluation can include checking the system's ability to handle conflicts, update dependencies, and manage version locks effectively.
6. Track ontology usage via the proxy:
   Analyze the logs generated by the proxy to ensure that ontology usage is being accurately tracked. The success would be measured by the system's ability to provide actionable insights into which ontologies are being accessed, potentially guiding decisions for ontology inclusion in Archivo.

- Implementation
  - For the implementation of the ontology wayback machine, transparent proxy, and FAIR assessment tool, I will use a combination of Python, which is widely used for web and network applications, and ontology management, respectively. I will use the Flask or Falcon framework that can serve as the backbone for the proxy server, handling HTTP requests and implementing the logic for the time-based, dependent-lock, and failover modes.
  - For the FAIR assessment, Python and libraries like RDFlib, SPARQLWrapper and OWLready2 can be used to evaluate and score the ontologies based on the FAIR principles.

RDFlib will be important in evaluating Findability and Interoperability by allowing us to handle RDF graphs efficiently, while OWLready2 will help in assessing the Reusability of ontologies via its robust OWL manipulation capabilities. For evaluating Accessibility, SPARQLWrapper will enable the interaction with RDF data stores via SPARQL queries. To facilitate the package manager functionality, I will also use Python and libraries or custom scripts to handle ontology dependencies and versions. Throughout the development process, I will integrate proper testing frameworks (like pytest) and use Git to manage my contribution effectively.

- Project timeline
    - Community bonding period (May 1 - 26)
      This period is a great chance to meet everyone and I'm excited to learn from the group. I'll spend time reviewing the community's guidelines and the documentation for the tools I'll be using to make sure that the work I do later on will be of good quality.
    - Week 1 - Week 2 (May 27 - June 9)
      The first task will be setting up the development environment, and ensuring all necessary tools and libraries are in place. There is already a Github repository for DBpedia Archivo which I will use as a basis, with its python environment. Following this, I will start the construction of the transparent proxy, focusing on basic routing and request handling. Additionally, I'll develop the time-based mode for the proxy, enabling it to serve ontology versions based on specific timestamps, and start the groundwork for the dependent-lock based mode to manage requests based on dependency manifest and lock files.
    - Week 3 - Week 4 (June 10 - 23)
      This phase will involve completing the dependent-lock based mode and developing the failover mode. Once the transparent proxy is finalized I will document the implemented functionalities and implement test cases to

guarantee that everything works as expected. I'll also start the development of the dependency analysis feature to ascertain the relationships between different ontology versions.
Midterm evaluation deadline: June 12

- Week 5 - Week 6 (June 24 - July 7)
  I will focus on finalizing the dependency analysis feature and start integrating the FAIR assessment tool. The goal is to establish a mechanism to evaluate the FAIRness of ontology versions, ensuring they adhere to the FAIR principles.
- Week 7 - Week 8 (July 8 - 21)
  During these weeks, I'll finalize the FAIR assessment tool integration. When the final version is reached, I will implement test cases to ensure the functionality and document the how the assessment scores were calculated. In the end, I'll start developing the extended ontology versioning system.
- Week 9 - Week 10 (July 22 - August 4)
  I will work on finalizing the extended ontology versioning system. This will include the development of the test cases and documenting the functionalities. I will begin the development of the package manager and lockfile functionality. This is important for managing ontology dependencies effectively and ensuring that the system can handle different ontology versions and their interdependencies.
- Week 11 - Week 12 (August 5 - 18)
  I will complete the package manager and lock file feature, ensuring they function correctly and manage ontology dependencies as intended and document the implemented functionalities. Additionally, the usage tracking feature within the proxy will be developed and integrated. I will implement the corresponding tests each week, however during this time I will go through them all making sure everything is extensively tested.
- Week 13 (August 19 - 26)

This final week is dedicated to wrapping up the project. I will finalize all documentation, ensuring that every aspect of the project is well-documented, understandable and reproducible. I'll address any remaining issues or refinements to ensure the project is fully polished and ready for deployment.

Technical skills

- Python: I have been coding in Python for 5+ years, and I have experience with all the libraries I am planning to use for this project.
- RDF/OWL: I have about 2 years of academic experience using RDF and OWL.
- HTTP: I have 3+ years of experience working with HTTP protocols and APIs.

Open-source

- This will be my first open-source project I have not yet participated in any other open-source projects. I find it extremely fascinating to see that projects like Python, and Linux can be successfully developed and maintained at such a high level purely by the open-source community. I would love to become part of this and contribute to DBpedia as my first open-source contribution.

Background and education

- 2017 - 2020: Eötvös Loránd University, Computer Science BSc
- 2020 - 2022: European Institute of Innovation and Technology (EIT) Digital Masters in Data Science (joint degree)
    - 1st year: Eötvös Loránd University
    - 2nd year: Polytechnic University of Madrid
- 2023 - present: Polytechnic University of Madrid, Artificial Intelligence PhD
    - Topic: Research software classification

Summer plans

- What city/country will you be spending this summer in?
- Do you have a full- or part-time job or internship planned for this summer?
- How many hours per week do you have available for a summer project?

GSOC experience

- Did you participate in a previous Summer of Code project?
  I did not participate in any previous Google Summer of Code project before.
- Have you applied or do you plan to apply for any other 2024 Summer of Code projects?
  I do not plan to apply to any other 2024 Summer of Code projects.
- Why did you decide to apply for a DBpedia project?
  I have studied ontologies and FAIRness during my studies and I was always fascinated by the topic. I believe in the goal of DBpedia, non-sensitive data should be open and freely available to everyone. I would enjoy working on a project that helps the community to extend the availability of these data.

If the mentors suggest, I'm willing to revise my proposal after submitting it because I'm eager to see this project come to reality this summer.

As for why I should be taken, my experience with ontologies in Python aligns well with the project's focus. I'm genuinely committed to the FAIR principles and enthusiastic about creating a tool that evaluates these standards in ontologies. Working on this project will deepen my understanding of ontology changes, the optimal approaches to address these changes, and how to assess an ontology's FAIR quality. This experience will be valuable in my future academic journey.