Time Agnostic Library

Release 3.1.0

Arcangelo Massari (https://orcid.org/0000-0002-8420-0696)

Apr 05, 2022

CONTENTS:

CHAPTER

ONE

TIME-AGNOSTIC-LIBRARY

1.1 time_agnostic_library package

1.1.1 Submodules

1.1.2 time_agnostic_library.agnostic_entity module

class time_agnostic_library.agnostic_entity.**AgnosticEntity**(*res: str*, *related_entities_history: bool = False*, *config_path: str = './config.json'*)

Bases: `object`

The entity of which you want to materialize one or all versions, based on the provenance snapshots available for that entity.

Parameters

- **res** (*str*) – The URI of the entity
- **related_entities_history** (*bool, optional*) – True, if you also want to return information on related entities, those that have the URI of the res parameter as an object, False otherwise. `False`
- **config_path** (*str, optional*) – The path to the configuration file. `'./config.json'`

get_history(*include_prov_metadata: bool = False*) → Tuple[Dict[str, Dict[str, rdflib.graph.Graph]], Dict[str, Dict[str, Dict[str, str]]]]

It materializes all versions of an entity. If **related_entities_history** is True, it also materializes all versions of all related entities, which have **res** as object rather than subject. If **include_prov_metadata** is True, the provenance metadata of the returned entity/entities is also returned. The output has the following format:

```
(
    {
        RES_URI: {
                TIME_1: ENTITY_GRAPH_AT_TIME_1,
                TIME_2: ENTITY_GRAPH_AT_TIME_2
        }
    },
    {
        RES_URI: {
            SNAPSHOT_URI_AT_TIME_1': {
                'generatedAtTime': GENERATION_TIME,
                'wasAttributedTo': ATTRIBUTION,
```

(continues on next page)

(continued from previous page)

```
                'hadPrimarySource': PRIMARY_SOURCE
            },
            SNAPSHOT_URI_AT_TIME_2: {
                'generatedAtTime': GENERATION_TIME,
                'wasAttributedTo': ATTRIBUTION,
                'hadPrimarySource': PRIMARY_SOURCE
        }
    }
)
```

Returns Tuple[dict, Union[dict, None]] – The output is always a two-element tuple. The first is a dictionary containing all the versions of a given resource. The second is a dictionary containing all the provenance metadata linked to that resource if **include_prov_metadata** is True, None if False.

get_state_at_time(*time: Tuple[Optional[str]]*, *include_prov_metadata: bool = False*) → Tuple[rdflib.graph.Graph, dict, Optional[dict]]

Given a time interval, the function returns the resource's states within the interval, the returned snapshots metadata and, optionally, the hooks to the previous and subsequent snapshots. The time can be specified in any existing standard. Snapshot metadata includes generation time, the responsible agent and the primary source. The output has the following format:

```
(
    {
        TIME_1: ENTITY_GRAPH_AT_TIME_1,
        TIME_2: ENTITY_GRAPH_AT_TIME_2
    },
    {
        SNAPSHOT_URI_AT_TIME_1: {
            'generatedAtTime': TIME_1,
            'wasAttributedTo': ATTRIBUTION,
            'hadPrimarySource': PRIMARY_SOURCE
        },
        SNAPSHOT_URI_AT_TIME_2: {
            'generatedAtTime': TIME_2,
            'wasAttributedTo': ATTRIBUTION,
            'hadPrimarySource': PRIMARY_SOURCE
        }
    },
    {
        OTHER_SNAPSHOT_URI_1: {
            'generatedAtTime': GENERATION_TIME,
            'wasAttributedTo': ATTRIBUTION,
            'hadPrimarySource': PRIMARY_SOURCE
        },
        OTHER_SNAPSHOT_URI_2: {
            'generatedAtTime': GENERATION_TIME,
            'wasAttributedTo': ATTRIBUTION,
            'hadPrimarySource': PRIMARY_SOURCE
        }
    }
)
```

Parameters

- **time** (*Tuple[Union[str, None]].*) – A time interval, in the form (START, END). If one of the two values is None, only the other is considered. The time can be specified using any existing standard.
- **include_prov_metadata** (*bool, optional*) – If True, hooks are returned to the previous and subsequent snapshots. `False`

Returns Tuple[dict, dict, Union[dict, None]] – The method always returns a tuple of three elements: the first is a dictionary that associates graphs and timestamps within the specified interval; the second contains the snapshots metadata of which the states has been returned. If the **include_prov_metadata** parameter is True, the third element of the tuple is the metadata on the other snapshots, otherwise an empty dictionary. The third dictionary is empty also if only one snapshot exists.

1.1.3 time_agnostic_library.agnostic_query module

class time_agnostic_library.agnostic_query.**AgnosticQuery**(*query: str*, *on_time: Tuple[Optional[str]] = (None, None)*, *config_path: str = './config.json'*)

Bases: `object`

class time_agnostic_library.agnostic_query.**DeltaQuery**(*query: str*, *on_time: Tuple[Optional[str]] = ()*, *changed_properties: Set[str] = {}*, *config_path: str = './config.json'*)

Bases: *time_agnostic_library.agnostic_query.AgnosticQuery*

This class allows single time and cross-time delta structured queries.

Parameters

- **query** (*str*) – A SPARQL query string. It is useful to identify the entities whose change you want to investigate.
- **on_time** (*Tuple[Union[str, None]], optional*) – If you want to query specific snapshots, specify the time interval here. The format is (START, END). If one of the two values is None, only the other is considered. Finally, the time can be specified using any existing standard. `()`
- **changed_properties** (*Set[str], optional*) – A set of properties. It narrows the field to those entities where the properties specified in the set have changed. `{}`
- **config_path** (*str, optional*) – The path to the configuration file. `'./config.json'`

run_agnostic_query() → Dict[str, Dict[str, str]]

Queries the deltas relevant to the query and the properties set in the specified time interval. If no property was indicated, any changes are considered. If no time interval was selected, the whole dataset's history is considered. The output has the following format:

```
{
    RES_URI_1: {
        "created": TIMESTAMP_CREATION,
        "modified": {
            TIMESTAMP_1: UPDATE_QUERY_1,
            TIMESTAMP_2: UPDATE_QUERY_2,
            TIMESTAMP_N: UPDATE_QUERY_N
        },
```

(continues on next page)

(continued from previous page)

```
        "deleted": TIMESTAMP_DELETION
    },
    RES_URI_2: {
        "created": TIMESTAMP_CREATION,
        "modified": {
            TIMESTAMP_1: UPDATE_QUERY_1,
            TIMESTAMP_2: UPDATE_QUERY_2,
            TIMESTAMP_N: UPDATE_QUERY_N
        },
        "deleted": TIMESTAMP_DELETION
    },
    RES_URI_N: {
        "created": TIMESTAMP_CREATION,
        "modified": {
            TIMESTAMP_1: UPDATE_QUERY_1,
            TIMESTAMP_2: UPDATE_QUERY_2,
            TIMESTAMP_N: UPDATE_QUERY_N
        },
        "deleted": TIMESTAMP_DELETION
    },
}
```

:returns Dict[str, Set[Tuple]] – The output is a dictionary that reports the modified entities, when they were created, modified, and deleted. Changes are reported as SPARQL UPDATE queries. If the entity was not created or deleted within the indicated range, the "created" or "deleted" value is None. On the other hand, if the entity does not exist within the input interval, the "modified" value is an empty dictionary.

class time_agnostic_library.agnostic_query.**VersionQuery**(*query: str*, *on_time: Tuple[Optional[str]] = ''*, *config_path: str = './config.json'*)

Bases: *time_agnostic_library.agnostic_query.AgnosticQuery*

This class allows time-travel queries, both on a single version and all versions of the dataset.

Parameters

- **query** (*str*) – The SPARQL query string.
- **on_time** (*Tuple[Union[str, None]], optional*) – If you want to query a specific version, specify the time interval here. The format is (START, END). If one of the two values is None, only the other is considered. Finally, the time can be specified using any existing standard. `''`
- **config_path** (*str, optional*) – The path to the configuration file. `'./config.json'`

run_agnostic_query() → Dict[str, Set[Tuple]]

Run the query provided as a time-travel query. If the **on_time** argument was specified, it runs on versions within the specified interval, on all versions otherwise.

:returns Dict[str, Set[Tuple]] – The output is a dictionary in which the keys are the snapshots relevant to that query. The values correspond to sets of tuples containing the query results at the time specified by the key. The positional value of the elements in the tuples is equivalent to the variables indicated in the query.

time_agnostic_library.agnostic_query.**get_insert_query**(*graph_iri: rdflib.term.URIRef*, *data: rdflib.graph.Graph*) → Tuple[str, int]

time_agnostic_library.agnostic_query.**init_lock**(*l*)

1.1.4 time_agnostic_library.prov_entity module

class time_agnostic_library.prov_entity.**ProvEntity**
Bases: object

Snapshot of entity metadata: a particular snapshot recording the metadata associated with an individual entity (either a bibliographic entity or an identifier) at a particular date and time, including the agent, such as a person, organisation or automated process that created or modified the entity metadata.

DCTERMS: ClassVar[rdflib.namespace.Namespace] = Namespace('http://purl.org/dc/terms/')

OCO: ClassVar[rdflib.namespace.Namespace] = Namespace('https://w3id.org/oc/ontology/')

PROV: ClassVar[rdflib.namespace.Namespace] = Namespace('http://www.w3.org/ns/prov#')

classmethod get_prov_properties()

iri_description: ClassVar[rdflib.term.URIRef] = rdflib.term.URIRef('http://purl.org/dc/terms/description')

iri_entity: ClassVar[rdflib.term.URIRef] = rdflib.term.URIRef('http://www.w3.org/ns/prov#Entity')

iri_generated_at_time: ClassVar[rdflib.term.URIRef] = rdflib.term.URIRef('http://www.w3.org/ns/prov#generatedAtTime')

iri_had_primary_source: ClassVar[rdflib.term.URIRef] = rdflib.term.URIRef('http://www.w3.org/ns/prov#hadPrimarySource')

iri_has_update_query: ClassVar[rdflib.term.URIRef] = rdflib.term.URIRef('https://w3id.org/oc/ontology/hasUpdateQuery')

iri_invalidated_at_time: ClassVar[rdflib.term.URIRef] = rdflib.term.URIRef('http://www.w3.org/ns/prov#invalidatedAtTime')

iri_specialization_of: ClassVar[rdflib.term.URIRef] = rdflib.term.URIRef('http://www.w3.org/ns/prov#specializationOf')

iri_was_attributed_to: ClassVar[rdflib.term.URIRef] = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasAttributedTo')

iri_was_derived_from: ClassVar[rdflib.term.URIRef] = rdflib.term.URIRef('http://www.w3.org/ns/prov#wasDerivedFrom')

1.1.5 time_agnostic_library.sparql module

class time_agnostic_library.sparql.**Sparql**(*query: str*, *config_path: str = './config.json'*)
Bases: object

The Sparql class handles SPARQL queries. It is instantiated by passing as a parameter the path to a configuration file, whose default location is "./config.json". The configuration file must be in JSON format and contain information on the sources to be queried. There are two types of sources: dataset and provenance sources and they need to be specified separately. Both triplestores and JSON files are supported. In addition, some optional values can be set to make executions faster and more efficient.

- **blazegraph_full_text_search**: Specify an affirmative Boolean value if Blazegraph was used as a triplestore, and a textual index was built to speed up queries. For more information, see https://github.com/blazegraph/database/wiki/Rebuild_Text_Index_Procedure. The allowed values are "true", "1", 1, "t", "y", "yes", "ok", or "false", "0", 0, "n", "f", "no".

- **graphdb_connector_name**: Specify the name of the Lucene connector if GraphDB was used as a triplestore and a textual index was built to speed up queries. For more information, see https://graphdb.ontotext.com/documentation/free/general-full-text-search-with-connectors.html.
- **cache_triplestore_url**: Specifies the triplestore URL to use as a cache to make queries faster. If your triplestore uses different endpoints for reading and writing (e.g. GraphDB), specify the endpoint for reading in the "endpoint" field and the endpoint for writing in the "update_endpoint" field. If there is only one endpoint (e.g. Blazegraph), specify it in both fields.

Here is an example of the configuration file content:

```
{
    "dataset": {
        "triplestore_urls": ["http://127.0.0.1:7200/repositories/data"],
        "file_paths": []
    },
    "provenance": {
        "triplestore_urls": [],
        "file_paths": ["./prov.json"]
    },
    "blazegraph_full_text_search": "no",
    "graphdb_connector_name": "fts",
    "cache_triplestore_url": {
        "endpoint": "http://127.0.0.1:7200/repositories/cache",
        "update_endpoint": "http://127.0.0.1:7200/repositories/cache/statements"
    }
}
```

Parameters **config_path** (*str, optional*) – The path to the configuration file. `'./config.json'`

run_construct_query() → rdflib.graph.ConjunctiveGraph
Given a CONSTRUCT query, it returns the results in a ConjunctiveGraph.

Returns ConjunctiveGraph – A ConjunctiveGraph containing the results of the query.

run_select_query() → Set[Tuple]
Given a SELECT query, it returns the results in a set of tuples.

Returns Set[Tuple] – A set of tuples, in which the positional value of the elements in the tuples is equivalent to the variables indicated in the query.

1.1.6 time_agnostic_library.support module

time_agnostic_library.support.**empty_the_cache**(*config_path: str = './config.json'*) → None
It empties the entire cache.

Parameters **config_path** (*str, optional*) – The path to the configuration file. `'./config.json'`

time_agnostic_library.support.**generate_config_file**(*config_path: str = './config.json'*, *dataset_urls: list = []*, *dataset_dirs: list = []*, *provenance_urls: list = []*, *provenance_dirs: list = []*, *blazegraph_full_text_search: bool = False*, *graphdb_connector_name: str = ''*, *cache_endpoint: str = ''*, *cache_update_endpoint: str = ''*) → None

Given the configuration parameters, a file compliant with the syntax of the time-agnostic-library configuration files is generated. :param config_path: The output configuration file path

`'./config.json'`

Parameters

- **dataset_urls** (*list, optional*) – A list of triplestore URLs containing data `[]`
- **dataset_dirs** (*list, optional*) – A list of directories containing data `[]`
- **provenance_urls** (*list, optional*) – A list of triplestore URLs containing provenance metadata `[]`
- **provenance_dirs** (*list, optional*) – A list of directories containing provenance metadata `[]`
- **blazegraph_full_text_search** (*bool, optional*) – True if Blazegraph was used as a triplestore, and a textual index was built to speed up queries. For more information, see https://github.com/blazegraph/database/wiki/Rebuild_Text_Index_Procedure `False`
- **graphdb_connector_name** (*str, optional*) – The name of the Lucene connector if GraphDB was used as a triplestore and a textual index was built to speed up queries. For more information, see [https://graphdb.ontotext.com/documentation/free/general-full-text-search-with-connectors.html{]}(https://graphdb.ontotext.com/documentation/free/general-full-text-search-with-connectors.html) `''`
- **cache_endpoint** (*str, optional*) – A triplestore URL to use as a cache to make queries on provenance faster `''`
- **cache_update_endpoint** (*str, optional*) – If your triplestore uses different endpoints for reading and writing (e.g. GraphDB), specify the endpoint for writing in this parameter `''`

1.1.7 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

S

T

V