Query formulation on a conceptual data model with explicit inferences of knowledge

Mandisa Baleni
Computer Science Department
University of Cape Town
South Africa
blnman002@myuct.ac.za

Bradley Malgas
Computer Science Department
University of Cape Town
South Africa
mlgabra002@myuct.ac.za

1 Introduction

Intelligent data access uses knowledge to efficiently manage data and improve decision making. Data access is often restricted by what a user knows about a database and how data is stored in it. But this is not the only restriction to effective data access. In order to optimize queries about data, users often need to be skilled in structured query languages. To give users the ability to optimize data access without the need to learn a query language, visual query formulation can remove the restrictions on most data systems which can be achieved by using a visual query system. A visual query system is a visual programming environment that uses user interaction in order to generate a query [1]. The visual aspect of this system pertains to data represented as a map with objects that express concepts in a subject and how they relate to each other. These are more formally known as conceptual data models like enhanced-entity relationship diagrams (EER) or ontologies (see figure 1).

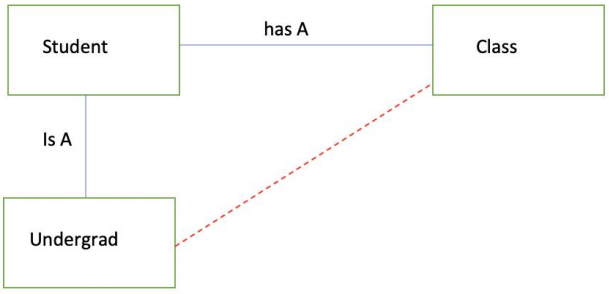


Figure 1: Arbitrary conceptual data model

Conceptual data models and ontologies need mechanisms to display domain expert knowledge in such a way that is expressive [2]. This happens when concepts and their interactions make the nature of the subject domain completely clear. Part of achieving this is explicitly capturing any implicit knowledge of the subject domain in the data model. For example, in Figure 1, one would implicitly acknowledge that an undergrad has a class since they are a student. That is not always explicitly included in a model. However, there are no effective mechanisms of doing this for most models. The process of acknowledging this implicit knowledge is known as reasoning and the process of explicitly adding it (or sometimes removing) is the materialisation of deductions.

An architecture addressing called *KnowID* [3] has been proposed with components that are meant to address the issues of query formulation and the materialisation of deductions (see figure 2). This project will dive into specification of KnowID's proposal of the circled components.

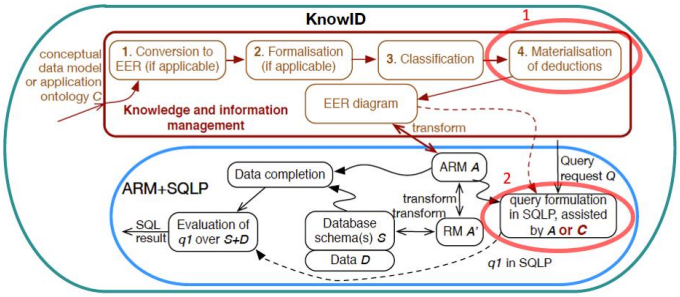


Figure 2: The KnowID architecture [3]

Our project focus is circled in red. Work allocation is equally distributed between team members. Each member will focus on a single component as follows:

1. **Materialisation of deductions**: Mandisa Baleni
2. **Query formulation in SQLP**: Bradley Malgas

1.1 Related Works

1.1.1 Related works for Visual Query Formulation

Different systems take different approaches to achieve visual query formulation. The GBLENDER system [4] for example, uses graphs and subgraphs in order to achieve visual querying. A query is represented as a subgraph and the database is then searched to see whether any graphs in the database contains this particular subgraph (called a *fragment*). Another system which employs graph strategies, is the OptiqueVQS system [5]. Here, user queries are represented as a tree-

like structure while making suggestions of which nodes to link while the user construct the query.

1.1.2 Related works for materialisation of deductions

The Intelligent Conceptual Modelling tool (ICOM) [6] is a tool that can partially reason and materialise over all EER's features. It uses what is known as a DIG protocol where the user must provide a URL for a reasoner that "follows" said DIG protocol. There are APIs and programming language modules for modifying ontologies using reasoning services. A java-based API is the OWL API [7] and a python-based programming language module is OWLready [8]. Both require the input ontology to be serialised in OWL 2 specification.

2 Problem Statement

2.1 The Query formulation problem

Experts in a specific domain need to use built-in queries or rely on IT professionals to create them, resulting in an inefficiency to create powerful and meaningful custom queries. Their main challenge is a lack of technical experience with structured query languages. Our main problem to address creating the ability to generate visual queries using an EER diagram. Currently, no existing tools allow users to provide EER diagrams or Abstract Relational Models (ARM) and generate queries in SQLP over them.

2.1.1 Issues with existing visual query system tools

The main issues of concern are:

1. Technical limitations: Some existing technical issues regarding visual query systems are the ability to deal with complex issues that involve cyclic or disjunctive queries and those involving negation. Other issues identified include not being able to undo or delete during the querying process as identified in the GBLENDER [4] visual query system. Another factor to consider is that most of the existing solutions work with SPARQL. There currently exists no support for SQLP.

2. User types: Each visual query system also targets different users. Highly specialized system such as the project ARIANE [9], which is exclusively for use within medical databases, have the advantage of leveraging existing ontologies to provide users with familiar concepts. Consequently, this means it does not cater for the broad, general user.

2.2 The materialisation of deductions problem

Fillotrani and Keet [3] propose a process that takes a logic-based formalisation of a subject domain to reason over and deduce any implicit facts in the model not captured in the diagram and then materialise thosse deductions by altering the diagram with the inferences made. However, there are no tools or algorithms that can fully complete the materialisation of deductions over ontologies and conceptual data models after being reasoned over and especially for the conceptual model known as Enhanced Entity Relationship diagram (EER) or an ontology encapsulated with all of its features. Another issue on a smaller scale is that most models and ontologies have materialisation of deductions done by reasoning that operates under open world assumption (assuming that a statement can be true whether it's known or not) as opposed to the more specific closed world assumption (assuming a statement is true only if its known) [3].

2.2.1 Issues with existing Materialisation of deductions tools

Main issues with existing tools to materialise deductions can be classified into 2 categories:

1. Compatibility: Some tools don't have reasoners (and subsequently inference capabilities) to imply knowledge for all EER features supported by KnowID. ICOM is not suited for EER's following features: weak vs. strong entity type, n-ary relationships, attributes and identifiers since its conceptual modelling languages can only express EER's following features: disjointess constraints, covering constraints, entities, and some cardinality constraints (mandatory participation, exactly one participation, at most one participation). Even though the OWL API supports ontologies of various syntaxes, it is not adjusted to any specific syntax or model as classes (entities) and axioms are based on OWL 2 Structural Specification [7]. Owlready also uses OWL 2 specification which does not support all of the EER features supported by KnowID. OWLready supports classes (and subclasses), disjointness, attributes, and datatypes (attributes) and object types (relationships) [8] but limited for other EER features.

2. Technical limitations: There aren't reasoners to express the features needed by an EER in KnowID and operate under closed world assumption (CWA). ICOM's DIG protocol is not actively developing meaning new reasoners don't support it [6]. The OWL API requires OWL reasoners like Racer Pro for reasoning which may not have the full reasoning capabilities we need for EER. Owlready has a HermiT reasoner which has limited reasoning capabilities for all of EER's features and OWL reasoners operates under open world assumption. Editing ontologies with the OWL API is

also more complex than using object programming languages to do so as done by Owlready.

2.2 Requirements

Some important requirements of the software system include:

- Creating accurate user queries in SQLP using EER diagrams
- Allowing users to recover from mistakes in the querying process (whether through deleting or undoing)
- Compliance with the already existing architecture (current transformation rules)
- Allow a modular design that can be used in a variety of use cases
- Reasoning suited for all of EER's features
- EER formalized in an ontology in suitable logic like OWL 2 DL
- Validation services after materialising deductions

2.3 Users

They are experts in the subject domain that the data interprets who want to process queries but with limited understanding of query languages and benefit from querying over a conceptual data model encapsulating the knowledge of their domain. They want to have the ability to create custom queries that suits their current needs and capture the importance of the relationships between the data contained without having the necessary technical knowledge.

2.4 Aims

1. Develop a visual query system that will allow users to leverage their understanding of the data contained in the database and use an EER diagram to create SQLP queries. The query system will be easy-to-use and needs to create correct and accurate SQLP queries.
2. Create an algorithm to facilitate with EER-to-SQLP transformation.
3. Achieve materialisation of deductions with open world assumption reasoning for the following reasoning tasks (over most of EER's listed features supported by KnowID):
    - Relationship subsumption
    - Entity subsumption (closed world assumption if there is enough time)
    - Entity consistency
    - Relationship consistency
4. Have a management tool that allows experts to approve or deny certain deductions inferred by the reasoner based on its sensibility in the real world and inform users about inconsistencies in specified models and choose what to do with them.

The first two aims will be achieved by the Visual Query Formulation section by Bradley Malgas. The remaining two will be achieved by the Materialisation of Deductions section by Mandisa Baleni.

3 Procedures and Methods

3.1 Procedures for the materialisation of deductions component

A python program will be implemented to materialise the deductions inferred by a reasoner when it is given an OWL file that encapsulates an EER model with all the features that KnowID supports. The program will include the following features:

a. *A simple text user interface* used to navigate the program. This will include services like file path specification of OWL files and requesting reasoning and materialisation of deductions. This will contribute to the 3rd and 4th aim listed in 2.4.

b. *An extension of Owlready*, a python-based module for ontology engineering (see figure 2) [8]. It will extend and modify the ontology modification algorithms implemented in Owlready to closely suit the features of EER. The module currently provides support for OWL 2 constructs. We will modify the ontology-specific python source files that make modifications based on reasoning and possibly the methods that map the OWL files to the python object model so that it better supports OWL 2 DL because it's closest to OWL full (EER-adjacent) [10]. The ontology will be sent to the HermiT reasoner to do the general open world assumption reasoning and the resulting inferences will be added to the ontology This satisfies aim 3 in 2.4. If there's time, Owlready's closed world algorithm will be used for each of the classes in the ontology to do whatever closed world reasoning it can do, the resulting inferences will be added to a new ontology. This will also address aim 3 in 2.4. We will analyse the consequences of including the close world reasoning and examine whether its worthwhile.

c. *Wrapper management tool for verifying materialisation of deductions* that will allow users to choose what to do with the deductions returned and inform them of the consequences of their decisions. This addresses the 4th aim in 2.4.

This aspect of our project will be implemented using an iterative and incremental development approach that is test driven. It will be done in three iterations with prioritized features addressed in earlier stages. The first iteration will focus on core EER features, the second iteration on the rest of EER features and the third iteration on the wrapper management tool. Each iteration will start of by (1) designing the test cases of sample models with expected I/O first, (2) followed by code implementation, and lastly (3)

followed by testing. Steps 2 and 3 will be continuously repeated if tests aren't passed.

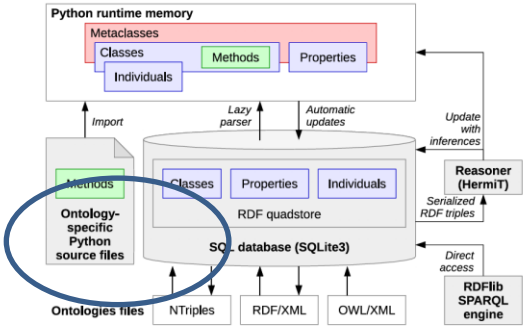


Fig. 2. General architecture of Owlready.

Figure 2: The architecture of Owlready [8]

3.2 Procedures for the query formulation component

A prototype of the software will be developed as a sub-component of what the full software will be capable of. These are the key features that will be prioritized in the protype design in order to demonstrate that the research problem has been adequately solved:

- ***a.*** *Visual querying ability:* The users should be able to make use of the querying environment to construct queries. They will be shown the status of the query through a "tree-graph" that represents the query. The available queries in the prototype will only be a subset of the availability and range of queries expected from the final software.
- ***b.*** *SQL^{path} (SQLP) Query support:* The system needs to be able to generate queries in SQLP. The query running will be simulated in the prototype for demonstration purposes only.
- ***c.*** *Checking of EER using ARM transformation:* Since our queries operate on the EER diagrams, we will make use of the existing algorithm to transform the EER diagram to an ARM that will integrate with the rest of the system. The classified information obtained from the ARM will be associated with the EER diagram.
- ***d.*** *A widget-based visual query environment:* While this feature is not core to the desired outcome, if time allows, we will provide an interface to users that will be intuitive, easy-to-use while being modular in its design. Each widget component serves as a separate application.

The implementation strategy will be to build a brand-new application from the ground up. The reason for taking this approach is due to the fact that there are currently no existing tools that can achieve what we are aiming to solve with this system. We will be using an Agile software development methodology. This will allow us to optimize the development process for rapid deployment. This decision lines up with the milestone of software feasibility demonstration – having the ability to make changes during the development process is why the Agile methodology was chosen.

The entire application is to be developed in Python. We will begin by designing a model that will be used to encapsulate the data contained in the EER diagram. This will be done using the Model representation in the *Django Framework*. Once we are able to have a useful way to interact with the models, including the relationships between them, we can extract this information. This is needed in order allow it to be used during the querying process. Each piece of extracted information will be classified as being either: a table name or a variable name. This information will be obtained easily from the ARM structure (Figure 3). We will be working under the assumption that the provided ARM diagram will be representative of information contained in the underlying relational database.

Once the information has been classified accordingly, we then able to link this information with EER diagram which will be used as the basis for actual user interaction. Users will then be able to design a query that uses will be matched according to the actual SQLP format. This can be achieved by designing an EER-to-SQLP query algorithm that will be based around the grammar shown in Figure 4.

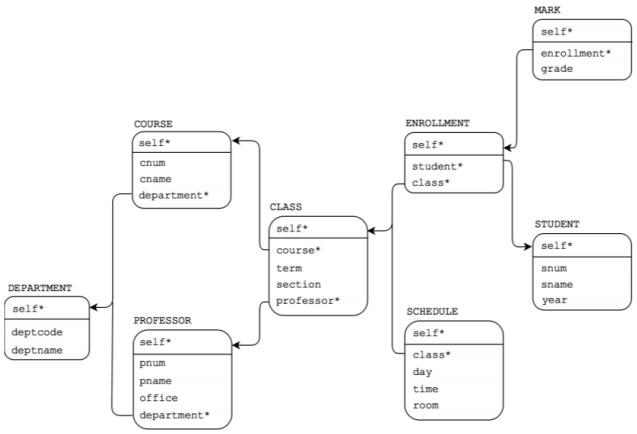


Figure 3: ARM Structure [11]

This structure as demonstrated by the grammar will be visually represented as a tree-like structure that will have nodes added to it, each time that a user adds a new element. In following this procedure, we will have completed designed a Visual query System.

$$\langle SQL^{path}{}_{query} \rangle ::= \texttt{select distinct } t_1, t_2, \ldots, t_m \; \langle SQL^{path}{}_{body} \rangle$$

$$\langle SQL^{path}{}_{body} \rangle ::= \texttt{from } T_1\ x_1, T_2\ x_2, \ldots, T_n\ x_n \; \texttt{where } \langle cond \rangle$$

$$\begin{aligned} \langle cond \rangle ::= {} & t \texttt{ op } t' \\ | \; & t \texttt{ op } c \\ | \; & \langle cond \rangle \texttt{ and } \langle cond \rangle \\ | \; & \texttt{exists (select } * \langle SQL^{path}{}_{body} \rangle) \\ | \; & \texttt{not } \langle cond \rangle \end{aligned}$$

Figure 4: SQLP Grammar [11]

3.3 Evaluation

We take several different testing methodologies to evaluate materialisation of deductions and query formulation respectively.

3.3.1. UI testing
We will begin with UI testing by assessing that each GUI component functions individually during each step of the querying process. This includes tests such as: checking that every button responds to presses in different scenarios, ensuring that resizing the window does impact on the ability to use the interface etc.

3.3.2. Performance testing
The system addressing query formulation will make use of performance testing. The way in which we will analyse the performance of the system is the speed at which queries are generated and how resource intensive the software is. This test will be performed on at least 2 different PC setups. Performance testing will only be conducted for the query formulation since this is the only component within our scope done at runtime.

3.3.3. Acceptance testing
Lastly, we will also employ acceptance testing. Since our main aim is to generate accurate queries, we will do acceptance tests on the generated queries against the query that was expected from the system.

3.3.4. System testing
To test the materialisation of deductions, we will use the sample model examples with EER features compatible to OWL 2 DL from a guide on mapping OWL full to EER [1l] and tweek them to test for correct subsumption and consistency modifications. We will introduce subclasses, sub relationships, and possible inconsistencies to test across most of EER's features supported by KnowID namely:

a. entity type (strong)
c. attributes
d. basic cardinality constraints
e. identifier
f. entity type subsumption
g. dis-jointness constraint
h. covering constraint

4 Ethical, Professional and Legal Issues

Extending Owlready is permissible. It is available under GNU GPL v3 license which is a free and copyleft license for software which aims to give freedom to share and change all versions of a program. Django is a Python-based free and open-source web framework released publicly under a BSD license. Both components will also be developed under GNU. Our programs will be open-sourced so that others are free to use and develop further for free.

5 Anticipated Outcomes

5.1 System

Our overall system will be modular so both components can be developed independently, but still able to be used together and in KnowID. We will speak more about the software and its key features as well as any design challenges.

5.1.1 Visual Query Environment
Based on approaches taken by related systems (see Section 5.2), the visual query environment will allow users to interact with a graph representation of the query being formulated. The system needs to allow users to link data elements from the database to form a query. Each element is represented as a node in the overall query graph. Data element to be used in the query will be pre-selected from the provided EER diagram. This approach presents a design.

5.1.2 Materialisation of deductions system
A major result is a system able to compute EER diagrams with explicit inferences that make sense to the logic of the reasoner and to the reality of the subject domain.

5.2 Impact

The impact of the results of the visual query formulation will be substantial and can go a long way into simplifying querying for the inexperienced users. We would be able to compare the time taken to construct a query using a structured query language vs using the visual querying environment.

If the materialisation of deductions can be realized for more features of EER, ontologies and conceptual data models can express knowledge more expressively and better suited to use over relational databases. Moreover, if that entire component of the project is successful, more reasoning and materialisation of deductions can be done under the closed world assumption.

5.3 Key success factors

In order to ensure that we have a successful project we will determine whether it can achieve the most basic tasks. These can be measured as completed or not using the following the following criteria:

- Can we load an EER diagram?
- Can we use the loaded diagram for queries?
- Can we generate an SQLP query using the visual query environment?
- Is the generated SQLP query an accurate representation of the visual query?
- Can materialisation of deductions be performed with open world reasoning on an ontology with classes and sub classes (entities), attributes, relationships and sub-relationships, cardinalities, disjoint relationship?
- Can a user make decisions about what to do with added features in an ontology?

If we can achieve all of the above tasks, then the project cab be deemed successful as these are the core requirements to solving the initial problem statement.

6 Project Plan

6.1 Risks

The risks for this project are shown in a risk matrix in **Appendix B**. It shows mitigation and management strategies for risks along with their impact and likelihood of taking place. The risk of the Covid-19 pandemic may greatly limit with access to communication amongst us as developers which may interfere with ensuring the materialisation of deductions component is compatible with query formulation developed separately.

6.2 Timeline

The project timeline is already in motion as of the 30th of March when work commenced on the Literature review of the various topics. This was followed up by this project proposal and will be furthered with other deliverables. This is reflected in the timeline (Gannt Chart) in **Appendix A**.

6.3 Required resources

6.3.1 Software Resources

- Python
- React
- Django Framework

6.3.2 Hardware Resources

- Personal Computers

6.3.3 Data Resources

- Owlready python module
- EER-ARM bi-directional algorithm module

6.4 Deliverables

The list of final deliverables that will be produced at the end of this project are as follows:

- Visual query system
- EER to SQLP algorithm
- Owlready extened
- Materialisation of deductions system with reasoning
- Final Research Paper
- Project Poster
- Project Web Page
- Reflection Paper

6.5 Milestones

As shown in the timeline, a few activities are marked with an orange diamond to signify that they are of importance. These major milestones will be used to determine when a significant portion of the project has been completed (key milestones indicated with diamonds in gantt chart):

- Completed Literature Review
- Rough Draft of Project Proposal Completed
- Completed Final Project Proposal
- Created data model to represent input
- Completed SQLP Query Generation Algorithm
- Designed Visual Query System
- Completing the first iteration of materialising deductions as shown in the gantt chart will be significant since the most important features would have been completed with owa reasoning
- Completing the 3rd iteration to implement a wrapper since it provides key functionality for a user to validate the model
- Software Feasibility Demo
- Final Draft of Research Paper

REFERENCES

[1] Patricia S. Abril and Robert Plant. 2007. The patent holder's dilemma: Buy, sell, or troll? *Communications of the ACM* 50, 1 (Jan, 2007), 36-44.

[2] Germán Braun, Laura Cecchi, Pablo Fillottrani, and Christian Gimenez. 2016. crowd: A Tool for Conceptual Modelling assisted by Automated Reasoning. In *Preliminary Report*, 2016, Argentina, 29-41.

[3] Pablo R. Fillotrani and C. Maria Keet. 2020. KnowID: An architecture for efficient Knowledge-driven Information and Data access. Universidad Nacional del Sur, Bah´ıa Blanca, Argentina. University of Cape Town, South Africa.

[4] Jin, C., Bhowmick, S. S., Xiao, X., Cheng, J. and Choi, B. GBLENDER: towards blending visual query formulation and query processing in graph databases. City, 2010.

[5] Soylu, A., Kharlamov, E., Zheleznyakov, D., Jimenez Ruiz, E., Giese, M. and Horrocks, I. Ontology-based visual query formulation: An industry experience. Springer, City, 2015.

[6] Pablo R. Fillottrani, Enrico Franconi, and Sergio Tessaris. 2012. The ICOM 3.0 Intelligent Conceptual Modelling tool and methodology. *Semantic Web* 3, 3 (2012), 293-306.

[7] Sean Bechhofer and Matthew Horridge. 2009. The OWL API: A Java API for Working with OWL 2 Ontologies. In *Proceedings of the 6th International Conference on OWL: Experiences and Directions (OWLED'09),* October, 2009. CEUR-WS.org, Aachen, Germany, 49-58.

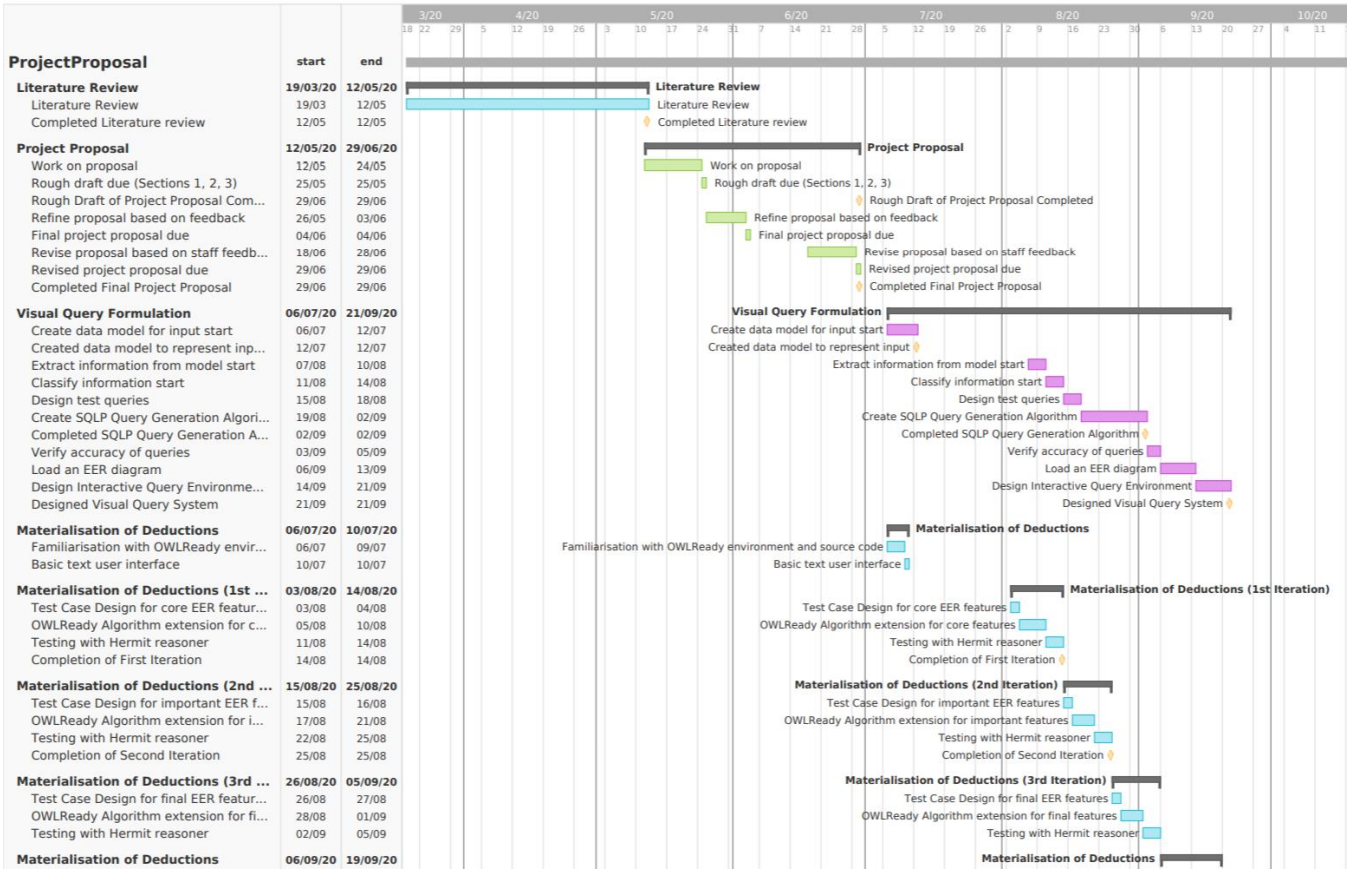
[8] Jean-Baptiste Lamy. 2017. Owlready: Ontology-oriented programming in Python with automatic classification and high-level constructs for biomedical ontologies. Artificial Intelligence in Medicine 80, 11-28.

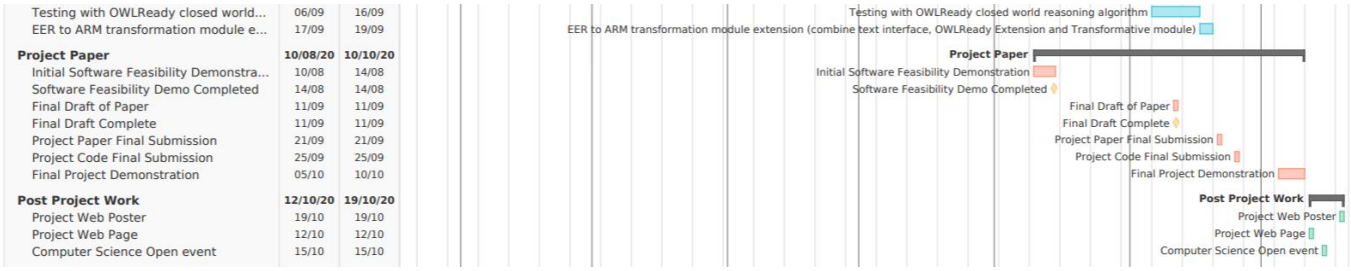
[9] Joubert, M., Fieschi, M., Robert, J.-J., Volot, F. and Fieschi, D. UMLS-based conceptual queries to biomedical information databases: an overview of the project ARIANE. Journal of the American Medical Informatics Association, 5, 1 (1998), 52-61.

[10] Sikha Bagui. 2009. Mapping OWL to the Entity Relationship and Extended Entity Relationship Models. In *International Journal of Knowledge and Web Intelligence* 1, 1-2 (September 2009), 125-149.

[11] Weicong Ma. 2018. *On the Utility of Adding an Abstract Domain and Attribute Paths to SQL*. Master's thesis. University of Waterloo, Ontario Canada.

Appendix A: Timeline (Gantt Chart)





Appendix B: Risk Matrix

Risk	Probability	Impact	Consequence	Mitigation	Monitoring	Management
Covid-19 prevents reliable communication between partners.	4	1	Lack of integration amongst the project as a whole but components can still contribute to KnowID architecture respectively	Reduce dependency on constant communication by arranging few meetings addressing concerns in large	Continue to check in with resources and network availability	Communicate with supervisor and continue development based on previous agreements
Team member drops out of honours.	5	1	Insignificant since both components are being developed independently	Continuous consolidation, enquiries and assistance where possible	Maintain contact and support of and enquire about headspace, health and circumstance during Covid	Communicate with supervisor and convener for suitable adjustment on project going forward
Owlready source code too complex and difficult.	6	9	Significant delay and obstruction to implementing extension to Owlready for materializing deductions	Engage with source material and other research papers on Owlready	Maintain regular communication with supervisor to consolidate understanding	Seek alternative API (Protégé Jena API) or ontology modification algorithms
Missing deadline.	1	10	Not handing in project report results in failure	Adhere to specified timeline in project plan	Check ins with supervisor and timeline to confirm progress	Consult supervisor for a possible readjustment and request to reduce scope and relay any significant troubles if any
Output produced by materialization of deductions system is incompatible for query formulation.	6	4	Query formulation can source other readily available EER/ARM models for testing and reporting	Small sample models for use case tests	Delivering sample ARM in segments	Use other tested/design ARM sample models for testing query formulation