Knowledge Graph Syntax Validation and Visual Navigation for Developing Intelligent Systems

by
Claude Asamoah

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

at

School of Computer Science and Information Systems
Pace University

May 2016

We hereby certify that this dissertation, submitted by **Claude Asamoah** satisfies the dissertation requirements for the degree of Doctor of Professional Studies in Computing and has been approved.

Lixin Tao — 5/11/16

Dr. Lixin Tao — Date
Chairperson of Dissertation Committee

Ronald Frank — 5/11/16

Dr. Ronald Frank — Date
Dissertation Committee Member

Meikang Qiu — 5/11/16

Dr. Meikang Qiu — Date
Dissertation Committee Member

Seidenberg School of Computer Science and Information Systems

Pace University

Abstract

Knowledge Graph Syntax Validation and Visual Navigation for Developing Intelligent Systems

by
Claude Asamoah

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

May 2016

Intelligent systems depend on effective knowledge representation and knowledge-based decision-making. While OWL is the dominant industry standard for knowledge representation, it has some limitations which include the lack of support for custom relations, it's reliance on the single "is-a" relation, and the emulation of other relations via complex object and data properties. Pace University has extended OWL to support knowledge graph as a replacement to better support knowledge representation and decision-making. One of the challenges is how to better support domain experts to create knowledge graphs and verify their correctness. Since a real-life knowledge graph can easily contain hundreds or thousands of classes with complex inter-relations, it is a major challenge for domain experts to review and validate their knowledge representation, and hard for application developers to fully understand the complex relations among the classes.

This research contributes a knowledge graph syntax validation algorithm and two knowledge graph visualization tools. Use cases such as the Cyber Security Communications Facilitator are used to verify the correctness and effectiveness of the contributed solutions.

Acknowledgements

I will like to thank my wife Betty Asamoah for supporting me in my quest to reach the apex of my educational aspiration. I also want to thank my children Brandon Asamoah, Ama Asamoah, and Alex Smith for their continued support in this endeavor. My special thanks to Dr. Lixin Tao, my academic advisor and Chairperson, Computer Science Department, Westchester, Chair of Ph.D. in Computer Science Program, Chair of Doctorate Professional Studies (DPS) in Computing Program for believing in me and guiding me through the completion of this Dissertation. Finally, I thank the Almighty God for giving me the strength to reach this great height of academic accomplishment.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Opportunities and Challenges in Knowledge-Based Decision-making

The Semantic Web has extended knowledge representation which was one of the goals of Artificial Intelligence (AI). AI started with pioneers such as Alan Turing (1912-1954) whose 1950 paper "Computing Machinery and Intelligence" is one of the most frequently cited in modern philosophical literature. His work is regarded by many scholars as the foundation of computer science and of the artificial intelligence program [3]. In the late 1950s and early 1960's notables such as Alan Turing, Marvin Minsky, John McCarthy and Allen Newell thought that computers that could "think" as humans do were just around the corner [4]. As cited by Harry Haplin [18], the goal of AI as stated by John McCarthy at the 1956 Dartmouth Conference is "the study to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it" [50]. The vision that machines could do practically anything humans can do did not materialize as envisaged by AI pioneers. Although AI had done well in "tightly-constrained domains," extending this ability was not sustainable [18]. Knowledge transfer that AI was supposed to address as creating a database for all the knowledge of the world did not materialize and became increasingly evident that it was rather going to create a virtual Tower of Babel of knowledge. Even within a specific knowledge representation domain such as semantic networks, it was established that a principal element such as a 'link' was interpreted in many different ways [18]. As such, Knowledge Representations were in-

accurate in determining the represented knowledge such as the use of first-order predicate logic, which was analogous to most of the knowledge representation systems used during that time [18]. Common-sense knowledge was formalized by researchers but their approach never merged into a universal platform for representing all knowledge, as stated by the influential Brachman-Smith survey [18].

Good decision-making and processing their results in an optimal manner is the consequence of quality Knowledge Representation (KR). Intelligence is "defined as the ability of a system to act appropriately in an uncertain environment, where appropriate action is that which increases the probability of success, and success is the achievement of behavioral sub goals that support the system's ultimate goal" [23]. Countless opportunities exist in knowledge based decision-making. For example, Web Mining is the extraction of pertinent data from distributed websites across the Internet. The Internet may be viewed as a huge database consisting of disparate and distributed data and at times there may be a need to traverse these various website collecting and collating pertinent data about a particular subject using agents or crawlers. Before the advent of Knowledge Based decision-making mechanism such as the Semantic Web, documents on the web contained a lot of information for computers to present them but were not understood by them. Tremendous opportunities exist for computers to understand some of the information embedded in the web documents and act upon them to the benefit of web users. The Semantic Web as described by its inventor Tim Berners-Lee as "an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation" [20]. The opportunity that Semantic Web brings by enabling machines to understand data and make decisions on

them will increase the viability of the web as well as present a means of doing complex and precision based activities to benefit mankind as we enter the age of Internet of Things (IoT) whereby persons, appliances, gadgets, and computers can communicate in unison.

OWL (Web Ontology Language) is the industry standard language for knowledge representation. There are tools that make it easier to build ontologies such as the Stanford Protégé. Stanford Protégé is used to create the OWL document in an OWL/XML, RDF/XML, and other formats. The RDF/XML based OWL document relates two classes together using the "is-a" relation. The "is-a" relation is currently the only first class relation used by OWL to relate two classes without user declaration. The use of this single relation "is-a" limits the ability and flexibility to relate two classes in profound ways by a subject domain expert who may want to express the relations between two classes in specific ways using custom relations. Though you can link two individual entities in a triple format via object properties and also link a data value to an individual entity via data properties, this approach of defining the relations of two classes using properties and their respective data properties if warranted via restrictions, is more complex than using the parsimonious approach of custom relations of Knowledge Graphs.

Another challenge is how to use OWL to represent knowledge in visual knowledge navigation and review. Although OWLViz, an add-on in Protégé, can graphically display the relations diagrams between classes, if the ontology is large, it becomes cumbersome to visualize all the relations in its entirety in the Knowledge Graph (KG) and therefore there is need for a visual navigation mechanism whereby all the entities, classes and relations, can be navigated easily.

1.2 Knowledge Representation Alternatives

1.2.1 Rule Based Approach

Judgmental knowledge is allowed by Rule-based inference systems about a specific problem domain to be represented as a collection of discrete rules. Each rule states that if certain assumptions are understood, then certain conclusions can be inferred [27]. W3C chartered the Rule Interchange Format (RIF) Working Group and tasked them to produce extensions in addition to a core rule language which in combination, allow rules to be translated between rule languages and subsequently between rule systems [34]. The RIF Working Group is challenged to amalgamate the needs of a diverse community including business rules and semantic web [34]. The Semantic Web Rule Language (SWRL) includes a high-level abstract syntax for Horn-like rules in both the OWL DL and OWL Lite sublanguages of OWL [35]. Rules are modes for representing knowledge and most often goes beyond OWL1 and are typically conditional statements in the Semantic Web for example the *if then clauses* [36]. In addition, rules expand the expressive power of SWRL. Rules are straight forward and mainly correspond to traditional operations available in major programming languages. Examples are Comparisons, Mathematical transformations, and modifiers. One of the popular rules used in the semantic web is the Jena Rule which includes a list of body terms or premises (the "if" clause) and a list of head terms or conclusions (the "then" clause) [36].

1.2.2 Web Ontology Language – OWL

According to Tom Gruber, an "ontology is a specification of a conceptualization" [1]. Ontologies enable the structuring of data in a hierarchical form. Recent work in Artificial

Intelligence (AI) is exploring the use of formal ontologies as a way of specifying content-specific agreements for the sharing and re-use of knowledge among software entities [11]. Web Ontology Language (OWL) is part of the Semantic Web which extends the current Web by extending current semantics to it [12]. OWL is one of the most used languages for creating ontologies which is the latest recommendation of W3C and is based on the RDF schema.

OWL = RDF schema + new constructs for expressiveness [13].

RDF is the basic block for supporting the Semantic Web and is all about vocabulary or metadata and supported by W3C. It is structured, machine readable, and capable of describing any resource independent of any domain [13].

OWL has three sublanguages OWL Lite, OWL DL, and OWL Full.

OWL Lite supports those users primarily needing a classification hierarchy and simple constraints [22].

OWL DL supports those users who want the maximum expressiveness while retaining computational completeness [22].

OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees [22].

Ontology is a vehicle to capture knowledge about a particular domain. It describes the relations between the entities within that domain. Web Ontology Language, OWL, is one of the most used languages for creating ontology which is the latest recommendation of W3C. The OWL consists of individuals, properties, and Classes.

Individuals are instance of a class. For example, John is an individual or instance of a Student class or USA is an individual or instance of a Country class.

Properties are links that relates two individuals together. For Example, *John livesIn Boston*. Properties have many characteristics such as inverse, transitive, asymmetric, or symmetric.

Classes: can be described as a set that has individuals as members. Examples of a class are Country and Car.

OWL uses the Triple, as agreed by W3C, to describe the relations between two entities. Figure 1 depicts this relation.

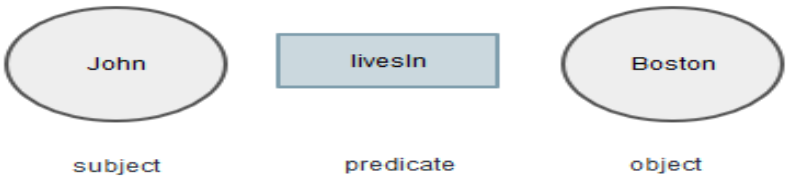


Figure 1: Triple Example

The most popular tool for creating OWL document is the Protégé. There are many versions of Protégé such as the Stanford University Protégé. The Stanford University Protégé supports only one first class relation, the "is-a" relation. For example, an *Audi is-a car*. In this triple, "Audi" is the object, "is-a" is the predicate, and "car" is the subject. With "is-a" as the only relation to link two classes, properties may have to be used to

capture a more vivid knowledge of the domain of interest. Figure 2 depicts an ontology created with the “is-a” relation of a class “Car” and its subclasses.

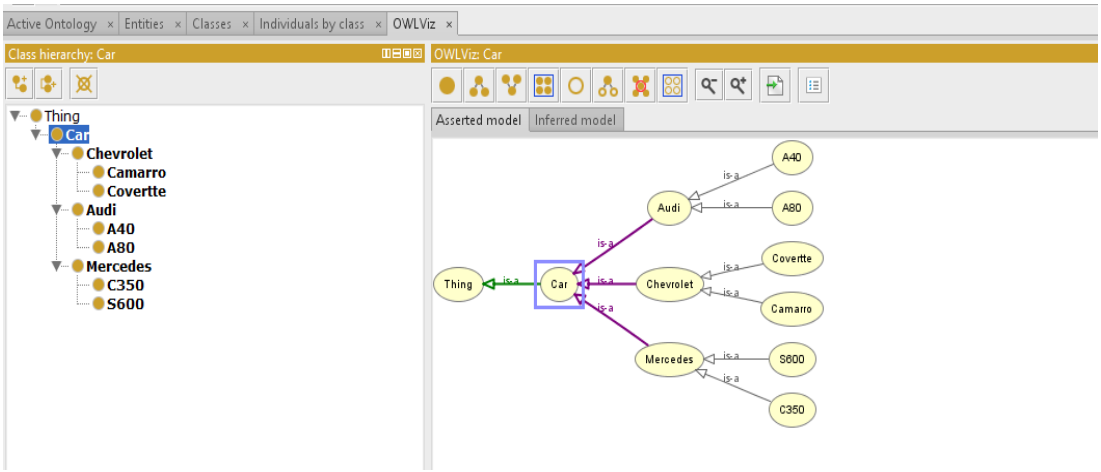


Figure 2: Car Relations Using “is-a” Predicate

It is clear that if there is a way to use more types of predicates to relate two classes, the capture of the knowledge of the domain class will be more vivid. This limitation necessitated the design and development of the Pace University Extended Protégé that has the ability to create custom relations to link two classes of a domain in question.

1.2.3 Knowledge Graph

Pace University Extended Protégé is a tool for extensible knowledge representation supporting custom relations. It is a tool for domain experts to describe and validate knowledge. Also, it is able to drive knowledge based decision-making and introduces minimal syntax extension to OWL so it can benefit from existing tools for OWL. It has the ability to relate two classes using various custom relations. Figure 3 depicts a KG document.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rel "http://www.pace.edu/rel-syntax-ns#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>
<rdf:RDF xmlns="http://www.pace.edu/body-73#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rel="http://www.pace.edu/rel-syntax-ns#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Ontology rdf:about="http://www.pace.edu/body"/>
    <!--
    // Relations
     -->
        <rel:NewRelation rdf:about="http://www.pace.edu/body#partOf"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/body#include"/>
    <!--
    //
    // Classes
    //
     -->
    <!-- http://www.pace.edu/body#finger -->
    <owl:Class rdf:about="http://www.pace.edu/body#finger">
        <rel:partOf rdf:resource="http://www.pace.edu/body#hand"/>
    </owl:Class>
    <!-- http://www.pace.edu/body#hand -->
    <owl:Class rdf:about="http://www.pace.edu/body#hand">
        <rel:include rdf:resource="http://www.pace.edu/body#finger"/>
    </owl:Class>
</rdf:RDF>
<!-- Generated by the OWL API (version 3.5.1) http://owlapi.sourceforge.net -->
```

Figure 3: RDF/XML OWL Document Example

Figure 4 below depicts a “State” knowledge graph with custom relations or predicates viewed via OWLViz, a plugin for the Protégé tool.

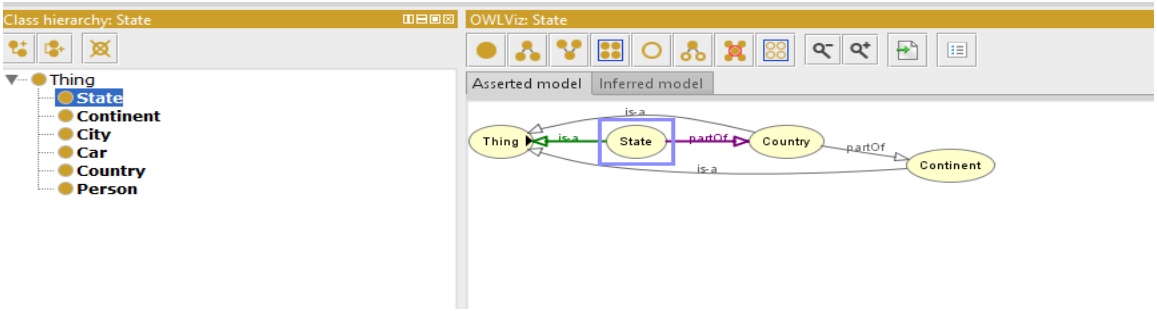


Figure 4: State Relation Using “partOf” Custom Relation

It will be more evident that if we add more custom relations to the State class, a more vivid knowledge description of the domain is presented as shown in Figure 5.

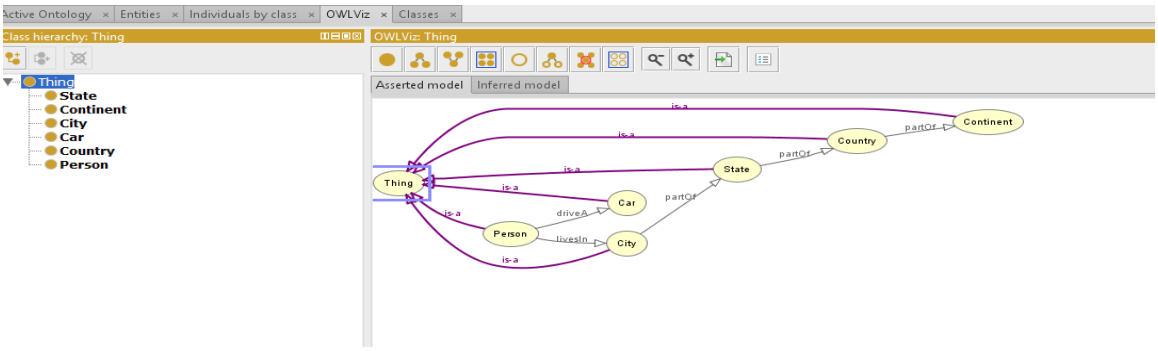


Figure 5: State Relations with More Custom Relation and Related Classes

Pace University knowledge graphs aid domain experts to design Knowledge Representation for Intelligent Systems more effectively than OWL in that custom relations can be created and used to relate classes of concepts in more vivid and expressive ways. When the KG is large, it is difficult to check for the correctness and to navigate each class relation in an efficient manner prompting the need of a visual navigation system that is easy and flexible. Before the KG can be navigated it has to be syntax validated to prevent unexpected eventualities and application system failure due to the KG not been well-formed. It is therefore of paramount importance that a syntax validation system be created to validate any KG created with customs relations. The syntax validation is a method designed in Pace Jena which is used to validate the KG before employing other methods designed in Pace Jena to parse the classes and relations of the KG and present a visual navigation capability. Though Pace Jena can parse and print out the class relations through its API, it was not intuitive as a tool in terms of flexibility, repeatability, and portability. It was therefore necessary to design a visual

navigation application facilitated by methods created in Pace Jena to display any KG currently. Pace Jena was developed by Pace University and it has the capability to parse and print out entities such as classes, custom relations, object properties, data properties, individual classes and other entities from KGs. It can also parse the KG into subject, predicate, and object in the Triple format. The subject and objects are classes that are related by customs relations or predicates. Before the Pace Jena parses the KG, it is prudent that it must first validate the KG and confirm that it is well-formed and OWL syntactically correct. The "DomParse.java" written by Dr Lixin Tao of Pace University [21] as a XML Validator is slightly modified to accommodate the ".owl" extension (which is the extension of the KG) in addition to the existing ".xml", "xsd", and "dtd" extensions. The Dom Parser will be used to validate the KG in conjunction with a designed Pace RDF/XML centric schema. Before the validation of the KG is realized, some challenges have to be overcome before the validation is possible. Since the KG is in RDF/XML format and includes custom relations, using the W3C XML OWL Schema.xsd to validate the KG with ".owl" extension is a challenge in that the W3C XML OWL Schema.xsd supports only owl documents serialized in OWL/XML format and will not work with RDF/XML serialized ".owl" extension KG. Using other RDF/XML Schemas from third parties on the web will not work either since the few encountered via research will not recognize the namespace of Pace University custom relations. The generated RDF/XML serialized KG has four distinct namespaces. They are "rdf", "rdfs", "owl", and "rel" namespaces respectively. Since an ".xsd" schema can only have one target Namespace, a unique method needed to be devised for all four namespaces to work together as a single schema. It was discovered that a separate

schema needs to be created for each namespace. The main schema will be the "rdf" namespace called main.xsd and will import the rel.xsd, rdfs.xsd, and owl.xsd schemas. The rel.xsd schema will import the main.xsd, the rdfs.xsd, and the owl.xsd schemas, the owl.xsd schema will import the main.xsd, rdfs.xsd, and rel.xsd schemas, and subsequently, the rdfs.xsd will import the rel.xsd, main.xsd, and owl.xsd schemas. By this configuration, all four namespaces persist in each individual schema even though only one target Namespace is permitted in a schema. Once the four namespaces are created and working together as one unit via the main schema, the next challenge is how to make the schema generic enough that it can validate any KG with custom relations. It is worthy to note that for future KGs created with custom relations by outside parties to be syntax validated successfully, they must use the "rel" namespace created by Pace University for their custom relation elements names. The validation of any KG will require that its relation elements be declared in the Pace KG schema. To solve this problem, a custom Pace Knowledge Graph Syntax Validator (KGSV) was designed to validate any inputted KG with custom relations before it is parsed by Pace Jena to facilitate visual navigation capabilities to the Pace KG. The designed Pace KGSV will first determine if the relation element in the KG have been declared in the Pace KG schema. If they are declared, the algorithm moves to validate the KG. If there are new elements that are yet to be validated, it employs the Multiple Pass algorithm to update the Pace KG schema by declaring the new relations elements in Pace KG schema before validating the KG. During the research to find a solution to syntax validate KGs, it was deemed that the designed Pace University's Syntax Validator should be able to syntax validate a subset of RDF/XML serialized documents with or without custom relations since there is no suitable

RDF/XML centric syntax validator. This effort led to the addition of two more namespaces, the “pace.xsd”, and the “xml.xsd” schemas to facilitate the syntax validation of RDF/XML serialized documents inclusive with object and data properties. Three functions were created in Pace Jena to facilitate visual navigation capabilities for the KG. The first function will load and return an array list with all unique classes in the KG. The second function will load and return all unique relations in the KG, and the third function takes a class name and a relation name as arguments and returns all instances of the triple namely a class instance as the subject, the custom relation as the predicate, and the related class instance as the object. The Pace Jena has been extended to provide visual navigation capabilities for the knowledge graph that can display all Classes and Relations of the KG and provide navigation in Web and Application models based on any inputted custom relation laden KG.

1.3 Problem Statement

Pace University knowledge graphs enable domain experts to codify their domain knowledge more effectively for driving knowledge-based decision-making. Since the authors can freely introduce new custom relations with various mathematical properties and use them in the same document, knowledge graph breaks the limitation of XML syntax so the standard XML syntax validator cannot validate knowledge graphs. Since a real-life knowledge graph can easily contain hundreds or thousands of classes with complex inter-relations, it is a major challenge for domain experts to review and validate their knowledge representation, and hard for application developers to fully understand the complex relations among the classes.

This research contributes firstly, a syntax validation algorithm for knowledge graphs with custom relation declarations and usage in the same document, and secondly, a knowledge graph visualization solutions based on an extension to the Pace Jena application and a web based approach.

1.4 Dissertation Roadmap

Chapter 2 will review different types of Knowledge Representation. It will discuss Knowledge Graph which is the result of extending OWL with custom relations. In this chapter the Semantic Web, RDF, RDFS, OWL, Inference Engines, Apache Jena, and Pace Jena will be discussed.

Chapter 3 will discuss Knowledge Graph Syntax Validation, Knowledge Graph Syntax Extension, challenges in Syntax Validation, The Multiple Pass Syntax Validation Algorithm with DOM Parsing, Knowledge Graph Syntax Validation Implementation, and small knowledge graph validation examples.

Chapter 4 will discuss Visual Knowledge Graph Navigation that will encompass the Generic API for Accessing Custom Relations, Web Based Knowledge Graph Navigation – Design and Implementation, and the Application-Based Graph Navigation – Design and Implementation.

Chapter 5 will focus on Experimental Validation using a sample Knowledge Graph for Cyber Security Communications. This chapter will demonstrate the Syntax Validation for the Cyber Security Communications sample as well as provide a visual navigation

capability to the Cyber Security Communications using the Web Based Knowledge Graph Navigation, and the Application Based Knowledge Graph Navigation models.

Chapter 6 will give the conclusion of the dissertation highlighting the key contributions and potential future works.

1.5 Conclusion

There is great anticipation for the Semantic Web to revolutionize the Internet but before its full potential is realized, more research is needed to explore and advance existing methods to create, validate, and visualize knowledge representation. For KG to be very useful to domain experts seeking to build effective knowledge representation and knowledge-based decision-making systems on the Web, challenges such as syntax validation of custom relation laden KGs needs to be resolved and supported. The Pace KG Syntax Validation provides a solution to this challenge. KGs, especially very large ones, need a medium whereby they can be navigated to ascertain the correctness of the KG and view the class relations embedded in it. Providing navigational ability to KGs necessitated the design and implementation of the Pace Knowledge Graph Visual Navigation applications. The extension of OWL to KG, the implementation of syntax validation of KGs, and the provision of visual navigation capabilities to KGs will contribute in facilitating the progression of the implementation of machine and human synthesis on the World Wide Web.

Chapter 2

Literature Review

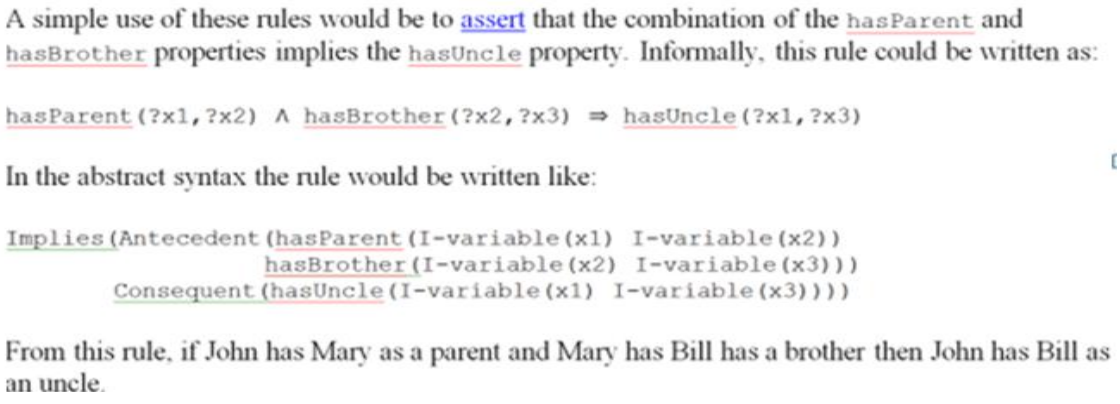
2.1 Knowledge Representation

2.1.1 Rule Based

Rule-based judgmental knowledge is allowed by rule-based inference systems about a specific problem domain to be represented as a collection of discrete rules. Each rule states that if certain assumptions are acknowledged, then certain conclusions can be inferred [27]. W3C chartered the Rule Interchange Format (RIF) Working Group and tasked them to produce extensions in addition to a core rule language which in combination, allow rules to be translated between rule languages and subsequently between rule systems [34]. The RIF Working Group is challenged to amalgamate the needs of a diverse community including business rules and semantic web [34]. The Semantic Web Rule Language (SWRL) includes a high-level abstract syntax for Horn-like rules in both the OWL DL and OWL Lite sublanguages of OWL [35]. Rules are modes for representing knowledge and most often goes beyond OWL1 and are typically conditional statements in the Semantic Web for example the *if then clauses* [36]. Rules are straight forward and are used for ontological mediation whereby resources are mapped between different ontologies. Another reason for rules is that resources are limited in that literals cannot be transformed such as concatenating a new string to a newly discovered string and also complex translations are difficult to express. Rules can

be used to limit OWL's open world assumption using a technique known as "Negation As Failure" (NAF). SWRL is based on OWL1 DL and OWL Lite species and uses a combination of Web Ontology Language (OWL) and Rule MarkUP Language (RuleML) modeled on Horn clauses. The Horn clause is a representation of the "if-then" conditional clauses also referred to as implications. An implication is the synthesis of an antecedent and a consequent. Antecedents and consequents consist of zero and more atoms. An atom is made up of any unary predicate (class inclusions such as "John belongs to class Person"), binary predicate, equality, inequality or built-ins [36]. The main goal of SWRL is to provide expressivity not permitted by OWL [36].

Below in Figure 6 is an example of SWRL



A simple use of these rules would be to assert that the combination of the hasParent and hasBrother properties implies the hasUncle property. Informally, this rule could be written as:

```
hasParent(?x1,?x2) ∧ hasBrother(?x2,?x3) ⇒ hasUncle(?x1,?x3)
```

In the abstract syntax the rule would be written like:

```
Implies(Antecedent(hasParent(I-variable(x1) I-variable(x2))
                   hasBrother(I-variable(x2) I-variable(x3)))
        Consequent(hasUncle(I-variable(x1) I-variable(x3))))
```

From this rule, if John has Mary as a parent and Mary has Bill has a brother then John has Bill as an uncle.

Figure 6: Example of SWRL

A Jena rule, another example of Rule based Knowledge Representation (KR) contains a list of body terms or premises (the if clause) and a list of head terms or conclusions (the then clause) [36]. Optionally, and for convenience, each rule may have a name

and a direction for hybrid rules. A Jena rule is bounded to a rule reasoner which in turn is bounded to a schema and fires based on its configuration [36].

Figure 7 below is an example of Jena Rule [28].

```
String NS = "urn:x-hp-jena:eg/";

// Build a trivial example data set
Model rdfsExample = ModelFactory.createDefaultModel();
Property p = rdfsExample.createProperty(NS, "p");
Property q = rdfsExample.createProperty(NS, "q");
rdfsExample.add(p, RDFS.subPropertyOf, q);
rdfsExample.createResource(NS+"a").addProperty(p, "foo")
```

Figure 7: Example of Jena Rule

Although there is no standard rule format for the Semantic Web currently, some of the rule languages and engines that have been used include F-Logic, Prolog, and Jess. Even though the W3C initiated the RIF Working Group in 2005 to produce a single rule for the Semantic Web, it is unlikely that the establishment of a single rule for the Semantic Web will materialize any time soon due to the fact that research and development in areas of rules and rule-based system is ongoing and is yet to be completed [36].

2.1.2 Logic Based

Logic based KR formalism include Descriptive Logic, Modal Logic, and Non-monotonic Logic. As stated by Franz Baader, "a knowledge representation formalism should allow for the symbolic representation of all the knowledge relevant in a given application domain" [29]. Knowledge representation formalism such as Semantics Network and Frames was motivated by attempts to provide a structured representation of knowledge [29]. Marvin Minsky who developed Frames and defined frames as "a data-structure for representing a stereotyped situation" combined his introduction of the frame idea with a

general rejection of logic as a KR formalization [30]. According to Nilsson, "many database systems and expert systems can be said to use declarative knowledge, and the 'frames' and 'semantic networks' used by several AI programs can be regarded as sets of declarative sentences" [31].

Semantic Network was developed by Quillian for representing the semantics of natural language that represents concepts and objects as nodes in a graph that has two different edges. The first is a property edge for example assigning properties such as *color* to a concept and the second an **"IS_A"** edge that introduces hierarchical relations among concepts. Figure 8 depicts such edges [29].

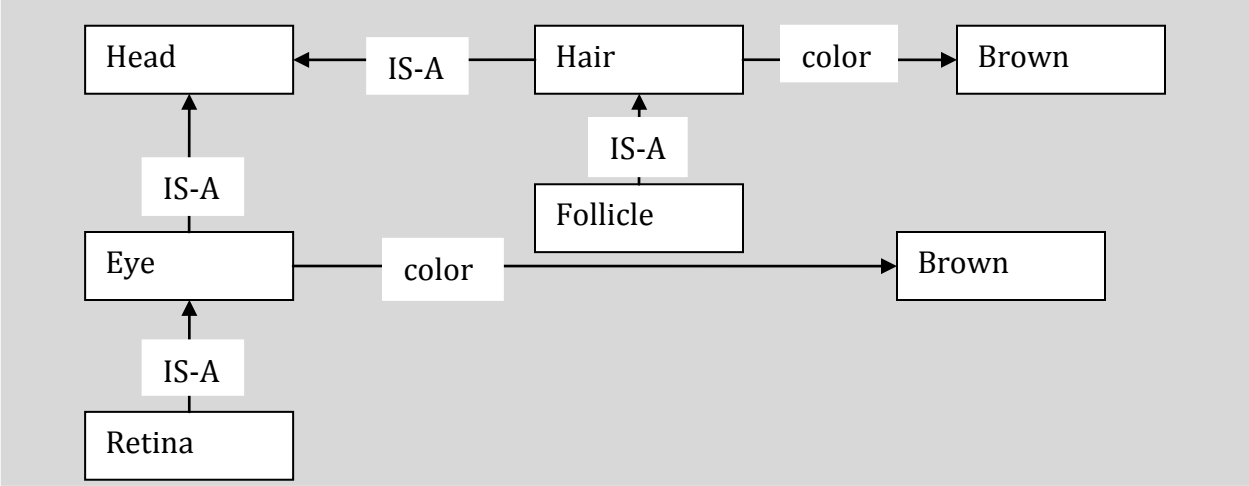


Figure 8 : Semantic Network Edges

The primary idea of Descriptive Logic (DL) is to commence with atomic concepts (unary predicates) and roles (binary predicates) to build complex concepts and roles using a small set of adequate constructors [30]. DL has been implemented in an array of application domains such as natural language processing [38], configuration of technical

systems [39], software information systems [40], optimizing queries to databases [41], and for support in planning [42].

“Modal logic is, strictly speaking, the study of the deductive behavior of the expressions ‘it is necessary that’ and ‘it is possible that’” [43]. Modal Logic KR extends propositional logic by unary operators which are called a box and diamond operators [30]. The box □ operator implies “Necessarily” and the diamond ◇ property implies “Possibly”. “It will snow today” is an example of a “Possibly” modality in that it implies the possibility to snow today.

Non-monotonic Logic is KR language based on classical logics such as the First order predicate in that if a statement Φ can be derived from a knowledge base then Φ can be derived from any larger knowledge base [30]. An advantage of this property is that once an inference is made it should not be revised when more information is received. A disadvantage to this property is that an inconsistency may result when additional information received contradicts the inference rendering the knowledge base useless. To avoid this situation, when plausible conclusions are drawn from available knowledge and the newly acquired knowledge show that some of the plausible conclusions are wrong, the plausible conclusions are withdrawn and do not result in inconsistencies [30]. In the Closed World Assumption (CWA) which by default assumes that the available information is complete, if an assertion cannot be derived using classical inference from the knowledge base, then CWA deduces to negation. Practical application of this assumption is employed in relational databases and in Logic Programming languages with “Negation of Failure” [30].

2.1.3 Ontology Based

Ontology is a vehicle to capture knowledge about a particular domain. It describes the relations between the entities within that domain. Web Ontology Language OWL is one of the most used languages for creating ontologies which is the latest recommendation of W3C. The OWL consists of individuals, properties, and Classes.

Individuals: are instance of a class. For example, a Mathew is an individual or instance of a Student class or Italy is an individual or instance of a Country class. Figure 9 shows the Representation of Individuals.

Figure 9: Representation of Individuals

Properties: relate two individuals together. For example, *Matthew livesIn England*. Properties have many characteristics such as inverse, reflexive, symmetric, transitive, or asymmetric. Figure 10 shows the Representation of Properties.

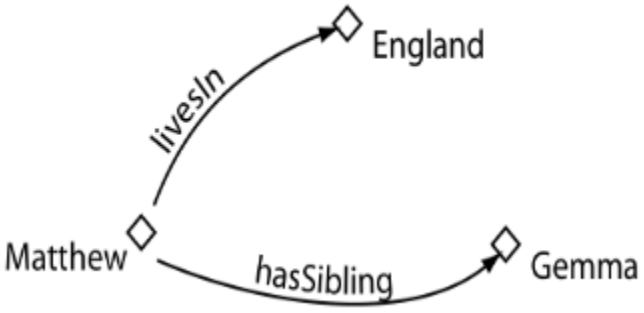


Figure 10: Representation of Properties

Classes: can be described as a set that has individuals as members. Examples of a class are Person, and Pet. Figure 11 shows the Representation of Classes.

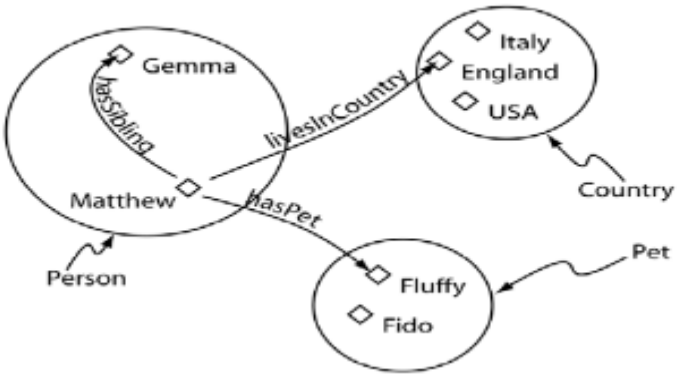


Figure 11: Representation of Classes

OWL uses the Triple to describe the relations between two entities. Figure12 depicts this relation.

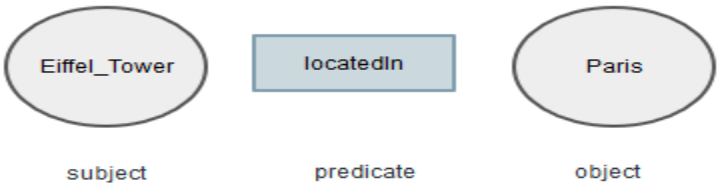


Figure 12: Triple Example

The most popular tool for creating OWL document is the Protégé. There are many versions of Protégé such as the Stanford University Protégé.

Stanford University Protégé: This protégé uses only one relation the "is-a" relation. For example, "Dog is-a Pet". In this triple, "Dog" is the subject, "is-a" is the predicate, and "Pet" is the object. With "is-a" as the only relation to use between two classes, properties may have to be used to capture a more vivid knowledge of the domain of interest. Figure 13 depicts an ontology created with the "is-a" relation of a class "Pet" and its subclasses

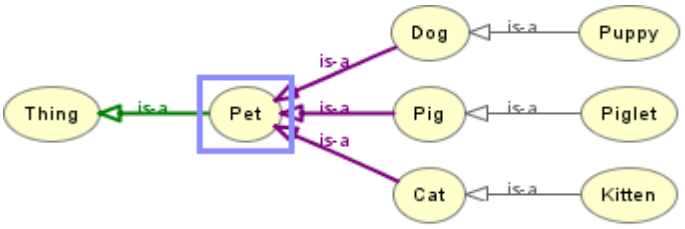


Figure 13: Pet Relations Using "is-a" Predicate

2.1.4 Knowledge Graph

Currently, there are a few systems or applications with different architectures and objectives that are referred to as knowledge graphs. One such example is the Google

Knowledge Graph which is employed by Google to improve search accuracies and relevancies and at times present Knowledge Graph boxes within the returned search results to provide direct answers [57]. The Google Knowledge Graph Search API enables users to query the Knowledge Graph database for specific information about entities residing in the Knowledge Graph database [58]. Unlike the Google's Freebase Knowledge Graph Search API discontinued in December 2014 and now taken over by Wikidata, which is part of Wikipedia, that allowed users to insert data in the knowledge graph database, the new Google Knowledge Graph Search API allows users to only query entities from the Knowledge Graph or Knowledge Vault databases and return results in JSON-LD format [59].

The Pace University Knowledge Graph (KG) on the other hand is engendered by extending OWL with custom relations. It is the extension of OWL with custom relations that transforms OWL into a knowledge graph. The Pace University KG introduces minimal syntax extension to OWL so it can benefit from existing tools for OWL. It has the ability to relate two classes using various custom relations. Figure 14 depicts a Hand class with custom created relations.

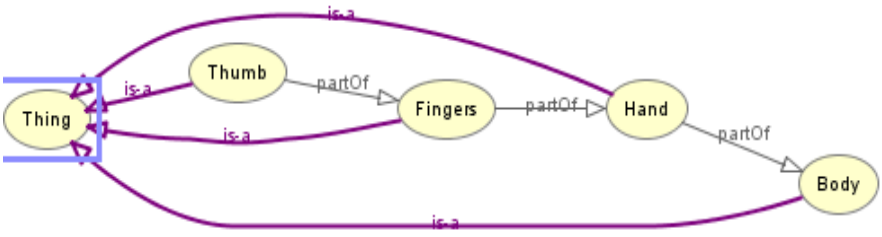


Figure 14: Hand Relations Using "partOf" Relation

It is evident that in addition to the “is-a” relation, the “partOf” custom relation was used to relate two classes in a meaningful way.

2.2 **Semantic Web**

2.2.1 RDF

The Resource Description Framework (RDF) is a language recommended by W3C [24]. RDF is the basic block for supporting the Semantic Web and is all about vocabulary or metadata. It is structured, machine readable, and capable of describing any resource independent of any domain [13]. The basic blocks of RDF are Resource, Property, and Statement**.**

Resource can represent anything or metadata. A Resource is anything being described by RDF expressions. A resource can be any object such as a car, a person’s name, or an address. A resource is identified by a uniform resource identifier (URI) and this URI is used as a name of the resource. An example of a resource is

http://www.yuchen.net/photography/SLR/#Nikon-D70 [13]

Property is a resource that has a name and can be used to describe a specific aspect, characteristics, attributes, or relations of a resource. An example of a resource is

http://www.yuchen.net/photography/SLR/#weight [13]

Statement is used to describe properties of resources. It has the following format known as the Triple and Table 1 shows some examples of the Triple below [13].

Table 1: Examples of the Triple

Subject	**Predicate**	**Object**
mySLR:Nikon-D70	mySLR:weight	1.4 lb
mySLR:Nikon-D70	mySLR:pixel	6.1 M
mySLR:Nikon-D50	mySLR:weight	1.3 lb

It is recommended that **rdf:about** be used in RDF document because it provides an absolute URI for the resource and that URI is taken verbatim as the subject [13].

Figure 15, an example of rdf:about is displayed below.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#
        Xmlns=http://www.yuchen.net/photography/Camera#>
<SLR rdf:about="http://www.yuchen.net/rdf/NikonD70.rdf#Nikon-D70">
        <weight>1.4 lbs</weight>
</SLR>
</rdf:RDF>
```

Figure 15: rdf:about Example

2.2.2 RDFS

RDFS stands for RDF Schema. RDF by itself is just a data model and does not have any semantics [25]. RDFS is used to create vocabulary to describe classes, subclasses and properties of RDF resources and is a recommendation from W3C [26]. RDF associates the properties it defines and can add semantics to RDF predicates and properties.

2.2.3 OWL

Recent work in AI is investigating the use of formal ontologies as a means of defining content-specific concordances in the reuse and sharing of knowledge among software entities [1]. Web Ontology Language (OWL) is a segment of the Semantic Web which expands the current Web by extending current semantics to it [12]. OWL is based on the RDF schema. OWL has three sublanguages OWL Lite, OWL DL, and OWL Full [71].

OWL Lite

- is a sublanguage of OWL DL
- supports only a subset of the OWL language
- has restrictions on class definitions such as the separation of classes, instances, properties and data values
- has a restriction on mixing rdf and owl constructs
- has its owl:DataTypeProperty and owl:ObjectTypeProperty defined as disjoint subclasses of rdf:property
- has OWL:class as a subset of RDF:class
- has “IntersectionOf” as the only class definition
- has Cardinality:0/1, MaxCardinality:0/1, and MinCardinality:0/1
- does not allow meta-modeling
- is not compatible to RDF

OWL DL

- supports same set of OWL language constructs

- has restrictions on class definitions such as the separation of classes, instances, properties and data values
- has a restriction on mixing rdf and owl constructs
- has its owl:DataTypeProperty and owl:ObjectTypeProperty defined as disjoint subclasses of rdf:property
- has OWL:class as a subset of RDF:class
- has class definition that includes “UnionOf”, “ComplementOf”, “IntersectionOf”, and enumeration
- has Cardinality>=0, MaxCardinality>=0, and MinCardinality>=0
- does not allow meta-modeling
- is not compatible to RDF

OWL Full

- supports same set of OWL language constructs
- does not have restrictions on class definitions in that classes can be instances or properties at the same time
- allows the mixing of rdf and owl constructs
- has its owl:ObjectProperty considered equivalent to rdf:property in that its owl:DataTypeProperty which is a subset of rdf:property is also a subset of owl:ObjectProperty.
- has its OWL:class and RDF:Class as equivalent
- has class definition that includes “UnionOf”, “ComplementOf”, “IntersectionOf”, and enumeration

- has Cardinality>=0, MaxCardinality>=0, and MinCardinality>=0
- allows meta-modeling so that RDF and OWL constructs can be re-defined and extended
- is compatible to RDF which indicates that a valid rdf document is also OWL Full
- document is equivalent to a valid rdf document in that owl:class and its sub classes are equivalent to rdf:class and its sub classes

2.2.4 Inference Engines

Inference Engines are designed to draw conclusions by analyzing statements from a repository of domain specific knowledge. Inference Engines uses a rule base to arrive at a conclusion. Inference engines can simulate human thought by rule chaining and using the knowledge embedded in ontology. Rule chaining inference can be categorized into two types: forward chaining and backward chaining whereby forward chaining is when the system starts with facts and progresses forward until it reaches a goal while backward chaining entails that the system starts with an objective and works backwards to validate the objective. Examples of inference engines are the HermiT [32], the Racer [51], and F-OWL [52].

HermiT is a Protégé plugin and an open-source OWL reasoner "that can determine whether or not the ontology is consistent, identify subsumption relations between classes" [32].

Racer is a Descriptive Logic Inference Engine and a highly optimized inference system which is freely accessible for research purposes. RICE is a tool for Racer that visualizes taxonomies [51].

F-OWL is an inference engine for the semantic web language OWL and is based on F-logic [52].

2.2.5 Apache Jena

Apache Jena developed by HP Labs is an open-source Java framework for building Semantic Web and Linked-Data applications that comprises different APIs such as SPARQL API, RDF API, Ontology API, and Inference API that interact together to process RDF data [56]. Apache Jena provides comprehensive Java libraries to support Java programmers develop RDF, RDFS, OWL and SPARQL related applications in accordance to published W3C recommendations [55]. SPARQL is the query language developed by the W3C RDF Data Access Working Group.

SPARQL API

Apache Jena Fuseki [63], an example of SPARQL API is a SPARQL server that provides multi-purpose services such as:

- can be run as an operating system service
- can be run as a Java web application (WAR document)
- can be run as a standalone server
- it can provide security using Apache Shiro
- it can provide server monitoring and administration services via a user interface

ARQ [64], another example of SPARQL API is a Jena query engine that:

- supports the SPARQL RDF Query language

RDF API

TDB [65], an example of RDF API is a component of Jena for RDF storage and query:

- that supports the full range of Jena APIs
- It can also be used as a high performance standalone RDF store on a single machine

Ontology API

Jena uses Ontology API [66] to provide

- a consistent programming interface for ontology application development that is independent of which ontology language being used in the programs

Inference API

The Jena Inference API [67]:

- is designed to accommodate a range of inference engines or reasoners that are incorporated into Jena
- uses Inference engines to extract additional RDF assertions which encompass some base RDF together with any elective ontological information as well as the axioms and rules related with the reasoned
- main use is to support the use of languages such as RDFS and OWL and permit additional facts to be deduced from instance data and class descriptions

2.2.6 Pace Jena

Pace University Jena is a simplified research application for Jena services developed by Dr. Lixin Tao [21] that allows the parsing and processing of OWL documents. Pace Jena has the capability to parse and print out entities such as classes, custom relations, object properties, data properties, individual classes and others from KG documents. This research has created four functions in Pace Jena. The first function will load and return an array list with all unique classes in the OWL document. The second function will load and return all unique relations in the OWL document, and the third function which takes a class name and a relation name as arguments and return all instances of the Triple namely a class instance as the subject, the custom relation as the predicate, and the related class instance as the object. The forth function will validate the KG for well-formedness and OWL syntax. Consult "Pace Jena Methods" section of this dissertation to view the functions that this research contributed to Pace Jena. These functions were developed to aid in the creation of the knowledge graph by providing syntax validation and visual navigation of the class relations embedded in the OWL documents.

2.3 Conclusion

Knowledge Representation (KR) whose origins can be traced to AI in the 1950's had come a long way. There are Rule based and Logic based KRs. Rules are modes for representing knowledge and most often goes beyond OWL1 and are typically conditional statements in the Semantic Web for example the *if then* clauses. A Jena rule is an

example of rule based KR and contains a list of body terms or premises (the *if* clause) and a list of head terms or conclusions (the *then* clause). SWRL is based on OWL1 DL and OWL Lite species and uses a combination of OWL and RuleML modeled on Horn clauses. The main goal of SWRL is to provide expressivity not permitted by OWL. Modal Logic is an example of logic based KR and extends propositional logic by unary operators which are called a box and diamond operators. It is the study of the deductive behavior of the expressions 'it is necessary that' and 'it is possible that'. Other forms of logic based KR are Descriptive Logic and Non-monotonic Logic. OWL is a language that could be created via tools such as Protégé. An example of such tool is the Stanford Protégé developed by the Stanford Center for Biomedical Informatics Research (BMIR) at the Stanford University School of Medicine [53]. Protégé 4 was developed by Matthew Horridge of Manchester University [54] who currently, is a research staff at BMIR at the Stanford University School of Medicine. OWL consists of classes that are related by relations. Currently, the "is-a" is the only first-class relation used in Stanford Protégé to relate classes. However, classes can be related to using object properties and data properties. KG extends OWL and uses Pace Jena to facilitate the syntax validation and visual navigation of KGs. Knowledge Representation via knowledge graphs can enable domain experts to develop effective systems to aid in their quest to facilitate machine and human cooperation on the Internet and extend the provisioning of essential functionalities to users on the World Wide Web.

Chapter 3

Knowledge Graph Syntax Validation

3.1 OWL Serialization Options

OWL/XML is a format proposed by University of Manchester UK and it represents OWL information in small XML structures similar to triples. OWL/XML does have standard XSD syntax definition. But OWL/XML is not suitable for people to read or write because it scatters information into many small structures. OWL/XML serialization has some similarities to RDF/XML in that they both

- support the single first-class relation “is-a”
- use object and data properties to relate individual classes
- are controlled under the Rule Interchange Format (RIF) a developed technology that facilitates the control of the different values of need across the Web
- can be expressed in Triples of subject, predicate, and object

OWL/XML and RDF/XML has differences as well that include

- OWL/XML has standard XSD syntax definition while RDF/XML does not have a standard syntax definition because it also needs to support other formats such as RDF, RDF Schema and N3
- OWL/XML does not have XML extensions to support custom relations while RDF/XML supports custom relations

- RDF/XML can represent multiple languages in the same document while OWL/XML does not
- RDF allows exportation of contents in different formats including RDF + XML and N3 which is a non-XML format

A KG with custom relations serialized in the RDF/XML format can be converted to OWL/XML format via a Protégé tool but the custom relations are not recognized and therefore discarded. Though an OWL document can be serialized in both OWL/XML and RDF/XML formats, this dissertation is focused on the RDF/XML serialization because currently, there is no OWL/XML extension for supporting custom relations. Also RDF/XML serialization is more concise and mostly used by researchers. RDF/XML was designed for RDF, but it can also be used to represent OWL ontologies with dedicated namespaces. It does not have a standard syntax definition because it also needs to support RDF, RDF Schema, N3, Turtle, and many others. RDF is the foundation and each other semantic web languages add syntax elements to it. RDF/XML can represent multiple languages in the same document.

Pace University opted to extend RDF/XML to KG instead of OWL/XML because

- OWL/XML serialized documents are not suitable for people to read or write since it scatters information into many small structures
- OWL/XML format has no extension for supporting custom relations but rather emulates custom relations with complex object and data properties
- Most Knowledge Representation (KR) applications needs custom relations
- RDF/XML document is more concise and used by most researchers

3.2 Knowledge Graph Syntax Extension to RDF/XML

The main objective for syntax extension to support custom relations is to enable domain experts (not IT experts) to be able to:

- declare custom relations and use them in the same document
- use custom relations directly and intuitively without using object property emulation
- declare and apply custom relations in IDEs like Protégé

When an OWL document serialized in the RDF/XML format is extended with custom relations, it is transformed into a KG. This extension can be categorized into three parts namely: the Definition of Namespace, the Definition of Custom Relations, and the Definition of Classes.

Definition of Namespace: An XML namespaces (xmlns) provides "a simple method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URI references" [68]. A namespace is declared using an attribute name such as xmlns or have "xmlns:" as a prefix. There are five namespaces defined for a KG which are "rdf", "owl", "rel", "xsd", and "rdfs" respectively as shown in the Definition of Namespace section of a KG depicted below.

```
<? xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
<!ELEMENT rdf:RDF (owl:Ontology,rel:NewRelation+,owl:Class+)>
<!ATTLIST rdf:RDF
xmlns:rdf CDATA #FIXED "http://www.w3.org/1999/02/22-rdfsyntax-ns#"
xmlns:owl CDATA #FIXED "http://www.w3.org/2002/07/owl#"
xmlns:rel CDATA #FIXED "http://www.pace.edu/rel-syntax-ns#"
xmlns:xsd CDATA #FIXED "http://www.w3.org/2001/XMLSchema#"
xmlns:rdfs CDATA #FIXED "http://www.w3.org/2000/01/rdf-schema#"]>
```

Figure 16: A KG Definition of Namespace

Each of these namespaces has a qualifying name identified by a URI. For example, the qualifying name for “rdf” namespace is “http://www.w3.org/1999/02/22-rdfsyntax-ns#” and that for “owl” namespace is “http://www.w3.org/2002/07/owl#”.

Definition of Custom Relations: The “NewRelation” element is used to declare custom relations created in the KG. The “rdf:about” element is used to provide an absolute URI for the resource (custom relation). Also a custom relation can be of FunctionalRelation, InverseFunctionalRelation, IrreflexiveRelation, ReflexiveRelation, SymmetricRelation, and TransitiveRelation rdf type. Figure 17 depicts the custom relations definition.

```
<? xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rel:NewRelation [
<!ELEMENT rel:NewRelation (rdf:type* )>
<!ATTLIST rel:NewRelation rdf:about CDATA #REQUIRED>
<!ELEMENT rdf:type EMPTY>
<!ATTLIST rdf:type rdf:resource
(&owl;AsymmetricRelation|&owl;FunctionalRelation|&owl;InverseFunctionalRelation|&owl;
IrreflexiveRelation|&owl;ReflexiveRelation|&owl;SymmetricRelation|&owl;TransitiveRela
tion) #REQUIRED>
]>
```

Figure 17: A KG Custom Relations Definition

Definition of Classes: The “rdf:about” element is used to provide an absolute URI for the resource (class) and the “rdf:resource” is used as a name for the resource (class). Figure 18 depicts the class definition of a KG.

```
<? xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE owl:Class [
<!ELEMENT owl:Class (rel:NewRelationName*)>
<!ATTLIST owl:Class rdf:about CDATA #REQUIRED>
<!ELEMENT rel:NewRelationName EMPTY>
<!ATTLIST rel:NewRelationName rdf:resource #REQUIRED>
]>
```

Figure 18: A KG Classes Definition

Currently, there is no known RDF/XML schema validator and the syntax validation of OWL documents serialized in the RDF/XML format is a considerable challenge. Since

knowledge graph breaks standard XML syntax rules to support easier introduction of custom relations, special algorithms are needed to support syntax validation and visualization. When the OWL document serialized in RDF/XML format is extended to a KG with custom relations, the syntax validation problem is further compounded because these custom relation elements needs to be declared and validated by an XSD schema. Also the custom relations are supposed to be declared in a XSD schema that will be used to validate the KG. The generated RDF/XML serialized OWL document has six distinct namespaces. They are "rdf", "rdfs", "owl", and "rel", "pace", and "xml" namespaces respectively. Since an ".xsd" schema can only have one target Namespace, a unique method is needed to be devised for all six namespaces to work together as a single schema. By experimentation it was discovered that a separate schema needs to be created for each namespace. The main schema will be the "rdf" namespace called "main.xsd" and will import the "rel.xsd", "rdfs.xsd", "pace.xsd", "owl.xsd", and "xml.xsd" schemas. The "rel.xsd" schema will import the "main.xsd", "pace.xsd", "rdfs.xsd", "xml.xsd", and the "owl.xsd" schemas, the "owl.xsd" schema will import the "main.xsd", "xml.xsd", "pace.xsd", "rdfs.xsd", and "rel.xsd" schemas, the "xml.xsd" schema will import the "main.xsd", the "rdfs.xsd", "pace.xsd", "rel.xsd", and the "owl.xsd" schemas, the "pace.xsd" will import the "rel.xsd", "main.xsd", "rdfs.xsd", "xml.xsd", and "owl.xsd" schemas and subsequently, the "rdfs.xsd" will import the "rel.xsd", "main.xsd", "pace.xsd", "xml.xsd", and "owl.xsd" schemas. By this configuration, all six namespaces persist in each individual schema even though only one target Namespace is permitted in a schema. Once the six namespaces are created and working together as one unit via the main schema as determined from an example of "xml.xsd" shown below in Figure 19, the

next challenge is how to make the schema generic enough that it can validate a subset of RDF/XML centric document with custom relations or without custom relations.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
target Namespace="http://www.w3.org/XML/1998/namespace"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:pace="http://csis.pace.edu/semweb#" xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:rel="http://www.pace.edu/rel-syntax-ns#">
  <xs:import namespace="http://csis.pace.edu/semweb#" schemaLocation="pace.xsd"/>
  <xs:import namespace="http://www.pace.edu/rel-syntax-ns#" schemaLocation="rel.xsd"/>
  <xs:import namespace="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
schemaLocation="main.xsd"/>
  <xs:import namespace="http://www.w3.org/2000/01/rdf-schema#"
schemaLocation="rdfs.xsd"/>
  <xs:import namespace="http://www.w3.org/2002/07/owl#" schemaLocation="owl.xsd"/>
  <xs:attribute name="base" type="xs:anyURI"/>
</xs:schema>
```

Figure 19: xml.xsd Schema

The Pace RDF/XML XSD schema is shown in Figure 20.

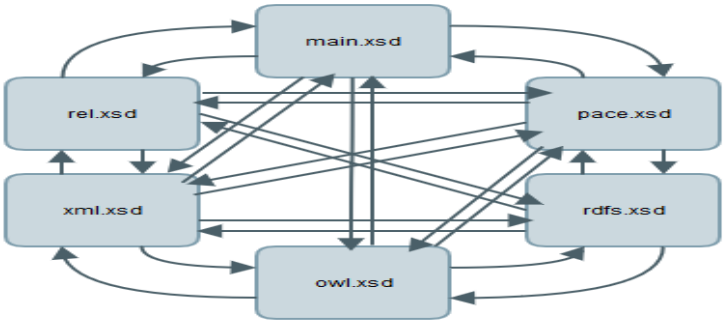


Figure 20: Pace Custom Schema

Below, Figure 21 portrays the main.xsd schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
 targetNamespace="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:pace="http://csis.pace.edu/semweb#"
 xmlns:owl="http://www.w3.org/2002/07/owl#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:rel="http://www.pace.edu/rel-syntax-ns#">
  <xs:import namespace="http://csis.pace.edu/semweb#" schemaLocation="pace.xsd"/>
  <xs:import namespace="http://www.pace.edu/rel-syntax-ns#" schemaLocation="rel.xsd"/>
  <xs:import namespace="http://www.w3.org/2000/01/rdf-schema#" schemaLocation="rdfs.xsd"/>
  <xs:import namespace="http://www.w3.org/2002/07/owl#" schemaLocation="owl.xsd"/>
  <xs:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="xml.xsd"/>
  <xs:element name="RDF">
```

Figure 21: main.xsd Schema Part of Pace Schema

Figure 22 below shows the five imported namespaces namely rel.xsd, main.xsd, rdfs.xsd, owl.xsd, and xml.xsd in to pace.xsd to persist the five namespaces in the pace.xsd schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
targetNamespace="http://csis.pace.edu/semweb#"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:pace="http://csis.pace.edu/semweb#"
 xmlns:owl="http://www.w3.org/2002/07/owl#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:rel="http://www.pace.edu/rel-syntax-ns#">
  <xs:import namespace="http://www.pace.edu/rel-syntax-ns#" schemaLocation="rel.xsd"/>
  <xs:import namespace="http://www.w3.org/1999/02/22-rdf-syntax-ns#" schemaLocation="main.xsd"/>
  <xs:import namespace="http://www.w3.org/2000/01/rdf-schema#" schemaLocation="rdfs.xsd"/>
  <xs:import namespace="http://www.w3.org/2002/07/owl#" schemaLocation="owl.xsd"/>
  <xs:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="xml.xsd"/>
  <xs:element name="level" type="xs:integer"/>
  <xs:element name="name" type="xs:NCName"/>
  <xs:element name="ref">
    <xs:complexType>
      <xs:attribute ref="rdf:resource" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 22: pace.xsd Schema with Imported Namespaces

The rdfs.xsd, owl.xsd, xml.xsd, and rel.xsd all follow the 2 examples shown in Figure 21 and 22 and import five namespaces into each other and persist their namespaces. Consult Appendix A to view all the six XSD schema documents in its entirety.

3.3 RDF/OWL Subset for Supporting Knowledge Graph Research

Since there is no standard syntax definition for RDF/XML, and the domain experts (not IT experts) can easily introduce bugs, there is a need for a clear syntax specification for a subset of RDF/XML to support this research on knowledge graphs. Custom relation declaration in a Knowledge Graph can be broken into two parts namely: Definition of Custom Relation and Application custom relations to Classes. Figure 23 shows an example of the definition of the “partOf” custom relation**.**

```
// Relations
 -->
<rel:NewRelation rdf:about=http://pace.edu/claude#partOf>
        <rdf:type  rdf:resource="TransitiveRelation">
        <rdf:type  rdf:resource="AsymmetricRelation">
```

Figure 23: Custom Relation Definition Example

Presented below in Figure 24 is an example of the syntax specification of a KG using the class relations example, the syntax of the classes "Finger" and "Hand" which are the nodes are linked with the custom relation "partOf" which is the predicate in the RDF triples of subject predicate and object. Another example in Figure 24 is the class "Hand" is related to class "Body" using the custom relation "partOf". As shown below in Figure 24, "*Finger*" "*partOf*" "*Hand*" is an example of asymmetric relation" since a *Finger* is part of the *Hand* but the *Hand* cannot be part of the *Finger*. Also, "*Hand*" "*partOf*" "*Body*" is an asymmetric relation since "*Hand*" is part of "*Body*" but the "*Body*" cannot be part of the "*Hand*". Since "*Finger*" is part of "*Hand*" and "*Hand*" is part of "*Body*", it can be inferred that "*Finger*" is part of "*Body*". Therefore "*Finger*" "*partOf*" "*Body*" is a transitive relation.

```
// Classes     -->
<!-- http://pace.edu/claude#Anatomy -->

 <owl:Class rdf:about="http://pace.edu/claude#Finger">
        <rel:partOf rdf:resource="http://pace.edu/Hand"/>
</owl:Class>

<owl:Class rdf:about="http://pace.edu/claude#Hand">
         <rel:partOf rdf:resource="http://pace.edu/claude#Body"/>
 </owl:Class>
```

Figure 24: Application Custom Relations to Classes Example depicting an Asymmetric and Transitive Relations

Below, Figure 25 portrays examples of asymmetric and symmetric custom relations.

```
// Relations
 -->
<rel:NewRelation rdf:about=http://pace.edu/claude#childOf>
        <rdf:type  rdf:resource="AsymmetricRelation">
        <rdf:type  rdf:resource="FunctionalRelation">

<rel:NewRelation rdf:about=http://pace.edu/family#siblingOf>
        <rdf:type  rdf:resource="SymmetricRelation">
        <rdf:type  rdf:resource="FunctionalRelation">
```

Figure 25 Asymmetric and Symmetric Custom Relations Example

Figure 26 shows that class "Daughter" is related to class "Father" using the "childOf" asymmetric custom relation since *"Daughter*" is child of "*Father*" but "*Father*" cannot be child of "*Daughter*". The symmetric relation of class "Daughter" to class "Brother" via custom relation "siblingOf" is also shown in Figure 26. Class "*Daughter*" is sibling of Class "*Brother*" and it can be inferred that Class "*Brother*" is also sibling of Class "*Daughter*" who is a child to Class "*Father*".

```
// Classes     -->
<!-- http://pace.edu/claude#Family -->

 <owl:Class rdf:about="http://pace.edu/claude#Daughter">
        <rel:childOf rdf:resource="http://pace.edu/Father"/>
</owl:Class>

<owl:Class rdf:about="http://pace.edu/claude#Daughter">
         <rel:siblingOf rdf:resource="http://pace.edu/claude#Brother"/>
 </owl:Class>
```

Figure 26: Asymmetric and Symmetric Class Custom Relations Example

Since in Figure 26 class "Daughter" is related to class "Brother" via a functional custom relation "siblingOf", it can be inferred that class "Brother" is a brother to Class "Daughter" through the functional custom relation "siblingOf" as depicted in Figure 27.

```
// Classes     -->
<!-- http://pace.edu/claude#Family -->

<owl:Class rdf:about="http://pace.edu/claude#Brother">
         <rel:siblingOf rdf:resource="http://pace.edu/claude#Daughter"/>
 </owl:Class>
```

Figure 27: Functional Class Custom Relations Example

Since in Figure 26 shows that class “Daughter” is related to class “Father” using the “childOf” functional custom relation, it can been inferred that class “Father” is the parent of class “Daughter” via the Inverse functional custom relation of “parentOf” as portrayed in Figure 28.

```
// Classes     -->
<!-- http://pace.edu/claude#Family -->

<owl:Class rdf:about="http://pace.edu/claude#Father">
        <rel:parentOf rdf:resource="http://pace.edu/Daughter"/>
</owl:Class>
```

Figure 28: Inverse Functional Class Custom Relations Example

Figure 29 below shows a “Person” class that relates to itself via the reflexive custom relations of “isKnownBy”.

```
// Classes     -->
<!-- http://pace.edu/claude#Human -->

<owl:Class rdf:about="http://pace.edu/claude#Person">
       <rel:isKnownBy rdf:resource="http://pace.edu/Person"/>
</owl:Class>
```

Figure 29: Reflexive Class Custom Relations Example

In Figure 26, class “Daughter” is related to class “Father” via the custom relation “childOf”. However class “Daughter” cannot relate to itself using the same custom relation “childOf” and likewise class “Father” cannot relate to itself using the same

custom relation “childOf”. So it can be inferred that class “Daughter” is “notChildOf” class “Daughter” and class “Father” is “notChildOf” class “Father” as shown in Figure 30

```
// Classes     -->
<!-- http://pace.edu/claude#Family -->

 <owl:Class rdf:about="http://pace.edu/claude#Daughter">
        <rel:notChildOf rdf:resource="http://pace.edu/Daughter"/>
</owl:Class>

<owl:Class rdf:about="http://pace.edu/claude#Father">
         <rel:notChildOf rdf:resource="http://pace.edu/claude#Father"/>
 </owl:Class>
```

Figure 30: Irreflexive Class Custom Relations Example

An example of how an RDF/XML document is validated is presented below. Figure 31 shows five namespaces of an RDF/XML document.

```
<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rel "http://www.pace.edu/rel-syntax-ns#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>

<rdf:RDF xmlns="http://www.pace.edu/claude#"
     xml:base="http://www.pace.edu/claude"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rel="http://www.pace.edu/rel-syntax-ns#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Ontology rdf:about="http://www.pace.edu/claude"/>
```

Figure 31: Namespaces in an RDF/XML Document Example

In this example, a set of XSD documents are needed to be created to meet the validation requirement for this KG document with five namespaces. The XML, RDF, RDFS, OWL, REL documents with “.xsd” extension were therefore created. Each XSD document

would cover all elements and attributes that start with the same prefix. For example in Figure 32,

```
<owl:Class rdf:about="http://www.pace.edu/claude#Body">
```

Figure 32: Class Element Example

The element ***Class*** will be defined in "owl.xsd" document and the attribute ***about*** will be defined in "rdf.xsd" document. In each XSD document, the other XSD documents will be imported into it, so when the RDF/XML owl document is being validated, only one of the XSD document needs to be inputted as the argument as shown in Figure 33.

```
<xs:import namespace="http://www.pace.edu/rel-syntax-ns#" schemaLocation="rel.xsd"/>
<xs:import namespace="http://www.w3.org/1999/02/22-rdf-syntax-ns#" schemaLocation="main.xsd"/>
<xs:import namespace="http://www.w3.org/2002/07/owl#" schemaLocation="owl.xsd"/>
<xs:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="http://www.w3.org/2001/xml.xsd"/>
```

Figure 33: XSD Document as Argument Example

If the internet is inaccessible and cannot be connected to, a new xml.xsd file needs to be created and stored locally to meet validation as shown in Figure 34.

```
<xs:import namespace="http://www.pace.edu/rel-syntax-ns#" schemaLocation="rel.xsd"/>
<xs:import namespace="http://www.w3.org/1999/02/22-rdf-syntax-ns#" schemaLocation="main.xsd"/>
<xs:import namespace="http://www.w3.org/2002/07/owl#" schemaLocation="owl.xsd"/>
<xs:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="xml.xsd"/>
```

Figure 34: Create Local xml.xsd Example

3.4 Knowledge Graph Syntax Validation Algorithm

The next step is how to surmount the challenge of validating a KG document. Through experimentation as explained in the previous chapter, it was determined that individual

namespace XSD schemas needed to be created to persist in the main.xsd schema. Also it was determined that for the custom relation to be syntax validated, it needed to be declared in the “owl.xsd” and “rel.xsd” schema documents with appropriate syntax before it is syntax validated with the same XSD schemas. Using a “cyberSecurityCommunications.owl” KG document which is the main Use Case for this research, the six custom schemas were created which consist of the main.xsd for the “rdf” namespace, rdfs.xsd for “rdfs” namespace, rel.xsd for the Pace University’s “rel” namespace, xml.xsd for the “xml” namespace, pace.xsd for the Pace University “semweb” namespace and the owl.xsd for the “owl” namespace. Figure 35 depicts the flow chart for validating a KG or a supported RDF/XML document with custom created main.xsd, owl.xsd, rel.xsd, pace.xsd, xml.xsd, and rdfs.xsd schemas in conjunction with functions in Pace Jena. These schemas can be viewed in its entirety in Appendix A and formed the pillars to syntax validate any KG with custom relations and a supported RDF/XML serialized document with or without custom relations.

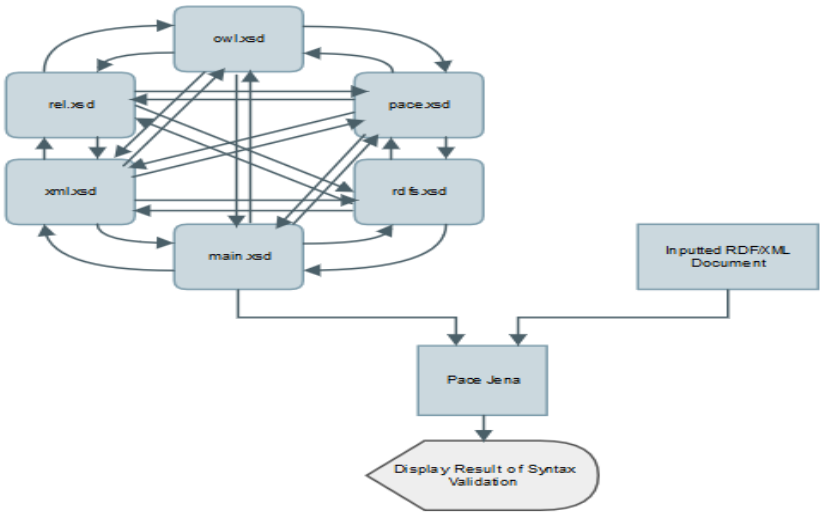


Figure 35: KG Syntax Validation Flowchart

After getting the “cyberSecurityCommunications.owl” KG document syntax validated successfully, the next challenge is to design an algorithm that will use these six namespaces and syntax validate a subset of supported RDF/XML serialized KG. First to be able to update the “owl.xsd” document in real time in-order to syntax validate a KG on the fly, a list of all relations in the KG needs to be retrieved and in a loop each custom relation is verified if it has indeed been declared in “owl.xsd” and “rel.xsd” schema or is yet to be declared. A function was created in Pace Jena as depicted in Figure 36 to retrieve the custom relations into an array list which can then be printed to console or piped into a syntax validation system.

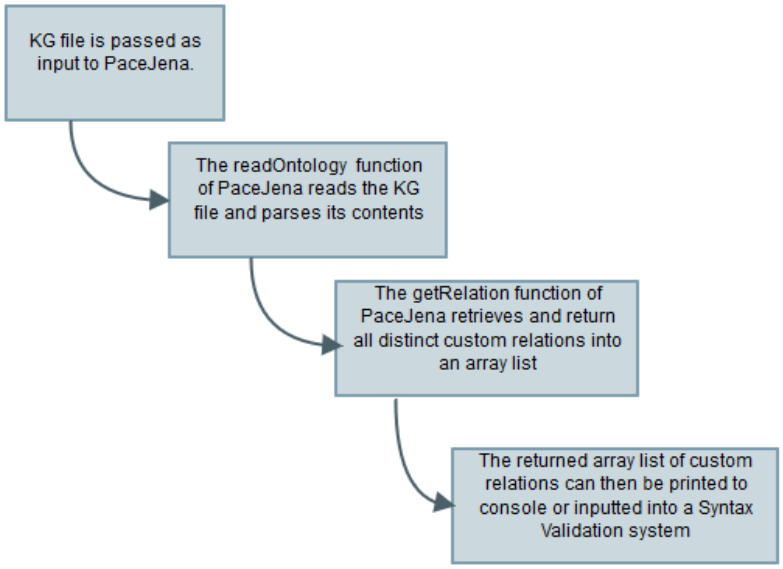


Figure 36: Process Flow Retrieving Custom Relations Using Function In Pace Jena

An algorithm displayed in Figure 37 was designed to syntax validate a supported inputted KG.

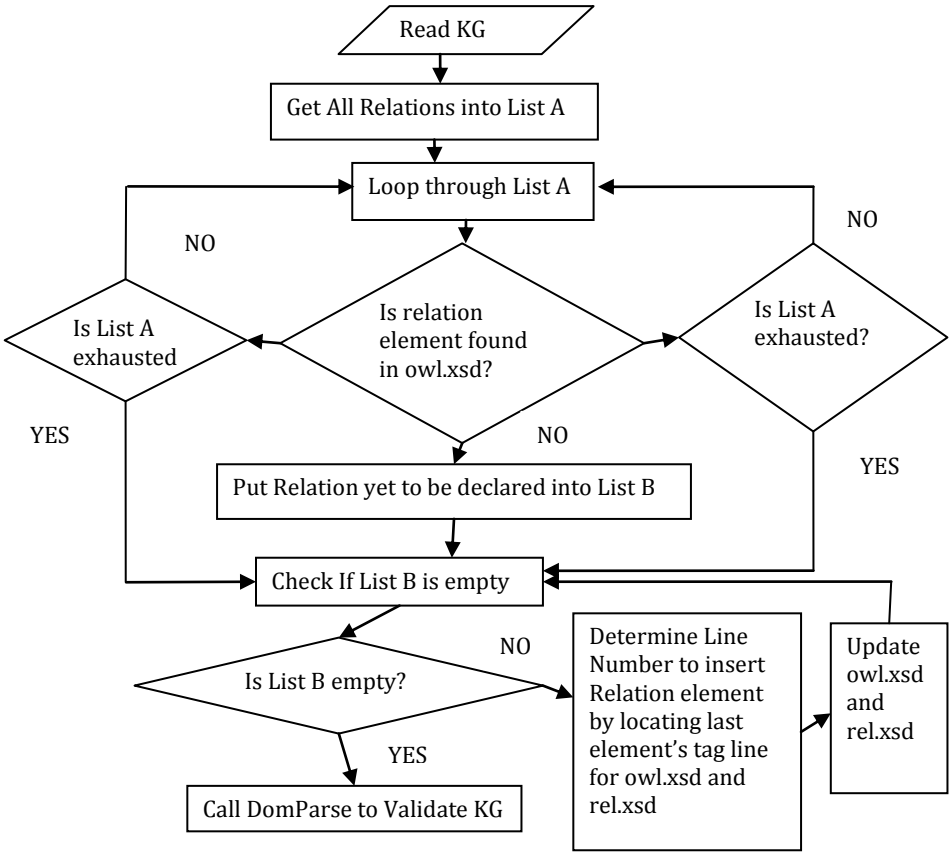


Figure 37: Algorithm to Syntax Validate Any KG on the Fly

This algorithm though successful in validating KGs with new custom relations, it failed to validate a KG that had its custom relations declaration in owl.xsd schema out of sync to the order in which they were declared in the KG document. This situation is manifested when any of the custom relations retrieved for declaration is already declared in the "owl.xsd" schema resulting in the order of the declaration to be out of order and no longer declared in a "First In First Out" (FIFO) manner. This prompted the algorithm of Figure 37 to be further evaluated. Upon careful examination and research, a problem related to the FIFO ordering of custom relations in "owl.xsd" schema was identified and is presented below. First, it is important to establish that it is beneficial to create the

“rel.xsd” and “owl.xsd” with pre-populated custom relations from an initial KG so that it is easier to locate the line numbers to insert new custom relations to be declared and validated using the pre-populated custom relations as a matching token to be identified to insert the new custom relations with appropriate syntax. Consider a Use Case of two custom relations namely “partOf” and “typeOf” in a new KG named KG_B. The “partOf” custom relation relates two classes “Engine” and “Car” in a triple as [Engine partOf Car]. The “typeOf” custom relation is used to relate two other classes “Cessna”, and “Airplane” respectively in the triple as [Cessna typeOf Airplane]. So in the array list of the custom relations named array list1, “partOf” is the first element and “typeOf” is the second element of array list1. A KG named KG_A was used as the initial KG to create the “rel.xsd” and “owl.xsd” schemas and syntax validated successfully with two custom relations namely “typeOf” and “includedIn” respectively and are declared in “owl.xsd” in that order. When KG_B is syntax validated, the “partOf” is written to “owl.xsd” and “rel.xsd” because it is a new relation and needs to be declared. The “typeOf” relation is not written to “owl.xsd” or “rel.xsd” because it has already been declared by KG_A. However during the syntax validation before calling “DomParse” program of the Multiple Pass Syntax Validation algorithm, an error will occur because the “typeOf” is encountered first before “partOf” in the “owl.xsd” schema (it is expecting the FIFO order of “partOf” then “typeOf” in the “owl.xsd”). To illustrate further the “owl.xsd” schema will now have the relations declared in this order (“typeOf”, “includedIn”, and “partOf”). To solve this problem, when a new custom relation encountered in a loop is flagged to be declared in “owl.xsd” and “rel.xsd” schemas, that custom relation name is used to check in the array list holding all custom relations retrieved from the KG and looks for the next

or adjacent custom relation name and retrieves it. It determines if the follow-up custom relation has been declared in “owl.xsd” and if so, gets its line number from the line number array list which has been retrieved from the “owl.xsd” schema using the corresponding index of the declared custom relation in the declared custom relation’s array list. The retrieved line number is decremented by one and the custom relation to be declared is inserted at that line number with appropriate syntax thus maintaining the FIFO order in the declarations of the new custom relation. This process is repeated if warranted for other custom relations that need to be updated. In our use case the order of the relations will now be (“partOf”, “typeOf”, and “includedIn”) in conformity with the FIFO custom relation order of the custom relations as listed in the Classes portion of KG_B knowledge graph. Note that for the declaration of custom relations in “rel.xsd” schema, the FIFO ordering does not apply. The only requirement is that the new custom relations be declared with appropriate syntax. After the KG has been validated, the “owl.xsd” and “rel.xsd” are reverted to their original templates (that has the initial custom relations of KG_A pre-populated and declared) by replacing them with fresh copies from a template folder. The challenge to check the custom created schema if the element names have been declared and if not, declare them instantaneously before the KG syntax is validated is resolved by using the Multiple Pass Syntax Validation Algorithm which encompasses the resolution of the FIFO custom relations declaration problem of “owl.xsd” schema. A revised algorithm was designed as shown in Figure 38 below which depicts the Multiple Pass Syntax Validation Algorithm with DOM Parsing used by “ValidateKGraph” function of Pace Jena. Table 2 describes a pseudo code for Multiple Pass Syntax Validation Algorithm with DOM Parsing.

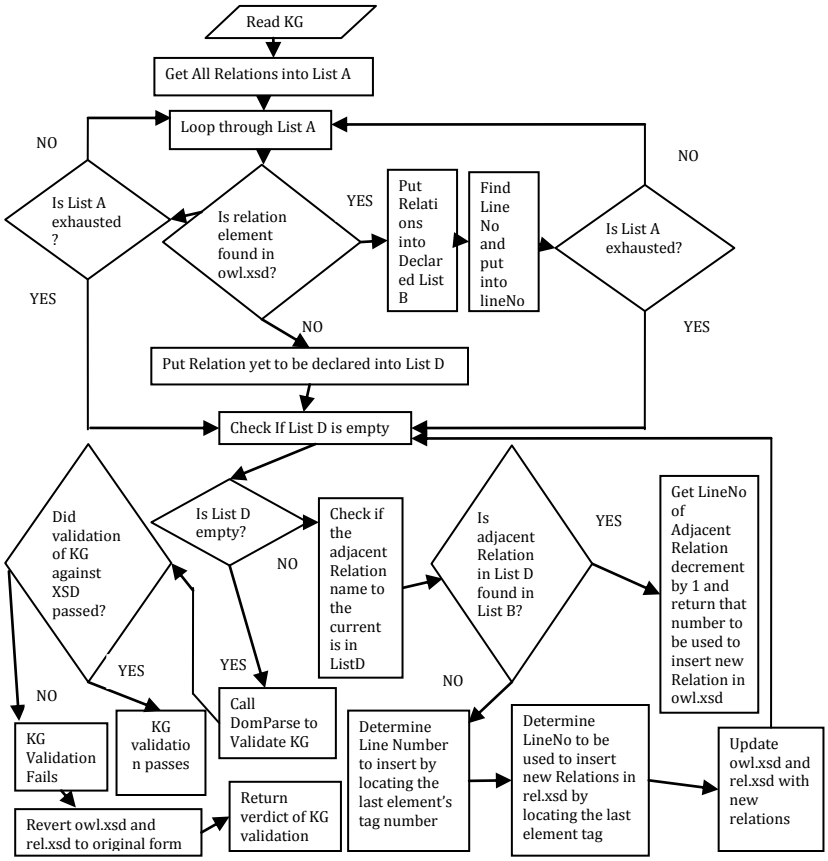


Figure 38: Multiple Pass Syntax Validation Algorithm with DOM Parsing

Table 2: Multiple Pass Syntax Validation Pseudo Code

Require: *KG*

Ensure: Updated KG

1: **Read KG**

2: **Get all relations into list*A***

3: **for ∀ relations in List*A* do**

4: **if Relation tag is in owl.xsd then**

5: **Put** in list*B*

6: **Get line number in owl.xsd and put in List*C***

7: **end if**

8**: if Relation tag is not in "owl.xsd" then**

9**: Put relation tags into list*D* to be declared as new relation**

10: **end if**

11: **end for**

12: **while List*D* is not empty do**

13: **if the adjacent relation name to the intended release to be updated is in List*B***

14: **Use the corresponding index to get the lineNo from list*C*, decrement lineNo by 1 to be used to update relation in "owl.xsd"**

15: **else locate last element tag in "owl.xsd" and get the lineNo to be used to update relation in "owl.xsd"**

16: **end if**

17: **Locate the last element tag in "rel.xsd" and get line number to be used to update relation in "rel.xsd"**

18: **Update "rel.xsd" and "owl.xsd" schemas by declaring new relation tag using determined lineNos**.

19: **end while**

20: **Call "DomParse.java" with "main.xsd" schema, KG document, -print flag as arguments**

21: **Validate KG against XSD**

22**. Revert "rel.xsd" and "owl.xsd" and to original state by discarding changes after**

validation

23: **Return verdict of validation whether successful or unsuccessful**

When an inputted knowledge graph document is sent for knowledge graph syntax validation, the “getFIFORelFromKG” function of “ValidateKnowledgeGraph” which is part of “ValidateKGraph” function of Pace Jena is used to retrieve all custom relations into an array list collection. In a loop each relation is verified in the “owl.xsd” schema document to ascertain if it is declared in the owl schema (“owl.xsd”). If the relation tag is found in “owl.xsd” document, it means it has already been declared so no declaration is needed but it is put into an array list and its line number is determined and put into another array list as well and the next relation element in the array list is fetched and checked if it is declared in the “owl.xsd” schema. If it is determined that the relation is not declared and therefore not found in “owl.xsd”, the relation is put in a new relation array list collection. This process is repeated until the first array list is exhausted. A second Pass is initiated if new relations are needed to be declared before the KG is validated. This is determined by testing if the new relation array list’s size is greater than zero. If the new relation array is determined to contain custom relations to be declared, then in a loop, each custom relation is declared in the “rel.xsd” and “owl.xsd” documents. However for the declaration in the “owl.xsd”, the adjacent custom relation to the custom relation to be declared is first determined by checking the array list holding all relations in a FIFO order and then check the array list holding the already declared relations if there is a match and if there is, the corresponding line number is fetched, decremented by 1 and will be used to update the “owl.xsd”. The decrementing by 1 ensures that the newly declared relation will precede the adjacent relation found to be already declared in the

“owl.xsd” document. Before the new relations tag in the “rel.xsd” document is declared by updating the schema document, the position (line number) to insert and declare the new relation had to be determined and a function ValidateKnowledgeGraph.getTokenNumber(String relation, String owlSchema, String tokenStr) imported into “ValidateKGraph” function of Pace Jena is used to return the line number that the relation tag to be declared is to be inserted. It then updates the “rel.xsd” document using theValidateKnowledgeGraph.insertStringToFile(String fileName, int lineno, String lineToBeInserted) function. The same process is repeated for the “owl.xsd” schema using the line number derived as described above. If the adjacent relation to the relation to be declared is not found to be declared in “owl.xsd”, the same method used to find the insertion point by determining the line number for “rel.xsd” is used for updating “owl.xsd” as well. Note that though the String relation will remain the same for both schema documents, their document names and “tokenStr” which is a pattern matching string will differ since the declaration syntax of the relation element in the “rel.xsd” and “owl.xsd” have different formats and will require different matching tokens to locate the position (line number) to insert the relation element. The update of both the “rel.xsd” and “owl.xsd” schemas documents is repeated until the new relations array list is exhausted. Once the update and declaration of new relations elements are completed, the third Pass is initiated. The “DomParse” program is called with the KG document and the “main.xsd” schema as arguments into the main function of “DomParse” to validate the syntax of the KG document. Note that the “main.xsd” document imports the ‘rel.xsd’, the “owl.xsd”, “pace.xsd”, “xml.xsd”, and the “rdfs.xsd” documents respectively. With this

configuration, the “rel”, “owl”, “pace”, “xml”, and “rdfs” namespaces persists in the “main.xsd” document.

A KG Syntax validation was tested using the “animal.owl” KG as shown in Figure 39. It was then put through ValidateKGraph function of Pace Jena. The validation was run with the output shown in Figure 40.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rel "http://www.pace.edu/rel-syntax-ns#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>
<rdf:RDF xmlns="http://www.pace.edu/animal#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rel="http://www.pace.edu/rel-syntax-ns#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Ontology rdf:about="http://www.pace.edu/animal"/>
    <!--
    // Relations
     -->
        <rel:NewRelation rdf:about="http://www.pace.edu/animal#isHuntedBy"/>
    <!--
    // Classes
     -->
    <!-- http://www.pace.edu/animal#Antelope -->
    <owl:Class rdf:about="http://www.pace.edu/animal#Antelope">
    </owl:Class>
    <!-- http://www.pace.edu/animal#Buffalo -->
    <owl:Class rdf:about="http://www.pace.edu/animal#Buffalo">
    </owl:Class>
    <!-- http://www.pace.edu/animal#Lion -->
    <owl:Class rdf:about="http://www.pace.edu/animal#Lion">
        <rel: isHuntedBy rdf:resource="http://www.pace.edu/animal#Buffalo"/>
        <rel: isHuntedBy rdf:resource="http://www.pace.edu/animal#Antelope"/>
    </owl:Class>
    <!-- http://www.pace.edu/animal#Skin -->
    <owl:Class rdf:about="http://www.pace.edu/animal#Skin">
    </owl:Class>
    <!-- http://www.pace.edu/animal#Tiger -->
    <owl:Class rdf:about="http://www.pace.edu/animal#Tiger">
        <rel: isHuntedBy rdf:resource="http://www.pace.edu/animal#Buffalo"/>
     </owl:Class>
</rdf:RDF>
<!-- Generated by the OWL API (version 3.5.1) http://owlapi.sourceforge.net -->
```

Figure 39: “animal.owl” Knowledge Graph

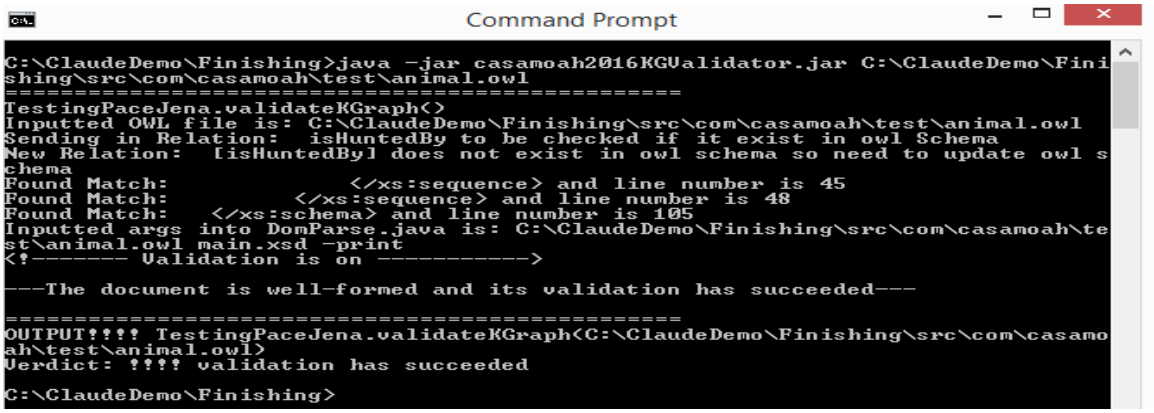


Figure 40: Output for “animal.owl” Knowledge Graph Syntax Validation

3.5 Conclusion

The Syntax Validation applies to a supported subset of RDF/XML serialized documents consisting of KGs with custom relations and an example of a non-custom relation XML/RDF document. It was tested on the Windows platform only and not on other OS platforms. Note that the “main.xsd” and “rdfs.xsd” schemas are not updated but are required for the proper functioning of the Pace custom schema by working in tandem with the “owl.xsd”, “pace.xsd”, “xml.xsd”, and “rel.xsd” schemas. Knowledge graph syntax validation was a challenge in that the Pace KG with custom relations could not be validated using W3C OWL schema because it is RDF/XML serialized. There is an RDF Validator on the Internet called the W3C RDF Validation Service [60] which takes an RDF/XML serialized owl document and parses the owl document in triples of the data model of subject, predicate, and object but this “service does not do any RDF Schema Specification validation” but rather it is an RDF parser [61]. Also the custom relations embedded in the knowledge graph could not be validated until the use of the Multiple Pass Syntax Validation (MPSV) Algorithm with DOM Parsing implemented in the

ValidateKGraph function of Pace Jena. The knowledge graph syntax validation was also made possible by designing a custom Pace Schema using a scheme of creating multiple schemas namely, “main.xsd”, “rdfs.xsd”, “xml.xsd”, “pace.xsd”, “rel.xsd”, and “owl.xsd” respectively. This approach was due to the fact that only one target Namespace is permitted per schema but the knowledge graph has multiple namespaces requiring the “main.xsd” schema to import the others schema into itself as well as the other schemas also importing the others into themselves to persist the multiple namespaces in the “main.xsd” schema.

Chapter 4

Visual Knowledge Graph Navigation

4.1 Generic API for Accessing Custom Relations

It is important that visual navigation and visualization be provided for KGs because it is very hard for domain experts to have a global view of their knowledge representation. Since a real-life knowledge graph can easily contain hundreds or thousands of classes with complex inter-relations, it is a major challenge for domain experts to review and validate their knowledge representation. It is hard for application developers to fully understand the complex relations among the classes especially very large KGs. IDEs like Protégé with a *plugin* such as OWLViz may not be suitable to display all class relations and the elements embedded in KGs it its entirety. Pace Extended Protégé is a generic API for accessing the custom relations. If the KG is small it is easy to see the classes and custom relations via OWLViz as shown in Figure 41. However if the KG is large, it is difficult to navigate the class relations and see all the inter-connections. Figure 42 shows Threat Types of cyber security communications and if more classes and custom relations are added say just about 50 more, it becomes increasing evident that the diagrammatical display will get crowded and make it difficult to navigate the numerous connections.

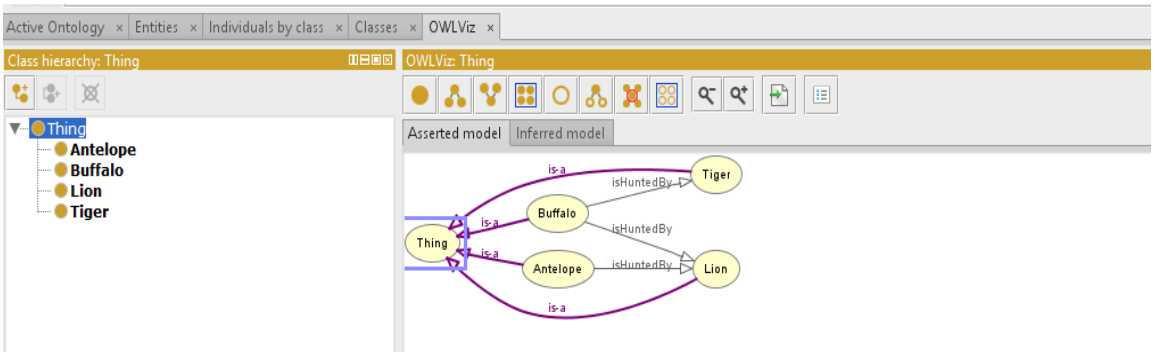


Figure 41: OWLViz Display of Classes and Relations within The "animal.owl" Knowledge Graph

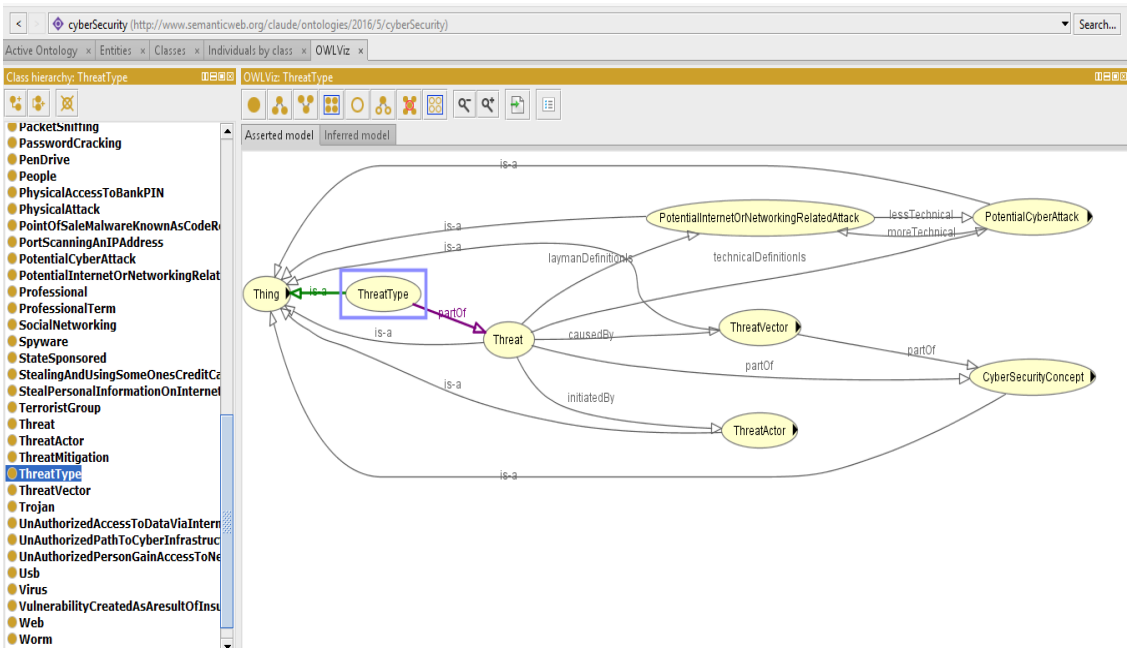


Figure 42: Threat Types Relations

4.2 Web Based Knowledge Graph Navigation

Before the visual navigation and visualization application can be developed, the entities in the KG namely the classes and the custom relations embedded in the KG needs to be retrieved via parsing and put into a collection lists so that it can be acted upon. The "readOntology" function of Pace Jena takes an RDF/XML serialized document as input and uses print functions in Pace Jena to print the classes, custom relations, properties, data types, and learning orders to console via the printClasses(), printRelations(), printProperties(), printDataTypes(), and printLearningOrders() functions respectively. Though these entities can be printed to console via their corresponding functions, it was not suitable to be used in a full scale visual navigation and visualization application. This prompted the creation of three functions in Pace Jena namely the getClassNames(), getRelations(), and getClassRelClass() to facilitate the visual navigation and visualization of the KG. The first function will load and return an array list with all unique classes in the KG. The second function will load and return all unique relations in the KG. The third function which takes a class name and a relation name as arguments and return all instances of the triple namely a class instance as the subject, the custom relation as the predicate, and the related class instance as the object. After testing these functions using different KG as inputs and verifying that the returned output was correct and consistent based on the function under test, the work flow of the visual navigation and visualization application design was embarked. Since it is prudent to syntax validate any KG to be well formed and OWL syntactically correct before using it in any application, the work flow of the visual navigation and visualization application was designed to first do the syntax validation first and if the validation is successful, it then proceeds to provide visual navigation and visualization

capabilities. The KG visual navigation and visualization work flow is portrayed in Figure 43 below.

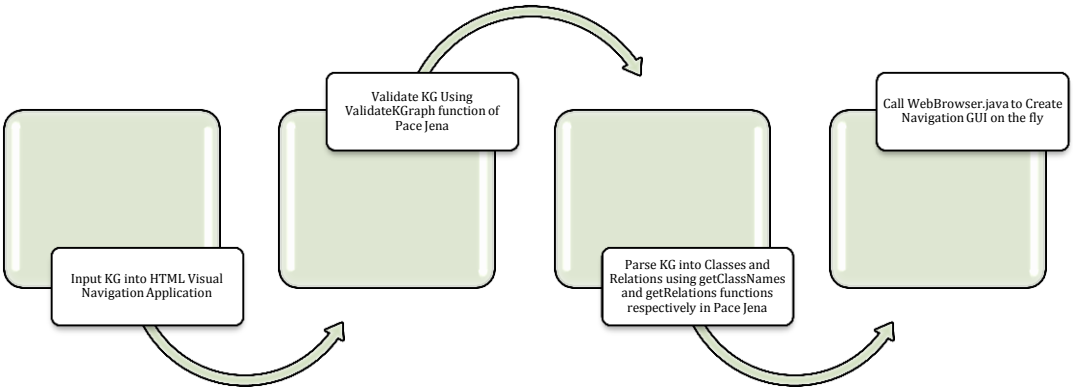


Figure 43: KG Visual Navigation Work Flow Web Model

The WebBrowser program was designed as an HTML version of the visual navigation application for the custom relation laden KG and takes as an input any supported KG document and uses functions created in a revised Pace Jena to syntax validate the KG and if it is successfully validated, all classes and relations are retrieved in the KG document and displayed in an html page using a program that creates the class relations on the fly. This html application presents to the user, the ability to navigate the classes and custom relations of the KG. After the KG has been syntax validated, visual navigation is provided by running the "WebBrowser" program. Presented below in Figure 44 is the process flow of "WebBrowser" program.

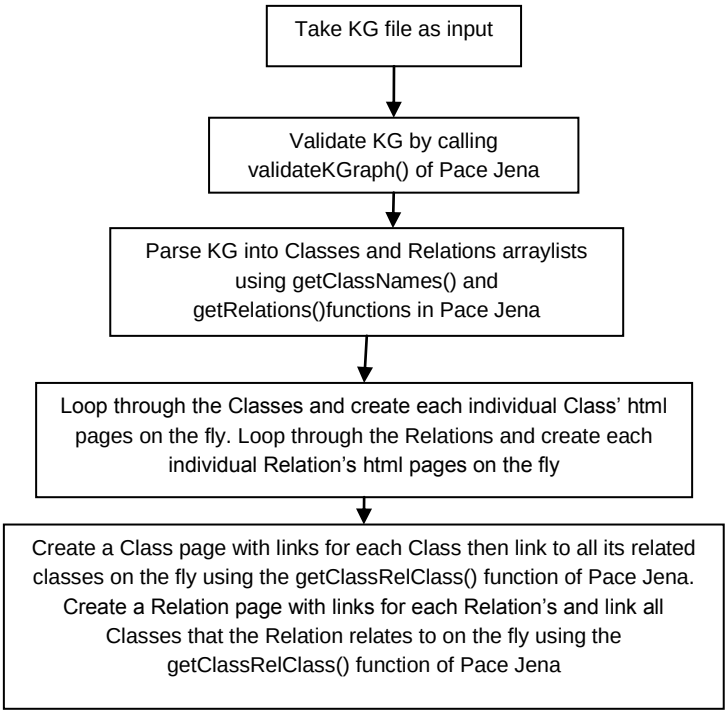


Figure 44: Flow Chart KG Visual Navigation Implementation

The process flow first takes a KG document as argument, syntax validate the KG using the validateKGraph() function of Pace Jena that returns the syntax validation verdict and if successful parse the KG into classes and relations array list using the getClassNames() and getRelations() functions of Pace Jena. It then sorts the array lists in alphabetical order and loops through the classes and relations lists and create their individual html pages on the fly. It then creates a Class page and in a loop of the classes' array list, creates a hyperlinked class with links to all its related classes instantaneously using the getClassRelClass() function of Pace Jena. It then creates a Relation page and in a loop of the relations' array list, creates a hyperlinked relation with links to all classes it is related

to instantaneously using the getClassRelClass() function of Pace Jena. Figure 45 portrays the logic design workflow.

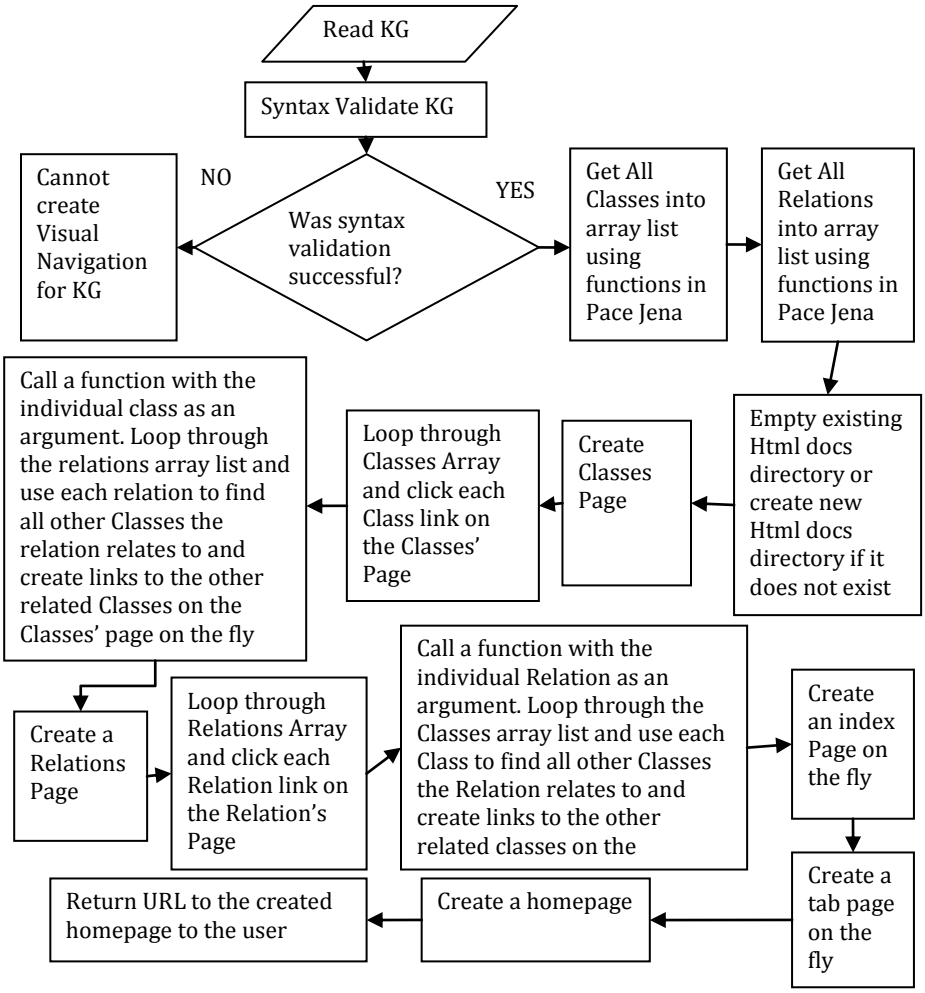


Figure 45: Visual Navigation Web Design Process Flow

The HTML version for the KG visual navigation program was run with the "animal.owl" KG document as input which is shown in Figure 46 below.

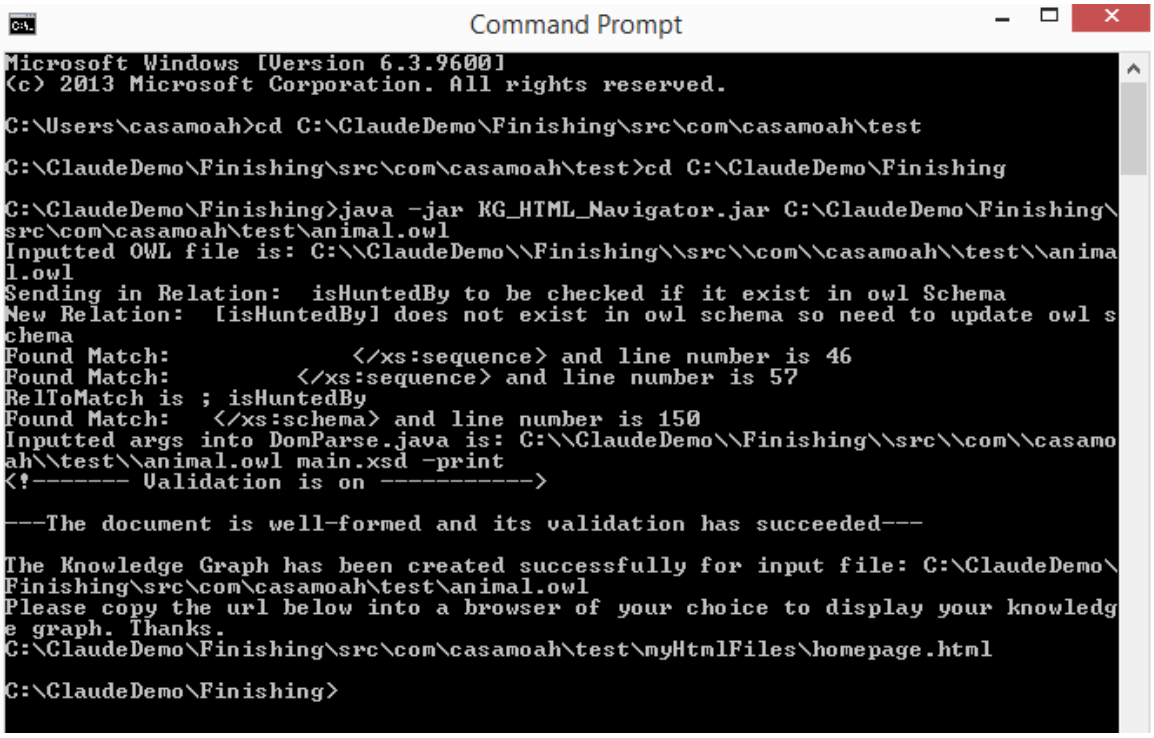


Figure 46: Output of HTML Version Visual Navigation for KG Launch

The URL is copied per the instruction of the output to a browser to launch the HTML version of the visual navigation for the “animal.owl” KG. Presented below in Figure 47 is the classes page showing all classes in the animal KG, Figure 48 is relations page, showing all relations in the animal KG, Figure 49 is *Lion* Class page showing all Class *Lion* relations to other classes, and *isHuntedBy* relation page at Figure 50 is showing all classes that are related by the *isHuntedBy* relation. When you click any class link, it displays the corresponding classes that it is linked to.

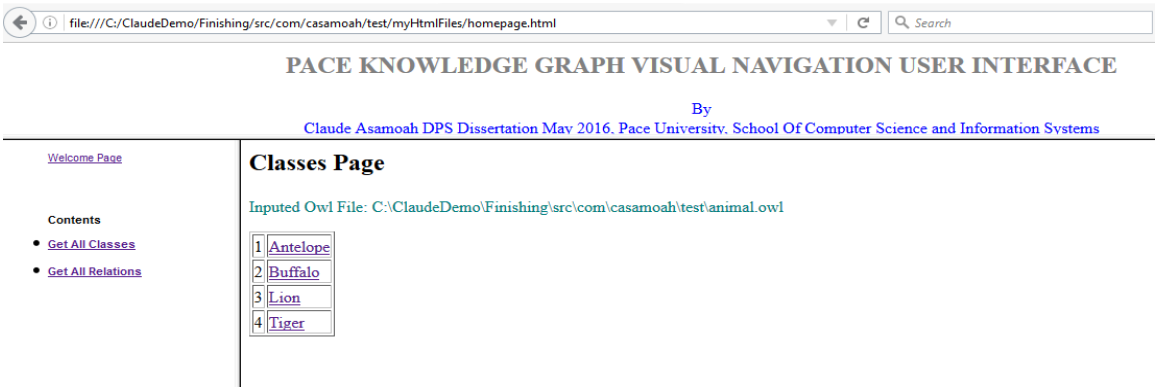


Figure 47: Class Page

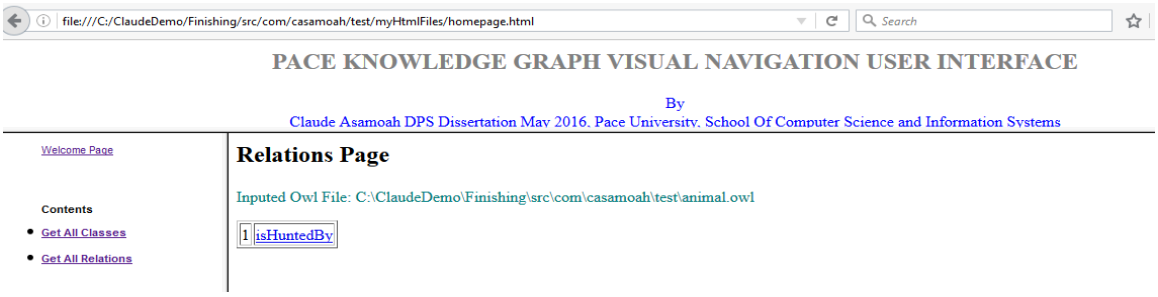


Figure 48: Relations Page

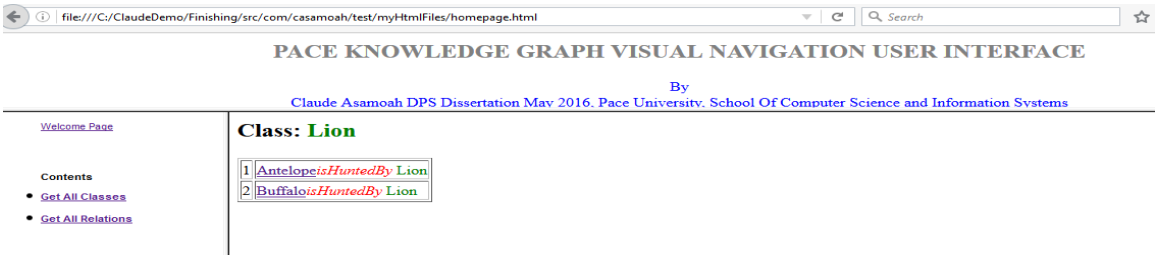


Figure 49: Class Lion Page

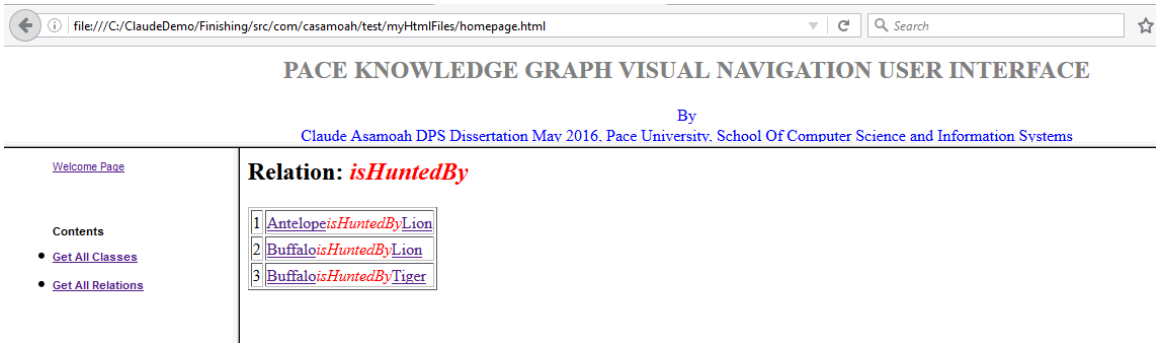


Figure 50: Relation "isHuntedBy" Page

The web based model of the visual navigation and visualization of the KG is rugged and requires no internet connection but just a browser. Neither does it require a database. A supported inputted KG is syntax validated and if the syntax validation is successful, a visual navigation and visualization tool is assembled and provided to the user on the fly. This will come in handy for domain experts working in remote regions with limited access to IT infrastructure needing to view the KG of their knowledge representation systems. The limitation of the web based model is that creating a visual navigation and visualization tool for an inputted KG on the fly requires that it be run from the command line and may not be appealing to some users.

4.3 Application-Based Graph Navigation

The application model of the visual navigation and visualization tool for KGs is a GUI based application that allows the user to select a KG as input from a File Explorer. The application model follows a similar work flow as its html counterpart. First, the KG has to be syntax validated to make sure it is well formed and OWL syntactically correct using

the Pace Multiple Pass Syntax Validation and Dom Parser as discussed earlier in Chapter 3. It then uses the getClassNames() and getRelations() functions created in Pace Jena to populate the “Classes” and “Relations” Combo Boxes’ drop down lists. But before the Combo Boxes drop down lists with its corresponding Grid Box that displays the class relations is displayed, an action driven “Display Classes” and “Display Relations” buttons needs to be clicked to trigger the population of the associated Grid Boxes with the class relations in the triple via an Action Listener. Note that the default selection of the “Classes” and “Relations” buttons are “ALL” that shows all class relations in the “Classes” Grid Box and all relations related classes in the Relations Grid Box. Selecting any individual class in the “Classes” dropdown list will display all related classes of the selected class. Likewise, selecting an individual relation from the “Relations” dropdown list will display all classes the selected relation relates to in a triple format. Note that the display of the resultant triple in the Grid Boxes are facilitated by the getClassRelClass() function of Pace Jena. Presented below in Figure 51 is the work flow of the application based KG visual navigation and visualization model.

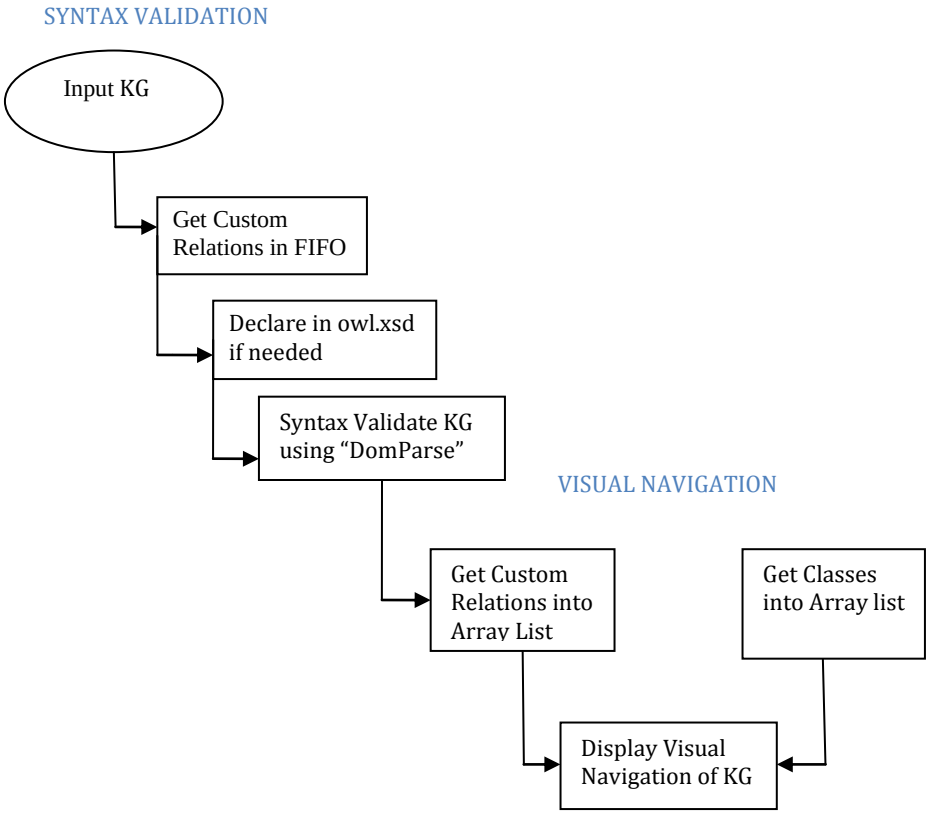


Figure 51 : Work Flow of Application-based KG Navigation

The Application model of the knowledge graph is a robust Java application that allows the user to browse a KG document as input into the application in-order to navigate the knowledge graph. Created is a jar file that can be double-clicked to launch the application. The application consists of a Pace Jena, "ClassesKnowledgeGraph", "RelationsKnowledgeGraph", and layout1 Java Classes.

Presented below in Figure 52 is the process flow of layout1.

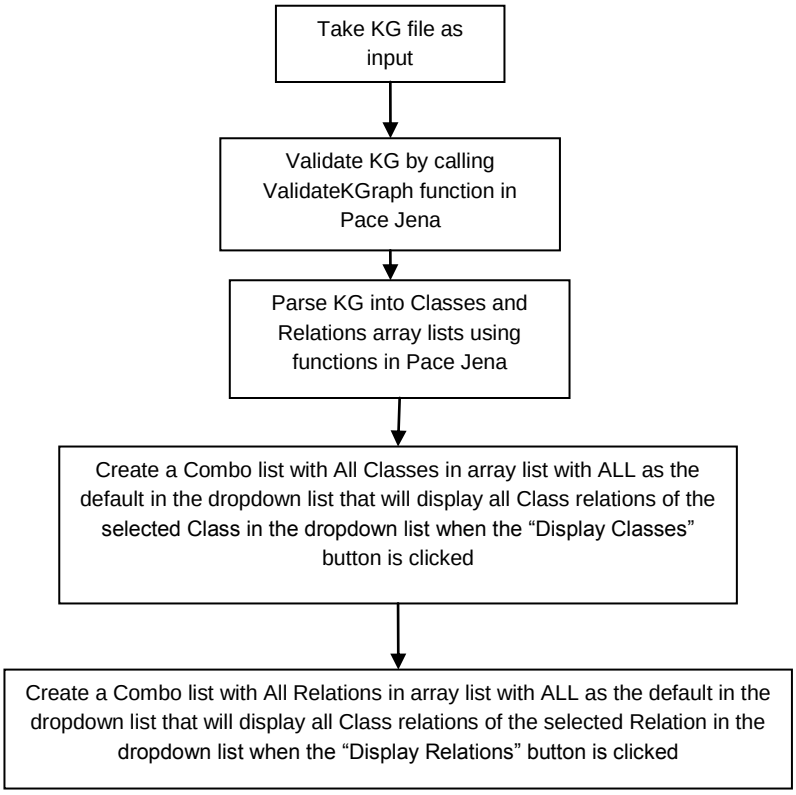


Figure 52: Process Flow of “layout1.java”

The layout1 has two combo boxes one for the Classes and the other for Relations. It uses an Action Listener if “Display Classes” button is clicked to populate the Class’ combo box’s dropdown list by calling “ClassesKnowledgeGraph” java class. Figure 53 depicts the visual navigation application web design process flow.

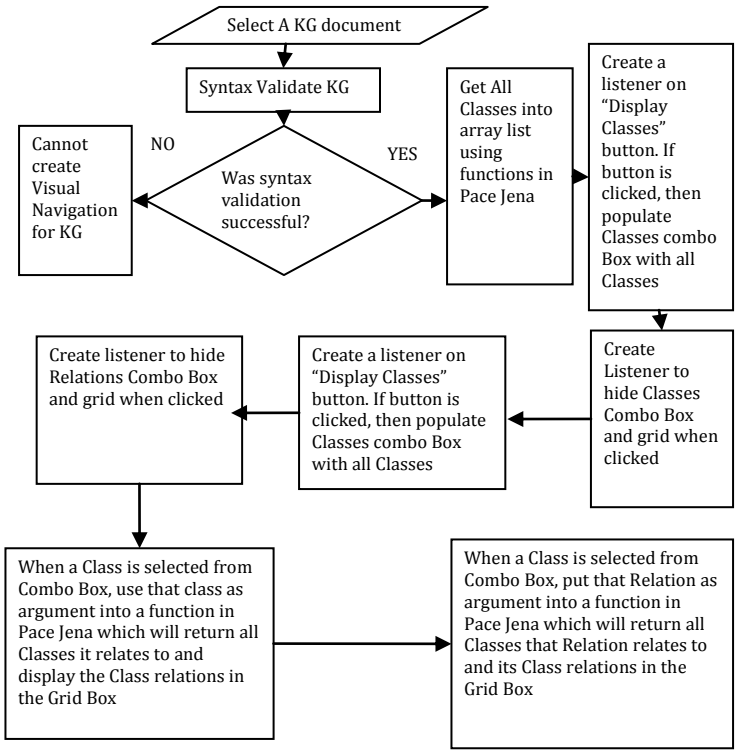


Figure 53: Visual Navigation Application Design Process Flow

When a class item is selected in the dropdown list, it displays its class relations in a grid box. Subsequently when a "Display Relation" button is clicked, it uses an action listener to populate the Relation combo box dropdown list by calling the "RelationsKnowledgeGraph()" Java class. Clicking the "Browse for OWL Document" will present to the user a Windows File explorer to select a KG document as input into the application model of the KG Visual Navigation GUI. When a KG document is selected, the application syntax validates it to determine if it is OWL syntactically correct and well-formed. If the KG validates successfully, it provides a visual navigation. If syntax validation fails or the document selected does not have the ".owl" extension, an "Error" message box is displayed and the user will have to fix the problem in the KG or

select a valid KG to proceed with the visual navigation. A “runnable” jar file of the application model shown below in Figure 54 is double click to launch the application.

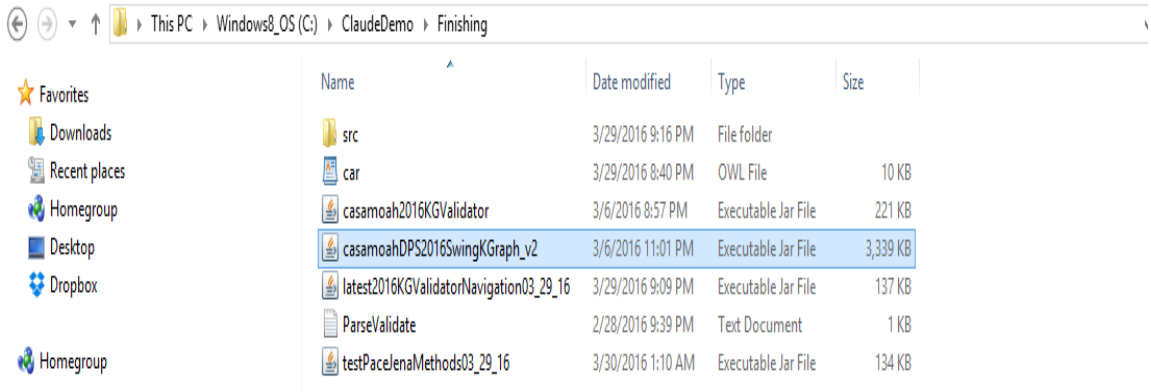


Figure 54: Launching the Java Application Model of the KG Visual Navigation

The “country.owl” KG document is navigated to and selected as input to the application as shown in Figure 55.

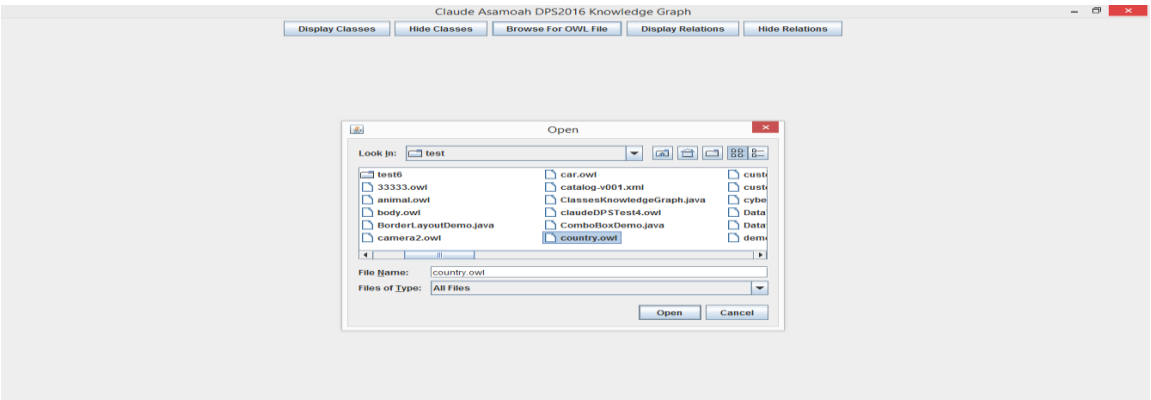


Figure 55: Selecting the “country.owl” KG as Input to the Application

Figure 56 shows the resultant output depicting the successful validation of the KG and the subsequent display of the all classes and relations embedded in the KG via their default “All” selected item of their respective dropdown lists.

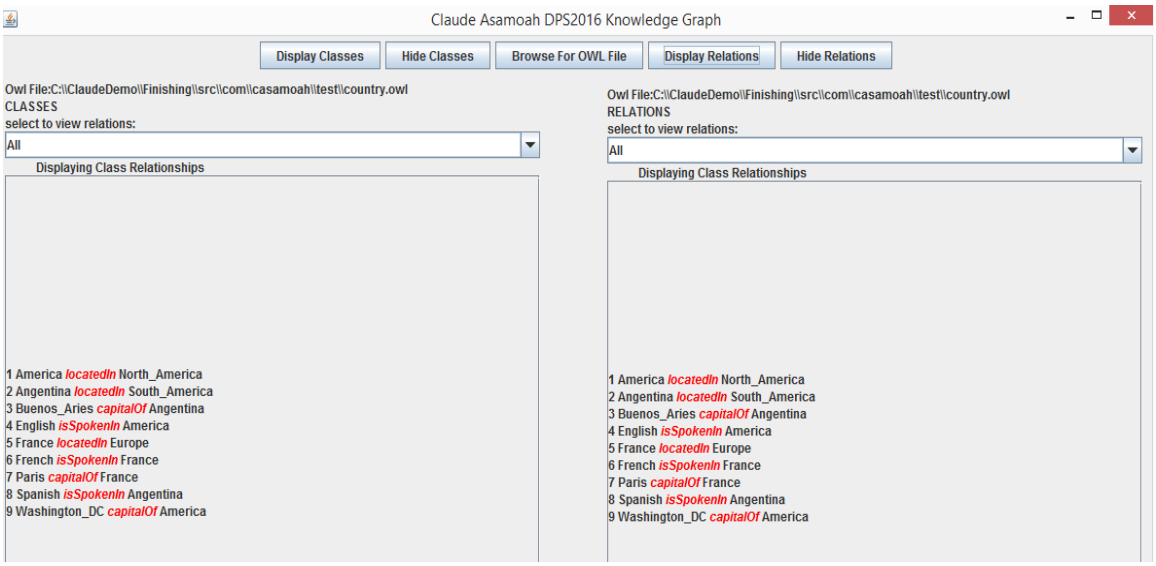


Figure 56: KG Visual Navigation with Default "ALL" Selected for both Classes and Relations

Selecting “America” as Class item and “capitalOf” as selected Relation item from their respective dropdown lists manifests the display illustrated in Figure 57.

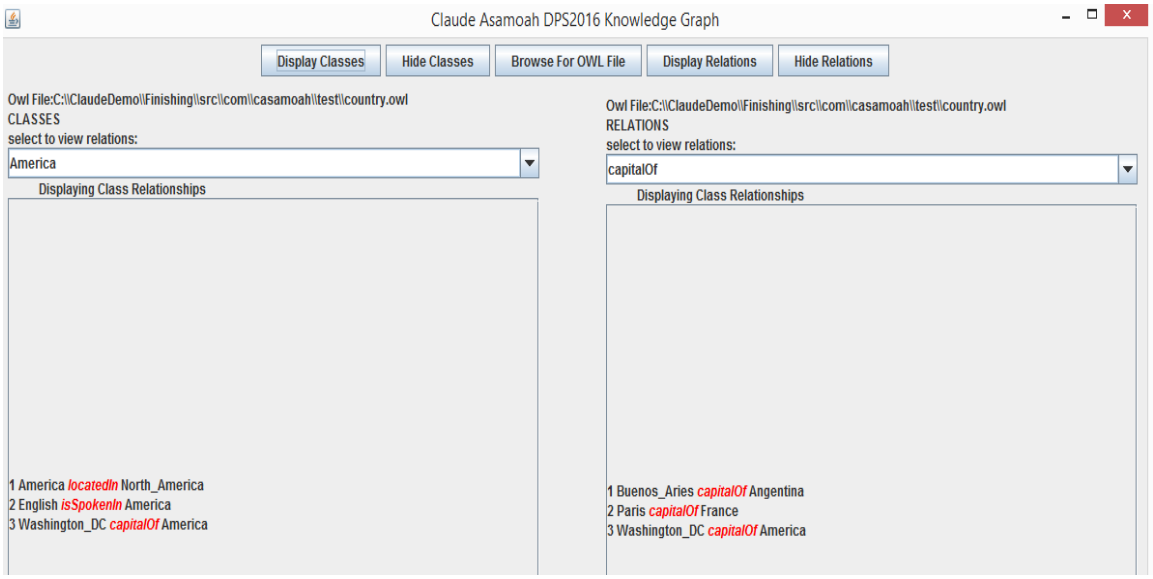


Figure 57: Application Model of the KG Visual Navigation GUI

4.4 Conclusion

The design and implementation of the visual navigation for KG is limited to KGs created with custom relations using the “rel” namespace. It does not support non-custom relations owl documents. OWL documents with object and data properties are not supported. Extending OWL to KG enables domain experts to design knowledge representation systems and applications using expressive custom relations. However, the KG needs to be navigated to ascertain its completeness and correctness before it can be used to design these knowledge-representation and knowledge-based decision-making systems. Moreover, if the KG is large, it is difficult to verify the correctness and inter-relations between the classes. The KG Visual Navigation system uses functions in Pace Jena in conjunction with the Web Browser program to facilitate visual navigation for KG in the Web or HTML format. Likewise, the KG Visual Navigation system uses functions in Pace Jena in conjunction with layout1 program to facilitate visual navigation for KG in the application model. The ability for the knowledge graph to accommodate any inputted KG document with custom relations makes it a powerful tool that enables domain experts to visualize the symbiosis between the classes and their custom relations and to review and validate their knowledge representation.

Chapter 5

Experimental Validation

5.1 A Sample Knowledge Graph for Cyber Security Communications

A cyber engineer wants to build a knowledge graph of cyber security terminology using a semantic approach to bridging the gap between different levels of technicality from layman to technological sophistication. He posits that for any topic, there are different versions of terminology. Some are formal, others are less formal thus necessitating the need to bridge the communication between different groups of people so that the terminology at the different levels of formality can work together and be reconciled to some extent. The benefit of reconciling these different versions of terminologies is to make searches of documents more effective. This research will use this cyber security communications Use Case as the basis for the validation of the knowledge graph syntax validation and visual navigation for developing intelligent systems.

Cyber security has threats, networking concepts, and cyber security terminology. Threats are cyber security concepts that have threat actors, threat vectors, threat types, and threat mitigations. In order to define and create the classes, a class hierarchy is first constructed of the cyber security terminology using a top down approach. Portrayed in Figure 58 are the cyber security concepts' relations and in Figure 59 are the cyber security concepts to threat relations. You will notice that only a section of the tree hierarchy is displayed

because displaying all the classes though possible will be clustered again emphasizing the need of a knowledge graph.

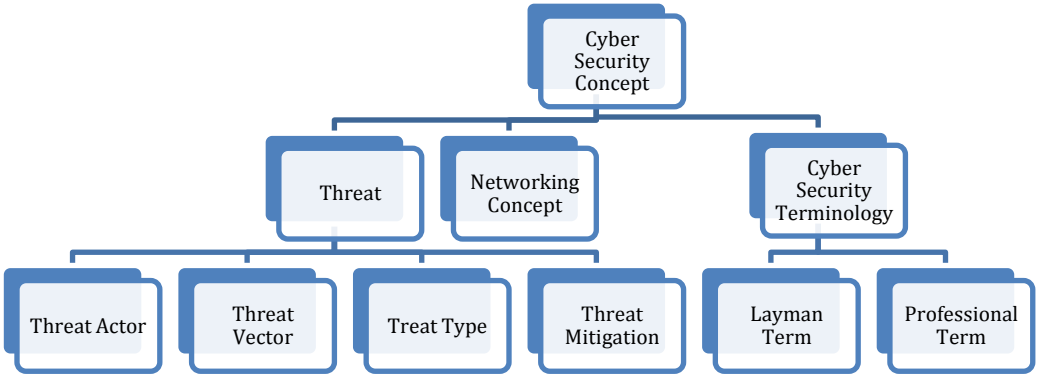


Figure 58: Cyber Security Concepts Relations

Figure 59 –Figure 61 below shows some of the KG relations derived from the class hierarchical tree of Figure 51.

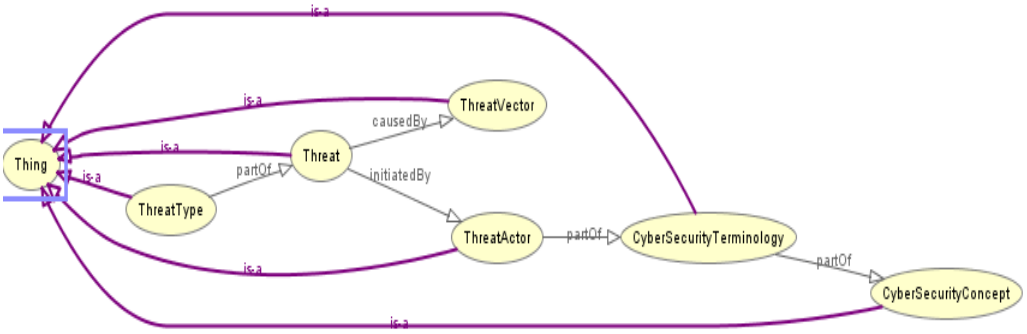


Figure 59: Cyber Security Concepts to Threat Relations

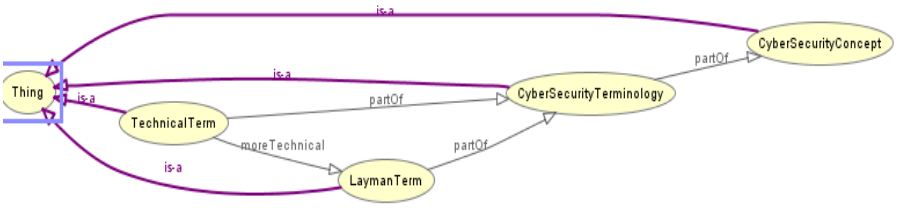


Figure 60: Layman Term, Professional Term, and Cyber Security Terminology Relations

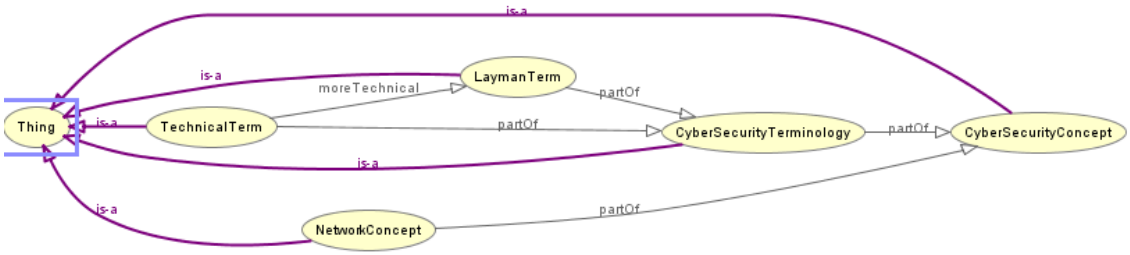


Figure 61: Networking Concept Relation to Cyber Security Concept

Cyber security concepts have many threat actors. Figure 62 shows the various threat actors within the cyber security concept hierarchical tree.

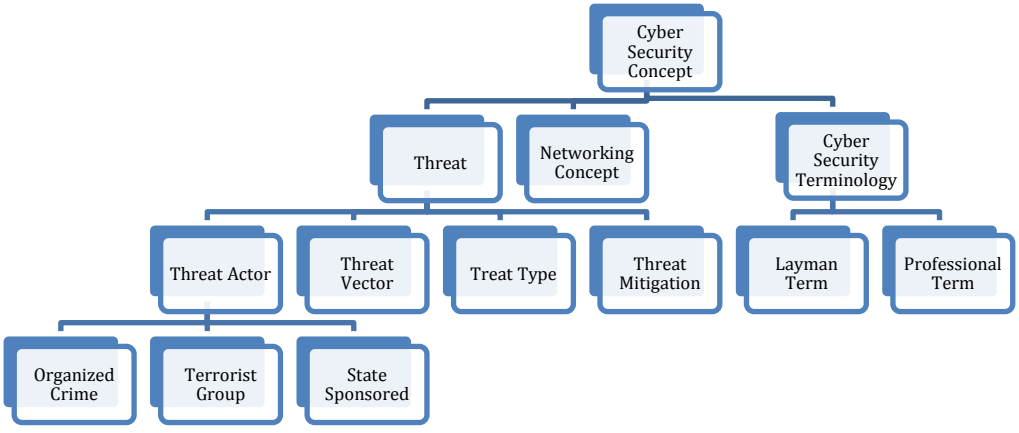


Figure 62: Threat Actor's Entities within the Cyber Security Concept Hierarchy

Cyber security threats have threat actors that strive to wreck-havoc on a country's or organization's cyber security infrastructure. Three categories of cyber threat actor namely state-sponsored, organized crime, and terrorist group have been identified [57]. Figure 63 describes the threat actor's entities KG class relations.

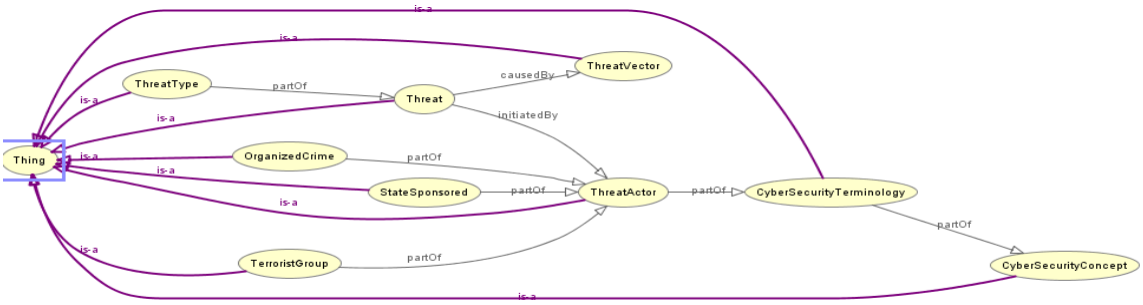


Figure 63: Threat Actor's Entities Relations

Moving down the cyber security hierarchy, Figure 64 shows the threat vector's entities within the hierarchy.

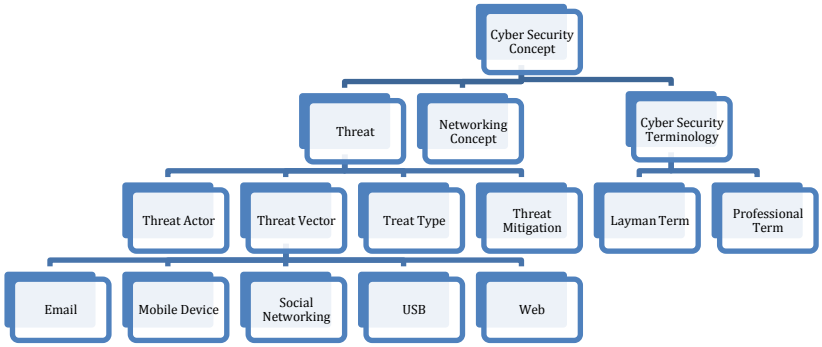


Figure 64: Threat Vector Entities within the Cyber Security Concept Hierarchy

Displayed in Figure 65 are the threat vectors' KG class relations.

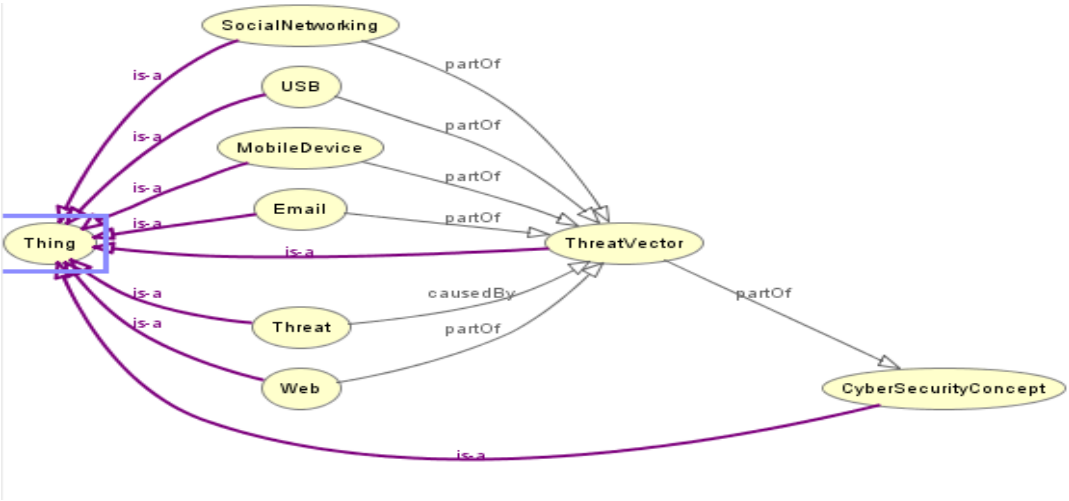


Figure 65: Threat Vector's KG Class Relations

Presented in Figure 66 below are the Threat Types entities within the cyber security concept's hierarchy.

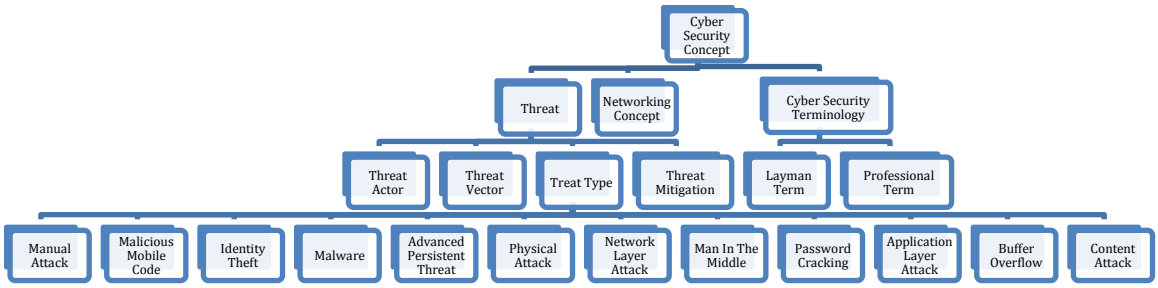


Figure 66: Threat Type's Entities within the Cyber Security Concept's Hierarchy

Presented below in Figure 67 are the threat type's KG class relations.

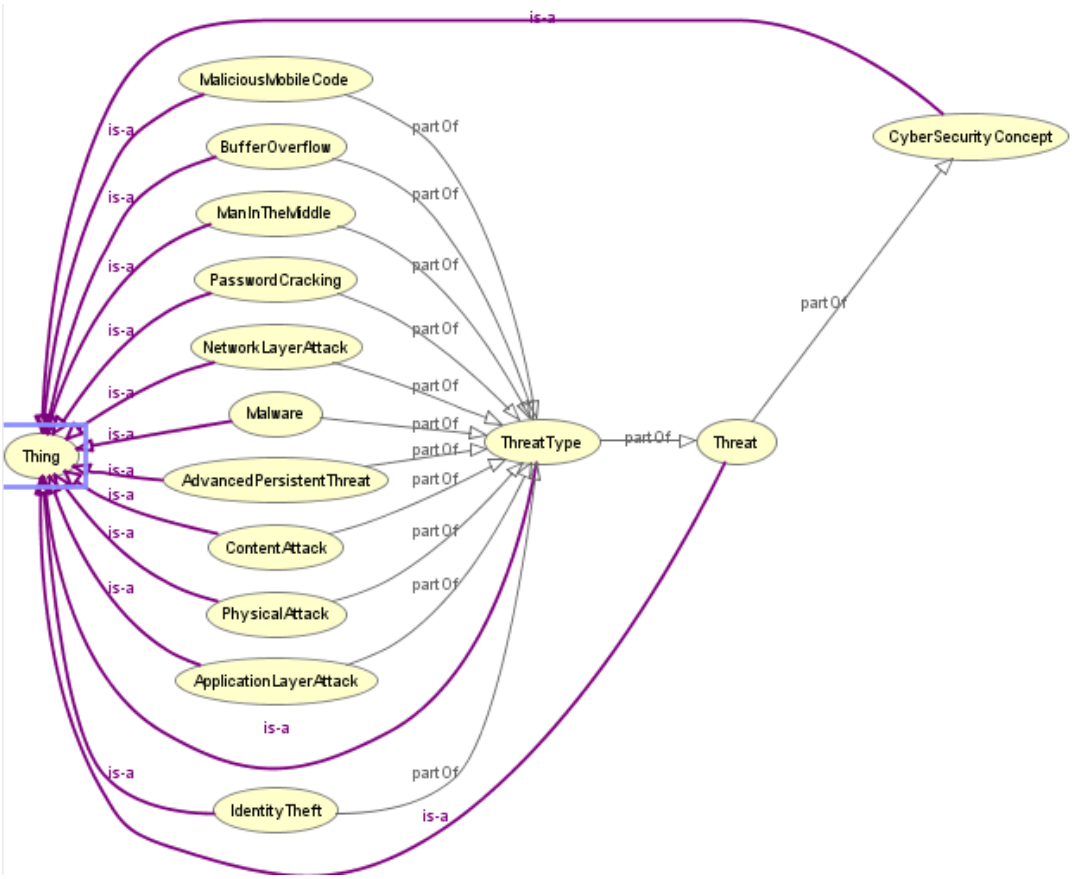


Figure 67: Threat Type's Class Relations

Moving down the cyber security hierarchy tree, Figure 68 depicts the threat mitigation entities in the cyber security threat vulnerability mitigation classes.

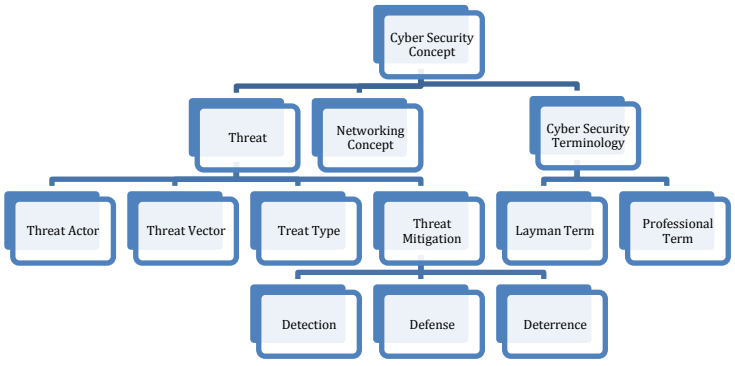


Figure 68: Threat Mitigation's Entities within the Cyber Security Concept's Hierarchy

Depicted below are three types of threat mitigation entities, notably; detection, defense, and deterrent which are displayed in Figure 69.

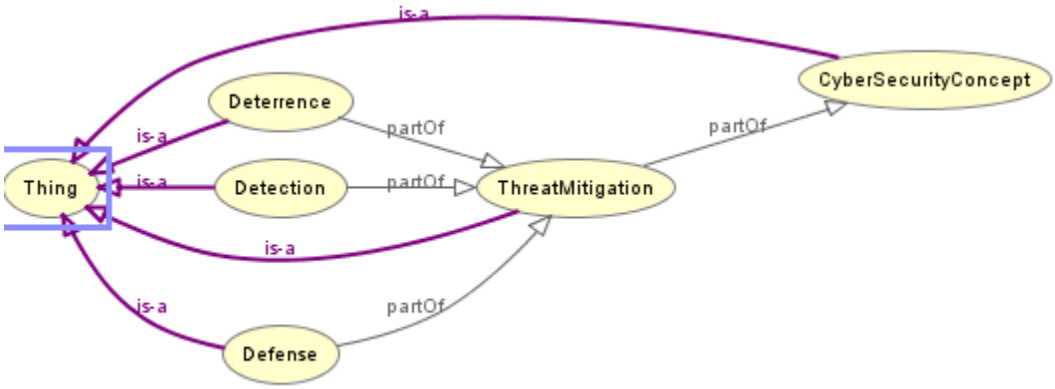


Figure 69: Threat Mitigation's Class Relations

Appendix B shows the owl document created for the Visual Navigation of cyber security terminology KG which is a KG to bridge the gap between different levels of technicality from layman to technological sophistication of cyber security concepts. The created KG document will first be syntax validated by the *validateKGraph* function in Pace Jena and then parsed into a collection of relations and classes using the *getRelations* and *getClassNames* functions of Pace Jena. The classes and relation collection lists are inputted into the visual navigation application of the KG in both the Web and the Application models.

5.2 Syntax Validation

The KG of Cyber Security terminology ("cyberSecurityCommunication.owl") is validated using the validateKGraph function of Pace Jena. A test program was created which takes the full path of the KG document as an argument was ran from the command line as depicted in Figure 70.

Figure 70: "cyberSecurityCommunication.owl" KG Syntax Validation Input Arguments

The output for the validation of the KG is displayed in Figure 71 below that displays the successful syntax validation of the "cyberSecurityCommunication.owl" KG document.


Command Prompt
C:\ClaudeDemo\Finishing>java -jar casamoah2016KGValidator.jar C:\ClaudeDemo\Finishing\src\com\casamoah\test\cyberSecurityCommunication.owl
==
TestingPaceJena.validateKGraph()
Inputted OWL file is: C:\ClaudeDemo\Finishing\src\com\casamoah\test\cyberSecurityCommunication.owl
Sending in Relation: areAlsoCalled to be checked if it exist in owl Schema
Found Match: <xs:element minOccurs="0" ref="rel:areAlsoCalled"/> in owl schema so no update necessary
Found Match: <xs:element minOccurs="0" ref="rel:areAlsoCalled"/> and line number is 40
Sending in Relation: technicalDefinitionIs to be checked if it exist in owl Schema
Found Match: <xs:element ref="rel:technicalDefinitionIs"/> in owl schema so no update necessary
Found Match: <xs:element ref="rel:technicalDefinitionIs"/> and line number is 45
Sending in Relation: laymanDefinitionIs to be checked if it exist in owl Schema
Found Match: <xs:element ref="rel:laymanDefinitionIs"/> in owl schema so no update necessary
Found Match: <xs:element ref="rel:laymanDefinitionIs"/> and line number is 46
Sending in Relation: partOf to be checked if it exist in owl Schema
Found Match: <xs:element minOccurs="0" ref="rel:partOf"/> in owl schema so no update necessary
Found Match: <xs:element minOccurs="0" ref="rel:partOf"/> in owl schema so no update necessary
Found Match: <xs:element minOccurs="0" ref="rel:partOf"/> and line number is 22
Found Match: <xs:element minOccurs="0" ref="rel:partOf"/> and line number is 49
Sending in Relation: isExampleOf to be checked if it exist in owl Schema
Found Match: <xs:element minOccurs="0" maxOccurs="unbounded" ref="rel:isExampleOf"/> in owl schema so no update necessary
Found Match: <xs:element minOccurs="0" maxOccurs="unbounded" ref="rel:isExampleOf"/> and line number is 38
Sending in Relation: lessTechnical to be checked if it exist in owl Schema
Found Match: <xs:element minOccurs="0" ref="rel:lessTechnical"/> in owl schema so no update necessary
Found Match: <xs:element minOccurs="0" ref="rel:lessTechnical"/> and line number is 35
Sending in Relation: moreTechnical to be checked if it exist in owl Schema
Found Match: <xs:element minOccurs="0" ref="rel:moreTechnical"/> in owl schema so no update necessary
Found Match: <xs:element minOccurs="0" ref="rel:moreTechnical"/> and line number is 25
Sending in Relation: isMaintainedBy to be checked if it exist in owl Schema
Found Match: <xs:element minOccurs="0" maxOccurs="unbounded" ref="rel:isMaintainedBy"/> in owl schema so no update necessary
Found Match: <xs:element minOccurs="0" maxOccurs="unbounded" ref="rel:isMaintainedBy"/> and line number is 28

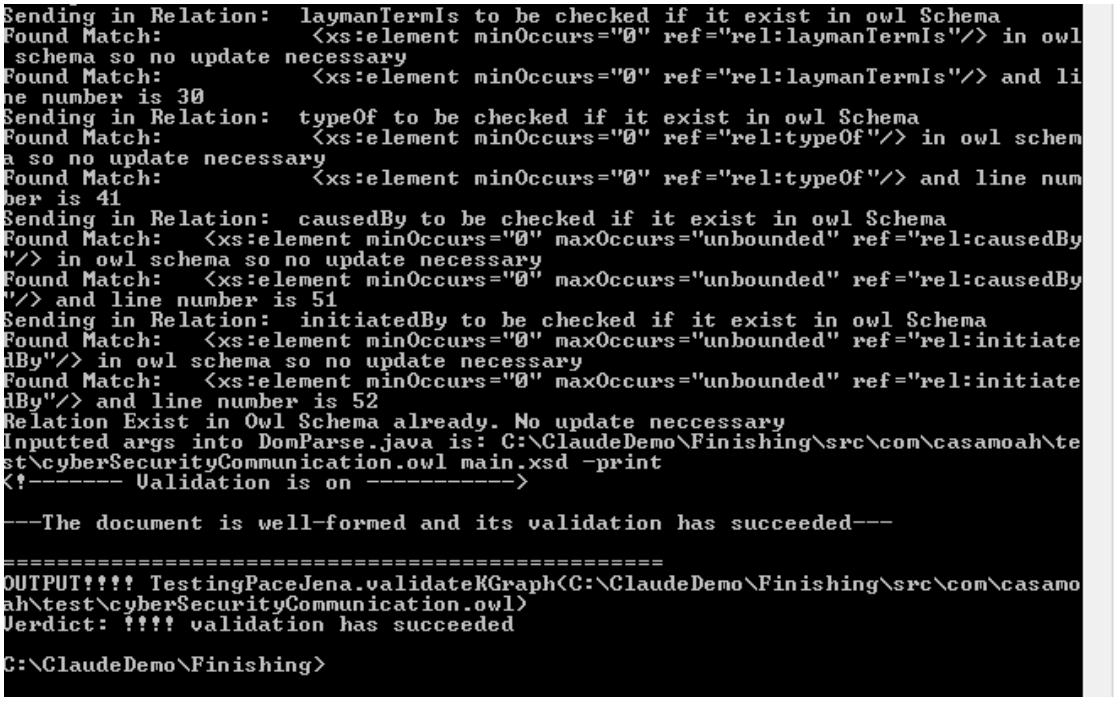


Figure 71: "cyberSecurityCommunication.owl" KG Syntax Validation Output

5.3 Web Based Knowledge Graph Navigation

The "WebBrowser" program takes as an input any KG document with custom relations and uses functions created in a revised Pace Jena to retrieve all classes and relations in the KG document and display them in an html page using a program that creates the class relations on the fly. This program presents to the user, the ability to navigate the classes and relations of the KG. The Web model implemented in "WebBrowser" program consists of Pace Jena and Owl Type java classes. Presented below are the screen shots of the application.

The application is launched from the command line using the test program and takes any KG document with custom relations as input. Note that the KG is syntax validated first

and if it passes, the visual navigation application is launched. Figure 72 shows the launch and portion of the KG being syntax validated.

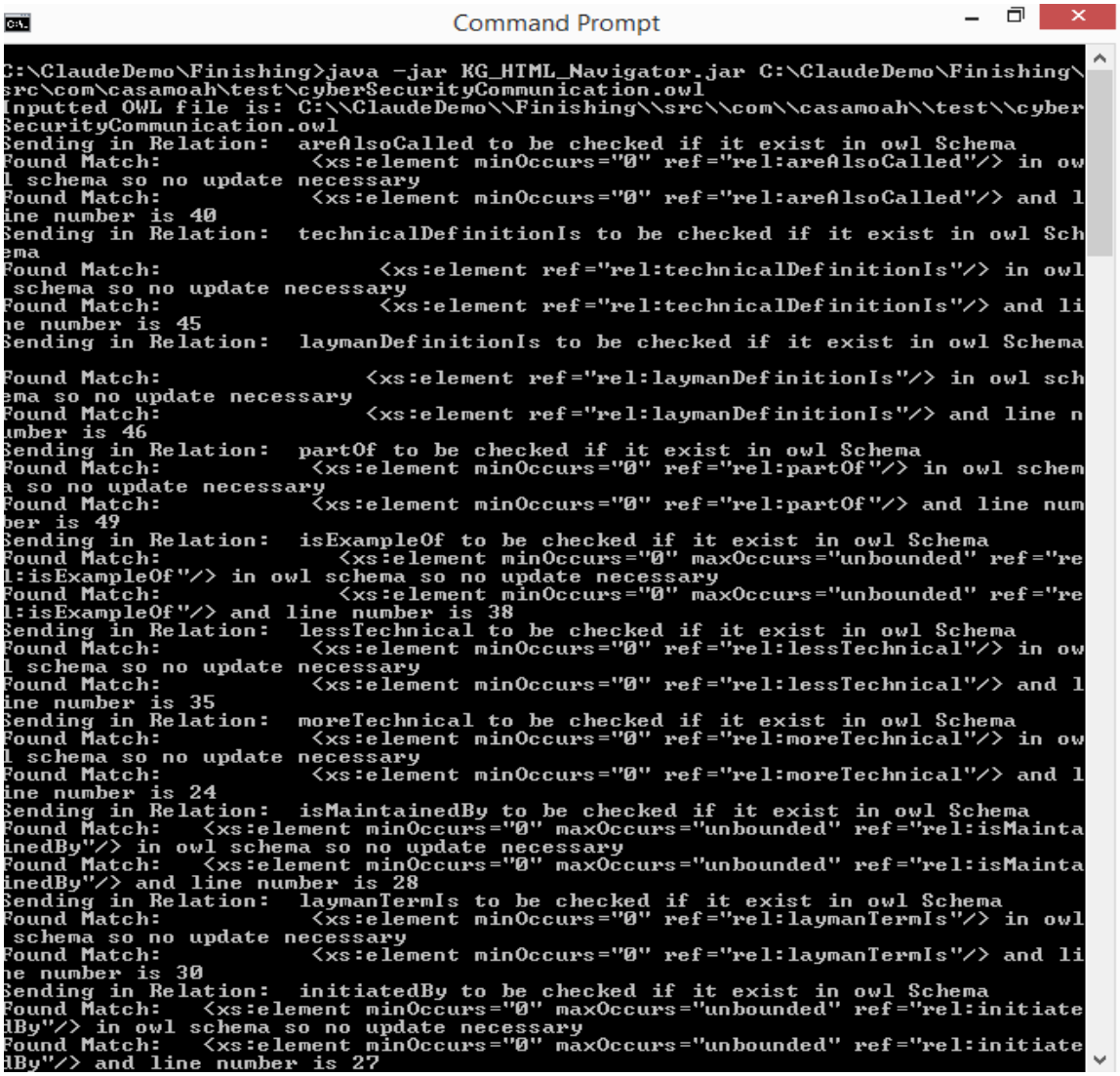


Figure 72: Launching the Application. Windows Command Prompt Example

When the program runs to completion, an output is presented as shown in Figure 73 below that the syntax validation has succeeded with instruction to copy the URL

provided and to paste it inside any browser. Note that the html model does not require any Internet connection.

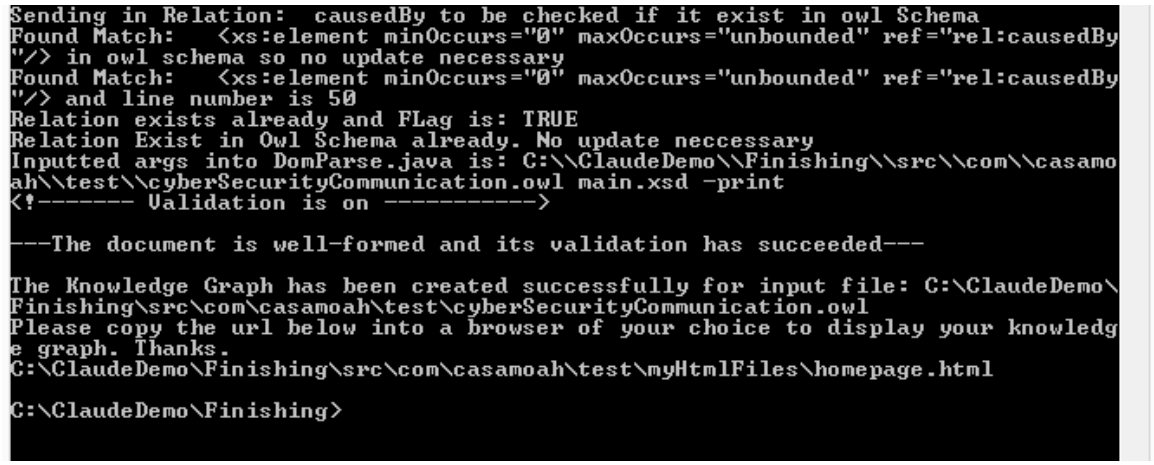


Figure 73: Program Output Presenting a URL that the User can Paste in any Browser Presented below in Figure 74 is the welcome page of the application.

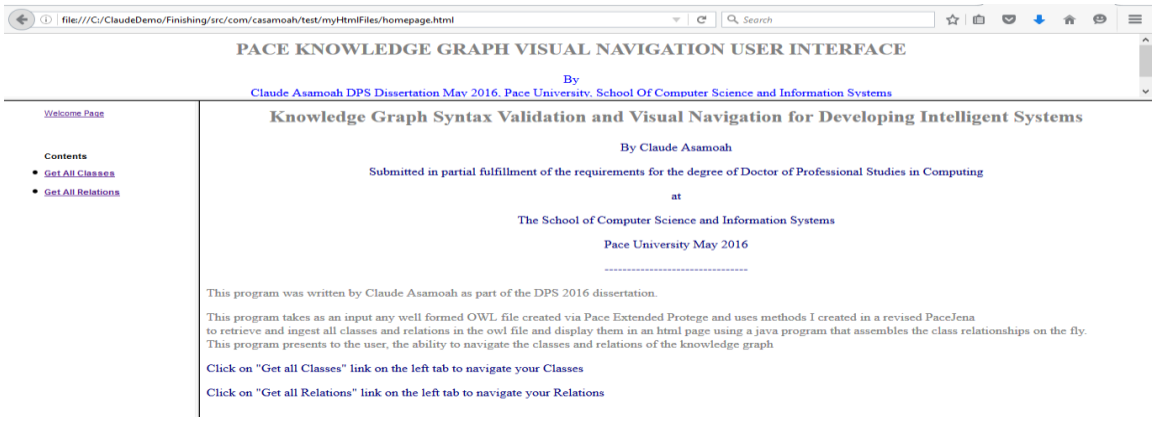


Figure 74: Knowledge Graph Prototype Welcome Page

When you click the “Get All Classes” link you will be presented with all classes retrieved from the KG as hyperlinks as depicted in Figure 75

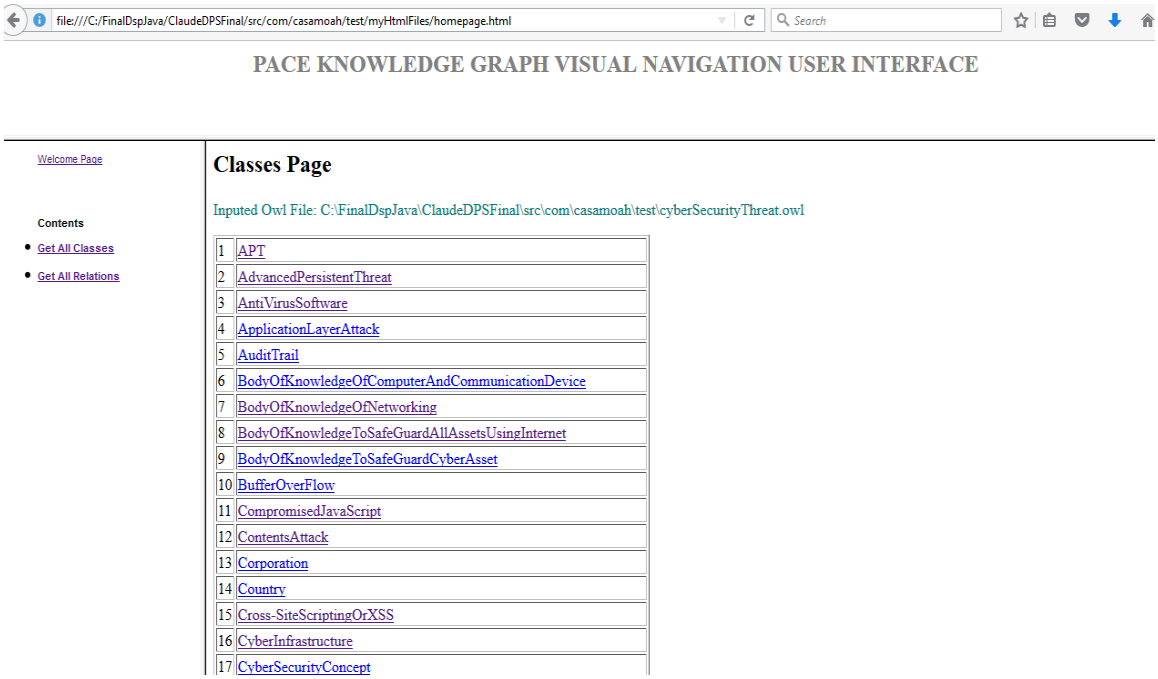


Figure 75: All Classes Page

When you click the link of any of the class, you will be presented with all links to other classes of the selected class showing you the “Triple” which is the subject, predicate, and object. Figure 76 displays “CyberSecurityConcept” relations. When you click any class link, it will display the other classes it is related to by a custom relation in the “Triple” format of subject, predicate, and object. Note that the predicate’s font color is red which is the custom relation embedded in the KG.

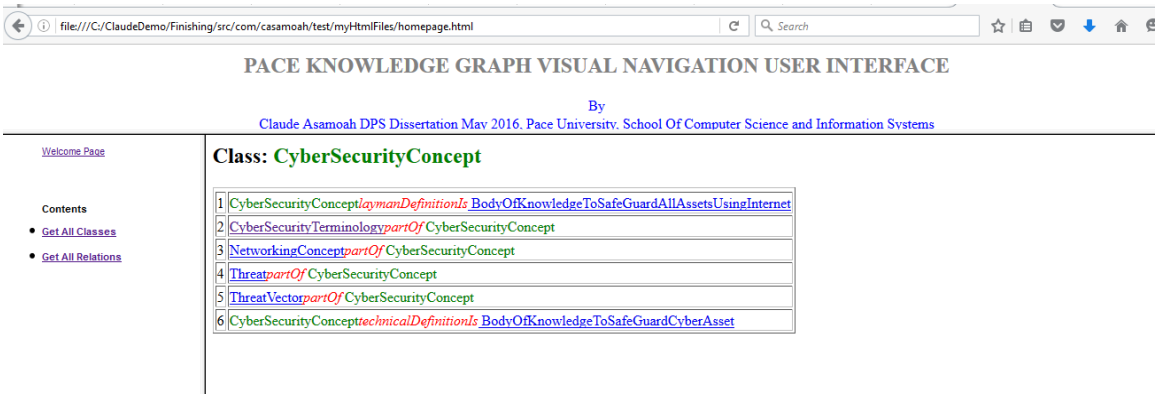


Figure 76: "CyberSecurityConcept" Class Relations Display Page

When you click the "Get All Relations" tab on the left banner of the application you will be presented with the Relations page that displays all relations as hyperlinks and when clicked, will display all the classes that use that relation to link two classes in the KG. Figure 77 shows the Relations Page.

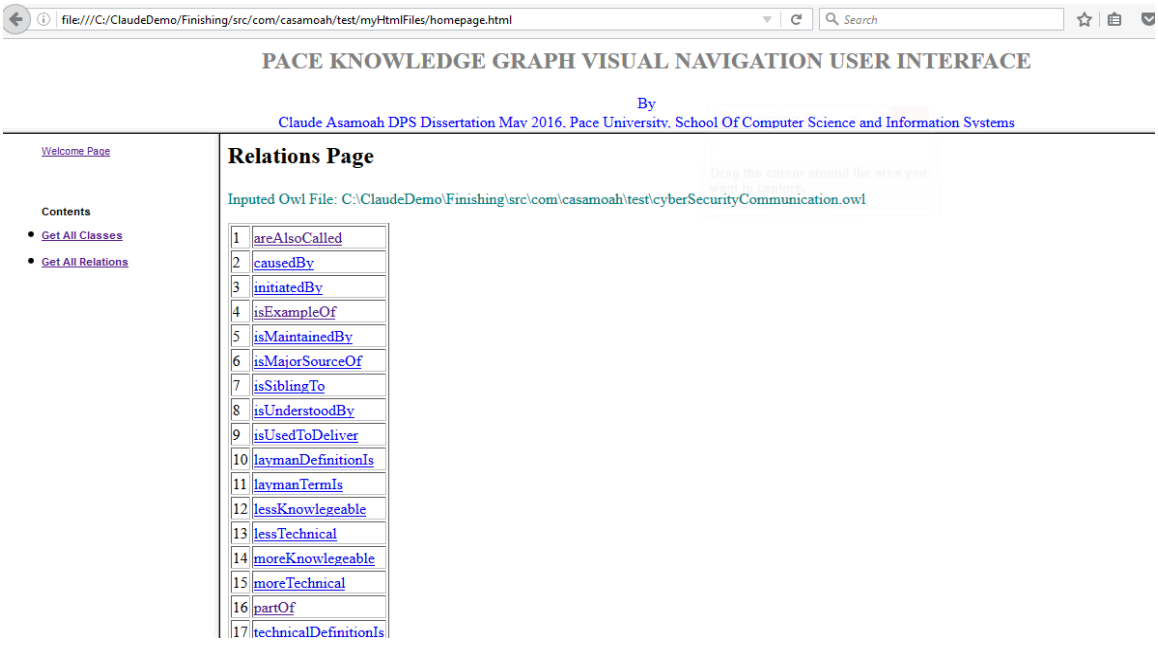


Figure 77: Relations Page

When you click any Relation for example the “partOf” relation link, it will display all classes that relate to other classes using the “partOf” custom relation. Figure 78 depicts the resulting page.

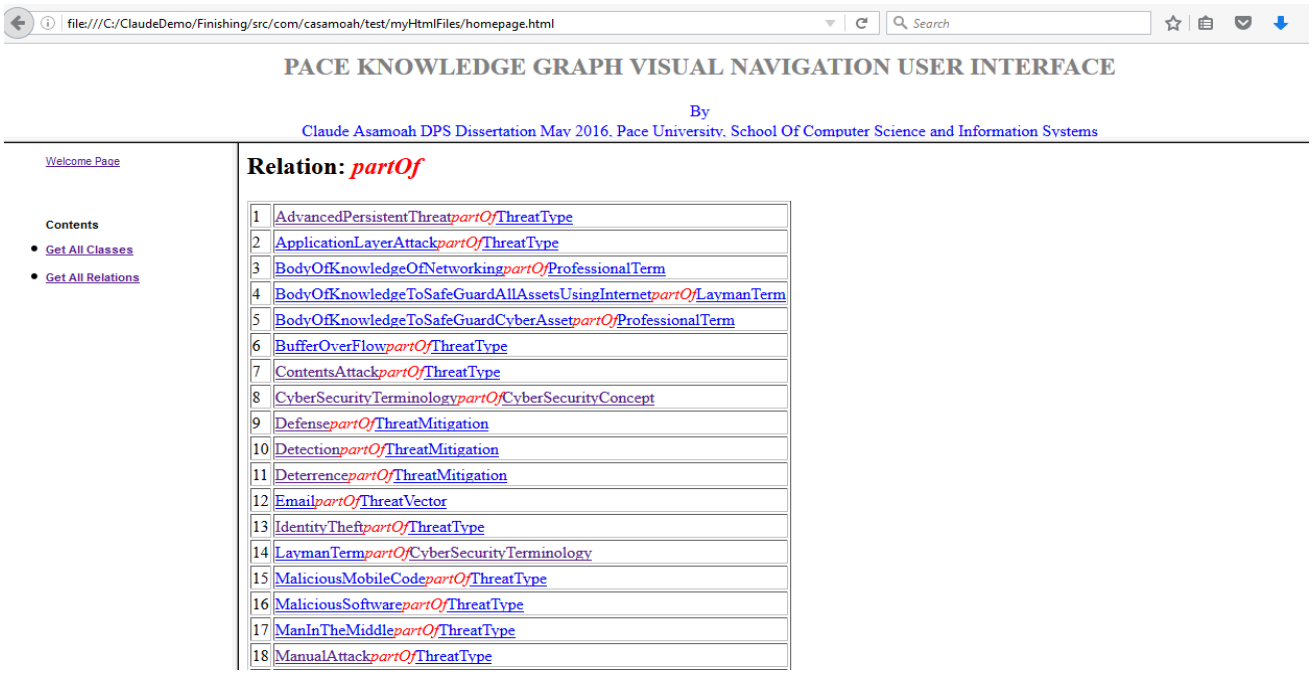


Figure 78: “partOf” Relation Page

The user may click any link of any class whether subject or object of the Triple to view any class relations. The Visual Navigation capability of the KG presents a useful means for any domain expert, developer, or student who is using KG in their research or application to be able to navigate the KG to ascertain the completeness of their KG. The visual navigation adds value to the KG and very convenient in viewing the KG in its entirety.

5.4 Application Based Knowledge Graph Navigation

The Application Model of the KG is a robust application that allows the user to browse for a KG owl document on their file system as input into the application in-order to navigate the KG. The application consists of a Pace Jena, “ClassesKnowledgeGraph”, “RelationsKnowledgeGraph”, “OwlTypes”, and “layout1” java classes. Figure 79 depicts the launching of the application by clicking the “KG_JavaSwing-model.jar” file.

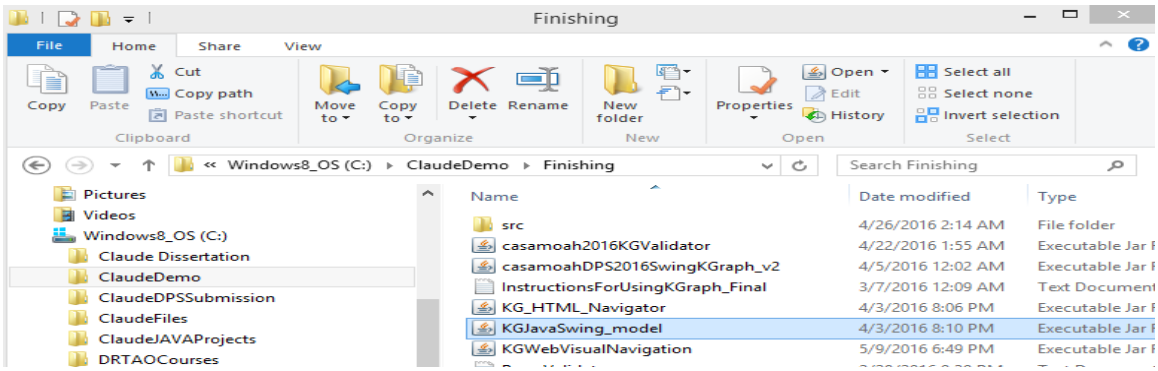


Figure 79: Clicking “KG_JavaSwing-model.jar” to launch the Application

Figure 80 below depicts the application when launched.

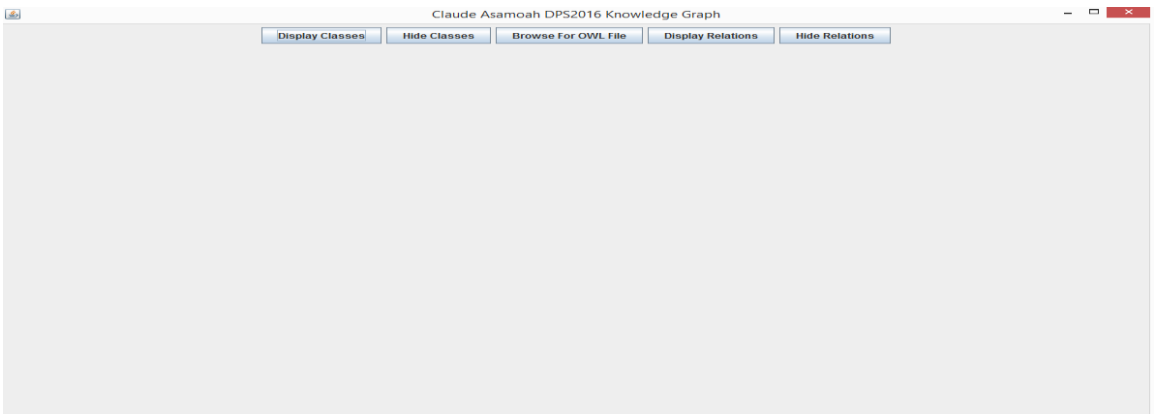


Figure 80: Java Swing Knowledge Graph

When the “Browse For OWL File” button is clicked, the user is presented with the File Explorer to browse and locate the KG document to be inputted into the application so that it can be navigated as shown in Figure 81.

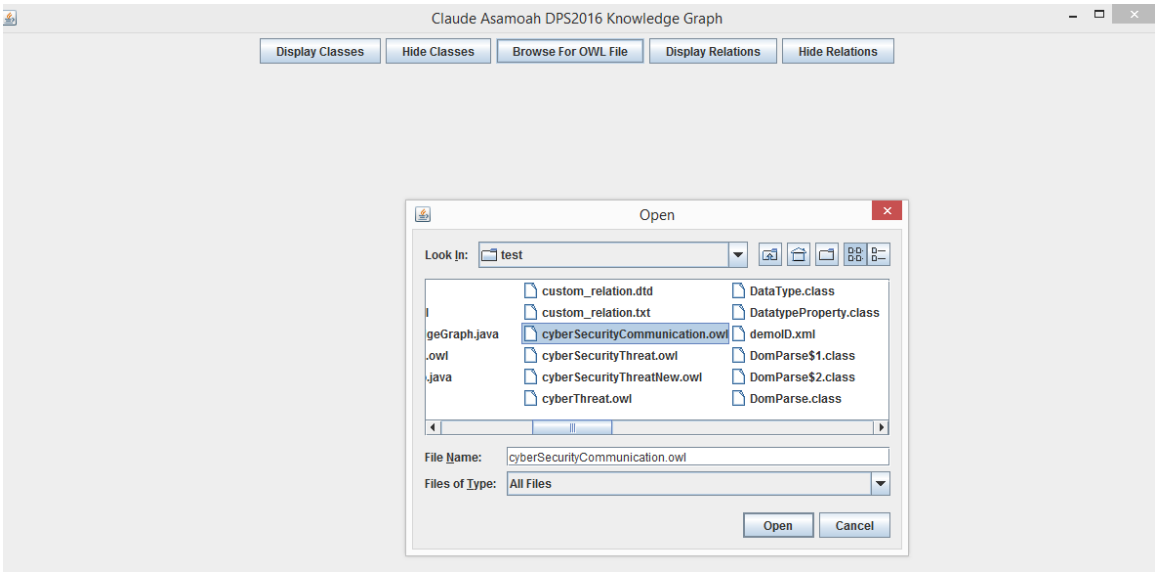


Figure 81: Browsing for an OWL Document to input into the KG Visual Navigation Application

Once the “Open” button is clicked, the KG is syntax validated and if the validation is successful, the application is loaded with the selected KG document otherwise you get a message box indicating that the validation failed and what type of failure and code line number and suggests a fix before being able to syntax validate the KG. When the “Display Classes” button is clicked, the application uses the default “All” of the dropdown list of the Combo Box to display all class relations initially as shown in Figure 82.

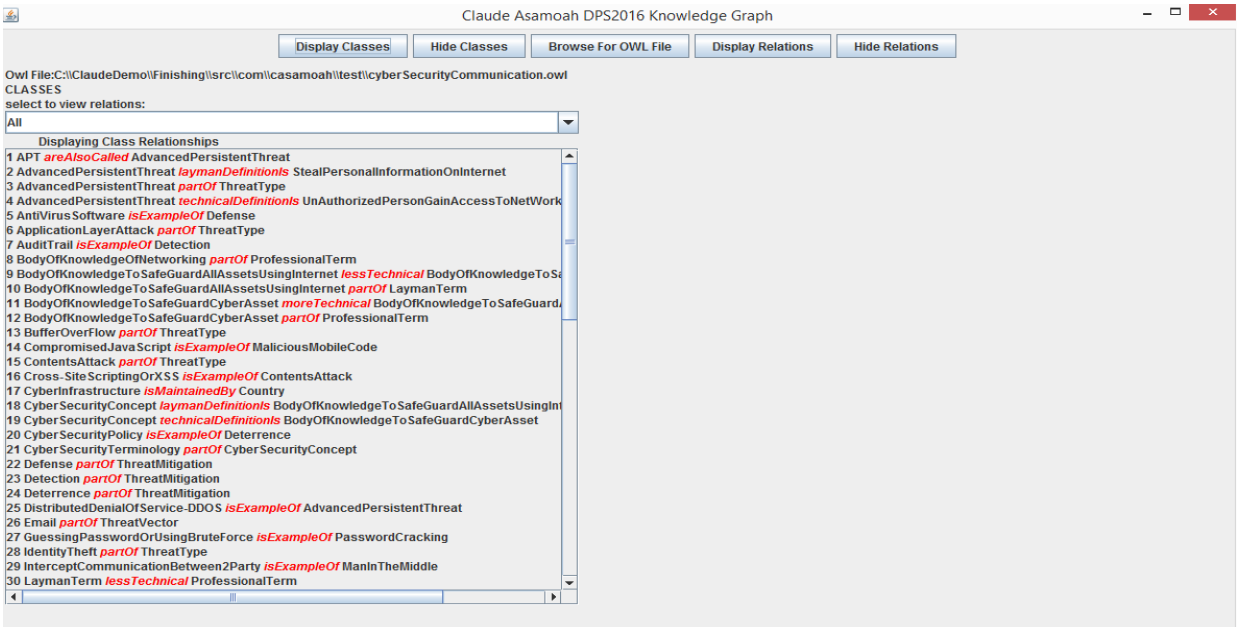


Figure 82: Displaying All Classes Relations of Knowledge Graph

When the "Display Relations" button is clicked, the application uses the default "All" of the dropdown list of the Combo Box to display all custom relations related classes as shown in Figure 83 below.

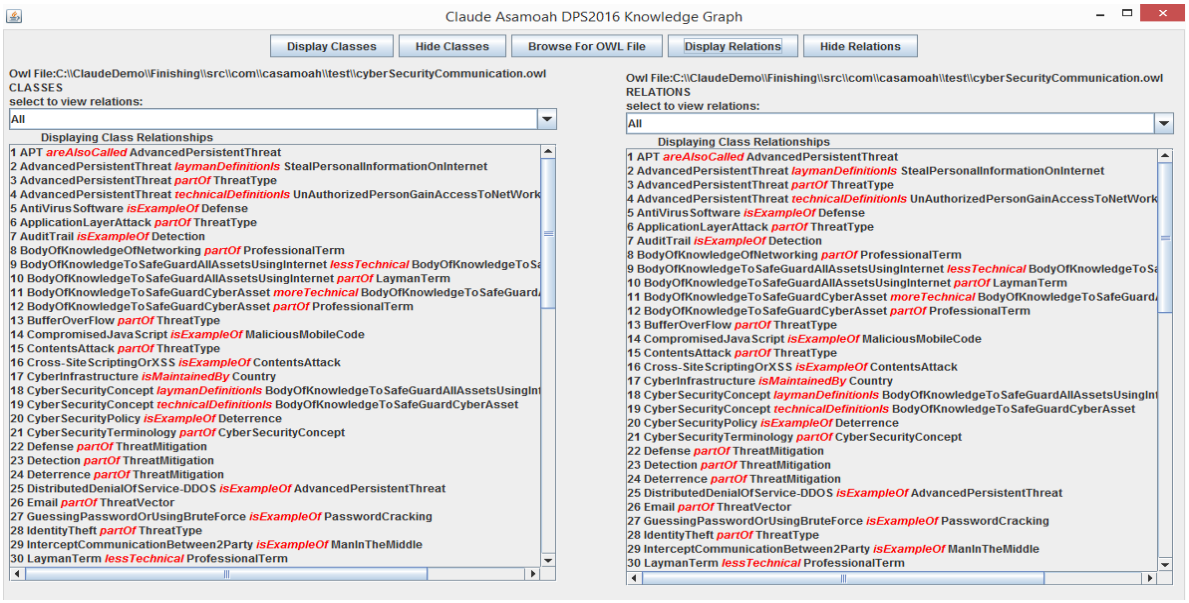


Figure 83: Displaying All Relations in Addition to All Classes

A user may choose to work on one display at a time by clicking the “Hide Relations” or “Hide Classes” buttons to hide one display window. The user can then select a class via the drop down list. Figure 84 shows the Relations Display hidden by clicking the “Hide Relations” button and selecting “ThreatType” from the “CLASSES” drop down list.

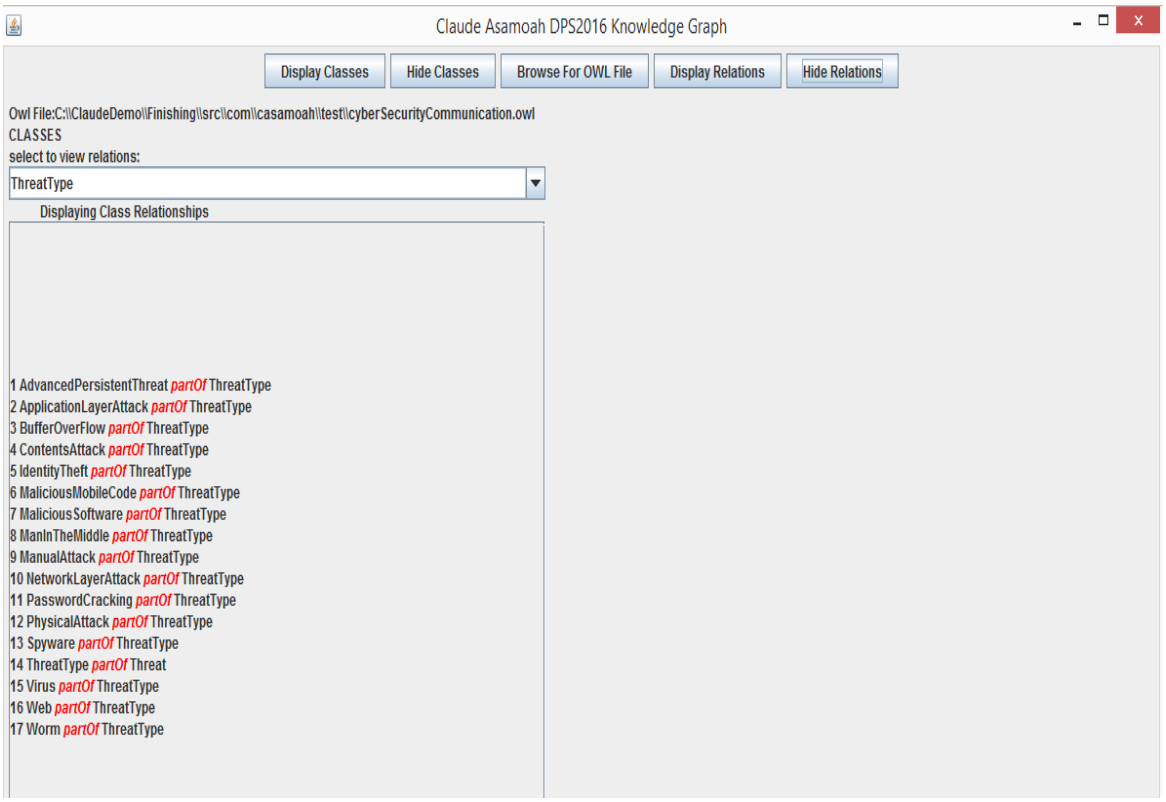


Figure 84: Threat Type Class Relations Display

Figure 85 shows the Classes display window hidden by clicking the “Hide Classes” button and displaying the “partOf” relation of all classes using that custom relation.

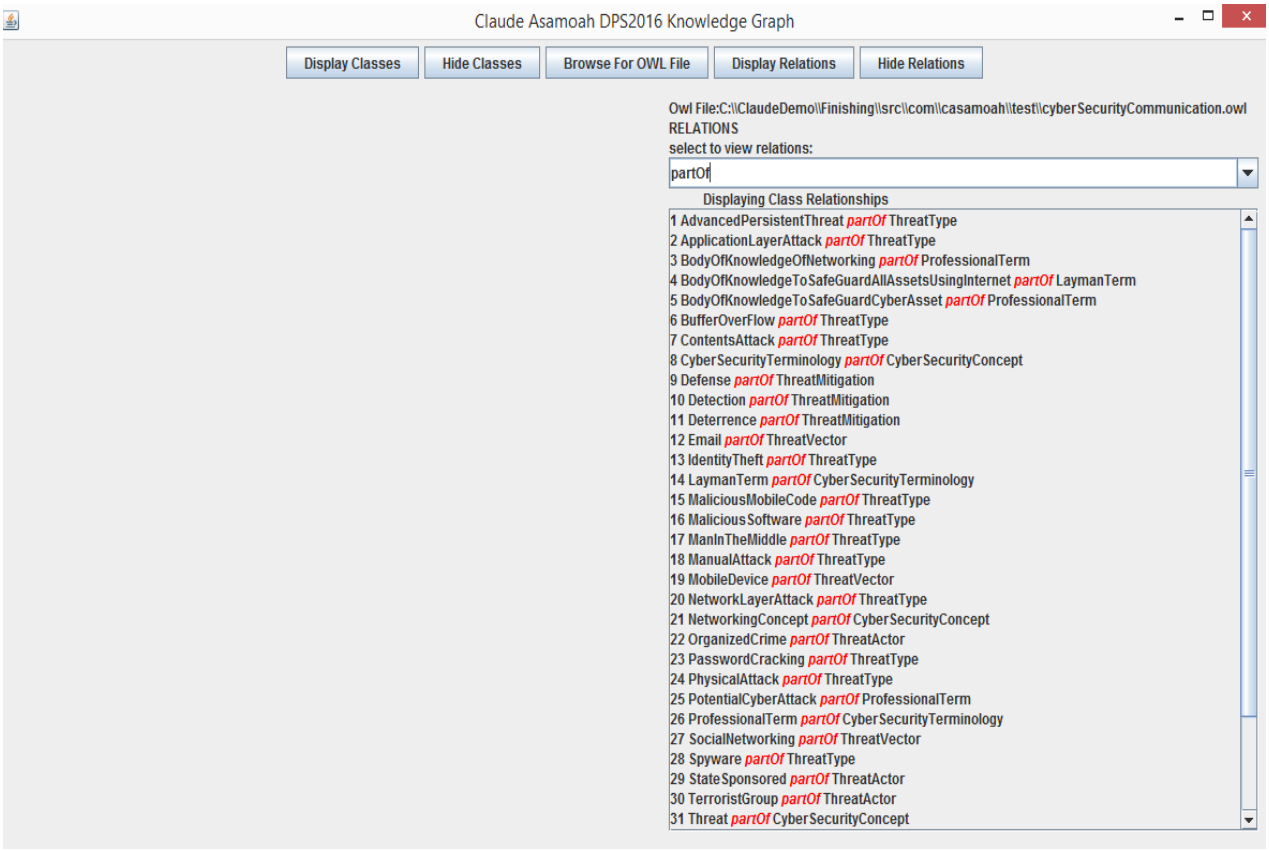


Figure 85: “partOf” Relations

5.5 Conclusion

In this chapter, it was successfully proven that a fairly large KG can be syntax validated and successfully visually navigated. The creation of a KG by a domain expert and the syntax validation and visual navigation of the KG by a domain expert was simulated using the cyber security communication use case. A KG of cyber security communication was designed by creating the hierarchical tree of the cyber security concept and deriving the class relations from them. The information gathered was then used to create a KG. The KG was syntax validated and visually navigated successfully using the knowledge graph syntax validation, and the knowledge graph visual navigation applications. It was

demonstrated how functions in Pace Jena are employed to provide syntax validation and visual navigation capabilities for both web and application models for a subset RDF/XML serialized document with custom relations also known as Knowledge Graph. The provision of syntax validation and visual navigation as demonstrated by the cyber security communication use case will enable domain experts to syntax validate their KGs to improve the correctness of knowledge encoding and to assist domain experts navigate, evaluate, and lookup the contents of their KG to support their research.

Chapter 6

Conclusion

6.1 Contributions Summary

Extending ontology to KG simplifies the relation of classes using custom relations as predicates. Custom relation provides flexibility in relating two individual classes by providing alternatives to the use of complex object and data properties to relate two individual classes. The KG needs to be syntax validated because domain experts (not IT experts) can easily introduce bugs, when designing KR systems using RDF/XML documents. Typical KR documents are large and complicated and document transmission can compromise documents requiring the need for syntax validation to safeguard the introduction of bugs. Also it is necessary to syntax validate a KG before it is inputted in the visual navigation mechanism to avoid potential unpredictability because the KG was not well-formed and OWL syntax validated. Validating the KG syntax-wise for the supported subset of RDF/XML for this research is a challenge in that the KG owl document contains custom relations which are not recognized by the W3C owl schema which validates only owl documents serialized in the OWL/XML format thus necessitating the creation of a custom schema that can syntax validate KGs serialized in the RDF/XML format and other supported non-custom relation RDF/XML documents. Another problem that needed to be surmounted was how to persist the six targeted Namespace in an XSD schema to validate a KG or a supported RDF/XML document. The six XSD schemas are the “main.xsd”, the “rdfs.xsd”, the “owl.xsd”, the “xml.xsd”,

the “pace.xsd”, and the “rel.xsd”. For the “main.xsd” schema to be able to syntax validate the KG supported by this research, the other namespaces need to persist in the main.xsd schema. However, an XSD schema can only have one target Namespace prompting the design of a scheme to import all other namespaces into each other for their namespaces to persist in the “main.xsd” schema. Once the namespace persistence problem was surmounted, the syntax validation of a supported KG challenge had to be solved. This was accomplished by designing and implementing the Multiple Pass Syntax Validation algorithm with Dom Parsing to declare any new custom relations in the Pace custom created schema before the KG is syntax validated. Once the KG is syntax validated, this dissertation also provided visual navigation capability for the KG by extending Pace Jena with functions to facilitate the creation of visual navigation applications created on the fly and provide the navigation of the KG in its entirety. The demonstration of this dissertation by using various types of custom relations to create a KG by extending OWL, providing syntax validation for the KG, and extending Pace Jena by creating functions that provides visual navigation capabilities for the KG has presented an alternate avenue to better support knowledge representation and decision-making and enable domain experts to be effective in designing their KGs, declare custom relations and use them in the same document, use custom relations directly and intuitively without using object property emulation, and declare and apply custom relations in IDEs such as Protégé.

6.2 Future Work

There is ample opportunity to continue to support syntax validations of other subsets of RDF/XML centric documents. Also KGs can be explored to support effective knowledge base decision-making. KGs can be explored as data sources for applications designed in the realms of Internet of Things (IoTs) and also can be used in the design of intelligent systems in the Cloud.

Pace Jena Methods

```
public ArrayList getRelations(){

     ArrayList<String> relRtn = new ArrayList<String>();

        for (String key: currentOntology.newRelationHash.keySet()) {
            Relation rel = currentOntology.newRelationHash.get(key);
            String id = rel.getIRI();


            String qName = id;
            relRtn.add(remove(qName).toString());



        }

         return relRtn;

}
```

Figure 86: “getRelations” function

```
public ArrayList getClassNames() {

      Enumeration<String> j = currentOntology.owlClassHash.keys();
      ArrayList<String> classRtn = new ArrayList<String>();

      while (j.hasMoreElements()) {

        OwlClass o = currentOntology.owlClassHash.get(j.nextElement());



        classRtn.add(remove(o.about).toString());


      }


      return classRtn;

   }
```

Figure 87: “getClassNames” function

```
public ArrayList getClassRelClass(String relation, String objectname) {

      Enumeration<String> j = currentOntology.owlClassHash.keys();
      ArrayList<String> classRtn = new ArrayList<String>();
      String classStr = " ";

      while (j.hasMoreElements()) {

        OwlClass o = currentOntology.owlClassHash.get(j.nextElement());

        for(Relation rel : o.relationsMap.keySet()){
            List<OwlClass> relatedClasses = o.relationsMap.get(rel);
            for(OwlClass b: relatedClasses)
             if(rel.getName() != null && remove(o.about) != null && remove(b.about) != null ){
                 classStr= "Class " + remove(o.about) +": "+ rel.getName() + "- " + remove(b.about);
            classStr= classStr.replace("Class", "");
            classStr= classStr.replace("- ", " ");
            classStr= classStr.replace(":", "");
            if(classStr.contains(relation) && classStr.contains(objectname) ){
             classRtn.add(remove( classStr).toString());
                }

             }
           }
        }

      return classRtn;
    }
```

Figure 88: “getClassRelClass” function

```
public String validateKGraph(String owlFile){
    String f2 = "src\\com\\casamoah\\test\\owl.xsd";
    String f3 = "src\\com\\casamoah\\test\\rel.xsd";
    String results = "";
    ValidateKnowledgeGraph myValidateKG = new ValidateKnowledgeGraph();
    String[] cmd = {owlFile,f2,f3};
    try{
        ValidateKnowledgeGraph.main(cmd);

        //now read validation results from ParseValidate.txt
        String sCurrentLine;

        String current_dir = new java.io.File( "." ).getCanonicalPath();
         BufferedReader br = null;
        current_dir = current_dir + "\\src\\com\\casamoah\\test\\";
         String verdictFilename = current_dir+"ParseValidate.txt";
        br = new BufferedReader(new FileReader(verdictFilename));

        while ((sCurrentLine = br.readLine()) != null) {
            results = sCurrentLine;
        }

    }
    catch (Exception e){

    }

    return results;
}
```

Figure 89 : “validateKGraph” function

Appendix "A KG Syntax Validation"

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
                  target Namespace="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                  xmlns:owl="http://www.w3.org/2002/07/owl#"
                  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
            xmlns:pace="http://csis.pace.edu/semweb#"
                  xmlns:rel="http://www.pace.edu/rel-syntax-ns#">
 <xs:import namespace="http://csis.pace.edu/semweb#" schemaLocation="pace.xsd"/>
  <xs:import namespace="http://www.pace.edu/rel-syntax-ns#" schemaLocation="rel.xsd"/>
  <xs:import namespace="http://www.w3.org/2000/01/rdf-schema#"
schemaLocation="rdfs.xsd"/>
  <xs:import namespace="http://www.w3.org/2002/07/owl#" schemaLocation="owl.xsd"/>
  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://www.w3.org/2001/xml.xsd"/>
  <xs:element name="RDF">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="owl:Ontology"/>
        <xs:element maxOccurs="unbounded" ref="rel:NewRelation"/>
        <xs:element maxOccurs="unbounded" ref="owl:Class"/>
      </xs:sequence>
      <xs:attribute ref="xml:base" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:attribute name="about" type="xs:anyURI"/>
  <xs:attribute name="resource" type="xs:anyURI"/>
</xs:schema>
```

Figure 90: "main.xsd" Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
                  target Namespace="http://www.w3.org/2000/01/rdf-schema#"
                  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                  xmlns:owl="http://www.w3.org/2002/07/owl#"
                  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
                  xmlns:pace="http://csis.pace.edu/semweb#"
 xmlns:rel="http://www.pace.edu/rel-syntax-ns#">
 <xs:import namespace="http://csis.pace.edu/semweb#" schemaLocation="pace.xsd"/>
  <xs:import namespace="http://www.pace.edu/rel-syntax-ns#" schemaLocation="rel.xsd"/>
  <xs:import namespace="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
schemaLocation="main.xsd"/>
  <xs:import namespace="http://www.w3.org/2002/07/owl#" schemaLocation="owl.xsd"/>
  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://www.w3.org/2001/xml.xsd"/>

  <xs:element name="subClassOf">
    <xs:complexType>
      <xs:attribute ref="rdf:resource" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 91: "rdfs.xsd" Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
target Namespace="http://csis.pace.edu/semweb#"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:pace="http://csis.pace.edu/semweb#"
 xmlns:owl="http://www.w3.org/2002/07/owl#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:rel="http://www.pace.edu/rel-syntax-ns#">
  <xs:import namespace="http://www.pace.edu/rel-syntax-ns#" schemaLocation="rel.xsd"/>
  <xs:import namespace="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
schemaLocation="main.xsd"/>
  <xs:import namespace="http://www.w3.org/2000/01/rdf-schema#"
schemaLocation="rdfs.xsd"/>
  <xs:import namespace="http://www.w3.org/2002/07/owl#" schemaLocation="owl.xsd"/>
  <xs:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="xml.xsd"/>
  <xs:element name="level" type="xs:integer"/>
  <xs:element name="name" type="xs:NCName"/>
  <xs:element name="ref">
    <xs:complexType>
      <xs:attribute ref="rdf:resource" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 92: “pace.xsd” Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
target Namespace="http://www.w3.org/XML/1998/namespace"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:pace="http://csis.pace.edu/semweb#" xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:rel="http://www.pace.edu/rel-syntax-ns#">
  <xs:import namespace="http://csis.pace.edu/semweb#" schemaLocation="pace.xsd"/>
  <xs:import namespace="http://www.pace.edu/rel-syntax-ns#" schemaLocation="rel.xsd"/>
  <xs:import namespace="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
schemaLocation="main.xsd"/>
  <xs:import namespace="http://www.w3.org/2000/01/rdf-schema#"
schemaLocation="rdfs.xsd"/>
  <xs:import namespace="http://www.w3.org/2002/07/owl#" schemaLocation="owl.xsd"/>
  <xs:attribute name="base" type="xs:anyURI"/>
</xs:schema>
```

Figure 93: “xml.xsd” Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
                  targetNamespace="http://www.w3.org/2002/07/owl#"
                  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
                  xmlns:owl="http://www.w3.org/2002/07/owl#"
                  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
                  xmlns:rel="http://www.pace.edu/rel-syntax-ns#"
                   xmlns:pace="http://csis.pace.edu/semweb#">

  <xs:import namespace="http://csis.pace.edu/semweb#" schemaLocation="pace.xsd"/>
  <xs:import namespace="http://www.pace.edu/rel-syntax-ns#" schemaLocation="rel.xsd"/>
  <xs:import namespace="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
schemaLocation="main.xsd"/>
  <xs:import namespace="http://www.w3.org/2000/01/rdf-schema#"
schemaLocation="rdfs.xsd"/>
  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://www.w3.org/2001/xml.xsd"/>
  <xs:element name="Ontology">
    <xs:complexType>
      <xs:attribute ref="rdf:about" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Class">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" ref="rel:isMajorSourceOf"/>
        <xs:element minOccurs="0" ref="rel:moreTechnical"/>
        <xs:element minOccurs="0" ref="rel:isUnderstoodBy"/>
        <xs:element minOccurs="0" ref="rdfs:subClassOf"/>
<xs:element minOccurs="0" maxOccurs="unbounded" ref="rel:initiatedBy"/>
<xs:element minOccurs="0" maxOccurs="unbounded" ref="rel:isMaintainedBy"/>
        <xs:element minOccurs="0" ref="rel:laymanTermIs"/>
        <xs:choice minOccurs="0">
        </xs:choice>
        <xs:element minOccurs="0" ref="rel:lessTechnical"/>
        <xs:choice>
          <xs:element ref="rel:isUsedToDeliver"/>
          <xs:element minOccurs="0" maxOccurs="unbounded" ref="rel:isExampleOf"/>
        </xs:choice>
        <xs:element minOccurs="0" ref="rel:areAlsoCalled"/>
        <xs:element minOccurs="0" ref="rel:typeOf"/>
        <xs:choice>
          <xs:element minOccurs="0" maxOccurs="unbounded" ref="rel:isSiblingTo"/>
          <xs:sequence>
                  <xs:element ref="rel:technicalDefinitionIs"/>
            <xs:element ref="rel:laymanDefinitionIs"/>
          </xs:sequence>
        </xs:choice>
        <xs:element minOccurs="0" ref="rel:partOf"/>
<xs:element minOccurs="0" maxOccurs="unbounded" ref="rel:causedBy"/>
      </xs:sequence>
      <xs:attribute ref="rdf:about" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 94: “owl.xsd” Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
                  target Namespace="http://www.pace.edu/rel-syntax-ns#"
                  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                  xmlns:owl="http://www.w3.org/2002/07/owl#"
                  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  <xs:import namespace="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
schemaLocation="main.xsd"/>
  <xs:import namespace="http://www.w3.org/2000/01/rdf-schema#"
schemaLocation="rdfs.xsd"/>
  <xs:import namespace="http://www.w3.org/2002/07/owl#" schemaLocation="owl.xsd"/>
```

```
  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://www.w3.org/2001/xml.xsd"/>
  <xs:element name="NewRelation">
    <xs:complexType>
      <xs:attribute ref="rdf:about" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="isMajorSourceOf">
    <xs:complexType>
      <xs:attribute ref="rdf:resource" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="moreTechnical">
    <xs:complexType>
      <xs:attribute ref="rdf:resource" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="isUnderstoodBy">
    <xs:complexType>
      <xs:attribute ref="rdf:resource" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="has">
    <xs:complexType>
      <xs:attribute ref="rdf:resource" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="laymanTermIs">
    <xs:complexType>
      <xs:attribute ref="rdf:resource" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="lessTechnical">
    <xs:complexType>
      <xs:attribute ref="rdf:resource" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="isUsedToDeliver">
    <xs:complexType>
      <xs:attribute ref="rdf:resource" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="isExampleOf">
    <xs:complexType>
      <xs:attribute ref="rdf:resource" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="areAlsoCalled">
    <xs:complexType>
      <xs:attribute ref="rdf:resource" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="typeOf">
    <xs:complexType>
      <xs:attribute ref="rdf:resource" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="isSiblingTo">
    <xs:complexType>
      <xs:attribute ref="rdf:resource" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="laymanDefinitionIs">
    <xs:complexType>
      <xs:attribute ref="rdf:resource" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="technicalDefinitionIs">
    <xs:complexType>
      <xs:attribute ref="rdf:resource" use="required"/>
    </xs:complexType>
```

```
  </xs:element>
  <xs:element name="partOf">
    <xs:complexType>
      <xs:attribute ref="rdf:resource" use="required"/>
    </xs:complexType>
  </xs:element>
<xs:element name="isAmodelOf">
<xs:complexType>
<xs:attribute ref="rdf:resource" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="continentFor">
<xs:complexType>
<xs:attribute ref="rdf:resource" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="locatedIn">
<xs:complexType>
<xs:attribute ref="rdf:resource" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="capitalOf">
<xs:complexType>
<xs:attribute ref="rdf:resource" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="include">
<xs:complexType>
<xs:attribute ref="rdf:resource" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="areHandled">
<xs:complexType>
<xs:attribute ref="rdf:resource" use="required"/>
</xs:complexType>
</xs:element>
</xs:schema>
```

Figure 95: “rel.xsd” Schema

Appendix B “KG Documents used in This Dissertation”

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rel "http://www.pace.edu/rel-syntax-ns#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>
<rdf:RDF xmlns="http://www.pace.edu/country-72#"
     xml:base="http://www.pace.edu/country-72"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rel="http://www.pace.edu/rel-syntax-ns#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Ontology rdf:about="http://www.pace.edu/country-72"/>
    <!--
    // Relations
     -->
<rel:NewRelation rdf:about="http://www.pace.edu/country-72#locatedIn"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/country-72#isSpokenIn"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/country-72#capitalOf"/>
    <!--
    // Classes
     -->
    <!-- http://www.pace.edu/country-72#America -->

    <owl:Class rdf:about="http://www.pace.edu/country-72#America">
        <rel:locatedIn rdf:resource="http://www.pace.edu/country-72#North_America"/>
    </owl:Class>
    <!-- http://www.pace.edu/country-72#Angentina -->

    <owl:Class rdf:about="http://www.pace.edu/country-72#Angentina">
        <rel:locatedIn rdf:resource="http://www.pace.edu/country-72#South_America"/>
    </owl:Class>
    <!-- http://www.pace.edu/country-72#Buenos_Aries -->

    <owl:Class rdf:about="http://www.pace.edu/country-72#Buenos_Aries">
        <rel:capitalOf rdf:resource="http://www.pace.edu/country-72#Angentina"/>
    </owl:Class>
    <!-- http://www.pace.edu/country-72#English -->

    <owl:Class rdf:about="http://www.pace.edu/country-72#English">
        <rel:isSpokenIn rdf:resource="http://www.pace.edu/country-72#America"/>
    </owl:Class>
```

Figure 96: “country.owl” KG

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
```

```
    <!ENTITY dc "http://purl.org/dc/elements/1.1/" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rel "http://www.pace.edu/rel-syntax-ns#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>
<rdf:RDF xmlns="http://www.pace.edu/cyber#"
     xml:base="http://www.pace.edu/cyberSecurity"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rel="http://www.pace.edu/rel-syntax-ns#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:dc="http://purl.org/dc/elements/1.1/">
    <owl:Ontology rdf:about="http://www.pace.edu/cyber#"/>

    <!--
    // Relations
     -->

        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#partOf"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#areAlsoCalled"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#laymanTermIs"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#moreTechnical"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#lessTechnical"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#causedBy"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#laymanDefinitionIs"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#isMaintainedBy"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#isMajorSourceOf"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#isSiblingTo"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#moreKnowlegeable"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#lessKnowlegeable"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#technicalDefinitionIs"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#isExampleOf"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#isUnderstoodBy"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#isUsedToDeliver"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#typeOf"/>
        <rel:NewRelation rdf:about="http://www.pace.edu/cyber#initiatedBy"/>

    <!--
    // Classes
     -->
    <!-- http://www.pace.edu/cyber#APT -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#APT">
        <rel:areAlsoCalled
rdf:resource="http://www.pace.edu/cyber#AdvancedPersistentThreat"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#AdvancedPersistentThreat -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#AdvancedPersistentThreat">
        <rel:technicalDefinitionIs
rdf:resource="http://www.pace.edu/cyber#UnAuthorizedPersonGainAccessToNetWorkToStealData"
/>
        <rel:laymanDefinitionIs
rdf:resource="http://www.pace.edu/cyber#StealPersonalInformationOnInternet"/>
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatType"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#AntiVirusSoftware -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#AntiVirusSoftware">
        <rel:isExampleOf rdf:resource="http://www.pace.edu/cyber#Defense"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#ApplicationLayerAttack -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#ApplicationLayerAttack">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatType"/>
    </owl:Class>
```

```
    <!-- http://www.pace.edu/cyber#AuditTrail -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#AuditTrail">
        <rel:isExampleOf rdf:resource="http://www.pace.edu/cyber#Detection"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#BodyOfKnowledgeOfComputerAndCommunicationDevice -->

    <owl:Class
rdf:about="http://www.pace.edu/cyber#BodyOfKnowledgeOfComputerAndCommunicationDevice">
    </owl:Class>

    <!-- http://www.pace.edu/cyber#BodyOfKnowledgeOfNetworking -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#BodyOfKnowledgeOfNetworking">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ProfessionalTerm"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#BodyOfKnowledgeToSafeGuardAllAssetsUsingInternet -->

    <owl:Class
rdf:about="http://www.pace.edu/cyber#BodyOfKnowledgeToSafeGuardAllAssetsUsingInternet">
        <rel:lessTechnical
rdf:resource="http://www.pace.edu/cyber#BodyOfKnowledgeToSafeGuardCyberAsset"/>
                    <rel:partOf rdf:resource="http://www.pace.edu/cyber#LaymanTerm"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#BodyOfKnowledgeToSafeGuardCyberAsset -->

    <owl:Class
rdf:about="http://www.pace.edu/cyber#BodyOfKnowledgeToSafeGuardCyberAsset">
        <rel:moreTechnical
rdf:resource="http://www.pace.edu/cyber#BodyOfKnowledgeToSafeGuardAllAssetsUsingInternet"
/>
                <rel:partOf rdf:resource="http://www.pace.edu/cyber#ProfessionalTerm"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#BufferOverFlow -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#BufferOverFlow">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatType"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#CompromisedJavaScript -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#CompromisedJavaScript">
        <rel:isExampleOf rdf:resource="http://www.pace.edu/cyber#MaliciousMobileCode"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#ContentsAttack -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#ContentsAttack">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatType"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#Corporation -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Corporation">
    </owl:Class>

    <!-- http://www.pace.edu/cyber#Country -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Country">
    </owl:Class>

    <!-- http://www.pace.edu/cyber#Cross-SiteScriptingOrXSS -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Cross-SiteScriptingOrXSS">
        <rel:isExampleOf rdf:resource="http://www.pace.edu/cyber#ContentsAttack"/>
    </owl:Class>
```

```
    <!-- http://www.pace.edu/cyber#CyberInfrastructure -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#CyberInfrastructure">
        <rel:isMaintainedBy rdf:resource="http://www.pace.edu/cyber#Corporation"/>
        <rel:isMaintainedBy rdf:resource="http://www.pace.edu/cyber#Country"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#CyberSecurityConcept -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#CyberSecurityConcept">
        <rel:technicalDefinitionIs
rdf:resource="http://www.pace.edu/cyber#BodyOfKnowledgeToSafeGuardCyberAsset"/>
        <rel:laymanDefinitionIs
rdf:resource="http://www.pace.edu/cyber#BodyOfKnowledgeToSafeGuardAllAssetsUsingInternet"
/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#CyberSecurityPolicy -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#CyberSecurityPolicy">
        <rel:isExampleOf rdf:resource="http://www.pace.edu/cyber#Deterrence"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#CyberSecurityTerminology -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#CyberSecurityTerminology">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#CyberSecurityConcept"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#Defense -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Defense">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatMitigation"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#Detection -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Detection">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatMitigation"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#Deterrence -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Deterrence">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatMitigation"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#DistributedDenialOfService-DDOS -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#DistributedDenialOfService-DDOS">
        <rel:isExampleOf
rdf:resource="http://www.pace.edu/cyber#AdvancedPersistentThreat"/>
    </owl:Class>


    <!-- http://www.pace.edu/cyber#Email -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Email">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatVector"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#GuessingPasswordOrUsingBruteForce -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#GuessingPasswordOrUsingBruteForce">
        <rel:isExampleOf rdf:resource="http://www.pace.edu/cyber#PasswordCracking"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#Hacker -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Hacker">
```

```
    </owl:Class>

    <!-- http://www.pace.edu/cyber#Hijacked -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Hijacked">
    </owl:Class>

    <!-- http://www.pace.edu/cyber#IdentityTheft -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#IdentityTheft">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatType"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#InterceptCommunicationBetween2Party -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#InterceptCommunicationBetween2Party">
        <rel:isExampleOf rdf:resource="http://www.pace.edu/cyber#ManInTheMiddle"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#Layman -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Layman">
    </owl:Class>

    <!-- http://www.pace.edu/cyber#LaymanTerm -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#LaymanTerm">
        <rel:lessTechnical rdf:resource="http://www.pace.edu/cyber#ProfessionalTerm"/>
              <rel:partOf
rdf:resource="http://www.pace.edu/cyber#CyberSecurityTerminology"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#MaliciousMobileCode -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#MaliciousMobileCode">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatType"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#MaliciousSoftware -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#MaliciousSoftware">
        <rel:areAlsoCalled rdf:resource="http://www.pace.edu/cyber#Malware"/>
                <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatType"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#Malware -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Malware">
        <rdfs:subClassOf rdf:resource="http://www.pace.edu/cyber#MaliciousMobileCode"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#ManInTheMiddle -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#ManInTheMiddle">
        <rel:laymanTermIs rdf:resource="http://www.pace.edu/cyber#Hijacked"/>
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatType"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#ManualAttack -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#ManualAttack">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatType"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#MobileDevice -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#MobileDevice">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatVector"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#NetworkLayerAttack -->
```

```
    <owl:Class rdf:about="http://www.pace.edu/cyber#NetworkLayerAttack">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatType"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#NetworkingConcept -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#NetworkingConcept">
        <rel:technicalDefinitionIs
rdf:resource="http://www.pace.edu/cyber#BodyOfKnowledgeOfNetworking"/>
        <rel:laymanDefinitionIs
rdf:resource="http://www.pace.edu/cyber#BodyOfKnowledgeOfComputerAndCommunicationDevice"/
>
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#CyberSecurityConcept"/>
       </owl:Class>

    <!-- http://www.pace.edu/cyber#OrganizedCrime -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#OrganizedCrime">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatActor"/>
    </owl:Class>



    <!-- http://www.pace.edu/cyber#Packed.Generic.236 -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Packed.Generic.236">
        <rel:isExampleOf rdf:resource="http://www.pace.edu/cyber#Worm"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#PacketSniffing -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#PacketSniffing">
        <rel:isExampleOf rdf:resource="http://www.pace.edu/cyber#NetworkLayerAttack"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#PasswordCracking -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#PasswordCracking">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatType"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#PenDrive -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#PenDrive">
    </owl:Class>



    <!-- http://www.pace.edu/cyber#People -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#People">
    </owl:Class>

    <!-- http://www.pace.edu/cyber#PhysicalAccessToBankPIN -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#PhysicalAccessToBankPIN">
        <rel:isExampleOf rdf:resource="http://www.pace.edu/cyber#PhysicalAttack"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#PhysicalAttack -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#PhysicalAttack">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatType"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#PointOfSaleMalwareKnownAsCodeRed -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#PointOfSaleMalwareKnownAsCodeRed">
        <rel:isExampleOf rdf:resource="http://www.pace.edu/cyber#Malware"/>
    </owl:Class>
```

```
    <!-- http://www.pace.edu/cyber#PortScanningAnIPAddress -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#PortScanningAnIPAddress">
        <rel:isExampleOf rdf:resource="http://www.pace.edu/cyber#ManualAttack"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#PotentialCyberAttack -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#PotentialCyberAttack">
        <rel:moreTechnical
rdf:resource="http://www.pace.edu/cyber#PotentialInternetOrNetworkingRelatedAttack"/>
     <rel:partOf rdf:resource="http://www.pace.edu/cyber#ProfessionalTerm"/>
       </owl:Class>

    <!-- http://www.pace.edu/cyber#PotentialInternetOrNetworkingRelatedAttack -->

    <owl:Class
rdf:about="http://www.pace.edu/cyber#PotentialInternetOrNetworkingRelatedAttack">
        <rel:lessTechnical
rdf:resource="http://www.pace.edu/cyber#PotentialCyberAttack"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#Professional -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Professional">
    </owl:Class>

    <!-- http://www.pace.edu/cyber#ProfessionalTerm -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#ProfessionalTerm">
        <rel:moreTechnical rdf:resource="http://www.pace.edu/cyber#LaymanTerm"/>
                <rel:partOf
rdf:resource="http://www.pace.edu/cyber#CyberSecurityTerminology"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#SocialNetworking -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#SocialNetworking">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatVector"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#Spyware -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Spyware">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatType"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#StateSponsored -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#StateSponsored">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatActor"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#StealPersonalInformationOnInternet -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#StealPersonalInformationOnInternet">
    </owl:Class>

    <!-- http://www.pace.edu/cyber#StealingAndUsingSomeOnesCreditCard -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#StealingAndUsingSomeOnesCreditCard">
        <rel:isExampleOf rdf:resource="http://www.pace.edu/cyber#IdentityTheft"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#TerroristGroup -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#TerroristGroup">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatActor"/>
    </owl:Class>
```

```
    <!-- http://www.pace.edu/cyber#Threat -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Threat">
         <rel:initiatedBy rdf:resource="http://www.pace.edu/cyber#ThreatActor"/>
                <rel:technicalDefinitionIs
rdf:resource="http://www.pace.edu/cyber#PotentialCyberAttack"/>
               <rel:laymanDefinitionIs
rdf:resource="http://www.pace.edu/cyber#PotentialInternetOrNetworkingRelatedAttack"/>
               <rel:partOf
rdf:resource="http://www.pace.edu/cyber#CyberSecurityConcept"/>
               <rel:causedBy rdf:resource="http://www.pace.edu/cyber#ThreatVector"/>

    </owl:Class>

    <!-- http://www.pace.edu/cyber#ThreatActor -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#ThreatActor">
        <rel:laymanTermIs rdf:resource="http://www.pace.edu/cyber#Hacker"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#ThreatMitigation -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#ThreatMitigation">
    </owl:Class>

    <!-- http://www.pace.edu/cyber#ThreatType -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#ThreatType">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#Threat"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#ThreatVector -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#ThreatVector">
        <rel:technicalDefinitionIs
rdf:resource="http://www.pace.edu/cyber#UnAuthorizedPathToCyberInfrastructure"/>
        <rel:laymanDefinitionIs
rdf:resource="http://www.pace.edu/cyber#UnAuthorizedAccessToDataViaInternet"/>
               <rel:partOf
rdf:resource="http://www.pace.edu/cyber#CyberSecurityConcept"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#Trojan -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Trojan">
        <rel:isExampleOf
rdf:resource="http://www.pace.edu/cyber#AdvancedPersistentThreat"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#UnAuthorizedAccessToDataViaInternet -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#UnAuthorizedAccessToDataViaInternet">
    </owl:Class>

    <!-- http://www.pace.edu/cyber#UnAuthorizedPathToCyberInfrastructure -->

    <owl:Class
rdf:about="http://www.pace.edu/cyber#UnAuthorizedPathToCyberInfrastructure">
    </owl:Class>

    <!-- http://www.pace.edu/cyber#UnAuthorizedPersonGainAccessToNetWorkToStealData -->

    <owl:Class
rdf:about="http://www.pace.edu/cyber#UnAuthorizedPersonGainAccessToNetWorkToStealData">
    </owl:Class>

    <!-- http://www.pace.edu/cyber#Usb -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Usb">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatVector"/>
    </owl:Class>
```

```
    <!-- http://www.pace.edu/cyber#Virus -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Virus">
        <rel:areAlsoCalled rdf:resource="http://www.pace.edu/cyber#Malware"/>
                <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatType"/>
    </owl:Class>

    <!--
http://www.pace.edu/cyber#VulnerabilityCreatedAsAresultOfInsuficientMemoryforProgramInput
-->

    <owl:Class
rdf:about="http://www.pace.edu/cyber#VulnerabilityCreatedAsAresultOfInsuficientMemoryforP
rogramInput">
        <rel:isExampleOf rdf:resource="http://www.pace.edu/cyber#BufferOverFlow"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#Web -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Web">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatType"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#Worm -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Worm">
        <rel:partOf rdf:resource="http://www.pace.edu/cyber#ThreatType"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#ZeroDayVulnerability -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#ZeroDayVulnerability">
        <rel:isExampleOf
rdf:resource="http://www.pace.edu/cyber#ApplicationLayerAttack"/>
    </owl:Class>

    <!-- http://www.pace.edu/cyber#Zombie -->

    <owl:Class rdf:about="http://www.pace.edu/cyber#Zombie">
        <rel:isExampleOf
rdf:resource="http://www.pace.edu/cyber#AdvancedPersistentThreat"/>
    </owl:Class>
</rdf:RDF>

<!-- Generated by the OWL API (version 3.5.1) http://owlapi.sourceforge.net -->
```

Figure 97:“cyberSecurityCommunications.owl”

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rel "http://www.pace.edu/rel-syntax-ns#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>
<rdf:RDF xmlns="http://www.pace.edu/country-71#"
     xml:base="http://www.pace.edu/country-71"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rel="http://www.pace.edu/rel-syntax-ns#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Ontology rdf:about="http://www.pace.edu/country-71"/>
```

```
    <!--
    // Relations
     -->
       <rel:NewRelation rdf:about="http://www.pace.edu/country-71#partOf"/>
       <rel:NewRelation rdf:about="http://www.pace.edu/country-71#continentFor"/>
    <!--
    // Classes
     -->
    <!-- http://www.pace.edu/country-71#England -->

    <owl:Class rdf:about="http://www.pace.edu/country-71#England">
        <rel:partOf rdf:resource="http://www.pace.edu/country-71#Europe"/>
    </owl:Class>
    <!-- http://www.pace.edu/country-71#Europe -->

    <owl:Class rdf:about="http://www.pace.edu/country-71#Europe">
        <rel:continentFor rdf:resource="http://www.pace.edu/country-71#England"/>
    </owl:Class>
</rdf:RDF>
<!-- Generated by the OWL API (version 3.5.1) http://owlapi.sourceforge.net -->
```

Figure 98: “europe.owl” KG

References

[1] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. Presented at the Padua workshop on Formal Ontology, March 1993, later published in *International Journal of Human-Computer Studies*, Vol. 43, Issues 4-5, November 1995, pp. 907-928.

[2] http://www.nsf.gov/awardsearch/showAward?AWD_ID=1021975.

[3] http://its2014.its-conferences.com/.

[4] The Stanford Encyclopedia of Philosophy, Alan Turing, First published Mon Jun 3, 2002; substantive revision Mon Sep 30, 2013

[5] Mark Urban-Lurain, "Intelligent Tutoring Systems: An Historic Review in the Context of the Development of Artificial Intelligence and Educational Psychology," College of Engineering Michigan State University, http://www.cse.msu.edu/rgroups/cse101/ITS/its.htm#_Toc355707494

[6] Intelligent Tutoring Systems, Chapter 37 / Corbett, Koedinger & Anderson / Chapter 37 (Original pp 849-874) 14 retrieved May 21, 2012 from http://act-r.psy.cmu.edu/papers/173/Chapter_37_Intelligent_Tutoring_Systems.pdf

[7] Beal, C. R., Beck, J., & Woolf, B. (1998). Impact of intelligent computer instruction on girls' math self concept and beliefs in the value of math. Paper presented at the annual meeting of the American Educational Research Association.

[8] Keles, A., Ocak, R., Keles, A., & Gulcu A. (2009). ZOSMAT: Web-based Intelligent Tutoring System for Teaching-Learning Process. [Elsevier.]. Expert Systems with Applications , 36 , 1229-1239.

[9] http://www.cs.iit.edu/~circsim/

[10] Lajoie, S. P., & Lesgold, A. (1989). Apprenticeship training in the workplace: Computer coached practice environment as a new form of apprenticeship. Machine-Mediated Learning , 3 , 7-28

[11] Eliot, C., & Woolf, B. (1994). Reasoning about the user within a simulation-based real-time training system. In Proceedings of the fourth international conference on user modeling , 121-126.

[12] International Journal of Human-Computer Studies, Volume 43, Issues 5–6, November 1995, Pages 907–928

[13] Liyang Yu, Introduction to the Semantic Web and Semantic Web Services, 2007

[14] Roy Ladner, Frederick E Petry, Net Centric Approaches to Intelligence and National Security, Page 3, 2005

[15] Ontology for Biomedical Investigations, http://obi-ontology.org/page/

[16] Semantic System Biology, http://www.cellcycleontology.org/

[17] Uhr, L. (1969). Teaching machine programs that generate problems as a function of interaction with students. Proceedings of the 24th National Confernece, 125-134.

[18] Harry Halpin,The Semantic Web: The Origins of Artificial Intelligence Redux, ICCS, School of Informatics,University of Edinburgh

[19] http://webfoundation.org/about/vision/history-of-the-web/

[20] Berners-Lee,T Hendler,J and Lassila,O, 2001 the Semantic Web, Sceintific American,284:34-43

[21] Dr Lixin Tao, Pace University, Chairperson, Computer Science Department, Westchester, Chair of Ph.D. in Computer Science Program, Chair of Doctorate Professional Studies (DPS) in Computing Program

[22] http://www.w3.org/TR/2004/REC-owl-features-20040210/#s1.2

[23] Outline for a Theory of Intelligence, James S. Albus, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, VOL. 21, NO. 3, MAY/JUNE 1991

[24] Resource Description Framework (RDF):Concepts and Abstract Syntax, W3C Recommendation 10 February 2004

[25] An Introduction To the OWL Web Ontology Language, Jeff Heflin, Lehigh University

[26] RDF Vocabulary Description Language 1.0: RDF Schema, W3C, http://www.w3.org/TR/rdf-schema

[27] Rule-Based Inference Systems Richard O. Duda, Peter E. Hart, Ntis J. Nilsson, and Georgia L. Sutherland Stanford Research Institute Menlo Park, CA 94025

[28] https://jena.apache.org/documentation/inference/

[29] Logic based Knowledge Representation, Franz Baader, LuFg Theoretical Computer Science, RWTH, Aachen, AhonstraBe, 55, 52074 Aachen Germany

[30] Framework for Representing Knowledge Marvin Minsky MIT-AI Laboratory Memo 306, June, 1974

[31] Nilsson, N.J., Logic and artificial intelligence, Artificial Intelligence 47 (1990) 31-56

[32] http://www.hermit-reasoner.com/

[33] http://www.gao.gov/products/GAO-12-666T

[34] https://www.w3.org/2005/rules/wg/charter.html

[35] https://www.w3.org/Submission/SWRL/

[36] Semantic Web Programming, John Hebeler, Mathew Fishert, Ryan Blake, Andrew Perez-Lopez

[37] http://www.daml.org/2003/11/swrl/examples.html

[38] J. Quantz and B. Schmitz, Knowledge-based disambiguation for machine translation. Minds and Machines, 4:39-57,1994

[39] J.R. Wright, E.S. Weixelbaum, K. Brown, G.T. Vesonder, S.R. Palmer, J.T. Berman, and H.H. Moore. A knowledge-based configurator that supports sales, engineering, and manufacturing at AT&T network systems. AI Magazine, 14(3):69-80, 1993

[40] P.Devanbu, R. J. Brachman, P.G. Selfridge, and B.W. Ballard. LASSIE: A knowledge-based software information system. Communications of the ACM, 34(5):34-49, 199

[41] M. Buchheit, M. Jeusfeld, W. Nutt, and M Staudt. Subsumption of queries to object-oriented databases. Information Systems, 19(1):33-54,1994.

[42] J Koehler. An application of terminological logics to case-based reasoning. In Proceeding of the fourth International Conference on Principles of Knowledge Representation and Reasoning, KR'94, pages 351 362, Bonn, Germany, 1994. Morgan Kaufmann.

[43] http://plato.stanford.edu/entries/logic-modal/

[44] B.F. Chellas. Modal Logic: An Introduction. Cambridge University Press, Cambridge, UK. 1980

[45] G.E. Hughes and M.J. Cresswell. A Companion to Modal Logic. Methuen & Co., London, 1984

[46] D.M. Gabbay, C.J. Hogger, and J.A. Robison, editors. Hand-book of logic in Artificial Intelligence and Logic Programming, Vol.3: Nonmonotonic Reasoning and Uncertain Reasoning. Oxford University Press, Oxford. UK, 1994

[47] G. Brewka. Nonmonotonic Reasoning: Logical Foundations of Commonsense. Cambridge University Press, Cambridge, UK, 1991

[48] G. Brewka, J. Dx, and K. Konolige. Nonmonotonic Reasoning: An Overview. CSLI Publications, Center for the Study of Language and Information, Stanford, Cal., 1997

[49] A Practical Guide To Building OWL Ontologies Using Protégé 5. A Revision of Matthew Horridge's "A Practical Guide To Building Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition 1.3" for Protégé 5

[50] http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html
[51] https://www.w3.org/2001/sw/wiki/OWL/Implementations
[52] http://fowl.sourceforge.net/
[53] http://protege.stanford.edu/about.php
[54] A Practical Guide To Building OWL Ontologies Using Protége 4 and CO-ODE Tools Edition 1.2, The University Of Manchester, by Matthew Horridge, March 13, 2009
[55] https://jena.apache.org/about_jena/about.html
[56] https://jena.apache.org/
[57] http://searchengineland.com/library/google/google-knowledge-graph
[58] http://searchengineland.com/google-launches-knowledge-graph-search-api-promises-to-close-freebase-api-in-future-238949
[59] http://www.seoskeptic.com/google-releases-knowledge-graph-api/
[60] https://www.w3.org/RDF/Validator/rdfval
[61] https://www.w3.org/RDF/Validator/documentation
[62] NF Noy, DL McGuinness, Ontology development 101: A guide to creating your first ontology, 2001
[63] https://jena.apache.org/documentation/fuseki2/
[64] https://jena.apache.org/documentation/query/
[65] https://jena.apache.org/documentation/tdb/
[66] https://jena.apache.org/documentation/ontology/#general-concepts
[67] https://jena.apache.org/documentation/inference/index.html
[68] https://www.w3.org/TR/1999/REC-xml-names-19990114/
[69] https://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/#section-Syntax-intro
[70] K. Patel, I. Dube, L. Tao, and N. Jiang. Extending OWL to support custom relations. In IEEE 2nd International Conference on Cyber Security and Cloud Computing, pages 494–499, New York, NY, USA, 2015. IEEE.
[71] https://www.w3.org/TR/owl-ref/#Sublanguages
[72] https://www.w3.org/TR/owl-guide/